# NANDRAD Model Reference

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

Version 1.1.0, September 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:

*Example 1. Definition of a NANDRAD Project File*

```
<?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 |

| Child tag | Description |
|---|---|
| `Location` | Climatic data and location settings → Climatic loads |
| `SimulationParameter` | Simulation model parameters → Simulation Parameters |
| `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 → Model Parametrization |
| `Schedules` | Definition of scheduled parameters → Schedules |
| `Outputs` | Output definitions → Outputs/Results |
| `ObjectLists` | Definition of object lists/object reference groups → Object Lists and Result References |

## 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 2. Parameters with Different Units*

```xml
<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 3. Whole Number (Integer) 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 4. 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. The child tags `X` and `Y` hold the actual values, always unitless. Number of x and y values must match.

*Example 5. Linear Spline Definition*

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

### 2.2.5. LinearSplineParameter

A linear spline parameter is effectively an extended `IBK:LinearSpline` parameter with additional attributes.

*Example 6. Linear Spline Parameter Definition*

```
<LinearSplineParameter name="ThermalLoad" interpolationMethod="linear">
  <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>
</LinearSplineParameter>
```

*Table 1. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `name` | Specific name that references to the space type the annual schedule will be set for | string | *required* |
| `interpolationMethod` | Specifies the interpolation method between the defined y values.<br>• `constant` - constant interpolation (values constant during time step)<br>• `linear` - linear interpolation (values linear interpolated between time steps) | key | *required* |
| `WrapMethod` | Specifies what should be done if values are requested with x values outside the x-value range.<br>• `continuous` - constant extrapolation (take first or last value, respectively)<br>• `cyclic` - apply cyclic adjustment with the model-specific period length (for example, a year) | key | *required* |

The child tags `X` and `Y` each hold a mandatory attribute `unit` with the respective value unit (see Unit Definitions).

## 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 7. Custom Directory Placeholders*

```xml
<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. The following child tags are supported.

| Child tag | Description | Format |
|---|---|---|
| `Comment` | General comment on the project. | *string* |
| `Created` | Date/time this project was created. | *string* |
| `LastEdited` | Date/time this project was last modified. | *string* |

The date/time strings for `Created` and `LastEdited` should stored the date and time in user-readible format, as they may be used to show lists of projects with change/creation date.

## 2.5. Embedded Databases

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 define constructions composed of such materials. These parameters are stored in databases, which are actually lists of XML objects.

### 2.5.1. Materials

In the NANDRAD project file the materials database section starts with an XML tag named `Materials`.

*Example 8. Materials with Parameters*

```xml
<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>
```

In this tag each material property set starts with an XML tag named `Material` with two XML attributes `id` and `displayName`.

*Table 2. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `id` | Unique id of the material. | > 0 | *required* |
| `displayName` | Name of material (used for informative/error messages). | string | *optional* |

Concerning the material parameters such as density, heat capacity and thermal conductivity they need to be defined within the XML tag `IBK:Parameter` (see IBK:Parameter):

| Name | Default Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `Density` | kg/m3 | Dry density of the material. | > 0.01 | *required* |
| `HeatCapacity` | J/kgK | Specific heat capacity of the material. | >= 100 | *required* |
| `Conductivity` | W/mK | Thermal conductivity of the dry material. | >= 1e-5 | *required* |

### 2.5.2. Construction Types

Constructions are defined inside the section starting with an XML tag `ConstructionTypes`.

*Example 9. 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 this section each construction definition starts with the XML tag named `ConstructionType` with the XML attributes `id` and optional `displayName`:

*Table 3. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `id` | Unique id number. | positive integer ( > 0 ) | *required* |
| `displayName` | Name of construction (used for informative/error messages). | string | *optional* |

A construction consists of one or more material layers. These are defined within the child XML tag named `MaterialLayers`. Each material layer is defined with the XML tag `MaterialLayer` with the following XML attributes:

| XML-Attribute | Description | Format | Usage |
|---|---|---|---|
| `thickness` | defines the thickness of the layer in `m` | > 0.0 | *required* |
| `matId` | refers to a material by unique material id number (`id` as defined in a `Material` tag), | string | *required* |

With the use of the *matId* attribute, layers of constructions reference the used materials:

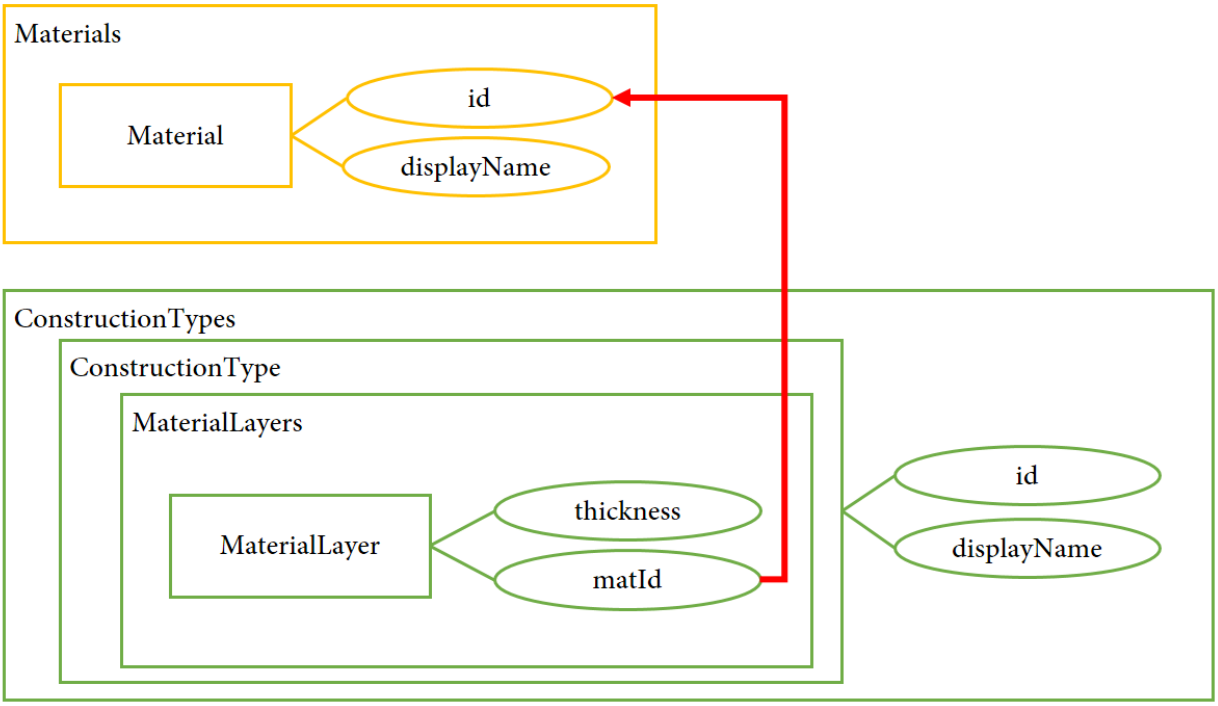



*Figure 1. Collaboration Diagram for ConstructionType and Material Objects*

The `MaterialLayer` does not have child tags since all needed data are defined as XML attributes as described above.

### 2.5.3. Glazing Systems

Glazing systems are defined in a list inside the XML tag `WindowGlazingSystems`.

*Example 10. Parameter definition for a glazing system*

```
<WindowGlazingSystems>
  <WindowGlazingSystem id="123" modelType="Standard">
    <IBK:Parameter name="ThermalTransmittance" unit="W/m2K">0.4</IBK:Parameter>
    <LinearSplineParameter name="SHGC" interpolationMethod="linear" wrapMethod="cyclic">
      <!-- X incidence angle - 90 deg = sun is perpendicular/normal to surface  -->
      <X unit="Deg">0 90 </X>
      <!-- Note: no constant parameter - if constant SHGC, define as below -->
      <Y unit="---">0.6 0.6 </Y>
    </LinearSplineParameter>
  </WindowGlazingSystem>
</WindowGlazingSystems>
```

Inside this section each galzing system definition starts with the XML tag named `WindowGlazingSystem` with the XML attributes `id`, `modelType` and optional `displayName`:

*Table 4. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `id` | Unique id number. | positive integer ( > 0 ) | *required* |
| `displayName` | Name of glazing system (used for informative/error messages). | string | *optional* |

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `modelType` | Identifies the model complexity:<br>• `Standard` - Standard glazing model, with a U-value (thermal transmittance) and incidence-angle dependent SHGC value | string | *optional* |

Scalar parameters are defined within an XML tag `IBK:Parameter` (see IBK:Parameter):

| Name | Default Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `ThermalTransmittance` | W/m2K | Thermal transmittance of glazing | > 0 | *required for model type Simple* |

Parameters, that depend on the incidence angle, are defined within an XML tag `LinearSplineParameter` (see LinearSplineParameter):

| Name | Default Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `SHGC` | --- | Solar heat gain coefficient | > 0 | *required for model type Simple* |

## 2.6. Zones

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

Objects of type `Zone` store all properties needed to compute zone temperature from energy density (the conserved quantity).

*Example 11. Zone Definition*

```
<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">
```

*Table 5. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `id` | Identifier of the Zone | ( > 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* |
| `type` | Defines whether zone is balanced and included in equation system.<br>• `Constant` as zone with constant/predefined temperatures. (schedule)<br>• `Active` as zone described by a temperature node in space<br>• `Ground` as ground zone (calculates temperature based on standard) | key | *required* |

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.

Parameters (see section IBK:Parameter for a description of the `IBK:Parameter` tag):

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `Volume` | m3 | Zone air volume | > 0.0 | *required* |
| `Area` | m2 | Net usage area of the ground floor (for area-related outputs and loads) | > 0.0 | *optional* |
| `HeatCapacity` | J/K | Additional heat capacity (furniture, etc.) | >= 0.0 | *optional* |
| `Temperature` | C | Temperature of the zone<br><br>only used if `Constant` | -70...120 | (*required*) |
| `RelativeHumidity` | % | Relative humidity of the zone<br><br>only used if `Constant` | 0...100 | (*required*) |
| `CO2Concentration` | g/m3 | CO2 concentration of the zone<br><br>only used if `Constant` | > 0.0 | (*required*) |

The parameters `RelativeHumidity` and `CO2Concentration` only need to be defined for *constant* zones, when the respective balance equation is enabled.

## 2.7. Construction Instances

Construction instances represent actually built one-dimensional parts of the building envelope, e.g. walls, floors, ceilings, roofs. The construction instance definition contains construction-specific parameters required by several models.

*Example 12. Definition of an Outside Wall with only Heat Conduction Boundary Condition*

```xml
<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>
```

The construction instances are defined inside the XML tag `ConstructionInstances`. Inside the section each construction definition starts with the XML tag named `ConstructionInstance` with attributes `id` and `displayName`.

*Table 6. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `id` | Identifier of the construction instance | > 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* child tag:

*Table 7. Construction Instance Child Tags*

| Tag | Description |
|---|---|
| `ConstructionTypeId` | Reference to `ConstructionTypeId` |
| `IBK:Parameter` | Different `IBK:Parameter` for constructions instance |
| `InterfaceA` | Interface for constructions instance side A |
| `InterfaceB` | Interface for constructions instance side B |

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

For the construction instance parameters the following XML tags named `IBK:Parameters` with the XML attributes `name` and `unit` with the following entries can be defined:

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `Orientation` | Deg | Orientation of the wall<br><br>if one interface has solar (short wave) radiation boundary condition it is *required* | 0…360 | *required / optional* |
| `Inclination` | Deg | Inclination of the wall<br><br>• 0 Deg - roof<br>• 90 Deg - vertical wall<br>• 180 Deg - facing downwards<br><br>if one interface has short and/or long wave radiation boundary condition it is *required* | 0…180 | *required / optional* |
| `Area` | m2 | Gross area of the wall (including potentially existing windows, holes etc.) | > 0 | *required* |

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.

### 2.7.1. Spatial Discretization (Finite Volume Method)

During calculation, each of the constructions is spatially discretized using a grid generation algorithm. This algorithm uses three influential parameters, defined in the Solver Parameters section:

- `DiscMinDx`
- `DiscStretchFactor`
- `DiscMaxElementsPerLayer`

Figure 2 illustrates the effect of different stretch factors.

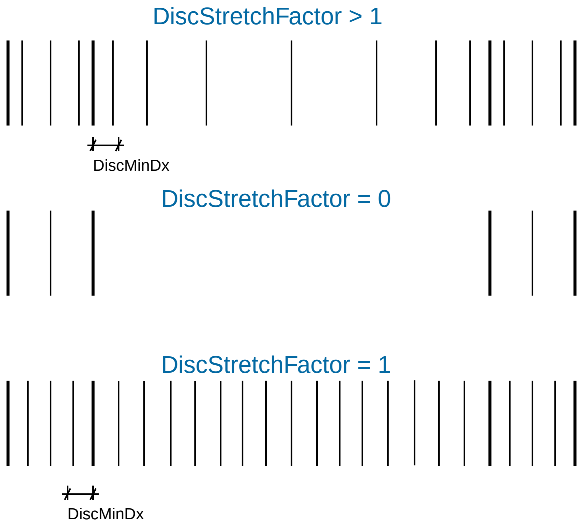



*Figure 2. Different Discretization Variants depending on DiscStretchFactor Parameter*

Basically three different grid generation operations are supported:

- **minimal grid:** when `DiscStretchFactor = 0` the algorithm creates one Finite Volume per material layer, except for the boundary elements, which are always split into two (needed for surface value extrapolation). So, for example, a 4-layered construction will result in 6 Finite Volumes.
- **equidistant:** when `DiscStretchFactor = 1` the algorithm generates equally spaced grid elements in each layer, whose thickness is close to, but always less than the `DiscMinDx` parameter. Since material layers may have different widths, a uniform grid element thickness may not be possible throughout the construction. Choose a `DiscMinDx` parameter where all material layer widths are whole-number multiples of this grid element thickness (e.g. *1 mm*)
- **regular grid:** for any `DiscStretchFactor > 1` a regular, variably-spaced grid is generated.

## Regular Grid Generation Algorithm

A regular stretching grid is generated using a double-sided *tanh*-stretching function. The factor `DiscStretchFactor` hereby determines roughly the ratio of first two grid element widths. Naturally, this growth factor varies and goes down to zero in the center of a material layer, but it nicely determines the overall grid detail. A factor of 4 is a good default value.

The parameter `DiscMinDx` defines the maximum width of the outermost grid elements in each layer. Hence, it is indirectly also used to determine the number of grid elements per material layer. With increasing number of grid elements per layer, the outermost grid elements will become smaller. This way, the algorithm determines the number of grid cells (for a given `DiscStretchFactor`), until the generated width if the outermost grid elements is equal or less than the `DiscMinDx` parameter. A minimum element thickness of *2 mm* is a good default value for very accurate calculations, but a value of *5 mm* may suffice in many situations (this reducing the number of unknowns and eventually simulation time significantly).

Finally, there is the parameter `DiscMaxElementsPerLayer` that can be used to limit the number of grid elements to be generated in a material layer. This is particularly useful when very thick material layers are present and a large number of grid cells are generated. Often, this accuracy is not needed (for very thick material layers anyways), so limiting the number may be meaningful to speed up calculation. As long as the number of generated grid cells per material layer exceeds `DiscMaxElementsPerLayer` the algorithm will gradually increase the `DiscStretchFactor` until the criterion is fulfilled. The solver will emit a warning message for each construction layer that this adjustment is applied to.

> As with all numerical solvers employing calculation grids, there is always a compromise between speed and accuracy. A grid sensitiviy study may be helpful, for example by starting with `DiscMinDx = 5 mm` and `DiscStretchFactor = 8` and then gradually reducing the values until the solution does no longer change. For small buildings/models, where performance is not an issue, you may want to use the default values `DiscMinDx = 2 mm` and `DiscStretchFactor = 4`.

## 2.8. Interfaces (construction boundary conditions)

Interfaces are defining boundary conditions and parameters for the one or two surfaces `InterfaceA` and `InterfaceB` of a constructions instance. If the construction instance defines an adiabatic wall only one interface is needed. All other cases require two interfaces. The `InterfaceA` links the first material layer from the construction type with the assigned zone via the `zoneId`. The `InterfaceB` links the last material layer from the construction type with the `zoneId` of `InterfaceB`.

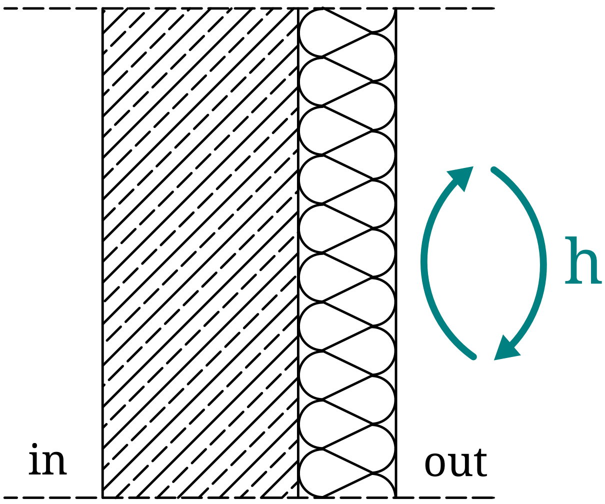
*Example 13. Interface Definitions for a Construction with Interfaces for either Side*

```
<ConstructionInstance id="1" displayName="All Surfaces Var01">
    ...
    <InterfaceA id="10" zoneId="1">
      <InterfaceHeatConduction modelType="Constant">
        <IBK:Parameter name="HeatTransferCoefficient" unit="W/m2K">2.5</IBK:Parameter>
      </InterfaceHeatConduction>
    </InterfaceA>
    <InterfaceB id="11" zoneId="0">
      <InterfaceHeatConduction modelType="Constant">
        <IBK:Parameter name="HeatTransferCoefficient" unit="W/m2K">8</IBK:Parameter>
      </InterfaceHeatConduction>
        <InterfaceSolarAbsorption model="Constant">
            <IBK:Parameter name="AbsorptionCoefficient" unit="---">0.6</IBK:Parameter>
        </InterfaceSolarAbsorption>
        <InterfaceLongWaveEmission model="Constant">
            <IBK:Parameter name="Emissivity" unit="---">0.9</IBK:Parameter>
        </InterfaceLongWaveEmission>
    </InterfaceB>
</ConstructionInstance>
```

`InterfaceA` and `InterfaceB` may have one or more child tags.

### 2.8.1. Heat Conduction

The convective heat conduction over the interface is described by the XML tag `InterfaceHeatConduction`.




*Example 14. Parameter Definition for Heat Conduction Boundary Condition*

```
<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`.

*Table 8. Attributes*

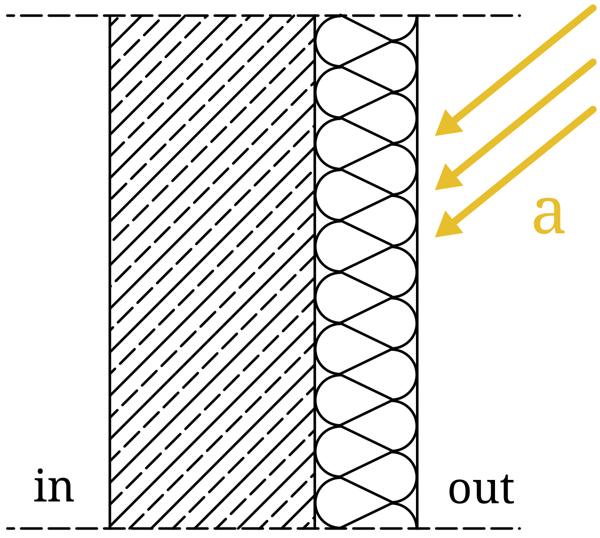
| Attribute | Description | Format | Usage |
|---|---|---|---|
| `modelType` | Sets the type of the heat conduction model<br>• `Constant` - Constant model used (currently the only option) | key | *required* |

Floating point parameters (see section IBK:Parameter for a description of the `IBK:Parameter` tag):

| Name | Default Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `HeatTransferCoefficient` | W/m2K | Constant convective heat transfer coefficient | > 0.0 | *required* |

### 2.8.2. Solar Absorption

The solar absorption over the interface is described by the XML tag `InterfaceSolarAbsorption`. This coefficient describes the solar short wave radiation that is absorpt by the interface.




*Example 15. Parameter Definition for Solar Absorption Boundary Condition*

```
<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`.

*Table 9. Attributes*

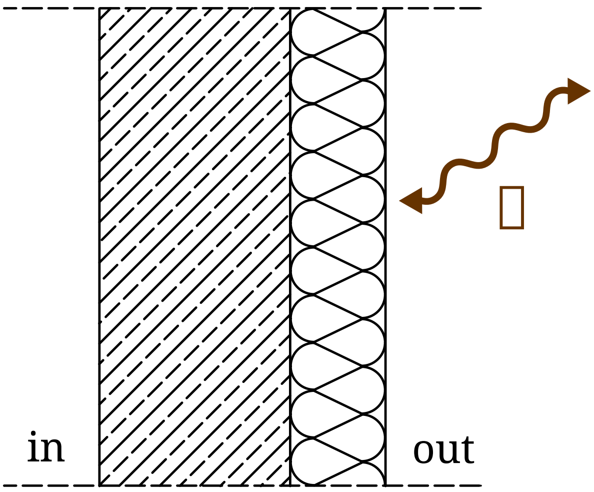
| Attribute | Description | Format | Usage |
|---|---|---|---|
| `modelType` | Sets the type of the heat conduction model<br>• `Constant` - constant model used (currently the only option) | key | *required* |

The XML tags named `IBK:Parameters` with the XML attributes `name` and `unit` with the following entries can be defined:

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `AbsorptionCoefficient` | --- | Constant absorption coefficient | 0...1 | *required* |

### 2.8.3. Long Wave Emission

The long wave emission over the interface is described by the XML tag `InterfaceLongWaveEmission`. This coefficient describes the long wave absorption and emission by the interface.




*Example 16. Parameter Definition for Long Wave Emission*

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

The `InterfaceLongWaveEmission` needs to be defined with the following XML attribute `modelType`.

*Table 10. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `modelType` | Sets the type of the heat conduction model<br>• `Constant` - constant model used (currently the only option) | key | *required* |

The XML tags named `IBK:Parameters` with the XML attributes `name` and `unit` with the following entries can be defined:

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `Emissivity` | --- | Constant absorption coefficient | 0…1 | *required* |

### 2.8.4. Vapour Diffusion

TO BE DEFINED LATER.

The vapour diffusion over the interface is described by the XML tag `InterfaceVaporDiffusion`.

*Example 17. Parameter Definition for Vapor Diffusion*

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

The `InterfaceVaporDiffusion` needs to be defined with the following XML attribute `modelType`.

*Table 11. Parameters for the InterfaceVaporDiffusion-Tag*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `modelType` | Sets the type of the heat conduction model<br>• `Constant` - constant model used (currently the only option) | key | *required* |

The XML tags named `IBK:Parameters` with the XML attributes `name` and `unit` with the following entries can be defined:

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `VaporTransferCoefficient` | s/m | Vapor Transfer Coefficient | > 0.0 | *required* |

### 2.8.5. Air Flow

TO BE DEFINED LATER.

The air flow over the interface is calculate with a pressure coefficient. It is described inside the XML tag `InterfaceAirFlow`.

*Example 18. Parameter Definition for Air Flow*

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

The `InterfaceAirFlow` needs to be defined with the following XML attribute `modelType`.

*Table 12. Attriubutes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `modelType` | Sets the type of the air flow<br>• `Constant` - constant model used (currently the only option) | key | *required* |

The XML tags named `IBK:Parameters` with the XML attributes `name` and `unit` with the following entries can be defined:

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `PressureCoefficient` | --- | Pressure Coefficient | 0...1 | *required* |

## 2.9. Embedded objects (windows, doors, openings…)

There can be several embedded object definitions.

*Example 19. Definition of a window inside a construction instance*

```
<ConstructionInstance id="1">
  <IBK:Parameter name="Area" unit="m2">12</IBK:Parameter>
    ...
  <EmbeddedObjects>
    <EmbeddedObject id="2000" displayName="A window">
        <!-- Area parameter is required. -->
      <IBK:Parameter name="Area" unit="m2">8</IBK:Parameter>
        ...
    </EmbeddedObject>
  </EmbeddedObjects>
</ConstructionInstance>
```

Embedded objects must have at least and `Area` parameter defined. This area must not exceed the gross surface area of the construction instance.

An embedded object is further qualified by embedded data objects.

### 2.9.1. Windows

A window is composed of a glazing and optionally frame and dividers. Without frame and dividers, the definition for such a window looks like:

*Example 20. Parameter definition for basic window without frame*

```
<EmbeddedObject id="2000" displayName="A window">
  <IBK:Parameter name="Area" unit="m2">8</IBK:Parameter>
    <Window glazingSystemID="123"/>
</EmbeddedObject>
```

Only the glazing system is referenzed by ID. Glazing systems are defined in the glazing systems database list, see Glazing Systems.

The window may have a frame and/or dividers. These are separate entities because frame and divider material (and hence thermal conductivity across these materials) may be different. These are defined in XML-tags `Frame` and `Divider`:

*Example 21. Parameter definition for basic window with frame and divider*

```
<EmbeddedObject id="2000" displayName="A window">
  <IBK:Parameter name="Area" unit="m2">8</IBK:Parameter>
  <Window glazingSystemID="123">
    <Frame materialID="1001">
      <IBK:Parameter name="Area" unit="m2">3</IBK:Parameter>
    </Frame>
    <Divider materialID="1002">
      <IBK:Parameter name="Area" unit="m2">2</IBK:Parameter>
    </Divider>
  </Window>
</EmbeddedObject>
```

The material properties (currently only thermal conductivity) of frame and divider elements are taken from the material referenced via ID.

The actual geometry of frame and divider elements is not important, but their total cross section area must be given as `Area` parameter.

The cross section occupied by frame and divider must not exceed the gross area of the embedded window object. The actual translucent glazing area is computed as difference between embedded object area and frame and divider areas.

When the window (or embedded object) is resized, the sizes of frame and divider must be adjusted accordingly. While it would have been possible to define frame and divider cross sections also as relative percentage, still this percentage needs to be updated when the window is resized.

### Window shading

It is possible to apply pre-computed shading to both opaque and translucent facade elements. Pre-computed shading is generally defined as global property in the `Location` tag (see Pre-computed shading).

When pre-computed shading is defined, for **each** opaque and translucent surface a factor will be provided.

As described in section Pre-computed shading, the association between provided data columns and object ID is done via identification string, composed from object type and ID number. For example, an embedded object with ID 14 would get the column header/caption *embObj.14* and a construction instance (opaque surface) with ID 29 would get *conInst.29*.

Alternatively or additionally to pre-computed shading it is possible to define controlled shading for the window.

*Example 22. Parameter definition for controlled shading*

```
<Window glazingSystemID="123">
    ...
  <Shading modelType="Standard">
    <ControlModelID>555</ControlModelID>
    <IBK:Parameter name="ReductionFactor" unit="---">0.7</IBK:Parameter>
  </Shading>
</Window>
```

The `Shading` needs to be defined with the following XML attributes:

*Table 13. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `modelType` | Sets the type of the shading model<br>• `Standard` - Standard model using only a single reduction factor and a separate control model. | key | *required* |

*Table 14. Child tags*

| Element | Description | Format | Usage |
|---|---|---|---|
| `ControlModelID` | Reference to a `ShadingControlModel` definition. | > 0 | *required for Standard model* |

The XML tags named `IBK:Parameters` with the XML attributes `name` and `unit` with the following entries can be defined:

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `ReductionFactor` | --- | Percentage of remaining solar gains when shading is closed | 0…1 | *required for Standard model* |

*Example 23. Calculation of the shading factor based on control signal*

```
ReductionFactor = 80%

Fz depending on control signal:

1   = full shaded:      Fz = 1 - (1 - 80%) * 1      = 0.8
0   = unshaded shaded:  Fz = 1 - (1 - 80%) * 0      = 1
0.5 = partially shaded: Fz = 1 - (1 - 80%) * 0.5    = 0.9
```

## 2.10. Climatic loads

### 2.10.1. Overview

Climatic loads in NANDRAD are provided by means of climate data files. For solar radiation calculation it needs information on the building location (usually provided in the climate file), and also the orientation and inclination of the various construction surfaces (defined for outside surfaces, see Geometry/Constructions).

### 2.10.2. Specification

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

*Example 24. Definition of Location*

```
<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 IBK:Parameter for a description of the `IBK:Parameter` tag):

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `Albedo` | --- | Used for diffuse solar radiation calculation (see Solar Radiation Calculation) | 0…1 | *required* |
| `Altitude` | m | later needed for specific altitude-related parameters (**TODO**) | >0 | *optional* |
| `Longitude` | Deg | If specified, overrides the location parameter `Longitude` of the climate data file (see Building/Station location). | -180…180 | *optional* |
| `Latitude` | Deg | If specified, they override the location parameter `Latitude` of the climate data file (see Building/Station location). | -90…90 | *optional* |

Flags and options (see section IBK:Flag for a description of the `IBK:Flag` tag):

| Name | Description | Default | Usage |
|---|---|---|---|
| `PerezDiffuseRadiationModel` | Defines whether to use the Perez-Model for diffuse solar radiation calculation | *false* | *optional* |

Lastly, 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 provided, 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).

### Cyclic (annual) and continuous (multi-year) climate data

The climate data file can either contain 8760 hourly values for an entire year. Anything else is considered as arbitrary range of time values indicating a specific time interval, possibly also with varying time intervals between data points. The latter climate data files cannot be used for annual/cyclic calculation, but require a specific (matching) simulation time interval (see also section Simulation time interval).

**Cyclic annual climate**

Climate data is provided in hourly values. The interpretation of these values depends on the type of quantity. NANDRAD distinguishes between state quantities and flux/load quantities.

State quantities are:

- temperatures
- relative humidities
- air pressures
- wind direction
- wind velocity

Flux/load quantities are:

- direct solar radiation intensity (in sun's normal direction)
- diffuse solar radiation intensity (on horizontal plane)
- rain load
- long wave sky emission

State quantities are expected to be monitored as *instantaneous values* at the *end of each hour*. Sub-hourly values are obtained through linear interpolation, as shown in Figure 3.

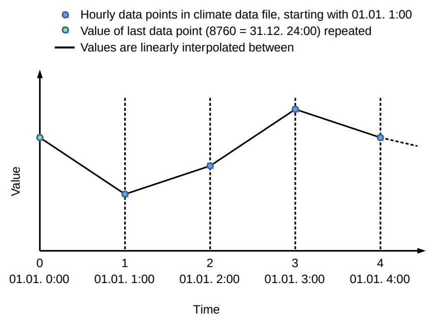



*Figure 3. Using Linear Interpolation to reconstruct sub-hourly Values from Hourly Data Points*

Flux/load quantities are expected as *mean/average values* over the *last hour*. Sub-hourly values are obtained through linear interpolation between the average values placed in the middle of each hour, as shown in Figure 4.

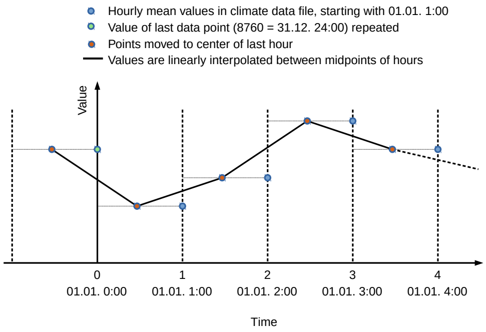



*Figure 4. Using Linear Interpolation to reconstruct sub-hourly Values from Hourly Mean Values*

**Continuous data**

The climate data file contains data points (at least 2), which also mark the earliest start and latest end point of the simulation.

If you continue the simulation past the available climate data, the last values in the climate data set will be kept constant, thus eventually leading to meaningless results (unless this is intendet in artificial test cases).

Since the user can choose arbitrary time steps in the climate data files, even down to minutely values, the accuracy of the input data depends on the user input. Between time points, the solver will linearly interpolate **all quantities** in the climate data file, and not distinguish between states and loads, as with hourly data.

To achive the same result as with annual hourly data, simply provide climatic data in 30 min intervals and compute interpolated values at end and middle of each hour, yourself.

### 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.

*Example 25. Definition of a Sensor in the Location*

```
<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>
```

*Table 15. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `id` | Identifier of the sensor | > 0 | *required* |

Parameters (see section IBK:Parameter for a description of the `IBK:Parameter` tag):

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `Orientation` | Deg | Orientation of the sensor | 0...360 | *required* |
| `Inclination` | Deg | Inclination of the sensor<br>• 0 Deg - facing upwards<br>• 90 Deg - e.g. like a vertical wall<br>• 180 Deg - facing downwards | 0...180 | *required* |

A sensor must be given a unique ID number and the mandatory parameters `Orientation` and `Inclination` (see section Construction Instances 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).

```xml
<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>
```

### 2.10.3. Solar Radiation Calculation

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

### 2.10.4. Pre-computed shading

A pre-processing software may calculate the percentage of surface area with sun exposure for each surface element of the building. For example, in Figure 5, a facade is partially shaded.

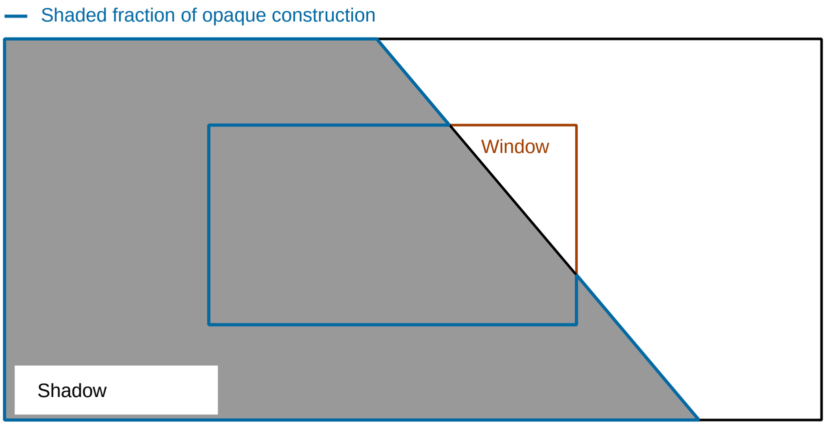



*Figure 5. Illustration of a partially shaded facade with a window.*

The software can now compute the percentage of shaded area for both the opaque facade element and for the embedded window object separately. The window is approximately 80% shaded, and about 20% of the surface is still exposed to the sun. The factor stored for this time and the window surface will be 0.2.

The factor stored for a construction always includes potentially embedded objects. Figure 6 shows a similar picture, yet easier to calculate.

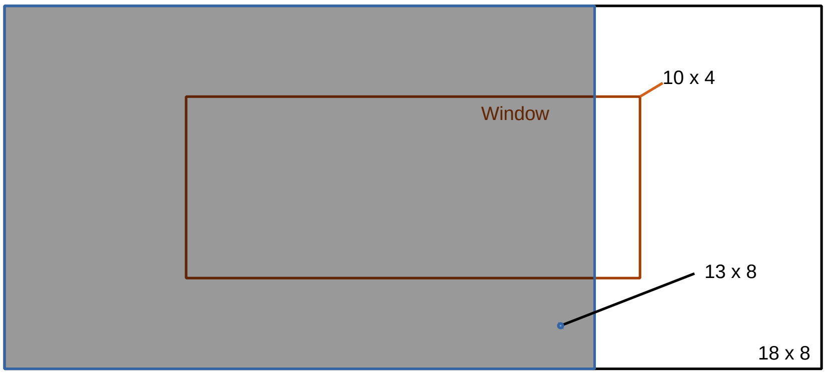



*Figure 6. Calculation example for a partially shaded facade with a window.*

The construction has a surface area of 18x8 = 144 m2. The window has a surface area of 10x4 = 40 m2. The shadow on the construction occupies 13x8 = 104 m2, thus the shaded fraction (factor) for the construction is: 104/144 = 72.2%.

The shadow onto the window alone occupies (13-(18-10)/2)x4 = 9x4 = 36 m2. Thus, the shaded surface factor for the window alone will be 90%.

The values **72.2%** and **90%** will be stored in the shaded fractions data file.

The shaded area on the opaque surface alone is 104 - 36 = 68 m2. The opaque construction alone has a surface area of 144 - 40 = 104 m2. Thus, the reduction factor to be used in the solar radiation load equation will be 68/104 = 65.4%. This calculation will be done internally by the solar radiation model.

Remember to compute the percentage of the shaded surface area for constructions **always including any embedded objects**.

### Pre-computed shaded fractions file format

The file containing pre-computed shading factors is referenced in the XML tag `Location`, as child tag `ShadingFactorFileName`. The path given here can be an absolute path, or a relative path following the `${Project Directory}` placeholder (see Path Placeholders).

The file follows the `tsv` file format rules (see *PostProc 2* documentation) and has a single header line. The first column is the time column, with either absolute time stamps or time offsets relative to midnight January 1st of the start year.

All other columns contain the compute shaded fraction values, where each column header identifies the respective surface with an ID-string.

The ID-string follows the pattern: `<objID>.<ID-number>`. Object-IDs are either `conInst` or `embObj`.

During the calculation, the values in the data table are linearly interpolated.

## 2.11. Object Lists and Result References

Whenever it is necessary to reference a calculation result (of a model object), this is done via *ObjectLists*.

In NANDRAD, physical equations are organized in terms of model objects, for example zones or constructions. These model objects can be uniquely identified by a model type and ID number. For example, all quantities computed for a room/zone are identified by model type *Zone* and id number of the respective zone. Table 16 lists the available reference type keywords.

*Table 16. Model Reference Types*

| Keyword | Description |
|---|---|
| `Zone` | Variables related to room (thermal zones) |
| `ConstructionInstance` | Variables related to constructions |
| `Schedule` | Scheduled parameters |
| `Location` | Variables from climate calculation model, including radiation sensor values |
| `Model` | Model-specific variables/results |

Example 26 shows several examples of object list definitions.

*Example 26. Definition of Several Object Lists*

```xml
<ObjectLists>
  <ObjectList name="All zones">
    <FilterID>*</FilterID>
    <ReferenceType>Zone</ReferenceType>
  </ObjectList>
  <ObjectList name="Zone Var01">
    <FilterID>1</FilterID>
    <ReferenceType>Zone</ReferenceType>
  </ObjectList>
  <ObjectList name="Wall_1_and_2">
    <FilterID>1,2</FilterID>
    <ReferenceType>ConstructionInstance</ReferenceType>
  </ObjectList>
  <ObjectList name="InfiltrationModel">
    <FilterID>501</FilterID>
    <ReferenceType>Model</ReferenceType>
  </ObjectList>
  ...
</ObjectLists>
```

### 2.11.1. Object List Definitions

All object lists are defined within the parent tag `ObjectLists`. Each object list definition begins with the XML-tag `ObjectList` with the mandatory attribute `name`, which uniquely identifies the object list.

XML-tag `ObjectList` has the following child tags.

*Table 17. Model Reference Types*

| Keyword | Description |
|---|---|
| `FilterID` | ID filter pattern (see description below) |
| `ReferenceType` | Model object reference type (see Table 16) |

### 2.11.2. ID-Filter Patterns

Objects (with same reference type) are uniquely identified by their ID number.

> ID numbers must only be unique for objects with the same reference type. Hence, it is possible to define Zone #1 and ConstructionInstance #1 at the same time.

A filter pattern can be composed of several parts, separated by , (comma), for example: `1,4,13-20`. Each part can be of the following format:

- a single ID number, e.g. *12*
- a range of ID numbers, e.g. *1-100*
- * (selects all IDs)

If you specify IDs several times, for example in `3, 1-10`, the resulting ID set will contain each ID only once.

## 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 values provided by the schedule. For example, they are used in the following objects or list of objects:

- **Occupancy rates**, **heat loads**, **clothing factors** in the person load model.
- **Heating/cooling set point temperatures** for thermostat controls
- **Mass flow rates** or **temperature set points** for plant components
- **Electrical power rates** for lighting and electrical devices

For example, a schedule defines a heating set point `HeatingSetPoint` for specific zones like living rooms. These are selected by an object list named "Living room", which selects objects with the type `Zone` and a certain ID range (will be described later in more detail).

There are two possible ways to describe a schedule:

- `ScheduleGroups`
- `AnnualSchedules`.

The two options are discussed in detail in Daily scheme based schedules and Annual schedules.

Furthermore scheduled data can be handled in two different ways, as

- **Cyclic** data, and
- **Non-cyclic** data.

**Cyclic data** means that schedule values will be repeated after the end of the schedule period. This means, for example, that an annual schedule will be run twice if the simulation time will be set to two years. **Cyclic-data** can be defined by `ScheduleGroups` and `AnnualSchedules`.

**Non-cyclic** data will always be used only once. This is useful if monitored data will be used to set up schedules. Then the simulation needs to be set only for the time span the monitored data is existing. **Non-cyclic** data can only be defined by `AnnualSchedules`.

*Example 27. Schedules Definition*

```
<Schedules>
    <Holidays>5,10</Holidays>
    <WeekEndDays>Fri,Sat</WeekEndDays>
    <FirstDayOfYear>Fri</FirstDayOfYear>
    <IBK:Flag name="EnableCyclicSchedules">true</IBK:Flag>

    <ScheduleGroups>
        ...
    </ScheduleGroups>
    <AnnualSchedules>
        ...
    </AnnualSchedules>
</Schedules>
```

Inside the object `Schedules` the following XML tags can also be specified

- `FirstDayOfYear`
- `Holidays`
- `WeekEndDays`
- `Schedule`
- `IBK:Flag` with the name `EnableCyclicSchedules`

*Table 18. XML tags that can be definned*

| XML tags | Description | Format | Usage |
|---|---|---|---|
| `FirstDayOfYear` | The day type of January 1st (offset of day of the week of the start year.<br>• `Mon` - Monday (*default*)<br>• `Tue` - Tuesday<br>• `Wed` - Wednesday<br>• `Thu` - Thursday<br>• `Fri` - Friday<br>• `Sat` - Saturday<br>• `Sun` - Sunday | String | *optional* |
| `Holidays` | List of holiday days, stored in a comma-separated list of numbers, where each number represents the "day of the year", not including leap days. | string | *optional* |
| `WeekEndDays` | Weekend days. | string | *optional* |

| XML tags | Description | Format | Usage |
|---|---|---|---|
| `IBK:Flag` | • name `EnableCyclicSchedules` - If set to true, schedules will be repeated after one year. If set to false (only applicable to annual schedules), these annual schedules are sampled just once. | *true (default) / false* | *optional* |

### 2.12.2. Daily scheme based schedules

Regular schedules are defined upon a daily cycle based scheme. Some parameters need to be defined inside the hereafter specified XML tags.

*Example 28. ScheduleGroup Definition*

```
<ScheduleGroups>
    <ScheduleGroup objectList="All Zones">
        <!-- AllDays constant  -->
        <Schedule type="AllDays">
            ...
        </Schedule>
        <Schedule type="WeekDays">
            ...
        </Schedule>
    </ScheduleGroup>
<ScheduleGroups>
```

*Example 29. ObjectList definition that selects zone objects and is named "All Zones"*

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

Regular schedules are defined within the XML tag `ScheduleGroup` with a mandatory XML attribute named `objectList` that references an `ObjectList` by name (see Table 19):

*Table 19. Attribute for the ScheduleGroup*

| Name | Description | Format | Usage |
|---|---|---|---|
| `objectList` | References to an object list with the specifed name | string | *required* |

Example 28 shows such a definition and Example 29 the corresponding object list.

#### Daily Cycles

Inside the `ScheduleGroup` several Objects called `Schedule` can be defined. The `Schedule` objects need an XML attribute called `type` with different names for specific day types (see Table 20). There must not be two `Schedule` objects with the same `type` inside a `ScheduleGroup`. Within each `Schedule` object a schedule is defined that is applied for all days of the given `type` during the course of a whole year. The following rules apply when constructing schedules.

At first priority the type `AllDays` will set specified daily schedule values (e.g. `HeatingSetPoint`) to all days of the whole year (Priority 0). Example 30 shows such a schedule definition.

After this the `type` named `WeekEnd` and `WeekDay` will, if defined, overwrite the already defined schedule values for only all week days or weekend days (Priority 1). Furthermore the weekdays named `Monday`, `Tuesday`, … define for which days the schedule values will be overwritten again (Priority 2). This continues with the day type `Holiday` (Priority 3) for the specified holidays inside the `Holidays` object.

It is possible to define different schedules for individual periods of the year, e.g. regular year and summer vacation period etc.. This way a schedule for the entire year can be defined.

*Example 30. Schedule definition with type "AllDays"*

```
<ScheduleGroup objectList="Zone01">
    <!-- AllDays constant  -->
    <Schedule type="AllDays">
        <DailyCycles>
            <DailyCycle interpolation="Constant">
                <TimePoints>0</TimePoints>
                <Values>InfiltrationRateSchedule [1/h]:0</Values>
            </DailyCycle>
        </DailyCycles>
    </Schedule>
</ScheduleGroup>
```

Table 20 shows the day types and their associated priorities.

*Table 20. Description of the schedule type attribute*

| `type` | Priority | Description |
|---|---|---|
| `AllDays` | 0 | Values will be set to all days of the period |
| `WeekEnd` | 1 | Values will be set to all weekend days of the period |
| `WeekDay` | 1 | Values will be set to all week days of the period |
| `Monday` | 2 | Values will be set to all Mondays of the period |
| `Tuesday` | 2 | Values will be set to all Tuesdays of the period |
| `Wednesday` | 2 | Values will be set to all Wednesdays of the period |
| `Thursday` | 2 | Values will be set to all Thursdays of the period |
| `Friday` | 2 | Values will be set to all Fridays of the period |
| `Saturday` | 2 | Values will be set to all Saturdays of the period |
| `Sunday` | 2 | Values will be set to all Sundays of the period |
| `Holiday` | 3 | Values will be set to all holidays of the period that are specified inside the `holidays` tag |

Example 31 illustrates the use of different schedules to define a weekly schedule. First, the basic every-day schedule is defined. Then, special rules are defined for tuesdays and weekends. Figure 7 illustrates the resulting schedule.

*Example 31. Schedule definition using different day types*

```
<Schedules>
    <WeekEndDays>Sat,Sun</WeekEndDays>
    <ScheduleGroups>
        <ScheduleGroup objectList="All zones">
            <!-- every day between 8-10 -->
            <Schedule type="AllDays">
                <DailyCycles>
                    <DailyCycle interpolation="Constant">
                        <TimePoints>0 6 10</TimePoints>
                        <Values>InfiltrationRateSchedule [1/h]:0 0.4 0</Values>
                    </DailyCycle>
                </DailyCycles>
            </Schedule>
            <!-- Tuesday no ventilation -->
            <Schedule type="Tuesday">
                <DailyCycles>
                    <DailyCycle interpolation="Constant">
                        <TimePoints>0</TimePoints>
                        <Values>InfiltrationRateSchedule [1/h]:0</Values>
                    </DailyCycle>
                </DailyCycles>
            </Schedule>
            <!-- Weekend only on afternoon -->
            <Schedule type="WeekEnd">
                <DailyCycles>
                    <DailyCycle interpolation="Constant">
                        <TimePoints>0 14 16</TimePoints>
                        <Values>InfiltrationRateSchedule [1/h]:0 0.1 0</Values>
                    </DailyCycle>
                </DailyCycles>
            </Schedule>
        </ScheduleGroup>
    </ScheduleGroups>
</Schedules>
```

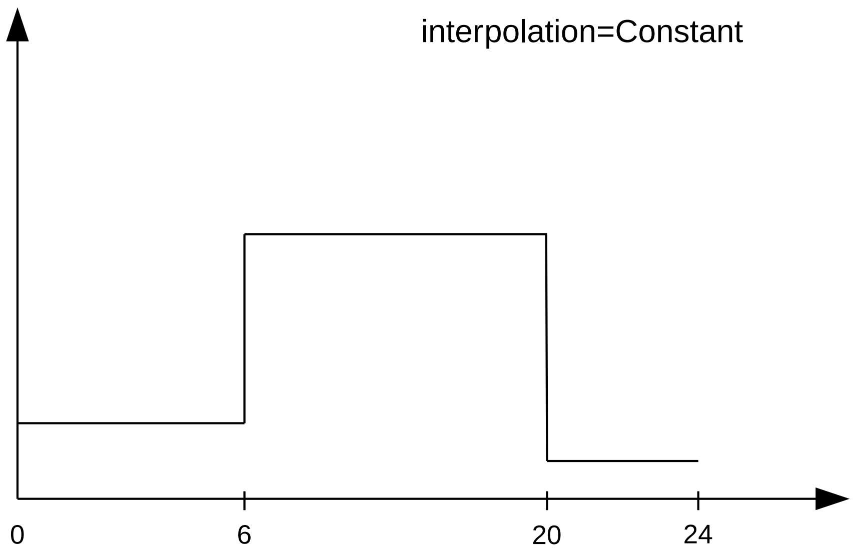



*Figure 7. Illustration of weekly schedule defined by example* [*Example 31*]

### Daily Cycle Time Intervals

A `DailyCycle` defines how one or more quantities change during the day. The child tag `TimePoints` defines space-separated time points in [h] (hours), and hereby the different time intervals of the day.

If the attribute `interpolation` is `Constant`, then the following rules apply:

- the time points are interpreted as **start** time of the next interval
- the first time point must be always 0, the last one must be < 24 h,
- the corresponding value is taken as constant during this interval

For example, a time point vector "0 6 20" defines three intervals: 0-6, 6-20, 20-24 and the data table must contain exactly 3 values.

If the attribute `interpolation` is `Linear`, then the following rules apply:

- the time points are points in time where associated values are given
- the first time point must be always 0, the last one must be < 24 h, because in cyclic usage, the time point at 24 h will be the same as for 0 h (and likewise the scheduled values)
- between time points the values are linearly interpolated

Figure 8 and Figure 8 illustrate the resulting value curve for time intervals given by *0, 6, 20* and corresponding parameter values *2, 7, 1*.

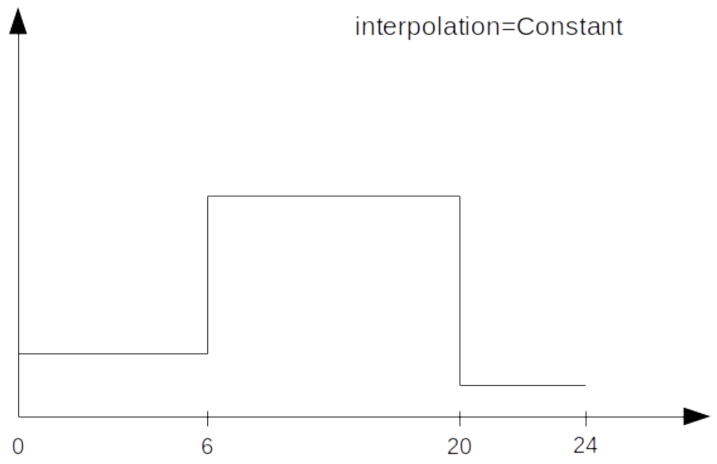



*Figure 8. Daily cycle with Constant interpolation mode*

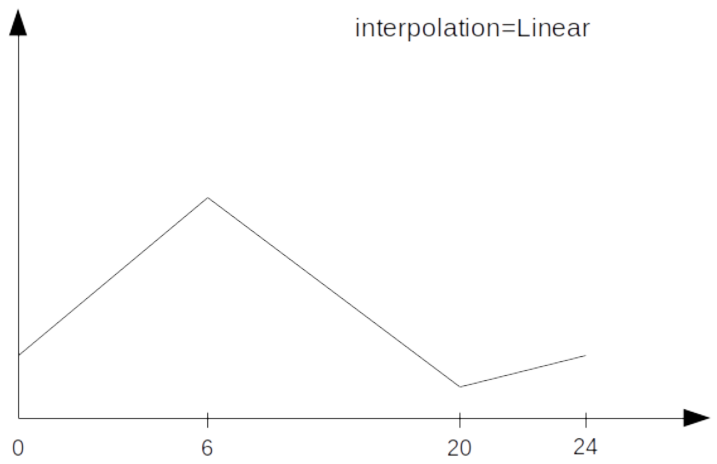



*Figure 9. Daily cycle with Linear interpolation mode*

When linear interpolation mode is used, the value at 24 h is taken from the start of the next daily cycle, that is defined in the schedule. For example, in Figure 7 the value at Monday 24:00 would be taken from the Tuesday schedule, whereas the value at Wednesday 24:00 would be taken from the regular *AllDays* schedule.

To define a single interval for the whole day, simply specify "0" as value in the `TimePoints` XML tag.

### Daily Cycle Parameter Values

For each interval given in the `TimePoints` tag, one or more quantities with associated units can be specified. This is done by defining the data table in the XML child tag `Values` of the `DailyCycle` tag. The data table data is formatted as:

```
quantity1 [unit]:val11 val12 val13; quantity2 [unit]:val21 val22 val23;...
```

Basically, each physical quantity is encoded in a string, whereby the strings for different quantities are combined into one string with ; (semi-colon) as separation character.

Each quantity string is composed of a header and the actual values. The values are simply values separated by spaces/tabs or comma (decimal numbers are written with . as decimal separator).

The header is a quantity keyword (see also Variable list) followed by its unit in brackets. So, for example, a heating set point temperature will have the header `HeatingSetPointTemperature [C]` and the values are then given in degree C.

There must be *exactly* as many values given as there are time points in the `TimePoints` XML tag. You can specify as many quantities as you need in this data table.

Example 32 shows a daily cycle with two scheduled quantities and three intervals.

*Example 32. Daily cycle with two scheduled quantities*

```
<DailyCycle interpolation="Constant">
    <TimePoints>0 6 10</TimePoints>
    <Values>
        InfiltrationRateSchedule [1/h]:0 0.4 0;
        HeatingSetPointTemperature [C]:18 22 18
    </Values>
</DailyCycle>
```

### Avoiding discontinuities / performance improvements

When defining daily cycles with interpolation mode `Constant`, the values will actually jump between intervals. These discontinuities are very expensive to compute, since the solver needs to cluster time steps around these jumps to accurately follow the step functions.

However, for practical applications these steps are often not desired - even though a set point may be switched momentarily to a new value, the resulting physical effect may indeed take a few minutes to be noticeable. This is taken into account when the solver interpretes scheduled values.

Instead of exactly providing the step-wise scheduled values, the solver implements an automatic 2 minute ramping just before the interval end. Figure 10 illustrates the 2 minute linear ramping applied directly before each new interval.

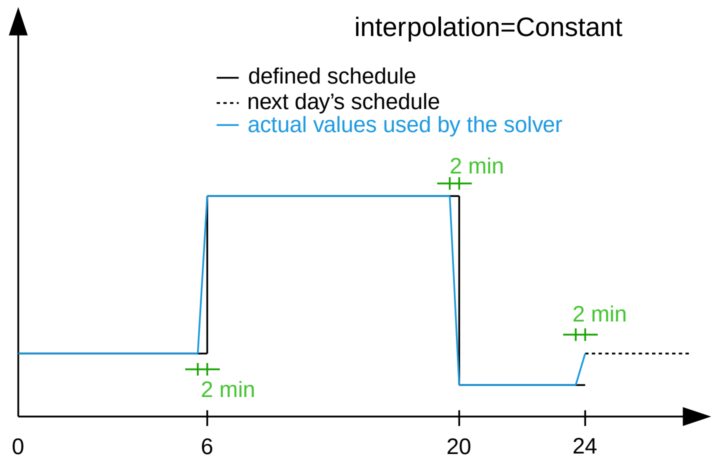



*Figure 10. Ramping/step smoothing applied for dialy cycles with step-wise defined values*

### 2.12.3. Annual schedules

Annual schedules are basically data tables with monotonically increasing X an Y-Values. Annual schedules can be defined as any linear/constant interpolated data tables. For example, hourly values of temperatures or control variables measured during the year can be specified.

Inside the XML tag `AnnualSchedules` the sub tag `SpaceTypeGroup` with an XML attribute `spaceTypeName` needs to be defined. This defined attribute needs to match to an defined `SpaceType` with the same name.

*Example 33. Definition of an Annual Schedule*

```
<AnnualSchedules>
    ...
  <SpaceTypeGroup spaceTypeName="Zone">
      <IBK:LinearSpline name="HeatingSetPointTemperature" interpolation="linear">
        <X unit="h">    0       2183    2184  6576  6577    8760   </X>
        <Y unit="C">    20      30      20    30    20      30     </Y>
      </IBK:LinearSpline>
      <IBK:LinearSpline name="TotalEnergyProductionPerPerson" interpolation="linear">
        <X unit="h">        0   2183    2184  6576  6577  8760    </X>
        <Y unit="W/Person"> 70  110     70    110   70    110     </Y>
      </IBK:LinearSpline>
      <IBK:LinearSpline name="EquipmentUtilizationRatio" interpolation="linear">
        <X unit="h">        0   2183  2184  6576  6577  8760</X>
        <Y unit="W/Person"> 10  20    10    20    10    20  </Y>
      </IBK:LinearSpline>
    </SpaceTypeGroup>
    ...
</AnnualSchedules>
```

*Table 21. Paramaters that can be set for the IBK:LinearSpline*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `name` | Specific name that references to the space type the annual schedule will be set for | string | *required* |
| `InterpolationMethod` | Specifies the interpolation method between the defined y values.<br>• `constant` - constant interpolation (values constant during time step)<br>• `linear` - linear interpolation (values linear interpolated between time steps) | key | *required* |

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `WrapMethod` not yet implemented | Specifies how to treat the values in multi-year simulations<br>• `cyclic` - Annual cyclic data<br>• `continuous` - Continuous data without repetition | key | *required* |

### 2.12.4. Variable list

The variable list describes all names and the units that can be used inside the schedules.

*Table 22. Variable List*

| Name | Unit | Description |
|---|---|---|
| `HeatingSetPointTemperature` | C | Setpoint temperature for heating. |
| `CoolingSetPointTemperature` | C | Setpoint temperature for cooling. |
| `AirConditionSetPointTemperature` | C | Setpoint temperature for air conditioning. |
| `AirConditionSetPointRelativeHumidity` | % | Setpoint relative humidity for air conditioning. |
| `AirConditionSetPointMassFlux` | kg/s | Setpoint mass flux for air conditioning. |
| `HeatingLoad` | W | Heating load. |
| `ThermalLoad` | W | Thermal load (positive or negative). |
| `MoistureLoad` | g/h | Moisture load. |
| `CoolingPower` | W | Cooling power. |
| `LightingPower` | W | Lighting power. |
| `DomesticWaterSetpointTemperature` | C | Setpoint temperature for domestic water. |
| `DomesticWaterMassFlow` | kg/s | Domestic water demand mass flow for the complete zone (hot water and equipment). |
| `ThermalEnergyLossPerPerson` | W/Person | Energy of a single persons activities that is not available as thermal heat. |

| Name | Unit | Description |
|---|---|---|
| `TotalEnergyProductionPerPerson` | W/Person | Total energy production of a single persons body at a certain activity. |
| `MoistureReleasePerPerson` | kg/s | Moisture release of a single persons body at a certain activity. |
| `CO2EmissionPerPerson` | kg/s | CO2 emission mass flux of a single person at a certain activity. |
| `MassFluxRate` | --- | Fraction of real mass flux to maximum mass flux for different day times. |
| `PressureHead` | Pa | Supply pressure head of a pump. |
| `OccupancyRate` | --- | Fraction of real occupancy to maximum occupancy for different day times. |
| `EquipmentUtilizationRatio` | --- | Ratio of usage for existing electric equipment. |
| `LightingUtilizationRatio` | --- | Ratio of usage for lighting. |
| `MaximumSolarRadiationIntensity` | W/m2 | Maximum solar radiation intensity before shading is activated. |
| `UserVentilationAirChangeRate` | 1/h | Exchange rate for natural ventilation. |
| `UserVentilationComfortAirChangeRate` | 1/h | Maximum air change rate = offset for user comfort. |
| `UserVentilationMinimumRoomTemperature` | C | Temperature limit over which comfort ventilation is activated. |
| `UserVentilationMaximumRoomTemperature` | C | Temperature limit below which comfort ventilation is activated. |
| `InfiltrationAirChangeRate` | 1/h | Exchange rate for infiltration. |
| `ShadingFactor` | --- | Shading factor [0...1]. |

## 2.13. 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:

*Example 34. Parameter Definition for Outputs*

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

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

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

### 2.13.1. 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 35. Global output parameters*

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

### 2.13.2. 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 36. 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.

| Name | Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `Start` | h | the start time of the interval (see explanation below) of the wall | >=0.0 | *optional* |
| `End` | h | the end time of the interval (see explanation below) | >=0.0 | *optional* |
| `StepSize` | h | the distance between outputs within the interval | >0.0 | *required* |

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)

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 within 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 37. 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 38. 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>
```

### 2.13.3. Output definitions

Below is an example of an output definition:

*Example 39. 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 | Usage |
|---|---|---|
| `Quantity` | Unique ID name of the results quantity, see also Quantity References | *required* |
| `ObjectListName` | Reference to an object list that identifies the objects to take results from | *required* |
| `GridName` | Reference to an output grid (output time definitions) | *required* |
| `FileName` | Target file name | *optional* |
| `TimeType` | Time averaging/integration method | *optional* |

The ID name of the quantity is the name of the result of a model object, or a schedule or anything else generated by the solver. The corresponding object or objects are selected by an object list. The grid name is the ID name of an output grid.

The `FileName` tag is optioned. It can be used to specifically select the name of an output file. Normally, output file names are generated automatically, depending on the type of output requested.

Lastly, the tag `TimeType` can be used to specifiy time averaging or time integration of variables, see section Time types.

### Variable names and 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)

### Output file names

The following sections describe the rules which determine the output file names.

#### When no filename is given

Target file name(s) are automatically defined.

All outputs a grouped depending on the quantity into:

- states
- fluxes
- loads
- misc

If `Integral` is selected as `TimeType`:

- for quantity of type *fluxes* the group *flux_integrals* is used instead,
- for quantity of type *loads* the group *load_integrals* is used instead

The outputs are further grouped by output grid name. The final output file name is obtained for each grid and group name:

- states → `states_<gridname>.tsv`
- loads → `loads_<gridname>.tsv`
- loads (integrated) → `load_integrals_<gridname>.tsv`
- fluxes → `fluxes_<gridname>.tsv`
- fluxes (integrated) → `flux_integrals_<gridname>.tsv`

There is one special rule: when only one grid is used, the suffix `_<gridname>` is omitted.

#### When a filename is given

The quantity is written to the specified file. If there are several output definitions with the same file name, then all quantities are written into the same file, regardless of type.

> All output definitions using the same file name must use the **same** grid (same time points for all columns are required!)

### Time types

The tag `TimeType` takes the following values:

- `None` - write outputs as computed at output time
- `Mean`- write value averaged over last output interval
- `Integral` - write integral value

By default (when the tag `TimeType` is not explicitely specified) the values are written as they are computed at the output time (corresponds to `None`). Figure Illustration of the various `TimeType` options illustrates the various options.

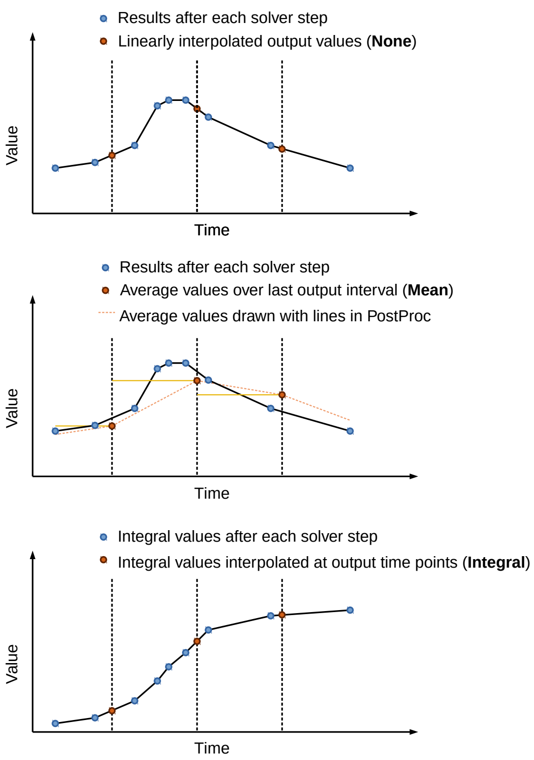



*Figure 11. Illustration of the various `TimeType` options*

> It is important to note that average values are always averages of the values in the *last output interval*. So, if you have hourly outputs defined yet the unit is `kW/d`, you will not get average values over a day, but over the last hour. The unit is only needed to convert the final value, but does not influence the way it is calculated.

### Examples

*Example 40. Requesting construction surface temperatures*

```
<Outputs>
    ...
    <Definitions>
        <OutputDefinition>
          <Quantity>SurfaceTemperatureA</Quantity>
          <ObjectListName>Walls</ObjectListName>
          <GridName>hourly</GridName>
        </OutputDefinition>
        <OutputDefinition>
          <Quantity>SurfaceTemperatureB</Quantity>
          <ObjectListName>Walls</ObjectListName>
          <GridName>hourly</GridName>
        </OutputDefinition>
        ... <!-- other definitions -->
    </Definitions>
</Outputs>
<ObjectLists>
  <ObjectList name="Walls">
    <FilterID>*</FilterID>
    <!-- object list must reference construction instances -->
    <ReferenceType>ConstructionInstance</ReferenceType>
  </ObjectList>
    ... <!-- other object lists -->
</ObjectLists>
```

*Example 41. Requesting energy supplied to layer in a construction (floor heating)*

```
<Outputs>
    ...
    <Definitions>
        <OutputDefinition>
            <!-- index 1 = heat source in layer #1, counting from side A -->
          <Quantity>HeatSource[1]</Quantity>
          <ObjectListName>FloorHeating1</ObjectListName>
          <GridName>hourly</GridName>
        </OutputDefinition>
        ... <!-- other definitions -->
    </Definitions>
</Outputs>
<ObjectLists>
  <ObjectList name="FloorHeating1">
    <FilterID>15</FilterID>
    <!-- object list must reference construction instances -->
    <ReferenceType>ConstructionInstance</ReferenceType>
  </ObjectList>
    ... <!-- other object lists -->
</ObjectLists>
```

### 2.13.4. Binary Format

The bindary variant of TSV files is very similar.

```
Header record:
- 64bit integer = n (number of columns)
- n times binary strings

Data section, each record:



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
```

### 2.13.5. Solver log files

Within the project’s result directory, the following files are automatically generated:

```
├── log
│   ├── integrator_cvode_stats.tsv
│   ├── LES_direct_stats.tsv
│   ├── progress.tsv
│   ├── screenlog.txt
│   └── summary.txt
├── results
│   └── ... (output files)
└── var
    ├── output_reference_list.txt
    └── restart.bin
```

| File | Description |
|---|---|
| `integrator_cvode_stats.tsv` | Statistics of the time integrator, written at end of simulation |
| `LES_direct_stats.tsv` | Statistics of the linear equation system (LES) solver, written at end of simulation |
| `progress.tsv` | Minimalistic runtime progress data, continuously written, can be used to follow simulation progress from GUI tools |
| `screenlog.txt` | Log file for solver output messages (same as console window outputs), continuously written |
| `summary.txt` | Statistics and timings of the simulation run, written at end of simulation |
| `output_reference_list.txt` | List of quantities generated in this project (see Quantity References) |
| `restart.bin` | Binary restart data (to continue integration) |

If you chose a different integrator or linear equation system solver (see section Solver Parameters), the files `integrator_cvode_stats.tsv` and `LES_direct_stats.tsv` are named accordingly.

## 2.14. Global parameters

The global simulation options control:

- how the model operates

- the calculation accuracy (impacts performance)
- the calculation performance

The individual settings are split into *simulation parameters* and *solver parameters*, the latter being centered on the numerical solution method.

### 2.14.1. Simulation Parameters

Hereafter all simulation parameters are described, see Example 42. All parameters are set as `IBK:Parameters`, `IBK:Flags` or `IBK:IntPara`.

*Example 42. Simulation Parameters*

```
<SimulationParameter>
  <IBK:Parameter name="InitialTemperature" unit="C">5</IBK:Parameter>
  <IBK:IntPara name="DiscMaxElementsPerLayer">30</IBK:IntPara>
  <Interval>
    <IBK:Parameter name="Start" unit="d">0</IBK:Parameter>
    <IBK:Parameter name="End" unit="d">730</IBK:Parameter>
  </Interval>
</SimulationParameter>
```

The `SimulationParameter` tag holds the following child tags:

| XML-tag | Description | Usage |
|---|---|---|
| `IBK:Parameter` | Floating point value parameters | multiple |
| `IBK:IntPara` | Whole number parameters | multiple |
| `IBK:Flag` | Flags | multiple |
| `Interval` | Defines simulation interval | none/once |

Floating point parameters (see section IBK:Parameter for a description of the `IBK:Parameter` tag):

| Name | Default Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `InitialTemperature` | C | Global initial temperature for all objetcs (`zones`, `constructionInstances`, etc) | positive double ( > 0.0 K) | *optional* |
| (*)`InitialRelativeHumidity` | % | Global initial relative humidity for all objects, that can have a humidity value set (`zones`, air flows in `models`, etc) | 0 ... 100% | *optional* |
| (*)`RadiationLoadFraction` | --- | Percentage of solar radiation gains attributed directly to the 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* |

| Name | Default Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| (*)`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* |
| (*)`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* |

*(*) - not yet used*

Whole number parameters (see section IBK:IntPara for a description of the `IBK:IntPara` tag):

| Name | Description | Default | Usage |
|---|---|---|---|
| `StartYear` | Start year of the simulation | *2001* | *optional* |

Flags and options (see section IBK:Flag for a description of the `IBK:Flag` tag):

| Name | Description | Default | Usage |
|---|---|---|---|
| (*)`EnableMoistureBalance` | Flag activating moisture balance calculation if enabled | *false* | *optional* |
| (*)`EnableCO2Balance` | Flag activating CO2 balance calculation if enabled | *false* | *optional* |
| (*)`EnableJointVentilation` | Flag activating ventilation through joints and openings. | *false* | *optional* |
| (*)`ExportClimateDataFMU` | Flag activating FMU export of climate data. | *false* | *optional* |

*(*) - not yet used*

### Simulation time interval

The tag `SimulationParameters` also contains the start and end of the simulation. By default, the simulation time interval is set to span a full year, starting at midnight January 1st. It is, however, possible to define a different time interval, thus also defining a simulation that runs longer than a year.

This is done in the child tag `Interval`:

*Example 43. Simulation interval starting on February 1st (just after the first 31 days of January are through), and running for 60 days*

```
<Interval>
  <IBK:Parameter name="Start" unit="d">31</IBK:Parameter>
  <IBK:Parameter name="End" unit="d">91</IBK:Parameter>
</Interval>
```

The start and end of a simulation are always defined in *simulation time*, explained in the next section.

#### Simulation time and absolute time reference

NANDRAD uses two time measures:

- **simulation time**, which always begins at 0 when the simulation starts, and
- **absolute time**, which is the time converted to a real date/time and is based on the actual simulation start time point.

*Simulation time* basically describes a time offset relative to the starting point of the simulation, and is typically expressed just as time delta, e.g. "20 d" or "15.5 h".

*Absolute time* is a specific time/date, like *20.09.2020 14:30*, which is obtained by adding the *simulation time* offset to a starting time point.

In NANDRAD this simulation start time points is given in two parameters:

- the `StartYear` and
- the offset of time since begin (midnight January 1st) of this year as `Start` interval parameter.

A `Start` offset of `1 d` lets the simulation start on *January 2nd, 0:00*. If, for example, the simulation shall start on *January 15th 2003, 6:00*, you need to specify

```
StartYear = 2003
Start = 14*24 + 6 = 342 h
```

And for the last day of the year, start the simulation at `Start = 364 d`.

> ❗ There are not leap years in NANDRAD. Even if you specify 2004 as start year, there won't be a February 29th! If you run a multi-year simulation every year has 365 days.

### 2.14.2. Solver Parameters

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

*Example 44. 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:IntPara name="MaxKrylovDim">30</IBK:IntPara>
  <IBK:Parameter name="DiscMinDx" unit="mm">2</IBK:Parameter>
  <IBK:Parameter name="DiscStretchFactor" unit="---">4</IBK:Parameter>
  <IBK:Flag name="DetectMaxTimeStep">true</IBK:Flag>
  <Integrator>CVODE</Integrator>
  <LesSolver>Dense</LesSolver>
</SolverParameter>
```

The `SolverParameter` tag holds the following child tags:

| XML-tag | Description | Usage |
|---|---|---|
| `IBK:Parameter` | Floating point value parameters | multiple |
| `IBK:IntPara` | Whole number parameters | multiple |
| `IBK:Flag` | Flags | multiple |
| `Integrator` | Defines time integrator | none/once |
| `LesSolver` | Defines linear equation system (LES) solver | none/once |
| `Preconditioner` | Defines preconditioner (iterative LES solver only) | none/once |

Floating point parameters (see section IBK:Parameter for a description of the `IBK:Parameter` tag):

| Name | Default Unit | Description | Value Range | Default | Usage |
|---|---|---|---|---|---|
| `RelTol` | --- | Relative tolerance for solver error check. | 0…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* |

| Name | Default Unit | Description | Value Range | Default | Usage |
|---|---|---|---|---|---|
| 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* |
| DiscStretchFactor | --- | Stretch factor for variable wall discretizations:<br>• **0** - no discretization<br>• **1** - equidistant<br>• > **1** - variable<br>see spatial discretization algorithm for details. | 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* |

*(*) - not yet used*

Whole number parameters (see section IBK:IntPara for a description of the IBK:IntPara tag):

| Name | Description | Default | Usage |
|---|---|---|---|
| PreILUWidth | Number of non-zeros in ILU | --- | *optional* |
| MaxKrylovDim | Max. size of Krylow-Dimension/max. number of linear iterations (iterative LES only) | 50 | *optional* |
| MaxNonlinIter | Max. number of non-linear/Newton iterations | 3 | *optional* |
| MaxOrder | Max. method order | 5 | *optional* |
| DiscMaxElementsPerLayer | Max. number of discretization elements per material layer | 20 | *optional* |
| (*)KinsolMaxNonlinIter | Max. iterations of Kinsol solver | *auto* | *optional* |

*(*) - not yet used*

Flags and options (see section IBK:Flag for a description of the `IBK:Flag` tag):

| Name | Description | Default | 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* |

*(*) - not yet used*

The options and parameters listed above partially depend on selected time integration algorithms, LES solvers and pre-conditioners, see table in section Solver Capabilities below.

### Integrator

The XML-tag `Integrator` contains a string to select a specific integrator (`CVODE` is used by default, when tag is missing).

*Table 23. Available Integrators*

| Name | Description |
|---|---|
| `CVODE` | Selects the **CVODE** integrator from the Sundials library: implicit multi-step method with error test based time step adjustment and modified Newton-Raphson for non-linear equation systems |
| `ExplicitEuler` | Explicit Euler integrator (only for debugging, `InitialTimeStep` parameter determines the fixed step-size) |
| `ImplicitEuler` | Implicit Euler integrator, single-step solver with error test based time step adjustment and modified Newton-Raphson for non-linear equations (only for debugging and specific tests) |

See Solver Capabilities for valid combinations.

### Linear equation system (LES) solver

The XML-tag `LesSolver` contains a string to select a specific solver for the linear equation systems (`KLU` is used by default, when tag is missing).

*Table 24. Available LES solvers*

| Name | Description |
|---|---|
| `Dense` | Direct dense solver (for debugging only) |
| `KLU` | Direct sparse solver |
| `GMRES` | Generalized Minimal Residual Method (iterative solver) |
| `BiCGStab` | Biconjugate Stabilized Gradient Method (iterative solver) |

See Solver Capabilities for valid combinations.

### Preconditioner

The XML-tag `Preconditioner` contains a string to select a specific preconditioner, to be used for iterative LES solvers (`ILU` is used by default, when tag is missing).

*Table 25. Available Preconditioners*

| Name | Description |
|---|---|
| `ILU` | Incomplete LU factorization (when `PreILUWidth` is given, ILU-T is used) |

Currently, two variants of the ILU preconditioner are implemented. One without threshold, where the factorization is stored only in the original Jacobi matrix pattern. If the user specified `PreILUWidth`, the routine will compute the factorization and keep in each row the highest n-values (where n is defined by `PreILUWidth`). This method is known as *ILU with threashold* (ILU-T).

An ILU-T method will only be effective for `PreILUWidth` > 3. The minimum number of non-zeroes in each matrix row is 3, since the Finite Volume discretization of the wall constructions will generate already a 3-diagonal pattern.

### Solver Capabilities

Not all integrators and LES solvers support all options mentioned above. Also, not all LES solvers can be combined with all integrators. The table below gives an overview of the supported combinations and options.

*Table 26. Capabilities and Supported Flags/Parameters for the provided Integrators*

| Integrator | LES solvers | Supported integrator parameters/flags |
|---|---|---|
| CVODE | Dense, KLU, GMRES, BiCGStab | RelTol, AbsTol, MaxTimeStep, MinTimeStep, InitialTimeStep, MaxOrder, NonlinSolverConvCoeff, MaxNonlinIter |
| ImplicitEuler | Dense | RelTol, AbsTol, MaxTimeStep, InitialTimeStep, NonlinSolverConvCoeff, MaxNonlinIter |
| ExplicitEuler | --- | InitialTimeStep |

*Table 27. Capabilities and Supported Flags/Parameters for the provided LES solvers*

| LES solver | Preconditioners | Supported integrator parameters/flags |
|---|---|---|
| DENSE | --- | --- |
| KLU | --- | --- |
| GMRES | ILU | PreILUWidth, MaxKrylovDim, IterativeSolverConvCoeff |
| BiCGStab | ILU | PreILUWidth, MaxKrylovDim, IterativeSolverConvCoeff |

## 2.15. Model Parametrization

This section describes the various model parametrization blocks.

### 2.15.1. Natural Ventilation Model (Infiltration)

A natural ventilation/infiltration model defines air exchange with the outside air.

*Example 45. Parameter definition for a natural ventilation model (Constant and Scheduled model variants)*

```xml
<NaturalVentilationModel id="501" displayName="Zone vent" modelType="Constant">
  <ZoneObjectList>All zones</ZoneObjectList>
  <IBK:Parameter name="VentilationRate" unit="1/h">0.5</IBK:Parameter>
</NaturalVentilationModel>
<NaturalVentilationModel id="502" displayName="Zone vent" modelType="Scheduled">
  <ZoneObjectList>All zones</ZoneObjectList>
</NaturalVentilationModel>
```

The `NaturalVentilationModel` needs to be defined with the following XML attributes:

*Table 28. Attributes*

| Attribute | Description | Format | Usage |
|---|---|---|---|
| `id` | Identifier of the natural ventilation model | > 0 | *required* |
| `modelType` | Sets the type of the heat conduction model<br>• `Constant` - Constant air change rate<br>• `Scheduled` - Air change rate changes accoring to defined schedule | key | *required* |

Floating point parameters (see section IBK:Parameter for a description of the `IBK:Parameter` tag):

| Name | Default Unit | Description | Value Range | Usage |
|---|---|---|---|---|
| `VentilationRate` | 1/h | Constant air change rate | >= 0.0 | *required for Constant model* |

If the model type `Scheduled` is being used, no further parameters are required, since the air change rate is taken from a scheduled parameter.

For the model to work, the parameter `InfiltrationRateSchedule` must be defined in a schedule (see Schedules).

Below is an example of a schedule that provides the `InfiltrationRateSchedule` for such a scheduled natural ventilation model:

*Example 46. Example schedule that provides a InfiltrationRateSchedule parameter*

```
<ScheduleGroup objectList="All zones">
  <!-- every day between 8-10 -->
  <Schedule type="AllDays">
    <DailyCycles>
      <DailyCycle interpolation="Constant">
        <TimePoints>0 6 10</TimePoints>
        <Values>InfiltrationRateSchedule [1/h]:0 0.4 0</Values>
      </DailyCycle>
    </DailyCycles>
  </Schedule>
  <!-- Tuesday no ventilation -->
  <Schedule type="Tuesday">
    <DailyCycles>
      <DailyCycle interpolation="Constant">
        <TimePoints>0</TimePoints>
        <Values>InfiltrationRateSchedule [1/h]:0</Values>
      </DailyCycle>
    </DailyCycles>
  </Schedule>
  <!-- Weekend only on afternoon -->
  <Schedule type="WeekEnd">
    <DailyCycles>
      <DailyCycle interpolation="Constant">
        <TimePoints>0 14 16</TimePoints>
        <Values>InfiltrationRateSchedule [1/h]:0 0.1 0</Values>
      </DailyCycle>
    </DailyCycles>
  </Schedule>
</ScheduleGroup>
```

### 2.15.2. Shading Control Model

A shading control model is a special type of control model, returning a signal value between 0 (no shading) and 1 (fully shaded). The actual amount of shading, or reduction of solar gains, is determined by the shading parameter block. Thus, you can use the same control model for different types of shadings.

*Example 47. Parameter definition for shading control model*

```
<Models>
  <!-- ShadingControlModel returns a value between 0 and 1
    0 = no reduction (shading open)
    1 = full reduction (shading closed)
  -->
  <ShadingControlModel id="555" displayName="Roof sensor" modelType="SingleIntensityControlled">
    <!-- Retrieves global radiation on given sensor. -->
    <SensorID>21</SensorID>
    <IBK:Parameter name="MaxIntensity" unit="W/m2">200</IBK:Parameter>
    <IBK:Parameter name="MinIntensity" unit="W/m2">100</IBK:Parameter>
    <!-- Shading control model returns 1 for values below Min, and 0 for values above Max.
      Switching is only allowed if Max/Min has been passed (hysteretic control).
    -->
  </ShadingControlModel>
</Models>
```

## 2.16. Reference

### 2.16.1. 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 / (slash) may be part of the unit name. All units following the slash are in the denominator of the unit. Exponents are just following the unit, for example `m2`. Dots in exponents are omitted, for example `h05` for square root of the hour. 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 | |
| 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 |

| Base SI unit | Convertible units |
|---|---|
| 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 | |
| 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 | |

| Base SI unit | Convertible units |
|---|---|
| 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 |
| W/mK2 | |
| W/Person | kW/Person |
| *undefined* | |

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

### 2.16.2. 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. |

| Reference/object type | Quantity | Unit | Description |
|---|---|---|---|
| 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. |
| 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. |

# 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:

```xml
<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:

```xml
<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 Construction Instances documentation.

Example:

```xml
<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 (see section Time types for a discussion).

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:

```xml
<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 (see section Output file names). An hourly time grid was selected as interval. The output takes place over the entire simulation duration.

Example:

```xml
<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:Parameter 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