NANDRAD Model Reference

Andreas Nicolai <andreas.nicolai@tu-dresden.de>, Dirk Weiß, Stephan Hirth, Katja Tribulowski

Version 1.0.0, July 2020

Table of Contents

1. Overview

This document contains a description of the various implemented models and the parametrization in the NANDRAD project file. It is primarily an input reference.

The section Project File Structure contains an overview of the project file structure, with references to the individual documentation sections. This is a good start to get an overview of the NANDRAD project specification.

The Tutorials chapter contains various tutorials that illustrate manual creation of project files with simple examples.

2. NANDRAD Input and Project File Reference

2.1. Project File Structure

The NANDRAD project specification is stored in an XML-file with the extension `nandrad`. The principle structure of the file looks like:

```
<?xml version="1.0" encoding="UTF-8" ?>
<NandradProject fileVersion="2.0">
  <!-- optional DirectoryPlaceholders section-->
  <DirectoryPlaceholders>...</DirectoryPlaceholders>

    <!-- the actual project specification -->
  <Project>
      <ProjectInfo>...</ProjectInfo>
      <Location>...</Location>
      <SimulationParameter>...</SimulationParameter>
      <SolverParameter>...</SolverParameter>
      <Zones>...</Zones>
      <ConstructionInstances>...</ConstructionInstances>
      <ConstructionTypes>...</ConstructionTypes>
      <Materials>...</Materials>
      <Models>...</Models>
      <Schedules>...</Schedules>
      <Outputs>...</Outputs>
      <ObjectLists>...</ObjectLists>
  </Project>
</NandradProject>
```

The optional `DirectoryPlaceholders` can be used to define relative path placeholders to be used for externally referenced files (see section Path Placeholders).

All project data is enclosed in the `<Project>` tag.

A project file may contain the following child tags (order is arbitrary):

Child tag	Description
`ProjectInfo`	General project meta information → Project Information
`Location`	Climatic data and location settings → Climatic loads
`SimulationParameter`	Simulation model parameters → [simulation_parameters]

Child tag	Description
SolverParameter	Numerical solver settings and performance options → [solver_parameters]
Zones	Zone specifications → [zones]
ConstructionInstances	Building components and boundary conditions → [construction_instances]
ConstructionTypes	Definition of multi-layered constructions → [construction_types]
Materials	Material properties → [materials]
Models	Model parameter blocks → [models]
Schedules	Definition of scheduled parameters → [schedules]
Outputs	Output definitions → Outputs/Results
ObjectLists	Definition of object lists/object reference groups → [object_lists]

2.2. Basic Data Types in NANDRAD Project File Specification

Within the various specificiation sections of the project file some basic data types / xml-tags are frequently used. The rules for specifying these parameters are defined below.

2.2.1. IBK:Parameter

An XML tag with name IBK:Parameter defines a floating point value parameter, identified by a name and physical unit (mandatory XML-attributes name and unit). The value of the xml tag is the actual parameter value.

Example 1. Parameters with different units

```
<IBK:Parameter name="Volume" unit="m3">30</IBK:Parameter>
<IBK:Parameter name="Temperature" unit="C">20</IBK:Parameter>
<IBK:Parameter name="Temperature" unit="K">293.15</IBK:Parameter>
<!-- unitless parameters take the --- unit -->
<IBK:Parameter name="RelTol" unit="---">0.7</IBK:Parameter>
```

The units must be selected from the global unit list, see section Unit Definitions. Not defining a parameter will mark it as *missing*, which means that either a default value is used or - in case of mandatory user parameters - an error is raised.

2.2.2. IBK:IntPara

Used for whole number parameters. Mandatory attribute name identifies the parameter. XML tag value is the parameter value. Not defining a parameter will mark it as *missing*, which means that either a default value is used or - in case of mandatory user parameters - an error is raised.

Example 2. Whole number parameter definition

```
<IBK:IntPara name="DiscMaxElementsPerLayer">30</IBK:IntPara>
```

2.2.3. IBK:Flag

Used for flags. Mandatory attribute `name` identifies the flag. Not defining a flag will mark it as *missing*, which means that either a default value is used or - in case of mandatory user parameters - an error is raised.

Example 3. Flag definition

```
<IBK:Flag name="EnableCyclicSchedules">true</IBK:Flag>
```

Recognized values for flag parameters are `true` and `1` or `false` and `0`.

2.2.4. IBK:LinearSpline

A linear spline is effectively a data table of x and y values, where x values are strictly monotonically increasing values. Mandatory attribute `name` identifies the linear spline parameter. The child tags `X` and `Y` hold the actual values, with mandatory attribute `unit` defining the respective value unit. Number of x and y values must match.

Example 4. Linear spline parameter definition

```
<IBK:LinearSpline name="ThermalLoad">
  <X unit="h">0    6    8   10   17   18  19  20</X>
  <Y unit="W">0  0.5  0.8  1.0  0.7  0.6 0.5   0</Y>
</IBK:LinearSpline>
```

2.3. Path Placeholders

In some parts of the NANDRAD project file, external files are referenced (for example climate data files, see Climate Data Files). To simplify exchange of projects or reference data files in common database directories, it is possible to use path placeholders in file paths.

For example, you can define `${MyDatabase}` to be `/home/sim/climate_DB` and then in your project reference a climate data file via `${MyDatabase}/ClimateData.epw`.

These mapping of the placeholders is done early in the project file, so when exchanging project files between computers, you may easily modify the placeholder paths to the directories on the local machine without any further changes in the project file.

The individual path placeholders are defined in the `DirectoryPlaceholders`:

Example 5. Custom directory placeholders

```
<DirectoryPlaceholders>
  <Placeholder name="Climate DB">/home/sim/climate_DB</Placeholder>
  <Placeholder name="DataFiles">/home/sim/data</Placeholder>
</DirectoryPlaceholders>
```

There is one builtin-placeholder `${Project Directory}` that will be automatically defined with the path to the directory of the project file.

2.4. Project Information

This section contains change times/dates and a brief description of the project.

2.5. Zones

In order to model buildings, it is necessary to define the individual rooms with the relevant parameters. A zone defines a thermal zone/room with a single air temperature.

The class Zone stores all properties needed to compute zone temperature from energy density (the conserved quantity).

Needed for the calculation is either the floor area and the height, or the zone volume. If all parameters are given, the volume property is computed from floor area and height. Zones can be either Constant or Active. For constant zones, the temperature is assumed to be fixed/predefined whereas in Active zones the temperature is computed (i.e. included in the model's unknowns). A constant zone only needs the temperature parameter. If an Active zone has a temperature parameter, this is used as initial condition.

Example 6. Zonedefinition

```
<Zones>
  <Zone id="1" displayName="Var01" type="Active">
    <IBK:Parameter name="Area" unit="m2">10</IBK:Parameter>
    <IBK:Parameter name="Volume" unit="m3">30</IBK:Parameter>
  </Zone>
</Zones>
```

Inside the XML-Tag named `Zones` each zone starts with the XML-Tag `Zone`. The following XML-Attributes need to be defined:

```
<Zone id="1" displayName="Var01" type="Active">
```

Attribute	Description	Format	usage
id	Identifier of the Zone	positive Integer (> 0)	*required*
displayName	Display Name of the Zone. Is needed to find the Zone in the Data Model and in Outputs more easily.	string	*optional*

Attribute	Description	Format	usage
type	Defines whether zone is balanced and included in equation system. • `Constant` *as zone with constant/predefined temperatures. (schedule)* • `Active` *as zone described by a temperature node in space* • `Ground` *as ground zone (calculates temperature based on standard)*	string	*optional*

The following XML-Tags named `IBK:Parameters` with the XML-Attributes `name` and `unit` with the following entries can be defined:

```
<IBK:Parameter name="Area" unit="m2">10</IBK:Parameter>
```

Table 1. Zone Parameters that can be set as **IBK:Parameter** *with the Attributes* `name` *and* `unit`

`name`	`unit`	Description	Format	usage
Volume	m3	Zone air volume	positive double (> 0.0)	*required*
Area	m2	net usage area of the ground floor (for area-related outputs and loads)	positive double (> 0.0)	*optional*
Temperature	C	Additional heat capacity	0...1	*optional*
CO2Concentration	g/m3	CO2 concentration of the zone, if set `constant`	0...1	*optional*
RelativeHumidity	%	Temperature of the zone, if set `constant`	0...1	*optional*

2.6. Constructioninstances

The construction instances define construction-specific parameters required by several models.

The construction instances are defined inside the Sections starting with an XML-Tag `ConstructionInstances`. Inside the Section each Construction instance starts with the XML-Tag named `ConstructionInstance` with the XML-Attributes `id` and `displayName`. Inside that it is necessary to specify the interfaces with the XML-Tag named `InterfaceA` and `InterfaceB`. Finally the Interfaces with the XML-Tag `InterfaceA` and `InterfaceB` need to be defined with the XML-Attributes `id` and `zoneId`. In the following it is described in detail.

Attribute	Description	Format	usage
id	Identifier of the Construction Instance	positive Integer (> 0)	*required*
displayName	Display Name of the Construction Instance. Is needed to find the Construction Instance in the Data Model and in Outputs more easily.	string	*optional*

The construction instance has the following `required subtag`:

- `constructionTypeId` - unique Id that defines the construction type of the construction instance

Example:

```
<ConstructionTypeId>10005</ConstructionTypeId>
```

The following XML-Tags named `IBK:Parameters` with the XML-Attributes `name` and `unit` with the following entries can be defined:

Table 2. Zone Parameters that can be set as **IBK:Parameter** *with the Attributes* `name` *and* `unit`

`name`	`unit`	**Description**	**Format**	**usage**
Orientation	m2	Orientation of the wall	positive double (> 0.0)	*required*
Inclination	m3	Inclination of the wall	positive double (> 0.0)	*required*
Area	C	Gross area of the wall	0...1	*required*

Example:

```
<ConstructionInstances>
  <!-- Surface Var 01 -->
  <ConstructionInstance id="1" displayName="All Surfaces Var01">
    <ConstructionTypeId>10005</ConstructionTypeId>
    <IBK:Parameter name="Area" unit="m2">62</IBK:Parameter>
    <InterfaceA id="10" zoneId="1">
      <!--Interface to 'Room'-->
      <InterfaceHeatConduction modelType="Constant">
        <IBK:Parameter name="HeatTransferCoefficient" unit="W/m2K">2.5</IBK:Parameter>
      </InterfaceHeatConduction>
    </InterfaceA>
    <InterfaceB id="11" zoneId="0">
      <!--Interface to outside-->
      <InterfaceHeatConduction modelType="Constant">
        <IBK:Parameter name="HeatTransferCoefficient" unit="W/m2K">8</IBK:Parameter>
      </InterfaceHeatConduction>
    </InterfaceB>
  </ConstructionInstance>
</ConstructionInstances>
```

2.7. Interfaces (construction boundary conditions)

Interfaces are defining boundary conditions and parameters for the two surfaces of a constructions instance.

The Interfaces can have the following `subtags`:

2.7.1. Heat Conduction

- `InterfaceHeatConduction`

```
<InterfaceHeatConduction modelType="Constant">
  <IBK:Parameter name="HeatTransferCoefficient" unit="W/m2K">2.5</IBK:Parameter>
</InterfaceHeatConduction>
```

The InterfaceHeatConduction needs to be defined with the following XML-Attribute `modelType`.`Constant` *as constant model.*

Table 3. Parameters for the InterfaceHeatConduction-Tag

Attribute	Description	Format	usage
modelType	Sets the type of the heat conduction model • `Constant` - Constant model used (currently the only option)	positive Integer (> 0)	*required*

The XML-Tags named `IBK:Parameters` with the XML-Attributes `name` and `unit` with the following entries can be defined:

Table 4. Zone Parameters that can be set as **IBK:Parameter** *with the Attributes* `name` *and* `unit`

`name`	`unit`	Description	Format	usage
HeatTransferCoefficient	W/m2	Constant heat transfer coefficient	positive double (> 0.0)	*required*

2.7.2. Solar Absorption

- `InterfaceSolarAbsorption`

```
<InterfaceSolarAbsorption modelType="Constant">
  <IBK:Parameter name="AbsorptionCoefficient" unit="---">0.6</IBK:Parameter>
</InterfaceHeatConduction>
```

The InterfaceSolarAbsorption needs to be defined with the following XML-Attribute `modelType`.`Constant` *as constant model.*

Table 5. Parameters for the InterfaceSolarAbsorption-Tag

Attribute	Description	Format	usage
modelType	Sets the type of the heat conduction model • `Constant` - constant model used (currently the only option)	positive Integer (> 0)	*required*

The XML-Tags named `IBK:Parameters` with the XML-Attributes `name` and `unit` with the following entries can be defined:

Table 6. Zone Parameters that can be set as **IBK:Parameter** *with the Attributes* `name` *and* `unit`

name	unit	Description	Format	usage
AbsorptionCoefficient	m2	Constant Absorption coefficient	0...1	*required*

2.7.3. Long Wave Radiation

Defined inside the XML-tag `InterfaceLongWaveEmission`

```
<InterfaceLongWaveEmission modelType="Constant">
  <IBK:Parameter name="Emissivity" unit="---">0.9</IBK:Parameter>
</InterfaceLongWaveEmission>
```

Table 7. Parameters for the InterfaceLongWaveEmission-Tag

Attribute	Description	Format	usage
modelType	Sets the type of the heat conduction model • `Constant` - constant model used (currently the only option)	positive Integer (> 0)	*required*

The XML-Tags named `IBK:Parameters` with the XML-Attributes `name` and `unit` with the following entries can be defined:

Table 8. Zone Parameters that can be set as **IBK:Parameter** *with the Attributes* `name` *and* `unit`

name	unit	Description	Format	usage
Emissivity	m2	Constant Absorption coefficient	0...1	*required*

2.7.4. Hygrothermic

Defined inside the XML-tag `InterfaceVaporDiffusion`

```
<InterfaceVaporDiffusion modelType="Constant">
  <IBK:Parameter name="VaporTransferCoefficient" unit="s/m">1</IBK:Parameter>
</InterfaceVaporDiffusion>
```

Table 9. Parameters for the InterfaceVaporDiffusion-Tag

Attribute	Description	Format	usage
modelType	Sets the type of the heat conduction model • `Constant` - constant model used (currently the only option)	positive Integer (> 0)	*required*

The XML-Tags named `IBK:Parameters` with the XML-Attributes `name` and `unit` with the following entries can be defined:

Table 10. Zone Parameters that can be set as **IBK:Parameter** *with the Attributes* `name` *and* `unit`

name	unit	Description	Format	usage
VaporTransferCoefficient	s/m	Vapor Transfer Coefficient	positive Double (> 0.0)	*required*

2.7.5. Air Flow

Defined inside the XML-tag `InterfaceAirFlow`

```
<InterfaceAirFlow modelType="Constant">
  <IBK:Parameter name="PressureCoefficient" unit="---">0.6</IBK:Parameter>
</InterfaceAirFlow>
```

Table 11. Parameters for the InterfaceAirFlow-Tag

Attribute	Description	Format	usage
modelType	Sets the type of the heat conduction model • `Constant` - constant model used (currently the only option)	positive Integer (> 0)	*required*

The XML-Tags named `IBK:Parameters` with the XML-Attributes `name` and `unit` with the following entries can be defined:

Table 12. Pressure Coefficient Parameters that can be set as **IBK:Parameter** *with the Attributes `name` and `unit`*

name	unit	Description	Format	usage
PressureCoefficient	---	Pressure Coefficient	0...1	*required*

2.8. Ambient climate boundary conditions

2.9. Interface between constructions and zones (internal boundary conditions)

2.10. Material and Construction Lists

2.10.1. Overview

In order to model building components such as Walls, Ceilings and Floors, etc. it is necessary to define some parameters for the Materials and then the Construction with the reference to an existing material.

[Mat ConstInst] | *Mat_ConstInst.png*

2.10.2. Materials

In the NANDRAD-Model the Materials-Section starts with an XML-Tag named `Materials`. In this each Material starts with an XML-Tag named `Material` with two XML-Attributes `id` and `displayName`. Concerning the Material Parameters such as Density, Heat Capacity and Conductivity they need to be defined within the XML-Tag `IBK:Parameter` with the

XML-Attributes `name` and `unit`. In the following it is described in detail.

Example 7. Materials with Material Layers and Parameters

```
<Materials>
  <Material id="1001" displayName="Brick">
    <IBK:Parameter name="Density" unit="kg/m3">2000</IBK:Parameter>
    <IBK:Parameter name="HeatCapacity" unit="J/kgK">1000</IBK:Parameter>
    <IBK:Parameter name="Conductivity" unit="W/mK">1.2</IBK:Parameter>
  </Material>
  <Material id="1004" displayName="Good Insulation">
    <IBK:Parameter name="Density" unit="kg/m3">50</IBK:Parameter>
    <IBK:Parameter name="HeatCapacity" unit="J/kgK">1000</IBK:Parameter>
    <IBK:Parameter name="Conductivity" unit="W/mK">0.02</IBK:Parameter>
  </Material>
</Materials
```

Inside the XML-Tag named `Materials` each Material starts with the XML-Tag `Material`. The following XML-Attributes need to be defined:

Attribute	Description	Format	usage
id	Dry density of the material.	positive Integer (> 0)	*required*
displayName	Specific heat capacity of the material.	string	*optional*

The following XML-Tags named `IBK:Parameters` with the XML-Attributes `name` and `unit` with the following entries need to be defined holding double values:

`name`	unit per default	Description	Format	usage
Density	kg/m3	Dry density of the material.	double	*required*
HeatCapacity	J/kgK	Specific heat capacity of the material.	double	*required*
Conductivity	W/mK	Thermal conductivity of the dry material.	double	*required*

2.10.3. Construction Types

The Construction is defined inside the Sections starting with an XML-Tag `Construction Types`. Inside the Section each Construction start with the XML-Tag named `Construction Type` with the XML-Attributes `id` and `displayName`. Inside that it is necessary to specify the Material Layers with the XML-Tag named `MaterialLayers` and finally the Material Layer with the XML-Tag `MaterialLayer` with the XML-Attributes `thickness` and `matId`. The `MaterialLayer` is always an empty Object since all needed Attributes are defined as XML-Attributes as mentioned before. In the following it is described in detail.

Example 8. Construction Types with References to Material Objects

```
<ConstructionTypes>
  <ConstructionType id="10005" displayName="Test Construction">
    <MaterialLayers>
      <MaterialLayer thickness="0.2" matId="1001" /> <!-- room side -->
      <MaterialLayer thickness="0.3" matId="1004" />
    </MaterialLayers>
  </ConstructionType>
</ConstructionTypes>
```

Inside the XML-Tag named `ConstructionTypes` each Construction starts with the XML-Tag `ConstructionType`. The following XML-Attributes need to be defined inside of this:

Attribute	Description	Format	usage
id	Unique id number.	positive Integer (> 0)	*required*
displayName	name of construction.	string	*optional*

Furthermore each defined Construction has Material Layers encapsulated within the XML-Tag `MaterialLayers`. Each Material Layer needs the be defined with the XML-Tag named `MaterialLayer`. This Tag needs the XML-Attributes `thickness` and `matId`:

XML-Attribute	Description	Format	usage
`thickness`	defines the Thickness of the Layer, per Default in `m`	positive Double (> 0.0)	*required*
matId	refers to the a Material with the `id` defined in the `Materials` Section,	string	*required*

2.11. Climatic loads

2.11.1. Overview

Climatic loads in NANDRAD are provided by means of climate data files. The climatic loads model is a purely time-dependent model without other input dependencies. For solar radiation calculation, it needs information on the building location, and also the orientation and inclination of the various surfaces.

2.11.2. Specification

Information about location and climate data is stored in the `Location` section of the project file:

```
<Location>
  <IBK:Parameter name="Latitude" unit="Deg">51</IBK:Parameter>
  <IBK:Parameter name="Longitude" unit="Deg">13</IBK:Parameter>
  <IBK:Parameter name="Albedo" unit="-">0.2</IBK:Parameter>
  <IBK:Parameter name="Altitude" unit="m">100</IBK:Parameter>
  <IBK:Flag name="PerezDiffuseRadiationModel">false</IBK:Flag>
  <ClimateFileName>${Project Directory}/climate/GER_Potsdam_2017.c6b</ClimateFileName>
</Location>
```

Parameters (see section Basic Data Types in NANDRAD Project File Specification for into on types):

- The parameter `Albedo` is required and used for diffuse solar radiation calculation (see Solar Radiation Calculation). Valid value range in [0,1].
- Parameters `Latitude` and `Longitude` are optional. If present, they override the location parameters stored therein (see Building/Station location).
- `Altitude` is optional and later needed for specific altitude-related parameters (**TODO**).
- The optional flag `PerezDiffuseRadiationModel` defines whether to use the Perez-Model for diffuse solar radiation calculation (when `true`), or the isotropic radiation model (`false`). Default setting is `false`.
- The `<ClimateFileName>` tag defines the path to the climate data file.

Climate Data Files

Currently, `c6b`, `wac` and `epw` files are supported (see also help for the CCM-Editor tool).

You need to specify the path to the climate data file in the `<ClimateFileName>` tag. Hereby, you can specify an absolute or relative path.

If a relative path is privided, it will be resolved using the current working directory as reference. For example, if you have specified

```
<ClimateFileName>GER_Potsdam_2017.c6b</ClimateFileName>
```

and the solver is run from the directory `/home/user/sim/Project1`, the climate data file will be searched in `/home/user/sim/Project1/GER_Potsdam_2017.c6b`. If the solver is run from a different directory, the referenced climate data file won't be found and an error message is raised.

To avoid this problem, you may specify directory placeholders to locate the climate data file *relative* to the project file's location. The builtin path placeholder `${Project Directory}` will be replaced by the directory the project file is located in. Use the placeholder just as a regular directory part, for example:

```
<ClimateFileName>${Project Directory}/climate/GER_Potsdam_2017.c6b</ClimateFileName>
```

It is possible to define custom placeholders in the project for all externally referenced files, see Path Placeholders.

Building/Station location

Climate data files contain information on latitude and longitude of the weather station, which is also taken to be the location of the building. This ensures that simulated time and position of the sun matches.

It is also possible to define latitude/longitude in the project file. If these parameters are specified in the project file, always **both** parameters must be given (and be valid) and then the these parameters from the project file are used instead of the climate data file location parameters.

By specifying latitude different from the climatic station, the computed sun position may no longer correspond to the sun position at the weather station, thus yielding probably wrong solar radiation loads.

Valid value range for `Latitude` is [-90,90] degrees (positive values are northern hemisphere), for `Longitude` it is [-180,180] degrees (positive values are east of Greenwich).

Additional radiation sensors

It is possible to specify additional planes (sensors) to generate solar radiation load outputs. This is done by specifying a `Sensor` definition.

```
<Location>
    ....
    <Sensors>
    <!-- Flat roof -->
    <Sensor id="1">
      <IBK:Parameter name="Orientation" unit="Deg">0</IBK:Parameter>
      <IBK:Parameter name="Inclination" unit="Deg">0</IBK:Parameter>
    </Sensor>
    <!-- North Wall 90 -->
    <Sensor id="2">
      <IBK:Parameter name="Orientation" unit="Deg">0</IBK:Parameter>
      <IBK:Parameter name="Inclination" unit="Deg">90</IBK:Parameter>
    </Sensor>
        ...
    </Sensors>
</Location>
```

A sensor must be given a unique ID number and the mandatory parameters `Orientation` and `Inclination` (see section << construction_interfaces>> for details on their definition).

For each sensor 4 output quantities are generated: * `DirectSWRadOnPlane[<sensor id>]` - direct solar radiation intensity on plane in [W/m2] * `DiffuseSWRadOnPlane[<sensor id>]` - diffuse solar radiation intensity on plane in [W/m2] * `GlobalSWRadOnPlane[<sensor id>]` - global radiation intensity on plane in [W/m2] (the sum of the former two) * `IncidenceAngleOnPlane[<sensor id>]` - the incidence angle onto the plane in [Deg] (0° when sun ray is perpendicular to the plane, 90° when ray is parallel to the plane or when sun is below horizon)

Example for a sensor output (see also output description in section Outputs/Results).

```
<OutputDefinitions>
    ...
    <!-- direct radiation intensive from sensor with id=2 -->
  <OutputDefinition>
    <Quantity>DirectSWRadOnPlane[2]</Quantity>
    <ObjectListName>Location</ObjectListName>
    <GridName>minutely</GridName>
  </OutputDefinition>
  <!-- incidence angle from sensor with id=42 -->
  <OutputDefinition>
    <Quantity>IncidenceAngleOnPlane[42]</Quantity>
    <ObjectListName>Location</ObjectListName>
    <GridName>minutely</GridName>
  </OutputDefinition>
    ...
</OutputDefinitions>
```

2.11.3. Solar Radiation Calculation

Solar radiation calculation follows the equations lists in section ... of the *Physical Model Reference*. The `Albedo` parameter is used in the diffuse radiation load calculation.

2.11.4. Implementation

The `Loads` model is a pre-defined model that is always evaluated first whenever the time point has changed. It does not have any other dependencies.

It provides all resulting variables as `constant` (during iteration) result variables, which can be retrieved and utilized by any other model.

With respect to solar radiation calculation, during initialization it registers all surfaces (with different orientation/inclination) and provides an ID for each surface. Then, models can request direct and diffuse radiation data, as well as incidence angle for each of the registered surfaces.

Registering surfaces

Each construction surface (interface) with outside radiation loads registers itself with the Loads object, hereby passing the interface object ID as argument and orientation/inclination of the surface. The loads object itself registers this surface with the climate calculation module (CCM) and retrieves a surface ID. This surface ID may be the same for many interface IDs.

The Loads object stores a mapping of all interface IDs to the respective surface IDs in the CCM. When requesting the result variable's memory location, this mapping is used to deliver the correct input variable reference/memory location to the interface-specific solar radiation calculation object.

2.12. Schedules

2.12.1. Overview

Schedules provide purely time-dependent quantities, similar to climatic loads.

Different to other results-producing models, schedules generate variables for sets of dependent models. As such, a schedule is formulated for an object list, which selects a set of objects taking the provided values.

For example, a schedule defines heating set points (HeatingSetPoint) for living room zones. These are selected by an object list "Living room", which selects *Zone*-type objects with a certain ID range.

2.12.2. Defining schedules

Simulation time to day type/local time mapping

Simulation time runs from t=0 over the duration of the simulation. For the lookup of schedules, this time needs to be mapped to the local (building) time.

TODO: Clarify (ticket: https://github.com/ghorwin/SIM-VICUS/issues/31) check this with the climate loads object, when cyclic is set, climatic loads **must not** be defined for continues data):

Time/day mapping in cyclic annual schedules

For cyclic schedule data, the flag "Cyclic" must be set in Schedules xml-block.

The following conventions apply:

- start year by default is 2001
- start time is given as parameter (as offset to Midnight January 1st 2001, or `"01.01.2001 00:00"`); for example, start time of `10.5 d` means simulation time 0 maps to `"10.01.2001 12:00"`
- if simulation duration exceeds 1 year, simulation time is wrapped at 365 d
- "schedule lookup time" is the same as simulation time
- leap days are never used, even if start year is set to 2000 and similar leap years
- parameter "DayOfTheWeekAtStart" indicates which day of the week corresponds to the first day of simulation (i.e. the day of start time, for example, one could specify "Wed" as day type and in the example above the 10.01. would become a wednesday)

Example:

Simulation takes 2 years, and starts in March 2nd, 12:00 (year 2003, but that is not important)

```
02.03. 12:00  -> t_start = (31+28+1)*24+12 = 1452 h = 60.5 d

t =   0 d -> t_sched = t_start + t = 60.5 d
t = 365 d -> t_sched = t_start + t = 425.5 d

t_sched > 365 ? -> t_sched = t_sched - 365 = 60.5 h

Evaluation at runtime:
scheduleData = scheduleTabulatedSplineData [t_sched=0...365 d]  -> interpolate at t_sched
```

Constructing spline data from input data

```
- loop over all days (d=0,1,...,364)
- determine day type:
  d_dayOfWeek = (startDayOffset + d) % 7 (modulo 7)

  startDayOffset = 0 for Monday, 1 for Tuesday, ... , 6 for Sunday

Example:

  d = 15  -> date = 16. January 2003
  startDayOffset = "Wed" -> 2  (1.1.2003 was a wednesday)

  d_dayOfWeek -> 15 + 2 = 17   17 % 7 = 3  -> DayType = "Thursday" (Check: 16. January 2003 was a Thursday)

- look up daily cycle:
  - find schedule (back to front) where d in range:
    - process daytypes in order Thursday, Weekdays, AllDays
      - if parameter is found in any of these days, take daily course and add to spline for this day,
      - if parameter not found, skip and search through next schedule
```

Continuous data

For continuous schedule data (i.e. flag "Cyclic" is off; meaningful when re-calculating monitored building data with real calander reference), the following procedure is used:

- start year is given as parameter
- start time is given as parameter
- flag indicates whether leap days are to be considered or not

If leap days are considered, a calender model is used to compute the actual local date and time based on given start year and start time, and also computes day of the week.

Without leap days, the following calculation is used: - simulation time is converted to date using regular 365 d years - parameter "DayOfTheWeekAtStart" indicates which day of the week corresponds to the first day of simulation

Data definition rules

A certain variable must be only defined once per object list

For example, if you have a regular daily-cycle-based schedule for "HeatingSetPoint" and zone object list "office spaces", there must not be an annual schedule for "HeatingSetPoint" and the same object list name "office spaces".

A variable must be defined unambiguously with respect to addressed object

For example, you may have a "HeatingSetPoint" for zone object list "office spaces" and this object list addresses zones with IDs 1 and 4. Now there is a second object list "all spaces", with wildcard `ID=*` (hereby addressing all zones). You **must not** define the variable "HeatingSetPoint" again for this object list, since otherwise you would get ambiguous defintions of this variable for zones 1 and 4.

Cyclic annual schedules must begin at simulation start (past the end, values are constant extrapolated)

For annual cyclic schedules, the schedule must start with time 0. For non-cyclic schedules, the schedule must start at latest at actual simulation start, so that start year ⇐ simulation start year and if same year, start time < simulation start time. Basically, the solver must be able to query a value at simulation start.

If simulation continues past the end of an annual schedule, the last value will be simply kept (constant extrapolation).

Regular schedules (based on daily cycles)

...

Annual schedules (as linearly interpolated splines)

Annual schedules are basically data tables with

2.12.3. Implementation

Schedules do not have any dependencies, and are not part of the model graph. They are updated just as climatic loads whenever time changes.

Instead of generated a (potentially large) set of variables for each object adressed by the object list, schedules provide result variable slots for each object list and scheduled quantity. The individual model instances requesting their scheduled parameters share the same variable slot.

For example, two zones of the same object list request a variable reference (pointer to variable slot) from the schedule object, and will get the same pointer for the same variable.

Schedules do not implement the regular model interfaces and are not included in the model graph. Instead, they are handled in a special way by the framework.

Variable lookup

1. Schedules define variables for object lists.
2. Object lists address a range of objects based on filter criteria, such as object reference type (e.g. Zone, ConstructionInstance, Interface), and id group/range (a set of IDs)

When a certain object (e.g. a zone with a given ID) wants to get access to a parameter defined for it, a `ValueReference` can be created with:

- reference type = `ZONE`
- id = zone-id
- variable_name = required scheduled parameter name

and the schedule object may then lookup the variable as follows:

- cycle through all known object lists (i.e. object lists used in schedule definitions)
- check if reference type matches, and if id-name is in ID group of object list
- if object list was found, resolve variable name (from enumeration `Results`)
- search map for this parameter name for a key that matches the object list's name
- if match was found, return offset/pointer to the respective result variable
- in all other cases, return nullptr

Variable lookup for outputs/lookup by schedule name

It may be possible to directly reference a scheduled parameter without going through the zone first. In this case, there is the problem, that an input reference cannot hold both quantity name **and** object list name.

With the current data structure it is not possible, to identify a quantity and objectlist by separate data members. Hence, we need to combine the information into the quantity name.

Such a reference could look like:

- reference type = `SCHEDULE` (or `OBJECT_LIST`???)
- id = 0 (unused)
- variable_name = <object list name>.<required scheduled parameter name>

For example. "All zones.HeatingSetPoint" would address the variable "HeatingSetPoint" defined for object list "All zones". Naturally, this implies that . characters are forbidden as object list or variable names.

2.12.4. Variable list

```
  /*! Available quantities from schedules.
    While this enum could be moved to NANDRAD::Schedule, we keep it here to reuse DefaultModel implementation
    for generating QuantityDescriptions.
  */
  enum Results {
    R_HeatingSetPointTemperature,     // Keyword: HeatingSetPointTemperature        [C]     'Setpoint temperature for
heating.'
    R_CoolingSetPointTemperature,     // Keyword: CoolingSetPointTemperature        [C]     'Setpoint temperature for
cooling.'
    R_AirConditionSetPointTemperature,    // Keyword: AirConditionSetPointTemperature     [C]     'Setpoint temperature
for air conditioning.'
    R_AirConditionSetPointRelativeHumidity, // Keyword: AirConditionSetPointRelativeHumidity  [%]     'Setpoint
relative humidity for air conditioning.'
    R_AirConditionSetPointMassFlux,     // Keyword: AirConditionSetPointMassFlux      [kg/s]    'Setpoint mass flux for
air conditioning.'
    R_HeatingLoad,              // Keyword: HeatingLoad               [W]     'Heating load.'
    R_ThermalLoad,              // Keyword: ThermalLoad               [W]     'Thermal load (positive or negative).'
    R_MoistureLoad,             // Keyword: MoistureLoad              [g/h]   'Moisture load.'
    R_CoolingPower,             // Keyword: CoolingPower              [W]     'Cooling power.'
    R_LightingPower,            // Keyword: LightingPower             [W]     'Lighting power.'
    R_DomesticWaterSetpointTemperature,   // Keyword: DomesticWaterSetpointTemperature    [C]     'Setpoint temperature
for domestic water.'
    R_DomesticWaterMassFlow,        // Keyword: DomesticWaterMassFlow         [kg/s]    'Domestic water demand mass
flow for the complete zone (hot water and equipment).'
    R_ThermalEnergyLossPerPerson,     // Keyword: ThermalEnergyLossPerPerson        [W/Person]  'Energy of a single
persons activities that is not available as thermal heat.'
    R_TotalEnergyProductionPerPerson,   // Keyword: TotalEnergyProductionPerPerson      [W/Person]  'Total energy
production of a single persons body at a certain activity.'
    R_MoistureReleasePerPerson,       // Keyword: MoistureReleasePerPerson        [kg/s]    'Moisture release of a
single persons body at a certain activity.'
    R_CO2EmissionPerPerson,         // Keyword: CO2EmissionPerPerson          [kg/s]    'CO2 emission mass flux of a
single person at a certain activity.'
    R_MassFluxRate,             // Keyword: MassFluxRate              [---]   'Fraction of real mass flux to maximum
mass flux for different day times.'
    R_PressureHead,             // Keyword: PressureHead              [Pa]    'Supply pressure head of a pump.'
    R_OccupancyRate,            // Keyword: OccupancyRate             [---]   'Fraction of real occupancy to maximum
occupancy for different day times.'
    R_EquipmentUtilizationRatio,      // Keyword: EquipmentUtilizationRatio       [---]   'Ratio of usage for existing
electric equipment.'
    R_LightingUtilizationRatio,       // Keyword: LightingUtilizationRatio        [---]   'Ratio of usage for
lighting.'
    R_MaximumSolarRadiationIntensity,   // Keyword: MaximumSolarRadiationIntensity      [W/m2]    'Maximum solar
radiation intensity before shading is activated.'
    R_UserVentilationAirChangeRate,     // Keyword: UserVentilationAirChangeRate      [1/h]   'Exchange rate for
natural ventilation.'
    R_UserVentilationComfortAirChangeRate,  // Keyword: UserVentilationComfortAirChangeRate   [1/h]   'Maximum air
change rate = offset for user comfort.'
    R_UserVentilationMinimumRoomTemperature,// Keyword: UserVentilationMinimumRoomTemperature [C]     'Temperature
limit over which comfort ventilation is activated.'
    R_UserVentilationMaximumRoomTemperature,// Keyword: UserVentilationMaximumRoomTemperature [C]     'Temperature
limit below which comfort ventilation is activated.'
    R_InfiltrationAirChangeRate,      // Keyword: InfiltrationAirChangeRate       [1/h]   'Exchange rate for
infiltration.'
    R_ShadingFactor,            // Keyword: ShadingFactor             [---]   'Shading factor [0...1].'
    NUM_R
  };
```

2.13. Global parameters

- parameters controlling how the model operates
- parameters controlling model accuracy
- parameters controlling performance

2.13.1. Simulation Parameters

Simulation time and absolute time reference

- how is the simulation time defined
- how is the absolute time obtained (start year and start time)
- how are cyclic annual simulations handled, how are continuous multi-year simulations handled

Example 9. Simulation Parameters

```
<SimulationParameter>
  <IBK:Parameter name="InitialTemperature" unit="C">5</IBK:Parameter>
  <IBK:IntPara name="DiscMaxElementsPerLayer">30</IBK:IntPara>
</SimulationParameter>
```

The XML-Tags named IBK:Parameter with the XML-Attributes `name` and `unit` with the following entries can be defined:

```
<IBK:Parameter name="InitialTemperature" unit="C">5</IBK:Parameter>
```

Table 13. Simulation Parameters that can be set as **IBK:Parameter** *with the Attributes* `name` *and* `unit`

name	unit	Description	Format	usage
InitialTemperature	C	Global initial temperature	positive double (> 0.0)	*optional*
InitialRelativeHumidity	%	Global initial relative humidity	0 ... 100%	*optional*
RadiationLoadFraction	---	Percentage of solar radiation gains attributed direcly to room 0..1.	0...1	*optional*
UserThermalRadiationFraction	---	Percentage of heat that is emitted by long wave radiation from persons.	0...1	*optional*
EquipmentThermalLossFraction	---	Percentage of energy from equipment load that is not available as thermal heat.	0 ... 1	*optional*
EquipmentThermalRadiationFraction	---	Percentage of heat that is emitted by long wave radiation from equipment.	0...1	*optional*
LightingVisibleRadiationFraction	---	Percentage of energy from lighting that is transformed into visible short wave radiation.	0...1	*optional*

name	unit	Description	Format	usage
LightingThermalRadiationFraction	---	Percentage of heat that is emitted by long wave radiation from lighting.	0...1	*optional*
DomesticWaterSensitiveHeatGainFraction	---	Percentage of sensitive heat from domestic water istributed towrads the room.	0...1	*optional*
AirExchangeRateN50	1/h	Air exchange rate resulting from a pressure difference of 50 Pa between inside and outside.	positive double (> 0.0)	*optional*
ShieldingCoefficient	---	Shielding coefficient for a given location and envelope type.	0 ... 1	*optional*
HeatingDesignAmbientTemperature	C	Ambient temparture for a design day. Parameter that is needed for FMU export.	positive double (> 0.0)	*optional*

The XML-Tags named **IBK:IntPara** holding integer values with the XML-Attributes `name` can be defined:

```
<IBK:IntPara name="DiscMaxElementsPerLayer">30</IBK:IntPara>
```

Table 14. Simulation Parameter that can be set as an **IBK:IntPara** *with an Attribute* `name`

name	Description	usage
StartYear	Start year of the simulation, per default set to **2001**	*optional*

The XML-Tags named **IBK:Flags** with the following XML-Attributes `name` can be defined.

```
<IBK:Flag name="DetectMaxTimeStep">true</IBK:Flag>
```

Table 15. Simulation Parameter that can be set as an **IBK:Flag** *with an Attribute* `name`

name	Description	usage
EnableMoistureBalance	Flag activating moisture balance calculation if enabled	*optional*
EnableCO2Balance	Flag activating CO2 balance calculation if enabled	*optional*
EnableJointVentilation	Flag activating ventilation through joints and openings.	*optional*
ExportClimateDataFMU	Flag activating FMU export of climate data.	*optional*

2.13.2. Solver Parameters

Hereafter all parameters that are required for the solver are described.

Example 10. Solver Parameters

```
<SolverParameter>
    <IBK:Parameter name="MaxTimeStep" unit="min">30</IBK:Parameter>
    <IBK:Parameter name="MinTimeStep" unit="s">1e-4</IBK:Parameter>
    <IBK:Parameter name="RelTol" unit="---">1e-005</IBK:Parameter>
    <IBK:Parameter name="AbsTol" unit="---">1e-006</IBK:Parameter>
    <IBK:Parameter name="NonlinSolverConvCoeff" unit="---">1e-05</IBK:Parameter>
    <IBK:Parameter name="MaxOrder" unit="---">5</IBK:Parameter>
    <IBK:Parameter name="MaxKrylovDim" unit="---">500</IBK:Parameter>
    <IBK:Parameter name="LESBandWidth" unit="---">15</IBK:Parameter>
    <IBK:Parameter name="PreBandWidth" unit="---">1</IBK:Parameter>
    <IBK:Parameter name="PreILUWidth" unit="---">1</IBK:Parameter>
    <IBK:Parameter name="DiscMinDx" unit="mm">2</IBK:Parameter>
    <IBK:Parameter name="DiscDetailLevel" unit="---">4</IBK:Parameter>
    <IBK:Flag name="DetectMaxTimeStep">true</IBK:Flag>
    <Integrator>CVODE</Integrator>
    <LESSolver>Dense</LESSolver>
    <Preconditioner>Band</Preconditioner>
</SolverParameter>
```

```
<IBK:Parameter name="MaxTimeStep" unit="min">30</IBK:Parameter>
```

Table 16. Parameters that can be set as an **IBK:Parameter** *with the Attributes* `name` *and* `unit`.

name	unit	Description	Format	initial	usage
RelTol	---	Relative tolerance for solver error check.	0...1	1E-04	*optional*
AbsTol	---	Absolute tolerance for solver error check.	0...1	1E-10	*optional*
MaxTimeStep	h	Maximum permitted time step for integration.	positive double (> 0.0)	1	*optional*
MinTimeStep	s	Minimum accepted time step, before solver aborts with error.	positive double (> 0.0)	1E-12	*optional*
InitialTimeStep	s	Initial time step size (or constant step size for ExplicitEuler integrator).	positive double (> 0.0)	0.1	*optional*
NonlinSolverConvCoeff	---	Coefficient reducing nonlinear equation solver convergence limit. Not supported by Implicit Euler.	0...1	0.1	*optional*
IterativeSolverConvCoeff	---	Coefficient reducing iterative equation solver convergence limit.	0...1	0.05	*optional*
DiscMinDx	mm	Minimum element width for wall discretization.	positive double (> 0.0)	2	*optional*

name	unit	Description	Format	initial	usage
DiscStretchFactor	---	Stretch factor for variable wall discretizations: • **0** - no disc • **1** - equidistance • > **1** - variable	positive integer (> 0)	50	*optional*
ViewfactorTileWidth	m	Maximum dimension of a tile for calculation of view factors.	positive double (> 0.0)	50	*optional*
SurfaceDiscretizationDensity	---	Number of surface discretization elements of a wall in each direction.	0...1	2	*optional*
ControlTemperatureTolerance	K	Temperature tolerance for ideal heating or cooling.	positive double (> 0.0)	1E-05	*optional*
KinsolRelTol	---	Relative tolerance for Kinsol solver.	0...1	-	*optional*
KinsolAbsTol	---	Absolute tolerance for Kinsol solver.	0...1	-	*optional*
IntegralWeightsFactor	---	Optional weighting factor for integral outputs.	0...1	1E-05	*optional*

```
<IBK:Flag name="DetectMaxTimeStep">true</IBK:Flag>
```

Table 17. Parameters set as IBK:Flag with an Attribute `name` *that enbables functionalities*

name	Description	initial	usage
DetectMaxTimeStep	Check schedules to determine minimum distances between steps and adjust MaxTimeStep.	false	*optional*
KinsolDisableLineSearch	Disable line search for steady state cycles.	false	*optional*
KinsolStrictNewton	Enable strict Newton for steady state cycles.	false	*optional*

All options for the integrator are described in the table below. The xml-tag **Integrator** contains a string to select the time integration method.

```
<Integrator>CVODE</Integrator>
```

Table 18. Integrator Parameters that are set as **Integrator**

Integrator	Description	usage
CVODE	Selects the Sundials library **CVODE**, Implicit multi-step method with adaptive time step width control and Modified Newton-Raphson for the resolution of non-linear couplings	*optional*
ExplicitEuler	Explicit Euler solver	*optional*

Integrator	Description	usage
ImplicitEuler	Implicit Euler solver with adaptive time step width control and Modified Newton-Raphson for the resolution of non-linear couplings	*optional*

```
<LESSolver>Dense</LESSolver>
```

Table 19. LESolver Parameters that are set as **LESolver**

LESolver	Description	usage
ILU	Incomplete LU preconditioner	*optional*
auto	System selects preconditioner automatically.	*optional*

```
<Preconditioner>Band</Preconditioner>
```

Table 20. Preconditioner Parameters that can be set as **Preconditioner**

Preconditioner	Description	initial	usage
PreILUWidth	Maximum level of fill-in to be used only for **ILU** preconditioner.	-	*optional*
MaxKrylovDim	Maximum dimension of Krylov subspace.	50	*optional*
MaxNonlinIter	Maximum number of nonlinear iterations.	3	*optional*
MaxOrder	Maximum order allowed for multi-step solver. Only used with **CVODE**	5	*optional*
KinsolMaxNonlinIter	Maximum nonlinear iterations for Kinsol solver.	-	*optional*
DiscMaxElementsPerLayer	Maximum number of elements per layer.	20	*optional*

2.14. Outputs/Results

In NANDRAD it is possible to retrieve output data for any computed and published quantity, see Quantity References for a complete list. Of course, not all quantities are available in all projects - much depends on what kind of models and geometry has been defined.

In order to define an output, the following information is needed:

- an output grid, that defines *when* outputs are to be written
- the variable/quantity name
- an object list, that selects the object or objects to retrieve data from
- (optional) time handling information, i.e. whether to average values in time or perform time integration
- (optional) target filename

In addition to manually defined outputs, NANDRAD also generate a number log and data files, automatically (see section Solver log files).

Outputs are stored in the XML-tag `Outputs`, with the following general structure:

```
<Outputs>
   ... <!-- global output parameters -->

    <Grids>
        ... <!-- Definition of output grids -->
    </Grids>

    <Definitions>
        ... <!-- Actual output definitions -->
    </Definitions>
</Outputs>
```

2.14.1. Global output parameters

The following parameters influence the output file generation:

- `TimeUnit` - the value of this XML-tag holds the time unit to be used in the output files
- `IBK:Flag`:
    - name `BinaryFormat`: if true, files will be written in binary format (see Binary Format).

Example 11. Global output parameters

```
<Outputs>
    <TimeUnit>d</TimeUnit>
    <IBK:Flag name="BinaryFormat">false</IBK:Flag>
    ....
</Outputs>
```

2.14.2. Output grids

Output grids define *when* outputs are written. An output grid contains a list of intervals, with an output step size defined for each interval. For example, if you want to have hourly output steps from start to end, you need to define a grid with one interval and a step size parameter of one hour:

Example 12. Output grid for entire simulation with hourly steps

```
<Grids>
  <OutputGrid name="hourly">
    <Intervals>
      <Interval>
        <IBK:Parameter name="StepSize" unit="h">1</IBK:Parameter>
      </Interval>
    </Intervals>
  </OutputGrid>
</Grids>
```

An output grid is uniquely identified by its name (mandatory XML-attribute `name`). It contains a single child tag `Intervals` which holds one or more intervals. The intervals (XML-tag `Interval`) are expected to follow temporally in consecutive order, optionally with a gap in-between.

Intervals can have up to 3 parameters:

- `Start` - the start time of the interval (see explanation below)
- `End`- the end time of the interval (see explanation below)
- `StepSize` - the distance between outputs within the interval

The parameters are stored in XML-tags of type `IBK:Parameter`, see IBK:Parameter.

Time points in `Start` and `End` parameters are defined with respect to Midnight January 1st of the year in which the simulation starts.

Rules

- the `Start` parameter is optional under the following conditions:
  - in the first interval, a missing `Start` parameter is automatically set to 0 (start of the year)
  - in all other intervals, the `End` time of the preceeding interval is taken (see next rule below)
- the end time of an interval is defined, either:
  - by defining the `End` parameter,
  - through definition of the `Start` parameter in next interval
  - through simulation end time (only in last interval)
- the parameter `StepSize` is mandatory in each interval

Basically, it must be clear for the solver when an interval starts and ends, and how long the step size is.

During simulation, an output is written exactly under the following condition:

- t must be in an interval defined by the grid
- the offset t from the start of the interval must be an exact multiple of the step size

Example 13. Output grid evaluation

Suppose an output interval is defined to start at 12.5 h, with a step size of 2 h. The simulation time shall be t=16.5 h. Then 16.5 - 12.5 = 4 h, which is an exact multiple of 2 h. Hence, the output grid is "active" at this simulation time and all outputs associated with this output grid will be written.

There may be gaps between intervals, in which no outputs are written:

Example 14. Output grid for daily values in first year and hourly values in third year (beginning at time "2 a")

```
<Grids>
  <OutputGrid name="first_and_last">
    <Intervals>
      <Interval>
        <IBK:Parameter name="StepSize" unit="d">1</IBK:Parameter>
        <IBK:Parameter name="End" unit="a">1</IBK:Parameter>
      </Interval>
      <Interval>
        <IBK:Parameter name="Start" unit="a">2</IBK:Parameter>
        <IBK:Parameter name="StepSize" unit="h">1</IBK:Parameter>
      </Interval>
    </Intervals>
  </OutputGrid>
</Grids>
```

2.14.3. Output definitions

Below is an example of an output definition:

Output of air temperature from all zones in object list All zones and using output grid hourly

```
<Definitions>
    <OutputDefinition>
      <Quantity>AirTemperature</Quantity>
      <ObjectListName>All zones</ObjectListName>
      <GridName>hourly</GridName>
    </OutputDefinition>
    ... <!-- other definitions -->
</Definitions>
```

The example shows the mandatory child tags of XML-tag `OutputDefinition`. Below is a list of all supported child tags:

XML-tag	Description
`Quantity`	Unique ID name of the results quantity, see also Quantity References
`ObjectListName`	Reference to an object list that identifies the objects to take results from
`GridName`	Reference to an output grid (output time definitions)
`FileName`	(optional) Target file name
`TimeType`	(optional) Time averaging/integration method

TODO below

Examples

- Requesting fluxes across construction interfaces: object list must reference interfaces
- Requesting energy supplied to layer in construction instance (floor heating): object list must reference construction instance, variable name must reference heat source + index of heat source (if several in construction): `<Quantity>HeatSource[1]</Quantity>` (first heat source in layer, counting from side A in

construction, see Heat sources in constructions/layers).

2.14.4. Variable lookup rules

Quantities in output definitions define the ID names of the output quantities, optionally including an index notation when a single element of a vectorial quantity is requested. Hereby the following notations are allowed:

- `HeatSource[1]` - index argument is interpreted as defined by the providing models, so when the model provides a vector-valued quantity with model ID indexing, then the argument is interpreted as object ID (otherwise as positional index)
- `HeatSource[index=1]` - index argument is explicitely interpreted as position index (will raise an error when model provides quantity with model ID indexing)
- `HeatSource[id=1]` - index argument is explicitely interpreted as object ID (will raise an error when model provides quantity with positional indexing)

2.14.5. Output file generation rules

When no filename is given

Target file name(s) are automatically defined.

All outputs a grouped by:

- grid and quantity type (Load, Schedule, State, Flux, Misc), quantity type is predefined for most quantity IDs (unknowns → Misc)
- for each used grid:
- states → `states_<gridname>.tsv`
- loads → `loads_<gridname>.tsv`
- fluxes → `fluxes_<gridname>.tsv`
- fluxes (integrated) → `flux_integrals_<gridname>.tsv`

Special rule: when only one grid is used, and grid has hourly steps all year round, the suffix `_<gridname>` is omitted.

When a filename is given

- check: all output definitions using this filename must use the **same** grid (same time points for all columns required!)
- all quantities (regardless of type) are written to this file

2.14.6. Binary Format

First record: unsigned int - n (number of columns) Next n records: binary strings, leading size (unsigned int) and termination character (sanity checking)

Next ?? records: unsigned int - n (for checking) and afterwards n doubles

2.14.7. Solver log files

3. Tutorials

This section contains several tutorials for different use cases in NANDRAD2. It will start with a simple tutorial for a single room.

3.1. Tutorial 1 - Simple Single Room

3.1.1. Introduction

In this example a single thermal zone modelling is described. The main focus is put on the geometry, material and construction parametrization. The temperature of the freely oscillating room is given as the result output. The dimensions of the room are `l = 2.0 m` length, `w = 5.0 m` width and `h = 3.0 m` height. This leads to an air volume of `V = 30.0 m³`. All further characteristic values are specified in the following.

PICTURE

3.1.2. Workflow

First all materials inside `Materials` and the used constructions inside `ConstructionTypes` are defined, which are needed for the test zone. Afterwards all enveloping surfaces inside `ConstructionInstances` are parametrized and the output parameters inside `Outputs` are set. Finally, the climate is specified inside `Location` and further simulation settings inside `SimulationParameter`.

3.1.3. Materials and Constructions

The building consists of a floor, a wall and a roof construction. The constructions are shown in the following table.

name	id	thickness [m]	λ [W/mK]	ρ [kg/m³]	ce [J/kgK]
floor	103				
concrete	1001	0.20	2.3	2000	1000
insulation	1004	0.05	0.04	50	1500
roof	102				
insulation	1004	0.20	0.04	50	1500
Wood	1002	0.05	0.17	500	2100
wall	101				
concrete	1001	0.20	2.3	2000	1000
insulation	1004	0.10	0.04	50	1500

Materials

For the materials the thermal parameters such as `thermal conductivity λ`, `density ρ` and `heat capacity ce` are required. Furthermore a unique Id `id` and name `displayName` is needed. Exemplary the description for concrete and insulation is given below. The detailed documentation is described in Materials.

Example:

```
<Materials>
  <Material id="1001" displayName="Concrete">
    <IBK:Parameter name="Density" unit="kg/m3">2000</IBK:Parameter>
    <IBK:Parameter name="HeatCapacity" unit="J/kgK">1000</IBK:Parameter>
    <IBK:Parameter name="Conductivity" unit="W/mK">2.3</IBK:Parameter>
  </Material>
  <Material id="1004" displayName="Insulation">
    <IBK:Parameter name="Density" unit="kg/m3">50</IBK:Parameter>
    <IBK:Parameter name="HeatCapacity" unit="J/kgK">1500</IBK:Parameter>
    <IBK:Parameter name="Conductivity" unit="W/mK">0.04</IBK:Parameter>
  </Material>
</Materials>
```

Execute ToDo hygric parameters

Constructions

Afterwards the Constructions in ConstructionTypes are assembled from the Materials via the Id matching the Id in the materials and also the layer thickness d. As with the materials, a construction is always assigned a unique identifier id and optionally a name displayName. Transfer and other parameters are not part of the construction and are defined inside the Constructions that represent an enveloping surface. For the later usage inside the ConstructionInstance the first material layer MaterialLayer inside the MaterialLayers List is linked to the InterfaceA and the last material layer to the InterfaceB. Thus, the inside or outside of the construction can be defined individually inside the Constructions.

The wall construction is examplarily shown below.

Example:

```
<ConstructionTypes>
  <ConstructionType id="101" displayName="Wall Construction">
    <MaterialLayers>
      <MaterialLayer thickness="0.2" matId="1001" /> <!-- Linked to InterfaceA -->
      <MaterialLayer thickness="0.1" matId="1004" /> <!-- Linked to InterfaceB -->
    </MaterialLayers>
  </ConstructionType>
</ConstructionTypes>
```

Zone

In this section the Zone and its parameters are defined. Geometrically, the zone represents the volume of air inside the room. All further geometrical properties are defined inside the tag named ConstructionInstances. Besides the Volume an Area is specified, which is needed for the conversion of area specific loads to room loads. These space loads are not described in this tutorial. The uniqueness of the zone is guaranteed by an identifier id. Optionally a name displayName can be assigned again. The type of the zone sets the calculation mode for the zone. Three types are distinguished:

- Active The zone is calculated by the solver via the energy balance equations and the room air temperature results from the gains and losses of all energy flows.
- Constant The zone is defined by a given temperature. It can be defined by a schedule, which does not have to be constant.

- `Ground` The floor temperatures from the default climate file are used for the room air temperature. The zone thus represents the adjacent soil.

The volume and the area are defined via so-called `IBK:Parameter`. The zone volume is defined in the example room with `V = 30 m³`. The base area is described with `A = 10 m²`.

Example:

```
<Zones>
  <Zone id="1" displayName="Single room model" type="Active">
    <IBK:Parameter name="Area" unit="m2">10</IBK:Parameter>
    <IBK:Parameter name="Volume" unit="m3">30</IBK:Parameter>
  </Zone>
</Zones>
```

Further setting options can be found in the detailed zone documentation.

Enclosing Surfaces

The Enclosing Surfaces are described in the `ConstructionInstances`. Each Enclosing Surfaces named `ConstructionInstance` is represented by an Id `id`, optionally a name `displayName`, a surface, a construction and the transition conditions represented by different models. The surface is described by an `IBC:Parameter` with the attribute `Area`. The construction is linked to the construction from `ConstructionTypes` via the `ConstructionTypeId`. The boundary conditions are defined via the interfaces `InterfaceA` and `InterfaceB`. As boundary conditions transfer coefficients and solar as well as thermal absorption coefficients are defined. These are each described by a separate model.

In the example the wall Enclosing Surfaces is shown. The selected wall is defined by an area `A = 15 m²`, a wall construction with the `id = 101` and an inside and outside boundary condition. The outside boundary condition is described with a constant transition coefficient of `h = 15 W/(m²K)`, a solar absorptance of `a = 0.6` and a long-wave absorption/emission of `ε = 0.9`. On the inside, only a transition coefficient `h = 10 W/(m²K)` is described.

Further setting options can be found in the detailed ConstructionInstance documentation.

Example:

```
<ConstructionInstances>
  <ConstructionInstance id="1" displayName="West Wall">
    <ConstructionTypeId>101</ConstructionTypeId>
    <IBK:Parameter name="Area" unit="m2">15</IBK:Parameter>
    <InterfaceA id="10" zoneId="1">
      <!--Interface to zone `Single room model` -->
      <InterfaceHeatConduction modelType="Constant">
        <IBK:Parameter name="HeatTransferCoefficient" unit="W/m2K">10</IBK:Parameter>
      </InterfaceHeatConduction>
    </InterfaceA>
    <InterfaceB id="11" zoneId="0">
      <!--Interface to outside-->
      <InterfaceHeatConduction modelType="Constant">
        <IBK:Parameter name="HeatTransferCoefficient" unit="W/m2K">15</IBK:Parameter>
      </InterfaceHeatConduction>
      <InterfaceSolarAbsorption modelType="Constant">
        <IBK:Parameter name="AbsorptionCoefficient" unit="---">0.6</IBK:Parameter>
      </InterfaceHeatConduction>
      <InterfaceLongWaveEmission modelType="Constant">
        <IBK:Parameter name="Emissivity" unit="---">0.9</IBK:Parameter>
      </InterfaceHeatConduction>
    </InterfaceB>
  </ConstructionInstance>
</ConstructionInstances>
```

Output

The requested outputs must be defined, otherwise a simulation will be started without obtaining output result variables. The `Outputs` are divided into `Definitions` and `Grids`. Inside `Grids` the interval step sizes and optionally the time points for the outputs are defined. The `Definitions` consist of individual outputs named `OutputDefinition` each with an object list name `ObjectListName`, an output grid name `GridName` and a result quantity `Quantity`. Additionally, the interval handling `timeType` and the output file name `FileName` can be specified. In the interval handling either momentary values at the end of the interval, average or integral values of the interval are output. In a separate discussion the [interval treatments] are described in detail.

The object list groups all IDs of objects, which are used to access the objects like zones, models, etc. themselves. The object list `objectlist` consists of a `FilterId`, a `ReferenceType` and a name `name`. With a `*` all existing Ids of a reference type can be addressed. The example below shows how the output of the models is referenced via the object list.

Example:

```
<ObjectLists>
  <ObjectList name="Zone">
    <FilterID>*</FilterID>
    <ReferenceType>Zone>/ReferenceType>
  </ObjectList>
</ObjectLists>
```

In the following example the air temperature is queried and written to the [standard output file]. An hourly time grid was selected as interval. The output takes place over the entire << simulation period, simulation duration>>.

Example:

```
<Outputs>
  <OutputDefinitions>
    <OutputDefinition>
      <Quantity>AirTemperatures>/Quantity>
      <ObjectListName>Zone</ObjectListName>
      <GridName>hourly</GridName>
    </OutputDefinition>
  </OutputDefinitions>
  <Grids>
    <OutputGrid name="hourly">
      <Intervals>
        <Interval>
          <IBK:Parameter name="StepSize" unit="h">1</IBK:Parameter>
        </Interval>
      </Intervals>
    </OutputGrid>
  </Grids>
</Outputs>
```

Location

The location and climate are described in the tag `Location`. Mandatory parameters are the albedo as [IBK:Parameter] and either a climate file `ClimateFileName` or a location description with the IBK:Parameters northern latitude `Latitude`, eastern longitude `Longitude` and the height above sea level `Elevation`.

TODO describe what to do if the climate file is missing.

Simulation parameters

4. Reference

4.1. Unit Definitions

Throughout the NANDRAD solver, units are *only* used for input/output purposes. Within the calculation functions, *always* the base SI units are used, hereby avoiding problems from unit conversions.

The unit system in NANDRAD uses the convention, that at maximum one / may be part of the unit definition. All units following the slash are in the denominator of the unit. Exponents are just following the unit, for example `m2`. Multiple units are just concatenated without . or * character, for example `kWh` or `kg/m2s`.

Units are case-sensitive! For example, `Deg` is correct whereas `deg` will not be recognized as correct unit.

Base SI unit	Convertible units
-	
---	%, 1
---/d	%/d
1/K	
1/logcm	

Base SI unit	Convertible units
1/m	1/cm
1/Pa	
1/s	1/min, 1/h
J	kJ, MJ, MWh, kWh, Wh
J/K	kJ/K
J/kg	kJ/kg
J/kgK	kJ/kgK, Ws/kgK, J/gK, Ws/gK
J/m2	kJ/m2, MJ/m2, GJ/m2, J/dm2, J/cm2, kWh/m2
J/m2s	W/m2, kW/m2, MW/m2, W/dm2, W/cm2
J/m3	Ws/m3, kJ/m3, MJ/m3, GJ/m3, J/dm3, J/cm3, kWh/m3
J/m3K	kJ/m3K
J/m3s	kJ/m3s, MJ/m3s, J/dm3s, J/cm3s, J/m3h, W/m3, kW/m3, MW/m3, W/dm3, W/cm3, W/mm3
J/mol	kJ/mol
J/s	J/h, J/d, kJ/d, W, kW, MW, Nm/s
K	C
K/m	
K/Pa	
kg	g, mg
kg/kg	g/kg, mg/kg
kg/m	g/m, g/mm, kg/mm
kg/m2	kg/dm2, g/dm2, g/cm2, mg/m2
kg/m2s	g/m2s, g/m2h, g/m2d, kg/m2h, mg/m2s, µg/m2s, mg/m2h, µg/m2h
kg/m2s05	kg/m2h05
kg/m3	kg/dm3, g/dm3, g/cm3, g/m3, mg/m3, µg/m3, log(kg/m3), log(g/m3), log(mg/m3), log(µg/m3)
kg/m3s	g/m3s, g/m3h, kg/m3h, mg/m3s, µg/m3s, mg/m3h, µg/m3h
kg/m3sK	g/m3sK, g/m3hK, kg/m3hK, mg/m3sK, µg/m3sK, mg/m3hK, µg/m3hK
kg/mol	g/mol
kg/ms	
kg/s	kg/h, kg/d, g/d, g/a, mg/s, µg/s
kWh/a	
kWh/m2a	
l/m2s	l/m2h, l/m2d, mm/d, mm/h
l/m3s	l/m3h
logcm	
logm	

Base SI unit	Convertible units
logPa	
Lux	kLux
m	mm, cm, dm
m/s	cm/s, cm/h, cm/d
m/s2	
m2	mm2, cm2, dm2
m2/kg	
m2/m3	
m2/s	cm2/s, m2/h, cm2/h
m2K/W	
m2s/kg	
m3	mm3, cm3, dm3
m3/m2s	m3/m2h, dm3/m2s, dm3/m2h
m3/m2sPa	m3/m2hPa
m3/m3	Vol%
m3/m3d	Vol%/d
m3/s	m3/h, dm3/s, dm3/h
m3m/m3m	m3mm/m3m
mm/m	
mol	mmol
mol/kg	mol/g
mol/m3	mol/ltr, mol/dm3, mol/cm3
Pa	hPa, kPa, Bar, PSI, Torr
Pa/m	kPa/m
Person/m2	
Rad	Deg
s	min, h, d, a, sqrt(s), sqrt(h), ms
s/m	kg/m2sPa
s/s	min/s, h/s, d/s, a/s
s2/m2	
W/K	
W/m2K	
W/m2K2	
W/m2s	W/m2h, kW/m2s, MW/m2s, W/dm2s, W/cm2s
W/mK	kW/mK

Base SI unit	Convertible units
W/mK2	
W/Person	kW/Person
undefined	

The unit `undefined` means *not initialized* (internally) and must not be used in input files.

4.2. Quantity References

The following list of quantities is an overview of all available results that can be requested as outputs. Which outputs are actually available depends on the project and will be printed into the file `var/output_reference_list.txt` (see discussion in section Outputs/Results).

Some of the quantities are vector-valued quantities, marked with a suffix `(id,xxx)` or `(index,xxx)`. To access these values, you need to specify the id/index in your output definition (see explanation and examples in section Outputs/Results).

Reference/object type	Quantity	Unit	Description
ConstructionInstance	FluxHeatConductionA	W	Heat conduction flux across interface A (into construction).
ConstructionInstance	FluxHeatConductionB	W	Heat conduction flux across interface B (into construction).
ConstructionInstance	LayerTemperature(index,xxx)	C	Mean layer temperature for requested quanties.
ConstructionInstance	SurfaceTemperatureA	C	Surface temperature at interface A.
ConstructionInstance	SurfaceTemperatureB	C	Surface temperature at interface B.
Location	AirPressure	Pa	Air pressure.
Location	Albedo	---	Albedo value of the surrounding [0..1].
Location	AzimuthAngle	Deg	Solar azimuth (0 - north).
Location	CO2Concentration	---	Ambient CO2 concentration.
Location	CO2Density	kg/m3	Ambient CO2 density.
Location	DeclinationAngle	Deg	Solar declination (0 - north).
Location	ElevationAngle	Deg	Solar elevation (0 - at horizont, 90 - directly above).
Location	LWSkyRadiation	W/m2	Long wave sky radiation.
Location	Latitude	Deg	Latitude.
Location	Longitude	Deg	Longitude.

Reference/object type	Quantity	Unit	Description
Location	MoistureDensity	kg/m3	Ambient moisture density.
Location	RelativeHumidity	%	Relative humidity.
Location	SWRadDiffuseHorizontal	W/m2	Diffuse short-wave radiation flux density on horizontal surface.
Location	SWRadDirectNormal	W/m2	Direct short-wave radiation flux density in normal direction.
Location	Temperature	C	Outside temperature.
Location	VaporPressure	Pa	Ambient vapor pressure.
Location	WindDirection	Deg	Wind direction (0 - north).
Location	WindVelocity	m/s	Wind velocity.
Model	InfiltrationHeatFlux(id,xxx)	W	Infiltration/natural ventilation heat flux
Model	InfiltrationRate(id,xxx)	1/h	Natural ventilation/infiltration air change rate
Zone	AirTemperature	C	Room air temperature.
Zone	CompleteThermalLoad	W	Sum of all thermal fluxes into the room and energy sources.
Zone	ConstructionHeatConductionLoad	W	Sum of heat conduction fluxes from construction surfaces into the room.
Zone	InfiltrationHeatLoad	W	Infiltration/natural ventilation heat flux into the room.