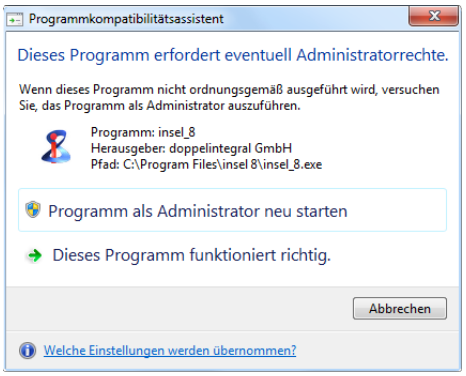
Module 3 :: Reading and writing data files

In this module, you will learn how to read data from files and write data to files and how these data can be used in simulations. We will concentrate on data files which contain meteorological data since meteorology is one of the most widely-spread applications of INSEL.

A general warning

Reading files means that you enter explosive ground. When you do not exactly know, what kind of information is saved in a data file that you are going to read, it is quite probable that your program crashes with a message similar to this:



Luckily in computing this means that only your program smashes, or if the blunder is too bad, you have to restart your computer.

The story of writing data is even more dangerous. Writing data to files means that you are manipulating bits on the hard disk of your computer. You can imagine, that if – by accident – you change some of the bits in the Windows operating system, this may lead to a really serious crash. In the worst case, you can throw away your computer and buy a new one.

So, it's really worth to study this Module with the necessary care!

Let us start with the seemingly trivial question "What is a data file?"

Naming conventions

Whenever you work with a computer you work with files – it is impossible to do something with a computer that is not related to files. Under Windows the Explorer – which probably everybody has seen crash due to wrong file handling – is a tool which lets you organise millions of files. All these files have names following a specific naming convention.

In the "historic" age of the eighties and nineties of the 20^{th} century the naming convention was: A file name may have a maximum of eight characters, followed by a dot and a file extension. The file extension was restricted to a maximum of three

characters. Under Windows a file could be saved under any path name with the same naming convention.

Today, everything is more comfortable – a file name can be quite lengthy and may even contain space characters, more than one dot and the length of path, name and extension is practically unrestricted. insel 8 can deal with any naming convention.

We are used to distinguish files by their extension: When we see a file with extension `.doc` we think "Aah, a Word document!," or a file with extension `.pdf` "Of course! This is an Acrobat file." Maybe one day people think "Yes, an INSEL file – what else!," when they see the extension `.insel`.

ASCII code

What makes up data files is that their content follows conventions, too. Like Enigma files are encoded in a specific "secret" code. One example for standardised code is the ASCII code: Every symbol – like letters and digits – is decoded by a series of seven bits, which can take either a value of zero, or a value of one – the dual system. Extended ASCII code uses eight bits, so that a larger set of symbols can be expressed. Eight bits are commonly called a byte. The trend goes to Unicode which uses sixteen bits, i. e. two bytes.

Hence, we can answer the question "What is a data file?" with the statement "A data file is nothing but a stream of bits. The meaning of this data stream needs to be known exactly – otherwise the data stream is completely useless."

`meteo82.dat`

The following lines show the head of a file that has been recorded in Oldenburg, North Germany, in the year 1982 – four years before INSEL 1.0 – named `meteo82.dat`.

```
 1 182 1   0.   0.   0.   0.   0.  4.5-40.  -40.0-40.0 265.  4.4
 1 182 2   0.   0.   0.   0.   0.  3.8-40.  -40.0-40.0 255.  4.3
 1 182 3   0.   0.   0.   0.   0.  3.4-40.  -40.0-40.0 255.  2.8
 1 182 4   0.   0.   0.   0.   0.  3.1-40.  -40.0-40.0 225.  2.3
 1 182 5   0.   0.   0.   0.   0.  3.0-40.  -40.0-40.0 225.  2.6
 1 182 6   0.   0.   0.   0.   0.  2.9-40.  -40.0-40.0 225.  3.0
 1 182 7   0.   0.   0.   0.   0.  3.2-40.  -40.0-40.0 225.  1.9
 1 182 8   0.   0.   0.   0.   0.  2.8-40.  -40.0-40.0 205.  2.5
 1 182 9   0.   0.   1.   0.   0.  2.3-40.  -40.0-40.0 205.  2.4
 1 18210  15.   6.   6.   3.   0.  2.7-40.  -40.0-40.0 195.  3.0
 1 18211  40.  28.  21.  19.   6.  3.4-40.  -40.0-40.0 195.  2.6
 1 18212  54.  25.  18.  15.   5.  3.8-40.  -40.0-40.0 195.  1.2
 1 18213  60.  24.  18.  15.   4.  4.3-40.  -40.0-40.0  85.  0.6
 1 18214  44.  17.  13.  10.   1.  3.9-40.  -40.0-40.0  75.  1.2
 1 18215  16.   9.   8.   5.   0.  3.9-40.  -40.0-40.0  85.  1.1
 1 18216   0.   3.   4.   1.   0.  4.0-40.  -40.0-40.0  85.  1.0
 1 18217   0.   0.   0.   0.   0.  4.1-40.  -40.0-40.0  85.  1.3
 1 18218   0.   0.   0.   0.   0.  4.1-40.  -40.0-40.0  85.  1.2
 1 18219   0.   0.   0.   0.   0.  4.3-40.  -40.0-40.0  65.  0.9
 1 18220   0.   0.   0.   0.   0.  4.2-40.  -40.0-40.0 165.  1.0
 1 18221   0.   0.   0.   0.   0.  4.4-40.  -40.0-40.0 175.  1.2
```

```
1 18222   0.   0.   0.   0.   0.  4.9-40.  -40.0-40.0 195.  1.9
1 18223   0.   0.   0.   0.   0.  5.2-40.  -40.0-40.0 225.  2.2
1 18224   0.   0.   0.   0.   0.  5.3-40.  -40.0-40.0 225.  2.5
```

Without going into the details of this file for the moment some comments may be useful.

Records

- As you see, the file is organised in well-formatted "lines." Every line is usually called a record. So we'd better say: This example shows the first 24 records of the file `meteo82.dat`.
- Well-formatted lines means that all "columns" look alike, i. e. the decimal points are all in the same column, every line (more accurate: every record) has the same length. Take your time and count the number of columns. You should come to a value of 64 columns per record, including the blank or space characters.

Record length

Every column represents one alphanumeric symbol, represented by the corresponding symbol which can be a "0", a ".", or a space " ", for example. When each of the symbols is encoded in extended ASCII code, every column represents one byte. Hence we speak of bytes rather than columns. So we can conclude that when we want to describe the file we say: The file `meteo82.dat` is formatted and has a record length of 64 bytes.

Records end with a line break – otherwise we would see only one long, long line. Unfortunately, different operating systems use different conventions for line separators. Windows uses two bytes, i. e. a CR (carriage return) and LF (line feed), Mac OS only a CR, Linux only an LF. The line separator is usually not considered in the value for the record length. Again and again these different conventions are a source of trouble poor programmers have to live with. So again, be careful when you work with files in programming environments!

- Record 12 starts with the bytes `_1_18212__54` (for more clearness, we have replaced the invisible space characters by an underscore `_`). Does it mean that there is a number 18212 encoded in the file? Of course not. We humans conclude immediately that the first two bytes `_1` stand for day one of the data recording, the next two bytes `_1` stand for the month January, the next two bytes `82` are an abbreviation for the year 1982, the next two bytes `12` stand for the hour, and so on.

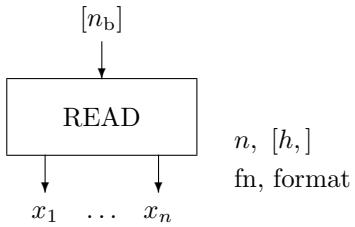
Fortran format

Computers don't conclude. They need to be told. This means that it is necessary to provide a "key" to any routine which shall interpret data files. Such a key is usually called a format. There are many conventions in computing that are used as formats. In INSEL the Fortran format conventions are used in order to describe the "keys" to data files.

3.1 Reading data

Reading data from files is a pre-requisite for many simulation runs, meteorological boundary conditions are required in practically all renewable energy simulations, for instance. Let us assume that some weather station has sent us a file with hourly ambient temperature data for one year, the file being named `temperature.dat`.

One INSEL block which can read data files sequentially is the READ block.



This block requires a parameter n for the number of values that is to be read per record, the file name fn, of course, and – very important – a parameter which describes the format of the file. In addition, there is an optional parameter h which allows us to start reading of the file not necessarily at the first record but at record number $h+1$, i. e. if we set $h > 0$, the READ block skips reading the first h records. And there is an optional input n_b which can be connected to any output of a block to express the dependence of the READ block from this block.

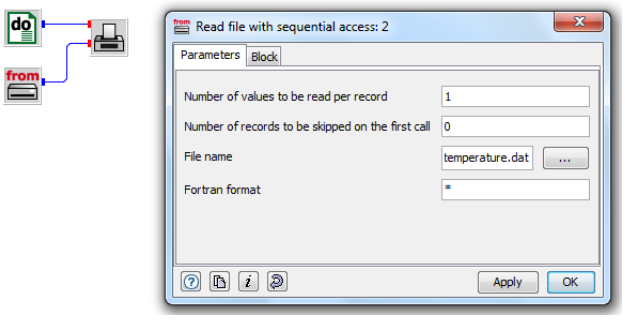
`temperature.dat`

The file `temperature.dat` is quite simple, it contains only one value per record. Here are the first ten records:

```
4.5
3.8
3.4
3.1
3.0
2.9
3.2
2.8
2.3
2.7
```

Star format

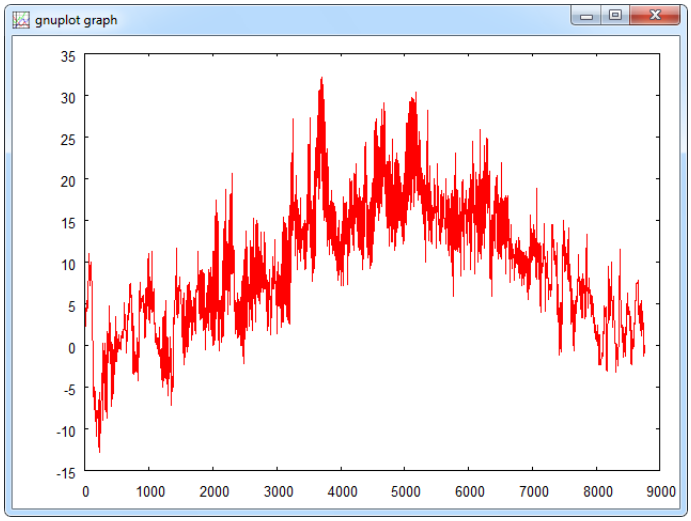
In this case and in cases where all data in a record are numbers and separated by a blank character (space) the star format is very easy to apply to reading such data files. In order to read the file `temperature.dat` it is easiest to use this star format like we did in `examples\blocks\inputOutput\read.vseit`:



We can identify all the above discussed parameters: The file name `temperature.dat`, the Fortran format `*` (both are string parameters which are entered without enclosing quotes), and the parameter $n = 1$ for the number of outputs and the skip parameter h, set to the default value zero here.

Well, one year has 8760 hours (if it is not a leap year), so we use a DO block which counts from 1 to 8760, and a PLOT block because we would like to see the time series. Please observe again, that the Standard block READ must not necessarily be connected to the Timer. If this disturbs you, you can add a data input terminal to the READ block and connect the DO block output – it makes no difference in this case, but perhaps it would make the model structure clearer.

When you run the application the graph with the temperature time series will show up.



Current directory

You may wonder, how INSEL found the file `temperature.dat` although no path information is given in the file name. By default, INSEL searches for files in the

directory of the model file. Of course, it is possible to include the full path to the file in the READ object. Either slashes or backslashes can be used, like `c:\myData\temperature.dat` or `c:/myData/temperature.dat`, for instance. The total length of the string is restricted to 1024 bytes – all string parameters in insel 8 are restricted to this length.

There are no problems to be expected when a file contains more than one value per record like a standard file, provided by the Fraunhofer Institute for Solar Energy Systems ISE in Freiburg. The file `data\weather\iseyear.dat` provides data in 15 minutes resolution of some meteorological parameters for the location of Freiburg im Breisgau, which is in the very south of Germany, close to the Swiss border. For this example we use only the July fraction of the file saved under `data\weather\iseyear7.dat`. The first records of this file are shown here:

iseyear7.dat

```
 7  1  0  7 30  17.5  95    0    0    0
 7  1  0 22 30  17.3  94    0    0    0
 7  1  0 37 30  17.1  92    0    0    0
 7  1  0 52 30  16.8  91    0    0    0
 7  1  1  7 30  16.6  90    0    0    0
 7  1  1 22 30  16.4  89    0    0    0
 7  1  1 37 30  16.2  88    0    0    0
 7  1  1 52 30  16.0  87    0    0    0
 7  1  2  7 30  15.8  86    0    0    0
 7  1  2 22 30  15.8  84    0    0    0
 7  1  2 37 30  15.7  83    0    0    0
 7  1  2 52 30  15.6  82    0    0    0
 7  1  3  7 30  15.6  81    0    0    0
 7  1  3 22 30  15.4  80    0    0    0
 7  1  3 37 30  15.3  79    0    0    0
 7  1  3 52 30  15.3  78    0    0    0
 7  1  4  7 30  15.3  76    0    0    0
 7  1  4 22 30  15.0  75    0    0    0
 7  1  4 37 30  14.7  74    5    0    5
 7  1  4 52 30  14.7  73   14    0   14
 7  1  5  7 30  14.6  72   31   52   27
 7  1  5 22 30  14.5  71   57  160   40
 7  1  5 37 30  14.7  69   83  228   50
 7  1  5 52 30  15.1  68  116  301   61
 7  1  6  7 30  15.6  67  150  366   70
 7  1  6 22 30  16.2  66  187  420   79
 7  1  6 37 30  16.9  65  226  466   88
 7  1  6 52 30  17.4  63  267  507   91
 7  1  7  7 30  18.1  60  309  543   99
 7  1  7 22 30  19.1  57  351  580  103
 7  1  7 37 30  20.2  53  395  620  107
 7  1  7 52 30  21.5  52  435  643  112
 7  1  8  7 30  21.2  52  473  658  117
 7  1  8 22 30  21.7  54  516  683  122
 7  1  8 37 30  21.6  55  557  700  129
 7  1  8 52 30  22.0  55  592  709  134
```

```
 7  1  9  7 30  21.6  55  629  713  145
 7  1  9 22 30  21.5  55  666  724  153
 7  1  9 37 30  22.1  55  701  736  158
 7  1  9 52 30  23.0  55  727  740  162
 7  1 10  7 30  23.3  52  758  751  166
 7  1 10 22 30  23.1  53  786  755  174
 7  1 10 37 30  23.0  52  811  758  180
 7  1 10 52 30  24.0  51  834  764  184
 7  1 11  7 30  24.5  49  851  762  191
 7  1 11 22 30  25.0  47  869  764  197
 7  1 11 37 30  25.0  45  882  763  202
 7  1 11 52 30  25.6  45  887  751  213
 7  1 12  7 30  25.7  45  894  748  217
 7  1 12 22 30  25.7  43  857  699  223
 7  1 12 37 30  26.0  41  906  768  209
 7  1 12 52 30  26.4  40  902  757  217
```

Interpretation

All data in the records are consequently separated by blanks so that we can use the star format to read this file. The file records contain

1 Month
2 Day
3 Hour
4 Minute
5 Second
6 Ambient temperature / °C
7 Relative humidity / %
8 Global horizontal irradiance / $\mathrm{W\,m^{-2}}$
9 Direct normal irradiance / $\mathrm{W\,m^{-2}}$
10 Diffuse horizontal irradiance / $\mathrm{W\,m^{-2}}$

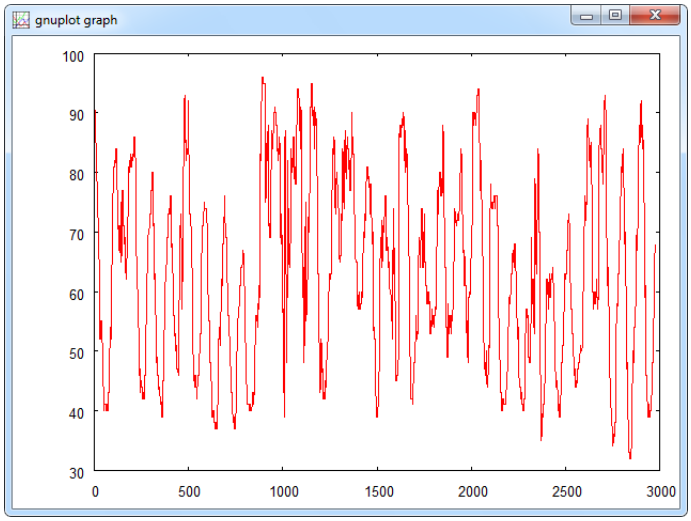
Exercise 3.1

The record length is 39 bytes. When you want to read the file, open a new VSEit network via *File* – *New* – or open the previously used file for the `temperature.dat` data – and save it under a new name, like `iseyear.vseit`, for example. The READ block so far has only one output, but we can add nine more via the block pane.

Now that your READ block has ten outputs you can give them understandable names, like M for month, d for day etc, by a double click on the respective ports.

Finally, choose some channels of your interest and analyze the file by plotting some time series portions from the file.

This graph shows the relative humidity, for example:



DWD data

Since many years monthly mean values of global radiation data are recorded by the DWD (Deutscher Wetterdienst – German Weather Service). We provide them for INSEL users in files named `dwdyyyy.dat` – `yyyy` is a place holder for the year 2011, for example. These lines show a part of the first records of file `dwd2011.dat`:

`dwd2011.dat`

```
Aachen          20   34   97  139  180  155
Augsburg        30   45  105  161  191  151
Berlin          18   41   92  132  185  181
Bonn            20   37   98  141  181  157
Braunschweig    19   34   90  140  174  177
Bremen          18   33   84  139  164  155
```

The file is well formatted in the above discussed sense, but – as a novelty – it does not only contain numerical data but also alphanumerical data, the name of the locations in this case. For such files, the Fortran star format can no longer be used.

Let us make a short excursion to some general Fortran format conventions.

3.1.1 Fortran format conventions

Edit descriptors

From the many so-called edit descriptors that exist in Fortran like `I`, `B`, `O`, `Z`, `F`, `E`, `EN`, `ES`, `D`, `G`, `A`, and `X` – to mention a few – INSEL makes use only of `F`, `E`, and `X`.

On the one hand this is a big advantage, because it is not necessary to read through pages and pages to understand all possible edit descriptors and their use. On the other hand this is a restriction, of course. But, you will see that almost all practical cases can be covered and in the (seldom) case that one of the other edit

descriptors is required, the advanced INSEL programmer can write and include his own code in INSEL to handle these cases.

Ergo, `F`, `E`, and `X`. What is their meaning? At first, `F` stands for floating point format and is used for real editing without exponents, `E` stands for exponential format, and `X` is used for positional editing.

Let us look at the definitions taken from the Microsoft Fortran documentation:

Syntax: `Fw.d` The `F` edit descriptor tells Fortran to treat a number as a simple decimal floating-point value. On output, the I/O list item associated with an `F` edit descriptor must be a single- or double-precision real or complex number, otherwise a run-time error occurs. On input, the number entered may have any real or complex form as its value is within the range of the associated variable. The field is `w` characters wide, with a fractional part which is `d` decimal digits wide.

Syntax: `Ew.d` An `E` edit descriptor means that there is an exponent in the syntax of the value. The I/O list item associated with the `E` edit descriptor for an output item must be a single- or double-precision real or complex number. A number input to a variable described with an `E` edit escriptor can have any real or complex form, as long as its value is within the range of the associated variable. The field is w characters wide. The input field for the `E` edit descriptor is identical to that described by an `F` edit decriptor with the same `w` and `d`.

Syntax: `nX` The `nX` edit descriptor advances the file position by n characters. If `n` is absent, the `X` edit descriptor defaults to `1X`.

So far the Microsoft text.

We are currently interested in the input cases, i. e. reading of files. Starting with `Fw.d` we have learnt that we can read floating-point numbers with a width of `w` bytes (including the decimal point by the way) and `d` bytes following the decimal point. Let's use the file `meteo82.dat` as an example. Recall the first four records of the file:

```
|...+...1...+...2...+...3...+...4...+...5...+...6...+...7.]
 1 182 1   0.   0.   0.   0.   0.  4.5-40.  -40.0-40.0 265.  4.4
 1 182 2   0.   0.   0.   0.   0.  3.8-40.  -40.0-40.0 255.  4.3
 1 182 3   0.   0.   0.   0.   0.  3.4-40.  -40.0-40.0 255.  2.8
 1 182 4   0.   0.   0.   0.   0.  3.1-40.  -40.0-40.0 225.  2.3
```

Here, in addition to the data a ruler is shown, which makes counting a bit easier.

We want the first eight bytes `_1_182_1` (the underscore representing the invisible space again) of the first record to be read in as 1 for the day, 1 for the month, 82 for the year (1982), and 1 for the hour.

Floating-point numbers in INSEL are described by the `F` edit descriptor. Hence, in order to read the first two bytes as 1 we need an `Fw.d` format where the number of bytes `w` is equal to two and since we have no decimals after the non-existing

decimal point `d` must be equal to zero. So `F2.0` must be used – four times to read day, month, year, and hour, so that we can write `F2.0,F2.0,F2.0,F2.0`.

The edit descriptor `F` is a so-called repeatable edit descriptor, which means that if – like in our case – a specific format appears identically several times, a repeat factor can preceed the edit descriptor. This means writing `F2.0,F2.0,F2.0,F2.0` is equivalent to writing `4F2.0`.

The next column contains `___0`. So ..., five bytes, no decimal fraction, `F5.0`. The next four values look alike, hence in total we have `5F5.0` – do you agree?

Then comes `__4.5-40.` – "What meaning of this?" as Peter Sellers said in the funny movie Murder by Death. In this special case the convention used in file `meteo82.dat` is, that whenever there is a `-40` in the file it means "lack of data." So we can guess that the `__4.5` is a datum and the concatenated "`-40.` is a datum which indicates "lack of data." This leads to an `F5.1` followed by an `F4.0` here.

The next two data seem to be missing too, so we interpret the bytes `__-40.0` as F7.1 and `-40.0` as F5.1. At the end of the record we see `_265.` followed by `__4.4` and have now understood that the edit descriptors are `F5.0` and `F5.1`.

To sum it up, the sequence of edit descriptors for the formatted file `meteo82.dat` is `4F2.0,5F5.0,F5.1,F4.0,F7.1,F5.1,F5.0,F5.1`, all separated by a comma.

Cross check

As we have already seen `meteo82.dat` has a record length of 64 bytes. We can cross-check this value with the sequence of edit descriptors: 4 times 2 bytes gives 8, plus 5 times 5 bytes gives 33, plus 5 bytes, gives 38, plus ... gives 64. Okay?

Parentheses

In Fortran, format strings have to be parenthesised, i. e. the final Fortran format string to read all data in a record of file `meteo82.dat` is

```
(4F2.0,5F5.0,F5.1,F4.0,F7.1,F5.1,F5.0,F5.1)
```

Big numbers

When numbers get too big, the representation like 123 000 000 000 is no longer practical. In Fortran we can use the exponential representation for such cases. The number then would be shown as `0.1230E+12`, which reads as 0.1230×10^{12}. In order to describe this with the `E` edit descriptor we first have to count the number of bytes of `0.1230E+12`, which is equal to 10, including the decimal point, the `E` and the plus sign. The number of decimals is 4 bytes following the decimal point, so that the format is `E10.4`.

The `nX` edit descriptor is used to ignore `n` bytes during the reading of a record. In case of the above mentioned file `dwd2011.dat` we are not interested in a numerical evaluation of the bytes which contain the name of a location.

In printed form it is difficult to exactly count the number of bytes used for the location name due to the many spaces. In `dwdyyyy.dat` in total 20 bytes (including all spaces) are used for the location names. This means, when we want

to read `dwdyyyy.dat` files we would like to always skip the first 20 bytes of each record. Hence, `n` is equal to twenty and we write `20X`.

In the files which contain data of complete years `12F6.0` values are saved for 12 monthly means of the global radiation, followed by one `F7.0` value for the sum of the 12 months radiation values and an `F5.0` value which contains the percentaged deviation of the annual value from the long-term mean value.

When we sum up the number of bytes in the edit descriptors we find $20 + 72 + 7 + 5 = 104$ bytes.

This ends our short excursion to the Fortran format conventions which are important for INSEL programmers.

You should now apply your newly gained knowledge for an analysis of the two example data files `meteo82.dat` and `dwd2010.dat`. Before you can really start, some more information about the file contents is necessary.

Contents of `meteo82.dat`

Let us start with the file `meteo82.dat`. It contains the following data:

1 Day
2 Month
3 Year
4 Hour (1-24)
5 Global irradiance horizontal / $\mathrm{W\,m^{-2}}$
6 Diffuse irradiance horizontal / $\mathrm{W\,m^{-2}}$
7 Global irradiance tilt angle 70 degrees, facing South / $\mathrm{W\,m^{-2}}$
8 Global irradiance tilt angle 70 degrees, facing South-East / $\mathrm{W\,m^{-2}}$
9 Global irradiance tilt angle 70 degrees, facing South-West / $\mathrm{W\,m^{-2}}$
10 Ambient temperature / °C
11 Relative humidity / %
12 Air pressure / hPa
13 Precipitation / mm
14 Wind direction / degrees from north via east, south, etc.
15 Wind speed / $\mathrm{m\,s^{-1}}$

Since three of the data columns are completely lacking (all `-40`s) we can skip these columns.

Exercise 3.2 Can you construct the Fortran format when these bytes are skipped by `nX`?

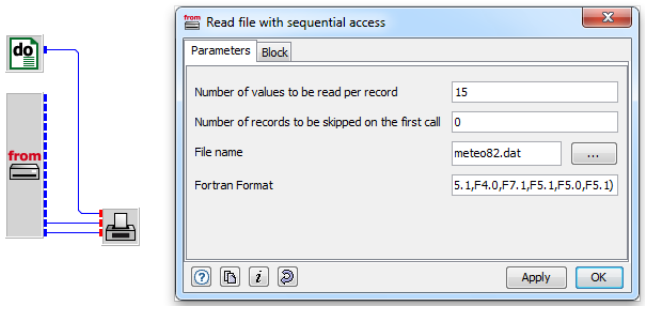
Example `meteo82.dat`

Now you have all the necessary information and can plot and analyse the time series saved in `meteo82.dat`.

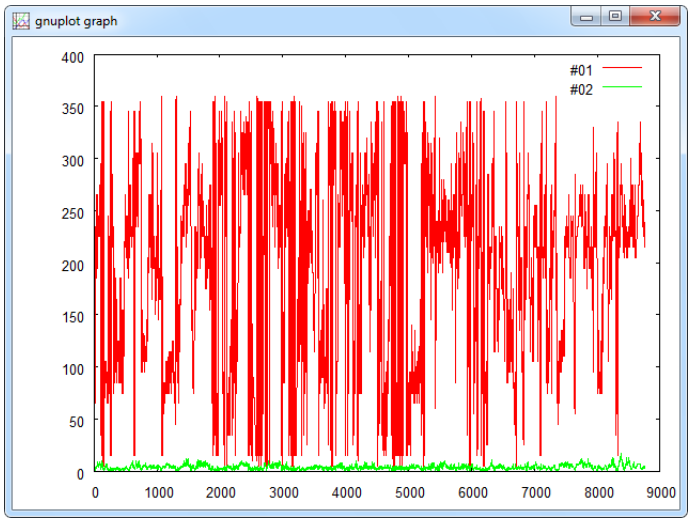
As usual, we provide example solutions. But you gain most from this Tutorial when you try and solve the problems on your own before you compare your solutions with ours.

Solution

We have constructed a READ block entity which fits the needs of `meteo82.dat` and plot the time series of wind direction and wind speed in one diagram.



Here comes the graph:



Rather than simply plotting the wind data it is much more interesting to analyse the radiation data on the different orientations, but we leave this task for you.

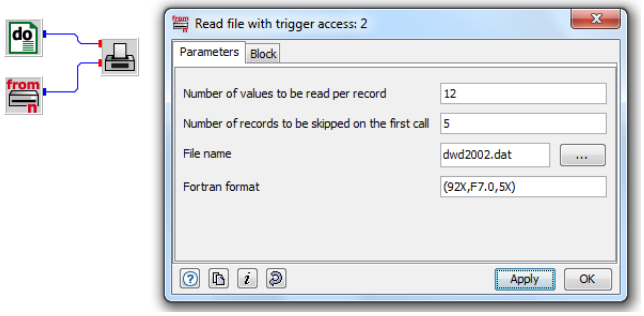
Example `dwd2002.dat`

As another exercise use the file `dwd2002.dat` to compare the radiation distribution over Germany at the different locations available in `dwd2002.dat`. When you open the file – which resides in the `data\weather` directory – with a text editor you find out that `dwd2002.dat` contains data for 62 locations.

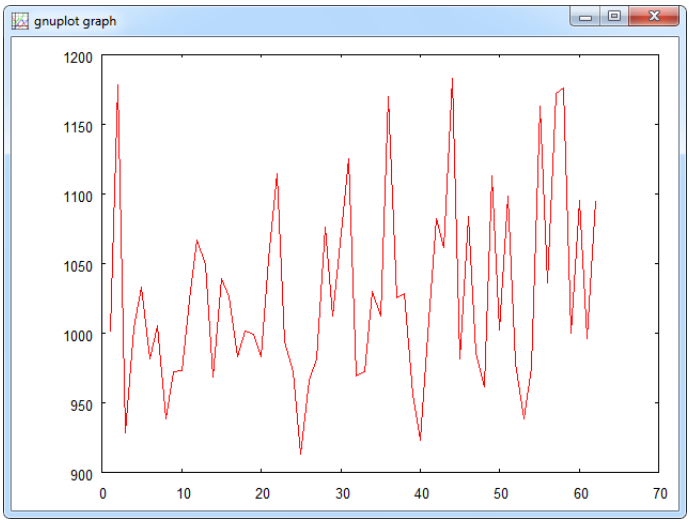
In the Fortran format conventions section we have seen that each record of the file contains a location name (20 bytes) followed by 12 monthly radiation values (`12F6.0`, in the unit kWh/m^2), the annual sum (`F7.0`, also in the unit kWh/m^2), and the deviation to long-term measurements (`F5.0`, in percent).

Exercise 3.3 Plot the annual radiation sums for all 62 locations.

Solution Since we are only interested in the annual radiation sums, we can skip to read all other information in the file. Hence, we can simply use the format string (`92X,F7.0,5X`).



This is the output:



The locations in the file are ordered alphabetically, so the sequence in the plot does not make much sense. But we can observe, that the level of global radiation on a horizontal surface in Germany is in the range between 900 (the pitiable people in North-Germany) and 1200 kWh/m^2 in the South.

3.1.2 The READN block

When you look at the structure of the data in file `dwd2002.dat` and our only approach to file reading – sequential access – so far, you may find yourself

confronted with the question “How can I use the READ block to access the monthly mean values saved in the file one after the other, i. e. plot the time series of monthly radiation data for a certain location?”

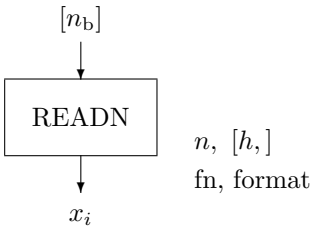
Exercise 3.4 Think about a solution for a few moments, please!

The problem is: One READ block execution reads one complete data record at a time. If we want to access all twelve data for a location, we can use a READ block with a Fortran format which reads all 12 values `(20X,12F6.0,12X)` with the consequence that we have all twelve values on output of the READ block at the same time. When this is what you want – no problem.

But if you want to plot the data for example, the PLOT block would require twelve inputs (all connected to the outputs of the READ block) plus one for the x-coordinate. What x-coordinate and what kind of a plot would this be?

So, what we really want is not to read a complete record in one step but only one datum like the radiation for January, plot it, read the next datum like the radiation for February and plot it and so on. It is obvious that the READ block as it is designed cannot solve this problem.

In such cases the READN block is one way out.[1] The READN block – like the READ block – reads one complete record as specified in the Fortran format parameter but outputs only one value at a time – it triggers the output values. The READN block expects a parameter which says how many calls of the block it has to wait until a next physical read access is to be performed on the file. The layout of the block is shown in the following graph:



The layout of the block is exactly identical to the READ block but – no matter what the format parameter is – it outputs only one value per call.

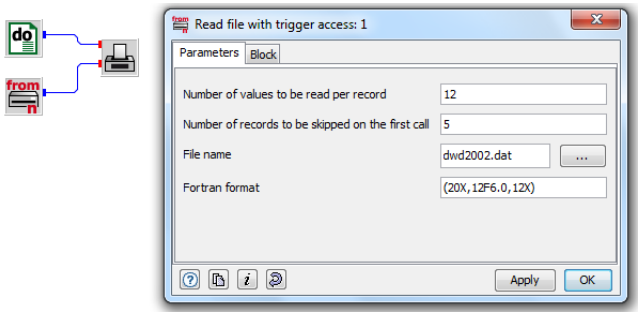
Let us use the block for a plot of monthly mean values of global radiation for the location of Stuttgart on the basis of file `dwd2002.dat`. By opening the file with a text editor we find out that Stuttgart is record number 55. So, the READN block

[1] Another possibility to solve this problem could be to use the multiplexer block MPLEX – see Block Reference Manual for details. But when the number of data per record gets large, the method is rather inconvenient since all outputs have to be connected.

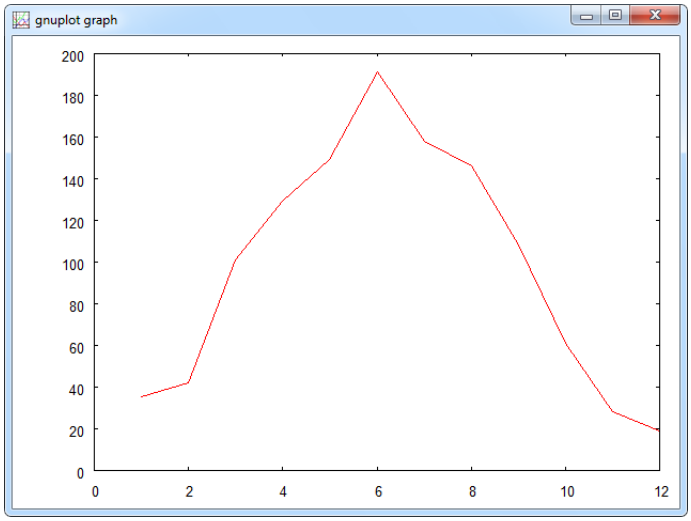
should skip the first 54 records. The Fortran format for reading is `(20X,12F6.0,12X)`. The file name is clear, so we can write the application.

Exercise 3.5 Do it, now!

Solution The solution is



and the time series plot is



3.1.3 The READD block

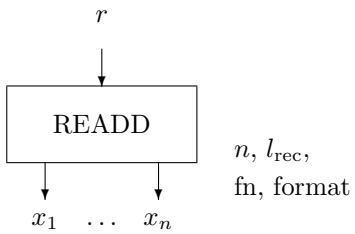
A third block, named READD for reading data files is available in INSEL which allows *direct access* to data files. In contrast to sequential access, where we can start reading a file at a given record and then the read operation returns the files records sequentially, i. e. one after the other, in direct access mode we can read records in any arbitrary order. In order to do that we have to deliver the record

number of the record we want to access. How will the operation system perform the reading when we, for example, want to read the tenth record of a file?

Well, as we have heard at the very beginning of this Module a file is nothing but a stream of bytes. Logically we have identified records as logical lines in the file. For direct access the operating system takes the record length – let us assume a record length of 80 bytes – of the file, multiplies it with the number of records to skip – nine in our example – and then knows the displacement from the start of the file to the position where we want to start reading. In our example reading the tenth record means that $80 \times 9 = 720$ bytes must be skipped (plus the line separator bytes).

This method works only if all records have exactly the same length – otherwise the calculation would lead to some arbitrary byte in the file and makes an interpretation of the data stream impossible.

This is the design of the READD block:



The parameters should be self explaining by now.

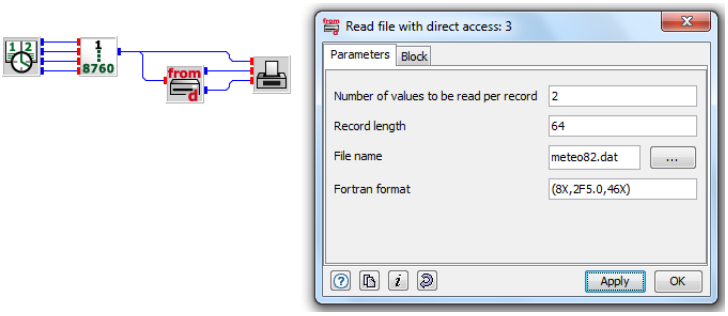
Exercise 3.6 As an example for direct access read and plot the time series of global and diffuse radiation on a horizontal surface for the month July as stored in file `meteo82.dat`.

Solution 1 Yes, it is not really necessary to use the READD block. We can also solve the problem by opening the file `meteo82.dat`, find the record number where the first of July starts (record number 4345 since 1982 was not a leap year), quickly calculate 31 days times 24 hours gives 744 records, use a DO block which counts from one to 744, use the READ block and set the skip parameter to 4344, plot the two curves and ready.

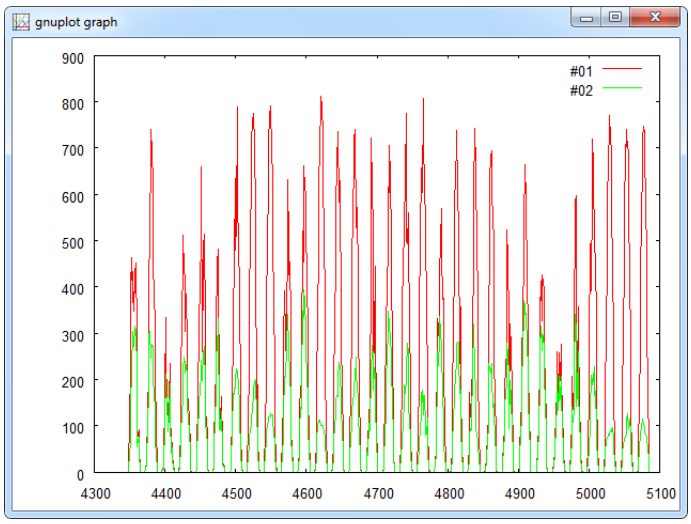
But how would you use the READD block? Continue reading when you know the solution.

Solution 2 Use a CLOCK block and set the parameters from 01.07. any year 00:00:00 to 31.07. same year 24:00:00 in steps of one hour. Remember that 1982 was not a leap year. The hour of year block HOY can be used to convert the Gregorian calendar date into the hour of the year – this value is exactly the record number that we need, connect it to the READD block's input, and plot the curves. Now it

is very easy to quickly change the parameters so that you can access the December data, for instance.



This is the plot of the radiation data:



3.1.4 File name qualifiers

File names in INSEL can be varied during a simulation run, when file name qualifiers (FNQs) are used in file name parameter strings. If the first character of a file name (without path) is a `#` character the file name is considered variable and is parsed by INSEL. Up to four qualifiers can be used as place holders for digits. The qualifiers have the format `%nX`, where `n` is a positive digit in the range of 0 to 9 and `X` is a character out of `YMDh` – reminding on their most frequent use of date and time information in data file names.

The numerical values for the `n` digits must be provided as block inputs. `Y` is used with the first input, `M` with the second, and so forth.

Example

The file name `#myName%4Y.dat` results in `myName2011.dat`, assuming that the first input of the block containing the file name has a value of 2011.

Adding a second input which contains values for the months of a simulation, a qualified file name would be `#myName%4Yplus%2M.dat`. For a simulation running over a complete year the generated file names would be `myName2011plus01.dat`, `myName2011plus02.dat`, ... `myName2011plus11.dat`, `myName2011plus12.dat`.

Suppress leading zeros

The months from January to September are interpreted with leading zeros when the qualifier `#myName%4Yplus%2M.dat` is used. Leading zeros are suppressed when the qualifier `#myName%4Yplus%0M.dat` is used.

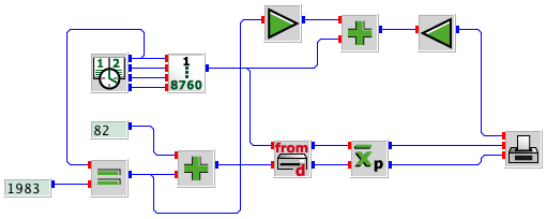
When a file name contains path information, the `#` sign is interpreted as qualifier only when it is the first character of the file name – not the path name. For example, running a CLOCK block over two years in steps of one month and connecting the first three outputs of the CLOCK block as inputs to an IO-block with the qualified file name `C:\Path\#prefix_%4Y_%0M_%2d_appendix.dat` results in something similar to

```
C:\Path\prefix_2000_1_01_appendix.dat
C:\Path\prefix_2000_2_01_appendix.dat
...
C:\Path\prefix_2000_9_01_appendix.dat
C:\Path\prefix_2000_10_01_appendix.dat
C:\Path\prefix_2000_11_01_appendix.dat
C:\Path\prefix_2000_12_01_appendix.dat
C:\Path\prefix_2001_1_01_appendix.dat
...
C:\Path\prefix_2001_12_01_appendix.dat
```

The file name parameter `#C:\Path\prefix_%4Y_%0M_%2d_appendix.dat` would not achieve the desired result but defines a constant file name `#C:\Path\prefix_%4Y_%0M_%2d_appendix.dat`.

readd_8283.vseit

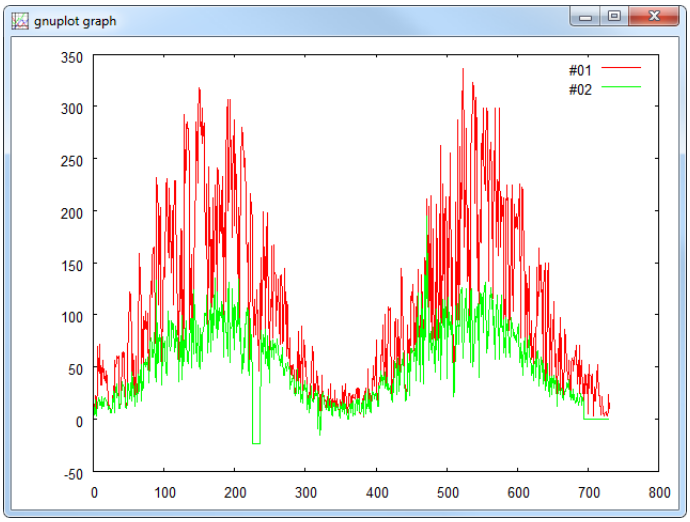
Assume, we want to read the data files `meteo82.dat` and `meteo83.dat` in a single simulation run. A file name qualifier which could be used for this purpose is `#meteo%2Y.dat`. As input to a READ or a READD block the two values 82 and 83 are required successively. One way to solve this problem is shown in the following model:



The CLOCK block varies date and time from 01.01.1982 00:00 to 31.12.1983 24:00 in steps of one hour. A CONST block with parameter 82 is used. As long as the CLOCK block runs through the year 1982 the logical result of the EQ block is false, i. e. 0. When the CLOCK block switches to the year 1983 the EQ block outputs the value 1. This value added to the CONST 82 gives the desired value 83.

Another solution could use a DO block with its output connected to the CLOCK block and the READ or READD block. The parameters of the DO block could be 82, 83, and 1. In this case the CLOCK block should run through any non-leap year, 2011, for instance.

We have used a AVEP block to calculate the daily means of the global and diffuse radiation data. Please observe, how we have used the EQ block to create an x-axis signal for two times 365 days. This is our result:



`readn_DWD.vseit`

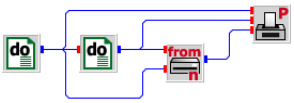
Another interesting application made possible though FNQs is the following: INSEL provides ground-measured DWD (German Weather Service) data for 62 German locations since the year 2000 in the `data\weather` directory. The files are organised as annual data files, named `dwd2000.dat`, `dwd2001.dat`, and so forth.

We have already used a READN block to read the Stuttgart data for one year earlier in this Module.

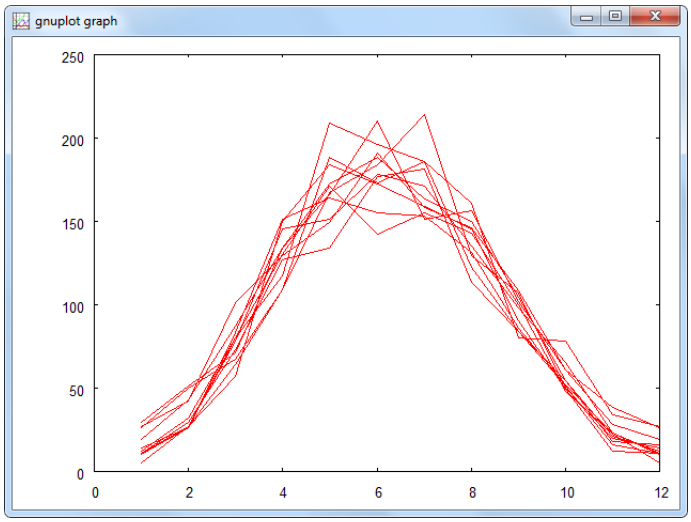
Exercise 3.7 Plot the monthly mean global radiation data for Stuttgart over the period from 2000 to 2010 with a single INSEL model.

Solution Using file name qualifiers makes this task easy as pie:

`readn_fnqs.vseit`



We need two DO blocks, one for the variation of the year from 2000 to 2010 in steps of 1 and one for the variation of the month from 1 to 12 in steps of 1. The READN block expects FNQs starting with input number 2. The qualified file name is `#dwd%4Y.dat` (plus path information) and, as usual, the PLOT block shows the result.



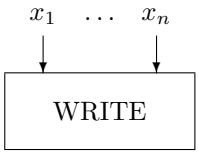
We are going to have one more example for the use of file name qualifiers at the end of the next section on writing data to files.

3.2 Writing data to files

With the WRITE block data can be written from an INSEL application to the computers hard disk, a USB flash memory disk, or any other media where you have write access.

There are basically two modes: You can

- Create a new file or overwrite an existing file
- Append data to a new or an already existing file



mode,
fn, format

If the *Overwrite* mode is chosen, then after opening the file specified in the file name parameter the write pointer points to the beginning of the file and overwrites its content.

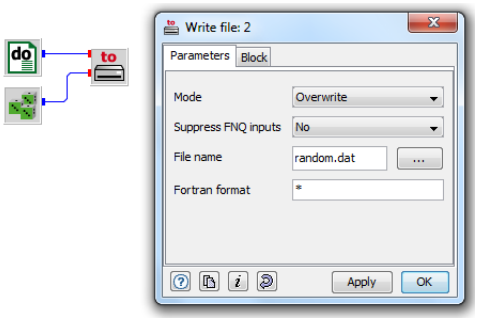
WARNING Be extremely careful with the option to overwrite files. When you choose this option, the block will definitively overwrite the file and the old file – if there was one – is definitively lost forever. There is no Undo. When it was a file which contained something very valuable before, now it is gone. So be careful. We deny any responsibility. So again, be careful!

If the *Append* option is chosen, the pointer is positioned at the end of the existing file before the first write operation.

If the *Generate error message* option is selected, the WRITE block will generate an error message if the file already exists and INSEL does not execute the simulation model.

As file name you can choose any name which is valid under the operating system version you are using. Of course, the file name can include path information. For example `c:\anyPath\myFile.dat` is a valid file name (assuming that the path `c:\anyPath` exists). If no path information is included in the file name INSEL writes the file into the current directory (usually the INSEL working directory `insel.work`).

`random.vseit` A simple application of the WRITE block is reformating of data files. `random.vseit` generates 100 Gauss distributed random numbers and writes the data to a file called `random.dat`.



`random.dat` The first ten records of the file look like this:

```
 1.0000000      1.6216065
 2.0000000     -0.39489648
 3.0000000     -0.33821103
 4.0000000      0.53852010
 5.0000000     -0.42136794
 6.0000000     -0.23904423
 7.0000000      1.2835248
 8.0000000      0.38314018
 9.0000000     -1.4969951
 10.000000     -1.0831203
```

Exercise 3.8 Please remember, that files like `random.dat` cannot be used for direct access with the READD block. Hence, write a small INSEL program which reformats the file to a file with format `(F12.7,1X,F5.1)`.

Solution Yes, you are right, we could have used a format string like `(F5.1,1X,F12.7)` in the `random.vseit` example already. But then you would have missed this one.

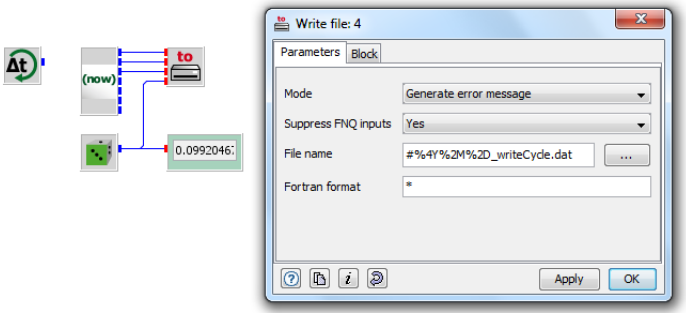
Three obvious blocks, DO, READ, WRITE and done. Please observe again, that the READ block does not need a connection to the DO block – because it is a Standard block and Standard blocks are always successors of the main Timer when no input is connected ...

... and everything is well formatted now:

```
 1.0    1.6216065
 2.0   -0.3948965
 3.0   -0.3382110
 4.0    0.5385201
 5.0   -0.4213679
 6.0   -0.2390442
 7.0    1.2835248
 8.0    0.3831402
 9.0   -1.4969951
10.0   -1.0831203
```

3.2.1 Monitoring and simulation

INSEL can be used to monitor real-life systems, the most prominent ones being grid-connected PV generators like the Trade Fair Munich Generator, for example. Have a look at the following block diagram:



A CYCLE block is used to continuously run this INSEL model. The execution speed of the model – measured in real-time seconds – can be specified by the parameter of the CYCLE block. For every time step the NOW block returns the current date and time. Year, month, and day are used as file name qualifiers to write files with daily data of some monitored and/or simulated data – in this case only dummy random numbers.

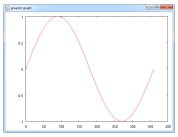
Blocks from the palette's *GUI objects* category can be used to display any data over the course of the simulation run. Perspectives can be used to create complete GUIs (graphical user interfaces) for the application. We will return to this topic in more detail in Module 9.

3.3 Plotting data

Another INSEL block of the data writing category – which you have used several times already – is the PLOT block. From the application point of view, x-y coordinates are connected to the block and the block generates a graphical output. Let us understand more deeply how the block operates.

At first, recall that every INSEL block works absolutely local. By this we mean that the block can perform only such operations, which depend on nothing but the actual values of the inputs, the parameters, and – in some block's cases – on the history.

Let us use the simple case of plotting the sine function as an example.



From the point of view of the PLOT block, on the first call the block receives an x-input zero and a y-input equal to zero, too. We know, that on the first call the first zero is an angle $\alpha = 0°$, and the second input is the $\sin(\alpha)$ – the PLOT block doesn't know anything about this.

The only point of interest from the PLOT block's point of view is that there is a data point (0,0) which I (the PLOT block) have to show as one of probably many data points in a graphical plot. What does the block do? It saves the data point in a data file and expects further actions – either it receives more data points, in which case the block will append them to the data file, or the instruction to display the complete graph.

An online plotter would show data points and their linear (or other) interpolation immediately. But the PLOT block is not an online plotter.

`insel.gpl` The data file always has the same name `insel.gpl` and is saved in the hidden-application-data directory. The instruction to display the graph comes from the inselEngine after the simulation run has been completed. The first ten records of file `insel.gpl` in this example are

```
0.0000000E+00  0.0000000E+00
0.1000000E+01  0.1745241E-01
0.2000000E+01  0.3489950E-01
0.3000000E+01  0.5233596E-01
0.4000000E+01  0.6975647E-01
0.5000000E+01  0.8715574E-01
0.6000000E+01  0.1045285E+00
0.7000000E+01  0.1218693E+00
0.8000000E+01  0.1391731E+00
0.9000000E+01  0.1564345E+00
```

INSEL uses the maximum number of significant digits for Fortran four-byte REAL numbers (which is seven).

If you want to make further use of the file – maybe you like to post-process it with a presentation software of your choice – you can copy or rename the file to your needs. The only thing you need to document is what the meaning of the records, i. e. the x-coordinate and the y-coordinate(s) is.

The PLOT block, by default, generates a second data file `insel.gnu`, which contains some basic commands which enables Gnuplot to display the graph. In the case of our sine application the file looks like this.

```
set autoscale xy
set style data lines
set nolabel
plot "C:/Users/name/AppData/Roaming/doppelintegral/INSEL/tmp/insel.gpl" using 1:2 title ""
pause mouse
```

The first command `set autoscale xy` leaves it up the Gnuplot to find reasonable settings for range and increment of the x- and the y-axis. The next command `set data style lines` requests from Gnuplot to draw a connection line, i. e. a linear interpolation between the data points. `set nolabel` leaves the plot clean of any label names, and `plot "/path/insel.gpl" using 1:2 title ""` lets

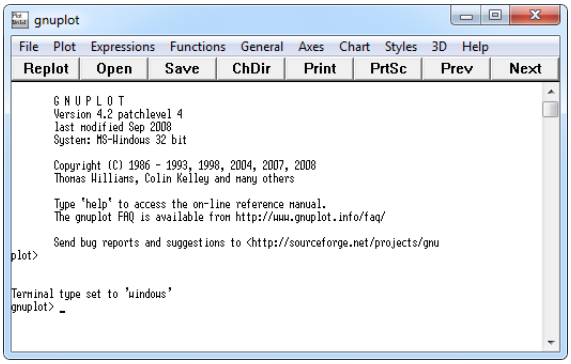
Gnuplot show the data plot based on the data in file `/path/insel.gpl` using the values in the first column as x-coordinate and the second as y-coordinate.

The value of `/path/` depends on the user's name and settings and is located on the lokal hard disk, by default. `title ""` suppresses any default legend of the plot, and finally `pause mouse` makes Gnuplot wait for a mouse click to close the window.

Such a default Gnuplot command file is always generated by the PLOT block when `insel.gnu` is given as PLOT block parameter. You can specify own Gnuplot command files for the PLOT block but this requires some knowledge about Gnuplot programming.

Interactive Gnuplot

One last hint to the PLOT block: You can start Gnuplot from the *Tools* menu or by a click on the icon in the tool bar. The Gnuplot window appears.



In the work area you see a prompt `gnuplot>` and a blinking cursor. Here you can enter Gnuplot commands. If, for example, you want to plot the last INSEL plot you made – this is file `insel.gpl` in the hidden application data directory – you can proceed as follows:

Type `pwd` at the gnuplot prompt (`pwd` is short for print working directory) and Gnuplot shows the actual directory name. You can use the change directory command `cd 'dirName`, where `dirName` stands for the target directory. Please notice the single quote in front of the directory name.

You can either specify a complete path – like `c:\myDirectory`, for example – or you can use relative directory names just like in a DOS box. Remember that for changing to a directory one level higher than the current directory the command is `ch ..` under DOS and `ch '..` under Gnuplot.

When the hidden application directory is the current directory you can use the command `load 'insel.gnu` (observe the quote again) and Gnuplot will display

the last graph – just like INSEL when it performs these default steps for you automatically.

The difference is now, that after closing the graph window you can interactively use the menus and buttons of Gnuplot to make modifications to the plot. For example, if you want to add a label to the x-axis use the *Axis –X Label* menu item and enter the text for the label, skip the offset by simply clicking *OK* and then click the *Replot* button in Gnuplot's tool bar.

Many things should be self-explaining in the Gnuplot window. When you are interested in a deeper understanding of Gnuplot, the complete Gnuplot manual is available under the *Help* menu of Gnuplot. It is definitively worth to have a look, because Gnuplot is really powerful.

Summary

- Data files are streams of bytes which must be interpreted by encodings, like ASCII, for example.
- INSEL uses Fortran format conventions. You should know now how to work with the edit descriptors `F`, `E`, and `X`.
- Sequential access to data files is possible with the READ block. Optionally, we can start reading of formatted or unformatted (star format) files either from the first record or start reading the file at a well-defined offset.
- Direct and trigger access to read files is possible with the blocks READD and READN.
- The WRITE block can be used to write data to arbitrary data files.
- The PLOT block turned out as a block which writes two data files, i. e. `insel.gpl` and `insel.gnu`.