NANDRAD Model Reference

Andreas Nicolai
andreas.nicolai@tu-dresden.de

Version 1.0.0, July 2020

Table of Contents

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

1. Overview

1.1. Simulation time and absolute time reference

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

2. Models and Input Reference

2.1. Zones

2.2. Constructions

2.3. Interfaces (construction boundary conditions)

2.3.1. Ambient climate boundary conditions

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

2.4. Construction Types

2.5. Materials

2.6. Climatic loads

2.6.1. Overview

The climatic loads model is a purely time-dependent model without other input dependencies. For solar radiation calculation, it needs information on the building location. Also, for most simulations, a climate data file is needed.

2.6.2. Specification

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

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

The parameter `Albedo` is used for diffuse solar radiation calculation. `Altitude` is needed for specific altitude-related parameters (**TODO**).

Parameters `Latitude` and `Longitude` are optional if a climate data file is given, otherwise they override the location parameters stored therein (see Building/Station location).

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).

Climate File Support

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

Building/Station location

Typically, climate data files contain information on latitude and longitude of the weather station.

- If these parameters are not specified explicitely in the project file, the latitude/longitude from the climate data file are used.
- If 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.

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.

2.6.3. Implementation

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

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

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

Registering surfaces

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

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

2.7. Schedules

2.7.1. Overview

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

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

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

2.7.2. Defining schedules

Simulation time to day type/local time mapping

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

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

Time/day mapping in cyclic annual schedules

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

The following conventions apply:

- start year is always 2003
- start time is given as parameter (as offset to Midnight January 1st 2003, or `"01.01.2003 00:00"`); for example, start time of `10.5 d` means simulation time 0 maps to `"10.01.2003 12:00"`
- if simulation duration exceeds 1 year, simulation time is wrapped at 365 d
- "schedule lookup time" is the same as simulation time
- leap days are never used, even if start year is set to 2000 and similar leap years
- parameter "DayOfTheWeekAtStart" indicates which day of the week corresponds to the first day of simulation (i.e. the day of start time, for example, one could specify "Wed" as day type and in the example above the 10.01. would become a wednesday)

Example:

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

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

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

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

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

Constructing spline data from input data

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

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

Example:

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

  d_dayOfWeek -> 16 + 2 = 18   18 % 7 = 3  -> DayType = "Thursay" (Check: 16. January 2003 was a Thursday)

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

Continuous data

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

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

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

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

Data definition rules

A certain variable must be only defined once per object list

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

A variable must be defined unambiguously with respect to addressed object

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

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

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

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

Regular schedules (based on daily cycles)

…

Annual schedules (as linearly interpolated splines)

Annual schedules are basically data tables with

2.7.3. Implementation

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

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

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

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

Variable lookup

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

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

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

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

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

Variable lookup for outputs/lookup by schedule name

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

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

Such a reference could look like:

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

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

2.7.4. Variable list

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

3. Global parameters

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

3.1. Simulation Parameters

3.2. Solver Parameters

4. Outputs: definitions, rules and implementation

4.1. Output definitions

Outputs are defined via:

- Grid (defines when outputs are to be written)
- Variable/Quantity ID name
- Object List (selects object to retrieve data from)
- Time handling (how to handle time averaging/integration)
- (optional) target filename

Example:

```xml
<!-- Definition of output grids -->
<Grids>
    <OutputGrid name="hourly">
        <Intervals>
            <Interval>
                <IBK:Parameter name="StepSize" unit="h">1</IBK:Parameter>
            </Interval>
        </Intervals>
    </OutputGrid>
</Grids>
<!-- Definition of outputs -->
<OutputDefinitions>
    <OutputDefinition>
        <Quantity>Temperature</Quantity>
        <ObjectListName>All zones</ObjectListName>
        <GridName>hourly</GridName>
    </OutputDefinition>
    <OutputDefinition>
        <Quantity>NaturalVentilationFlux</Quantity>
        <ObjectListName>All zones</ObjectListName>
        <GridName>hourly</GridName>
        <TimeType>Integral</TimeType>
    </OutputDefinition>
</OutputDefinitions>


<!-- Referenced object list -->
<ObjectLists>
    <ObjectList name="All zones">
        <FilterID>*</FilterID>
        <ReferenceType>Zone</ReferenceType>
    </ObjectList>
</ObjectLists>
```

`TimeType` is optional, same as `None` by default. `FileName` is optional (see below).

4.1.1. 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.

The intervals are expected to follow in consecutive order, optionally with a gap in-between.

Intervals can have up to 3 parameters:

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

Time definitions

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 in last das Intervall das letzte Intervall ist (in diesem Fall läuft das Interval bis zum Simulationsende) das darauffolgende Interval hat ein Start-Wert hat. Bei Angabe eines Start-Parameters muß der Zeitpunkt mindestens dem End-Zeitpunkt des vorangegangenen Intervalls entsprechen → die Intervalle sind sequentiell geordnet und überschneiden nicht. Die Angabe des Parameters StepSize ist zwingend.

4.1.2. Examples

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

4.2. 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)

4.3. Output file generation rules

4.3.1. When no filename is given

Target file name(s) are automatically defined.

All outputs a grouped by:

- grid and quantity type (Load, Schedule, State, Flux, Misc), quantity type is predefined for most quantity IDs (unknowns → Misc)
- for each used grid:

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

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

4.3.2. When a filename is given

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

4.4. Binary Format

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

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