File formats in INCABuilder

Magnus Dahler Norling

March 20, 2019

Contents

1 Introduction

This document describes how to use the parameter and input file formats that the INCABuilder framework has built-in functionality to load or generate.

2 The standard .dat format

2.1 General

The .dat format is the format that is the easiest to work with when setting up index set structure, parameters and inputs for a new dataset. The .dat files are utf-8 encoded text files. We recommend editing them in a plain text editor like Notepad++.

Note 1. If you find this section too technical, just skip to the examples.

i You can put a comment in one of these files using the # symbol. This will treat the rest of the line as a comment that is ignored when parsing the file.

ii Quoted strings are sequences of characters enclosed in a pair of quotation marks " ". These can contain contain utf-8 encoded strings, but not newlines.

iii Outside quoted strings, everything has to be ascii.

iv Except for inside quoted strings and except for newlines after a #, all consecutive sequences of whitespace (space characters, tabulars, newlines) are treated as one single space.

v Each .dat file is treated as a series of tokens. A token is a sequence of characters that encode one value or symbol. An overview of the types of tokens are given in Table 2.1.

Token type	**Example**
Colon	`:`
Open brace	{
Close brace	}
Number	`8` or `0.05` or `1e-9` or `NaN`
Boolean	`true` or `false`
Unquoted string	`index_sets`
Quoted string	`"Timesteps"`

Table 1: Types of allowed tokens in the standard format

Both quoted strings and unquoted strings are always case sensitive, so you can for instance not type "landscape units" if you want to talk about the index set "Landscape units".

2.2 The parameter file

Example 1. A shortened example of a parameter file

```
index_sets :
# You can add in as many landscape units as you like,
# but remember to fill in corresponding parameter values!
"Landscape units" : {"Forest" "Agricultural"}
"Soils" : {"Direct runoff" "Soilwater" "Groundwater"}
"Reaches" : {"Reach 1" {"Reach 2" "Reach 1"}}

parameters :
"Timesteps" :
365

"Start date" :
"1999-1-1"

"Growing degree threshold" :
1 2.0

"Percolation matrix" :
0.2 0.8 0.0
1.0 0.5 0.5
0.0 0.0 1.0

0.1 0.9 0.0
1.0 0.6 0.4
0.0 0.0 1.0

"Reach has effluent input" :
false true

"%" :
9 81
33.0 67.0
```

Every parameter file, like the file in Example 1, needs to have two sections (in that order)

i `index_sets :` This section describes the indexes of each index set that is present in the model. Here you can for instance configure the number of landscape units or reaches in your dataset.

ii `parameters :` This section lets you fill in values for individual parameters. If a parameter indexes over one or more index set, you need to provide a number of values corresponding to the number of (tuples of) indexes.

2.2.1 Index sets

Which index sets that are present are determined by the model you want to run. Most often you are free to determine which indexes they have, but sometimes, the indexes of an index set are also specified by the model. For instance, INCA-N requires you to have the indexes {"Direct runoff" "Soilwater" "Groundwater"} for soils. In any case, you have to provide a set of indexes for each index set in the parameter file. There are two types of index sets, basic and branched.

i Basic index sets are index sets with no additional structure except for the order of the indexes. One example is "Landscape units" in Example 1. To provide the indexes of a basic index set, type the quoted name of the index set followed by a colon and a brace-enclosed space-separated (no commas) list of the quoted names of the indexes.

ii In a branched index set, each index can have zero or more branch inputs, where each input is another index. In INCA type models, this is used to encode a river network of reaches, where each reach can be an input to other reaches. The indexes of a branched index set are provided as a brace-enclosed list of items, where each item is either a quoted name of an index or a brace-enclosed list of quoted names of indexes. If the item is a name appearing alone, it signifies an index that does not have any inputs. If an item is a list, the first string of the list is a new index, and the rest of the strings are names of previously declared indexes that are inputs to this one.

Example 2. The river system has reaches "R1", "R2" and "R3", where "R1" is an input to "R2" and "R2" is an input to "R3".

```
"Reaches" : {"R1" {"R2" "R1"} {"R3" "R2"}}
```

Example 3. The river system has reaches "R1", "R2" and "R3", where "R1" and "R2" have no inputs of their own, but are both inputs to "R3" (forming a Y-like shape).

```
"Reaches" : {"R1" "R2" {"R3" "R1" "R2"}}
```

Example 4. The river system consist of three separate reaches "R1", "R2" and "R3", where none of them are inputs to any others. This could potentially happen if you want to simulate three separate streams at the same time that for instance are inputs to a lake (where the lake itself is not modelled by this framework).

```
"Reaches" : {"R1" "R2" "R3"}
```

2.2.2 Parameters

The parameter section consists of a series of quoted names of parameters followed by their values. Any parameter in the model will have one of four types (that are specified by the model).

Parameter type	Example value
double	`8` or `0.05` or `1e-9` or `NaN`
uint	`10`
boolean	`true` or `false`
time	`"1999-1-1"`

Table 2: Parameter value types

i Parameter values of type **double** (double-precision floating point numbers) can written as numbers with or without a decimal separator '.', or in scientific notation. Technically you could also provide a NaN (not-a-number), but that will probably crash the model in most cases.

ii Values of type **uint** (unsigned integers) have to be written as a non-negative number without any commas. These are often used to denote a number of timesteps or days of duration for some process.

iii **Boolean** values are either `true` or `false`. These are typically switches that turn on or off a process.

iv Values of **time** type are provided as a date on the format "y-m-d".

Every parameter indexes over a list of index sets determined by the model. Note that an index set can appear more than one time in this list. If a parameter indexes over no index sets, it has only one value. If it indexes over one index set it has one value for each index in that index set. If it indexes over more than one index set, it has one value for each element in the cartesian product[1] of these index sets, i.e. the number of values is equal to the product of the number of indexes.

In the following examples, the index sets are as in Example 1. These examples will also describe the order that the values should appear in.

Example 5. The parameter "Timesteps" does not index over any index sets, so it has only one global value.

```
"Timesteps" :
365
```

Example 6. The parameter "Growing degree threshold" indexes over the index set "Landscape units", which has two indexes "Forest" and "Agricultural". "Growing degree threshold" thus has two values.

```
"Growing degree threshold" :
1 2.0
```

Here `1` is the value for "Forest", while `2.0` is the value for "Agricultural".

Example 7. The parameter "%" indexes over the index sets "Reaches" and "Landscape units" (the order of these are important), both having two indexes. "%" thus has 4 values. Which reach you are in is indicated by the row, while which landscape unit is indicated by the column.

```
"%" :
9 81
33.0 67.0
```

Here `9` is the value for {"Reach 1" "Forest"} (signifying that the catchment of reach 1 has a 9% forest cover). Similarly, {"Reach 1" "Agricultural} is 81, {"Reach 2" "Forest} is 33.0 and {"Reach 2" "Agricultural} is 67.0

[1] Remember that the Cartesian product of sets X and Y is the set of tuples (x, y), with $x \in X$ and $y \in Y$.

Example 8. The parameter "Percolation matrix" indexes over the index sets "Landscape units", "Soils" and "Soils" (Soils appearing twice). "Landscape units" has 2 indexes, while "Soils" has 3, so "Percolation matrix" has $2 * 3 * 3 = 18$ values.

```
"Percolation matrix" :
0.2 0.8 0.0
1.0 0.5 0.5
0.0 0.0 1.0

0.1 0.9 0.0
1.0 0.6 0.4
0.0 0.0 1.0
```

The landscape unit signifies which block matrix you are in (the first block matrix is for "Forest", the second for "Agricultural"). The first soil index is the row, while the second is the column. In this case, the "Percolation matrix" of each landscape unit appears exactly as they are specified in the PERSiST model [1].

The general rule for the order values appear in is that you index over the rightmost index set first, then the next to the left, then the next to the left and so on.

Note 2. It is the order that the values appear in that is important. You could remove the newlines between value rows. The newlines are only there for visual clarity for a human editor.

If you provide the name of a parameter, you have to provide all of its values. However, you can omit a parameter from the file entirely. In that case, that parameter will get its model-specified default value.

Usually you will work with parameter files that have been autogenerated by the model. In that case, the files will contain useful comments about which parameters index over which index sets, what are their units and model-recommended min and max values, and often a description of the parameters.

Example 9. A shortened example of an autogenerated parameter file

```
# Parameter file generated for PERSiST V1.1 at 2019-02-11 15:30:40

index_sets :
"Landscape units" : {"Forest" "Agricultural"}
"Soils" : {"Direct runoff" "Soilwater" "Groundwater"}
"Reaches" : {"Reach 1" {"Reach 2" "Reach 1"}}

parameters :
####################### (no index sets) ######################
"Timesteps" :   #(days) [0, 18446744073709551615]
365

"Start date" :   # ["1000-1-1", "3000-12-31"]
"1999-1-1"

####################### "Landscape units" ######################
"Growing degree threshold" :  #(°C) [-4, 4] The temperature at or above which plant
    growth and hence evapotranspiration are assumed to occur
1 2

####################### "Landscape units" "Soils" "Soils" ######################
"Percolation matrix" :  #(dimensionless) [0, 1]
0.2 0.8 0.0
1.0 0.5 0.5
0.0 0.0 1.0

0.1 0.9 0.0
1.0 0.6 0.4
0.0 0.0 1.0

####################### "Reaches" ######################
"Reach has effluent input" :
false true

####################### "Reaches" "Landscape units" ######################
"%" :    #(%) [0, 100] The percentage of a subcatchment occupied by a specific land
    cover type
9 81
33.0 67.0
```

2.3 The input file

Example 10. A shortened example of an input file

```
start_date :
"1999-1-1"

timesteps :
5

additional_timeseries :
"Discharge"

index_set_dependencies :
"Actual precipitation" : {"Reaches"}
"Discharge" : {"Reaches"}

inputs :

"Air temperature" :
-10.7251
-18.1251
-14.8251
-23.6251
-24.7251

"Actual precipitation" {"Reach 1"} :
2.75
3.3
2.75
15.07
8.03

"Actual precipitation" {"Reach 2"} :
1.54
1.65
0.88
7.15
1.54

"Discharge" {"Reach 1"} :
"1999-01-01"  0.001982
"1999-01-02"  0.001982
"1999-01-04"  0.001428
"1999-01-05"  0.00404
end_timeseries

"Discharge" {"Reach 2"} :
"1999-01-01"  0.001486
"1999-01-03"  0.003005
"1999-01-05"  0.002082
end_timeseries
```

Input files, like the file in Example 10, can or must have the following sections

i `start_date :` (optional) The start date of the input data. This can be earlier or equal to the model run start date that is provided by the "Start date" parameter in the parameter

file. If this section is not in the input file, the input start date will be assumed to be equal to the model run start date.

ii `timesteps :` (required) The number of timesteps (days) that you wish to allocate for your input data. You have to provide enough timesteps to cover the entire model run, meaning that the input start date plus the input timesteps must be greater than or equal to the model run start date plus the model run timesteps.

iii `additional_timeseries :` (optional) Here you can provide the name of additional timeseries that you want to put into the dataset, that are not required by the model. This is for instance used to be able to load measured comparison data for viewing together with the model results in INCAView, or for use in calibration routines.

iv `index_set_dependencies :` (optional) If one or more of your timeseries varies over some of the model index sets, you have to declare so here. For instance, you can choose whether or not you have an "Air temperature" timeseries that is the same for every subcatchment, or if you have a separate timeseries for each reach. You can add dependencies on multiple index sets at the same time, so "Air temperature" could depend on both "Reaches" and "Landscape units" (not that that is likely), but if you do something too crazy you may break the model. This is because the model may adapt to evaluate equations such as "Precipitation falling as snow" to evaluate for every index you have a precipitation and air temperature timeseries for, and this will propagate to evaluations of other equations that depend on the value of "Precipitation falling as snow". Generally you should be safe if you do something that feels sensible, such as having one timeseries per reach.

v `inputs :` (required) The actual input data. If you did not declare any index set dependencies for an input, you only provide one timeseries. Otherwise you provide one for each (combination of) index(es) in the index set(s) it depends on.

You can completely omit providing data for a timeseries (i.e. don't type its name or any values for it). In that case it will be asumed to be equal to 0 for every timestep. This is usually not recommended. There are two formats you can choose.

i If you use the dense format, just list all the values after one another. If you use this format you have to provide exactly as many values as the number of timesteps you declared for the input data.

ii If you use the sparse format, list pairs of dates (format "y-m-d") and values. If you use this format, dates in the start date - timesteps range that do not receive a value will be assigned NaN. This means that this format is mostly suitable for observation timeseries and not for model required inputs, since the model will most likely crash on getting a NaN value. The exception is if you fill in all the values (maybe using constant periods as described below). Every sparse timeseries has to be ended with a `end_timeseries` token. The dates do not have to be provided in order.

Input values can be provided in the same format as parameters of type double in the parameter file. I.e. they can be numbers with or without decimal separator or in scientific notation.

Note 3. If you have a timeseries of, say, "Air temperature" that you want to use as a model input timeseries (as opposed to an additional observation timeseries) and it has missing values, you have to do preprocessing on it yourself to interpolate the missing values before you put it in the input file.

There are a few more advanced options you can use when providing inputs.

Example 11. Declaring a unit for an additonal timeseries.

```
additional_timeseries :
"Discharge" unit "m3/s"
"Nitrate concentration"   #NOTE: you can freely declare units for some timeseries
    while omitting them for others
```

Example 12. Providing the same timeseries for multiple (combinations of) indexes at the same time

```
"Actual precipitation" {"Reach 1"} {"Reach 2"} :
1.54
1.65
0.88
7.15
1.54

"Actual precipitation" {"Reach 3"} {"Reach 4"} :
2.75
3.3
2.75
15.07
8.03

"Fertilizer nitrate" {"Reach 1" "Agricultural"} {"Reach 2" "Agricultural"} {"Reach 3"
     "Agricultural"} :
0.4
0.4
0.5
0.5
0.6
```

As in Example 12 you can make sure the same timeseries is used in more than one, but typically not all, instances (if you wanted one series to be used in all instances, you would just not give it an index set dependency).

Example 13. Declaring constant periods of values

```
"Nirate dry deposition" :
"1999-01-01" to "1999-05-31" 0.01
"1999-06-01" to "1999-12-31" 0.02
end_timeseries
```

As in Example 13 you can declare that the timeseries should have a given constant value between two dates. Both the start date and end date is inclusive. Note that this otherwise counts as using the sparse format, that is missing dates will be given the NaN value. Moreover, the periods do not have to be declared in order. You can also combine these advanced features freely.

Example 14. Combining various advanced formats.

```
"Nirate dry deposition" {"Reach1"} {"Reach2"}:
"1999-01-01" to "1999-05-31" 0.01
"1999-06-01" to "1999-12-31" 0.02
"2000-01-01" 0.01  #NOTE: you can provide values for single days in-between periods
"2000-01-02" 0.015
"1999-01-03" to "1999-05-31" 0.01
end_timeseries
```

2.3.1 Converting from .csv or .xlsx

In many cases, your input data will a priori exist in a .csv format or an .xlsx format. Since there are very many ways your data could be formatted it would be difficult for us to make a converter that can handle every possibility. If you don't know any python or other scripting languages you could use to convert them, you may have to learn some tricks. We will just give some quick tips here. A more complete tutorial may be for the future.

i If your data are opened in excel (or open office calc) your can click the letter of a column in the top bar to select the entire column, then ctrl-c to copy it. You can now paste it in a text editor.

ii You need all values to have a '.' decimal separator rather than a ','. You can either do a search-replace in a text editor after copying the values, or in excel go to File→Options, On the Advanced tab, under Editing options, uncheck the Use system separators box, then type in the right Decimal separator in its box.

iii To format a column of dates to the "y-m-d" format, select the colum, then right click it and select Format Cells. Under Date select the right format.

iv If you have one date column and many data columns and you want to copy just the date column together with one data column, you can temporarily copy the data column next to the date column and ctrl-select and copy the two together.

v You may still need to enclose all dates in quotation marks " ". This can be tedious to do manually. It is pretty easy to do it using macros.

To use macros in Notepad++, look for the buttons in Figure 1.

Figure 1: Notepad++ macro buttons

Say that you have a file that looks like this

Example 15.

```
1999-01-01    0.001982
1999-01-02    0.001982
1999-01-04    0.001428
1999-01-05    0.00404
```

and you want to put quotation marks around the dates. You can use Notepad++ macros to record your keyboard strokes in order to repeat the same operation many times.

i Move your cursor to the left of the first date.

ii Click the red circle to start recording.

iii Type in a ” symbol.

iv Press the right arrow on the keyboard until the cursor is at the right of the date.

v Type another ” symbol.

vi Press the left arrow until the cursor is at the beginning of the line again.

vii Press the down arrow once to move the cursor to the beginning of the next line.

viii Click the black square symbol to stop the recording.

ix You can now click the triangle symbol to repeat these keystrokes for one line at a time, or the double triangle to do it for multiple lines.

2.4 Error messages

Most often if you get an error during the parsing of a parameter or input file, it will tell you exactly at which line and column in the file that the error happened. Right now an exception to that is if you provided a name of a model entity (index set, index, parameter or input) that the model did not recognize. In that case it will tell you the name that it did not recognize, but it will not say where or in which file it happened. This will hopefully be improved soon. Here is a non-exhaustive list of error messages you could encounter.

Example 16. The message:

```
ERROR: Tried to look up the handle of the IndexSet "landscape units", but it was not
    registered with the model.
```

The culprit line:

```
"landscape units" : {"Forest" "Wetland"}
```

Explanation: The model only recognizes ”Landscape units” as the name of one of its index sets. This is case sensitive.

Example 17. The message:

```
ERROR: In file langtjernparameters.dat line 5 column 75: Found a token of unknown
    type.
```

The culprit line:

```
"Soils" : {"Direct runoff" "Organic layer" "Mineral layer" "Groundwater"} æ
```

Explanation: ’æ’ is not a valid token (though you can still use that character inside quoted strings if you want to).

Example 18. The message:

```
ERROR: In file langtjerninputs.dat line 12 column 27: Did not get the right amount of
     indexes for input Air temperature
```

The culprit line:

```
"Air temperature" {"Bork"}:
```

In this case you have declared "Air temperature" to depend on either 0 or 2 or more index sets.

Example 19. The message:

```
ERROR: Tried the index name Bork with the index set Reaches, but that index set does
    not contain that index.
```

The culprit line:

```
"Air temperature" {"Bork"}:
```

In this case you have declared "Air temperature" to depend on the index set "Reaches", but "Bork" is not one of the indexes you provided for that index set in the parameter file.

3 The Json format

3.1 General

Json (Javascript object notation) is another text format that follows a standard that is recognized by many programming language libraries and applications. This format is typically used to serialize the dataset, for instance in order to easily read it from other programs or send it over the web. The reason this is not used as the standard format is that it is less convenient for manual editing and does not support having comments.

3.2 The parameter file

The format of the parameter file is similar to the dat format. Apart from having to obey the json standard, the file has to obey a structure similar to that given in Example 20

Example 20. The structure of the parameter file in json

```
{
       "index_sets" : {
              "Name of regular index set" : ["index 1", "index 2", "index 3"],
              "Name of branched index set" : [
                     ["index", "branch input 1", branch input 2"],
                     <...etc...>
              ]
       },
       "parameters" : {
              "Parameter 1" : [1, 2, 3],
              "Parameter 2" : [true, false],
              <...etc...>
       }
}
```

i The file is a dictionary containing two keys, `"index_sets"` and `"parameters"`.

ii The value of `"index_sets"` is a dictionary where every key is the name of an index set.

iii The value of a basic index set is a list of the names of its indexes.

iv The value of a branched index set is a list of lists, where every sub-list is first the name of an index and then all the branch inputs to that index.

v The value of `parameters"` is a dictionary where every key is the name of a parameter.

vi The value of every parameter is a list of values. These values are given in the same order as in the .dat format (but have to be comma separated as per the json standard).

3.3 The input file

The format of the input file is similar to the dat format. Apart from having to obey the json standard, the file has to obey a structure similar to that given in Example 21

Example 21. The structure of the input file in json

```
{
	"creation_date": "2019-03-07 12:36:29",
	"additional_timeseries": [
		"Observed discharge"
	],
	"index_set_dependencies" : {
		"Air temperature" : ["Reaches"]
	},
	"start_date": "1981-01-01",
	"timesteps": 10
	"data": {
		"Air temperature": [
			{
				"indexes": ["Reach 1"],
				"values": [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
			},
			{
				"indexes": ["Reach 2"],
				"values": [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
			},
		],
		<...etc...>
	}
}
```

i The file is a dictionary containing 6 keys, `creation_date`, `additional_timeseries`, `index_set_dependencies`, `start_date`, `timesteps` and `data`.

ii `creation_date` is optional and its value is a string containing the time the file was created (if it was autogenerated).

iii The value of `additional_timeseries` is a list of names of additional input timeseries.

iv The value of `index_set_dependencies` is a dictionary where every key is the name of an input. The value associated to every key is a list of names of index sets.

v The value of `start_date` is a string with a date on the format ”y-m-d”.

vi The value of `timesteps` is a number.

vii The value of `inputs` is a dictionary where every key is the name of an input.

viii The value of every input in `data` is a list of dictionaries. Each of these dictionaries has the two keys `indexes` and `values`. The value of `indexes` is a list of names of indexes for this timeseries. The value of `values` is a list of numeric values. Unlike in the dat format, `NaN` is not a valid json identifier, so `null` is used to signify a missing value instead.

4 The sqlite3 format

4.1 General

Sqlite is a database format. This format is typically used to communicate with INCAView.

#TODO Complete the description!

References

[1] M. N. Futter, M. A. Erlandsson, D. Butterfield, P. G. Whitehead, S. K. Oni, and A. J. Wade. PERSiST: a flexible rainfall-runoff modelling toolkit for use with the INCA family of models. *Hydrol. Earth Syst. Sci.*, 18:855–873, 2014.