NANDRAD Model Reference

Andreas Nicolai
andreas.nicolai@tu-dresden.de

Version 1.0.0, July 2020

Table of Contents

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

2.1. Climatic loads

2.1.1. Overview

- time-dependent
- climate data file reading
- location settings in climate data file
- location settings in NANDRAD project file → override rules
- albedo
- radiation sensors

2.1.2. 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.

2.2. Schedules

2.2.1. 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.

2.2.2. Defining schedules

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 **most 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.

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

2.2.3. 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

When model input variables are resolved, schedules are the

3. Output definitions, conventions/rules and implementation aspects

3.1. Output definitions

Outputs are defined via:

- Grid (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).

3.1.1. 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).

3.2. Rules

3.2.1. 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.

3.2.2. 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

3.3. 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