Module 13 :: INSEL without GUI

13.1 Running `.insel` files

We hope, that it has been wonderful to see how the completely graphical approach to programming with graphical programming elements like INSEL blocks and their interconnections works. When you "look behind the stages" it doesn't require much information to interpret a user-written block diagram simulation application. By generating a graphical block diagram, let us ask and answer the question, what actually happens and which kind of information is provided to the simulation environment.

In block diagram based simulation environments like INSEL the information consists basically only of two information types – the block diagram structure and the used parameters.

As an alternative to graphical programming in VSEit, you can also write INSEL models in a text editor. In order to write an INSEL simulation program in its ASCII representation you must use a text editor like Windows Notepad or Kedit, for example. You then need to know the syntax of the INSEL programming language, which is very simple and consists of only few keywords.

Hello, world!

In a book of Kerninghan and Ritchie on programming in C the now famous Hello, world example was given, a C program which displays the string "Hello, world!" on the computer's display. If you want to solve this task with INSEL, you need a block which is able to display information on the screen. One of these blocks is the SCREEN block. It has an optional parameter for the format of the displayed information.

Please notice that two different types of information have to be provided for the SCREEN block and – more general – each INSEL application:

Structure and parameters

First, the model structure fixes which blocks are used in a certain application and how they are interconnected. Second, the model parameters fix what the current values of the block parameters are.

`hello.insel`

In this example, model structure and parameters are extremely simple, because all we need is one single block. These two statements do the job:

```
s 1 screen
p 1 '(''Hello, world!'')'
```

S or s statement

The first record starts with an `s` which is an INSEL keyword (short for structure). It is followed by an arbitrary block number (which has to be unique for every block that is used in a given INSEL program) and the block's name, SCREEN in this case. Please observe that the entries are separated by a delimiter, one blank (space character) in this case. Usually, a list of block inputs follows after the block name, but in this example no inputs need to be connected.

P or p statement

The second record provides the necessary parameter information starting with the keyword `p` (short for parameter). In order to enable INSEL to uniquely identify the given values with a certain block, the above mentioned user-defined block number follows the `p`-keyword.

The parameter list comes next, in this case the `'(''Hello, world!'')'` string. Because the format parameter is a string, it has to be embedded in quotes. Concerning the string value pay attention to use two single quotes `''` and not one combined `"`. The parentheses in the string follow the Fortran format conventions.

Now you are ready to save the information under a file name like `hello.insel`, for example. It is necessary to use the `.insel` extension for INSEL source code files. The next step is to tell INSEL that you like to execute the `hello.insel` application.

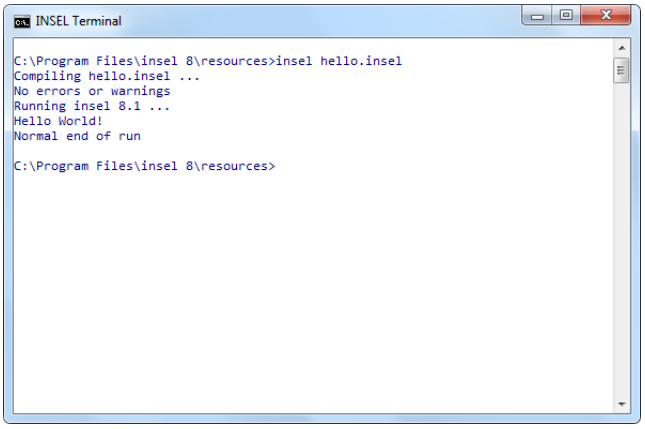
Execute `hello.insel`

There are two options how the model can be executed: either from the VSEit interface via File – Open .insel File... and the Run button, or from a DOS box via `insel hello.insel`. The second option requires that `insel.exe` is in the current `%PATH%` and that `hello.insel` is available in the current directory.

Exercise 13.1

Please open an INSEL Terminal from the tool bar and try it.

Solution



Photovoltaics

As a second more applied example with inputs and outputs we now write a `.insel` model which calculates the power output of a photovoltaic module as a function of the voltage. We start with the timer block DO, which outputs the voltage from 0 to 40 V in steps of 0.01 V to the PVI block.

```
s 10 do
p 10  0       % Initial value
     40       % Final value
      0.01    % Increment
```

Comments

Please observe that comments can be used in `.insel` files: everything starting with a `%` symbol to the end of the record is gobbled by the INSEL compiler.

The PVI block uses the first output from the DO block as an input, i. e. the output from block number 10. This first output is written as `10.1`, the second output would be `10.2` etc. The PVI block also needs the irradiance and module temperature as inputs. To keep the model simple, irradiance and temperature are set constant. Hence, we define two different constant blocks.

```
s 11 const
p 11 1000.0   % Irradiance in W/m2
s 12 const
p 12   25.0   % Module temperature in degrees celsius

s 20 pvi 10.1 11.1 12.1
```

The next block is the multiplication block MUL, where the voltage (output from the DO block number 10) and the current (output from the PVI block number 20) are multiplied.

```
s 30 mul 10.1 20.1
```

Finally, the result of the MUL block – the DC power of the PV module – is plotted against the voltage.

```
s 40 plot 10.1 30.1
```

What is still missing, are the parameters for the PVI block. INSEL provides a data base for several thousand modules that are or have been on the world market. In the `data` directory you find a file named `pvModules.dat` which contains a list of modules which are in the data base. The first few records of this file look similar to:

`pvModules.dat`

```
PRODUCER 2009              PVTYPE                      pvxxxxxx      Pnenn
-------------------------------------------------------------------------
3S Swiss Solar Systems AG  Fassadenmodul                 001531      178.0
3S Swiss Solar Systems AG  MegaSlate-Indachmodul mono    008917      148.0
3S Swiss Solar Systems AG  MegaSlate-Indachmodul poly    001189      136.0
Aavid Thermalloy           ASMC-150M                     009458      150.0
Aavid Thermalloy           ASMC-175M                     009459      175.0
Aavid Thermalloy           ASMC-180M                     009460      180.0
Aavid Thermalloy           ASMC-190M                     009461      190.0
Advent Solar, Inc.         Advent 210                    005405      210.0
Advent Solar, Inc.         Advent 215                    005406      215.0
```

The records should be self explaining, except the `pvxxxxxx` column. The parameters for the modules (or more general, all parameter sets in the INSEL data base) are saved in files with the extension `.bp` which is short for block parameters. The file name in case of the PV parameters is `pvxxxxxx` with the

place holder xxxxxx. Column pvxxxxxx provides this placeholder. For example, if you want to simulate the Advent 210 module of Advent Solar, Inc., the parameters are provided in file pv005405.bp in the data\bp directory of your INSEL installation. This is the content of file pv005405.bp:

pv005405.bp

```
                % File name      pv005405.bp
                % Photon ID      005623
                % Module         Advent 210
                % Manufacturer   Advent Solar, Inc.
                % Cell type      poly

                % Mode must be set externally
  60            % Number of cells in series N_s per module
   1            % Number of cells in parallel N_p per module
   1            % Number of modules in series M_s
   1            % Number of modules in parallel M_p
   0.0275       % Cell area A_c (m^2)
   1.663        % Module area A_m (m^2)

                % Electrical parameters
   1.12         % Band gap (eV)
   0.2542       % Short-circuit current parameter C_0 (V^-1)
   0.153E-03    % Isc temperature coefficient  C_1 (V^-1 K-^1)
   0.169663E+05 % Shockley saturation parameter C_01 (A m-2 K^-3)
   0            % Recombination saturation parameter C_02 (A m-2 K^-5/2)
   0.00012389   % Series resistance r_s (Ohm m^2)
   0.03129369   % Parallel resistance r_p (Ohm m^2)
   1.0165366    % Shockley diode ideality factor alpha
   2            % Recombination diode quality beta
   0            % Bishop parameter-1
   0            % Bishop parameter-2
   0            % Bishop parameter-3
   3.0          % Module tolerance plus
  -3.0          % Module tolerance minus

                % Thermal parameters
   1.680        % Characteristic module length l_m (m)
  22.700        % Module mass m_m (kg)
   0.70         % Default absorption coefficient a
   0.85         % Default emission factor epsilon
 900.0          % Default specific heat of a module C_mod (J kg^-1 K^-1)
  47.0          % Nominal operating cell temperature NOCT (degrees C)
  25            % Intial value of cell temperature (degrees C)

                % Numerical parameters (optional)
   1E-5         % Error tolerance of voltage of single cell (V)
 100            % Maximal number of iterations to solve I/V-equation
```

When you look at the file and into the documentation of the PVI block, you see that the temperature mode is not part of the .bp file but must be set as an extra parameter.

I or i statement

Rather than copying the whole file into your `.insel` file the include statement can be used. Its syntax is simply

```
i 'file name'
```

When the INSEL compiler finds this statement in a `.insel` file it replaces the statement with a verbatim copy of the file content.

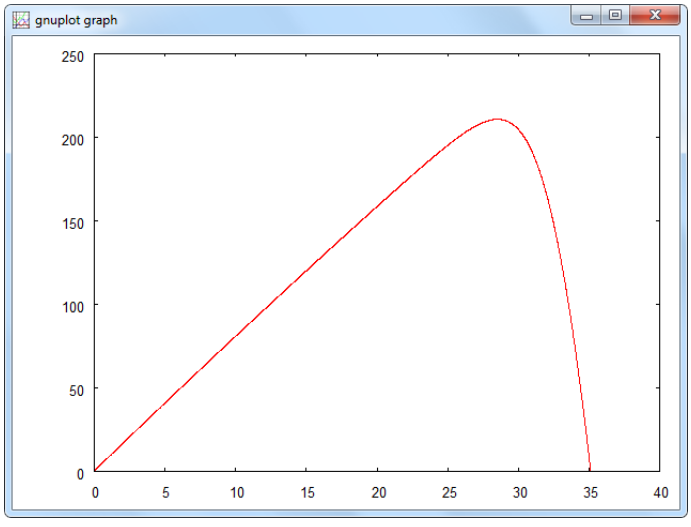
In conclusion, the complete program for calculating the DC power of a PV module as a function of voltage looks like this:

```
s 10 do
p 10  0       % Initial value
      40      % Final value
       0.01   % Increment
s 11 const
p 11 1000.0   % Irradiance in W/m2
s 12 const
p 12 25.0     % Module temperature in degree celsius

s 20 pvi 10.1 11.1 12.1
p 20 0        % Mode
     i 'c:\Program Files\insel~8\data\bp\pv005405.bp'
s 30 mul 10 20.1

s 40 plot 10 30
```

Please observe the syntax used by the PLOT block: when no output number is specified this defaults to output number one.

Exercise 13.2 Save the file under any name, for example `pv.insel`, and run it.

Solution



Arbitrary order of statements

One special feature of graphical programming languages like INSEL is that the order of statements in the source code is completely free. We could shuffle the model into any arbitrary order, like

```
s 40 plot 10 30
p 10  0       % Initial value
     25       % Final value
      0.01    % Increment
s 11 const
s 12 const

s 20 pvi 10.1 11.1 12.1
p 20 0        % Mode
     i 'c:\Program Files\insel~8\data\bp\pv1129.bp'
s 30 mul 10 20.1

s 10 do
p 11 1000.0   % Irradiance in W/m2
p 12 25.0     % Module temperature in degree celsius
```

In a conventional programming language it would be impossible to use variables like the PLOT block's inputs from blocks 10 and 30 before the values are defined. This makes it possible – and you probably used this feature without notice – to construct the VSEit applications in any order, delete VSEit objects, add new ones etc. By the way, the VSEit objects appear in the `.vseit` file in the order in which you placed them on the screen.

C or c statement

One last statement completes the set of only four statements in total – INSEL is perhaps the simplest programming language in the world with only four statements (the earlier versions had even only two: `s` and `p`). The `c` statement can be used to define constants by name and value. The syntax is

```
c name value
```

The variable `name` (no enclosing quotes) defines the name of the constant, `value` specifies its value, which can be either a valid numerical or string parameter with the usual INSEL conventions (i. e. strings enclosed by quotes, numerical values not enclosed by quotes). Variable names can be constructed from the characters `[A-Z][a-z][0-9]` but have to start with an alphabetic character.

In addition, the special character `#` is allowed in variable names. Its use should however be restricted to developers of "`.include/.insel`" applications. What is this?

13.2 `.include/.insel` applications

Program development (not only) in the field of renewable energy simulation can be classified into two different aspects: (i) the calculation model formulation and

(ii) program parts which provide convenient user interfaces. In many cases both program parts are combined into one software project.

The c- and i-statements enable a concept which we call ".include/.insel" applications. With this method INSEL provides a programming environment for the experienced INSEL user and C/C++ programmer (or any other high-end programming language software developer), where both calculation kernel and user interface can be written completely independent. The results are applications which look like common Windows applications, but which give the experienced user of such a program access to the modeling level, without having to recompile the user interface code.

To understand this in more detail let us use the previous example, in which the DC power of a PV module was calculated. In this example we had defined two constants for the irradiance and module temperature, blocks 11 and 20, respectively. The values 1000 W/m^2 and 25 °C were inspired by the standard test conditions for PV modules. Additional parameters were the voltage range and increment and a .bp file name for a specific PV module.

PV module flasher

We have modified the example so that it can be used as a "laboratory flasher" which can be used with a convenient user interface for any real (simulated) PV module.

flasher.include

The model is split up into two files. The first one contains only c-statements:

```
% Include file for the definition of the free parameters
c #PVincludeFile   'c:\Program Files\insel~8\data\bp\pv005405.bp'
c #InitialValue     0
c #FinalValue      40
c #Increment        0.01
```

flasher.insel

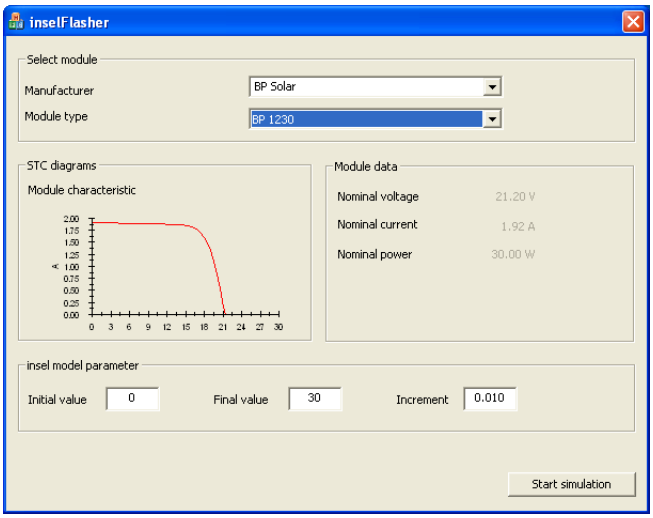
The second file contains the model which makes use of the variables defined in the include file.

```
% INSEL file to plot the STC I-V curve
i 'flasher.include'
s 10 do
p 10  #InitialValue
      #FinalValue
      #Increment
s 11 const
p 11 1000.0   % Irradiance in W/m2
s 12 const
p 12   25.0   % Module temperature in degree celsius

s 20 pvi 10.1 11.1 12.1
p 20 0        % Mode
     i #PVincludeFile

s 40 plot 10 20
```

In order to adapt the model to any given PV module – or in other words, to flash a certain module in the laboratory and create an I-V curve protocol – only the values in the include file must be changed. There is no need to touch the `.insel` file.

In consequence this means that the user interface needs to manipulate the `.include` file only. Once a user of such an interface has entered the parameters the interface tool can execute the complete model by calling the inselEngine. This is what was meant when we said that interface and calculation model are completely disjoint.

The details for programming of C++ interfaces is beyond the scope of this Tutorial. This screenshot shows a possible implementaion of the flasher example.



13.3 Parameter variations with Ruby scripts

Core component: File `lib/insel.rb`

```
require File.join(File.dirname(__FILE__), 'core_exts')
require 'fileutils'

module Insel
  # DEFINE THE RIGHT PATH FOR YOUR SYSTEM
  # It could be
  InselPath = "/Users/juergenschumacher/Documents/insel/VSEit/"
  #InselPath=File.join('/opt', 'insel8', 'resources')

  ## This class launches insel.exe with a temporary file containing insel_content
  ## It parses insel output, and returns the results as Float, Array or Array of Arrays
  ## insel_content needs to be defined separately, either with a Block or with a Template
  class Model
    # Parses insel output and return the results.
    # Results are supposed to be between "Running insel" and "Normal end of run"
    # A single value gets returned as Float
    # Multiple lines with single value get returned as an Array
    # One line with multiple values get returned as an Array
    # Multiple lines with multiple values get returned as an Array of Arrays
    def results
      rr = raw_results
      if rr =~/Running insel [\d\w \.]+ ...\s+([^\*]*)Normal end of run/m then
        $1.split(/\n/).map{|line|
          floats = line.split(/\s+/).reject{|f|f.empty?}.map{|r| r.to_f}
          floats.extract_if_singleton
        }.extract_if_singleton
      else
        raise "problem with INSEL #{rr}"
      end
    end

    # Returns the r-th output
    def [](r)
      @outputs_number=r+1
      results[r]
    end

    private

    # Writes a temporary .insel file with insel_content
    # Runs insel
    # Returns the raw output coming from insel
    # Deletes the temporary .insel file
    def raw_results
      temp_file = File.expand_path(File.join(File.dirname(__FILE__), 'test.insel'))
      FileUtils.cd(InselPath){
        File.open(temp_file, 'w+'){|f|
          f.write insel_content
        }
        @raw_results=%x(./insel #{temp_file})
```

```
      FileUtils.rm temp_file
    }
    @raw_results
  end
end


## This class is not exactly an insel Block, but an insel Model with one interesting block
## and the needed CONST blocks for input and SCREEN block for output.
## The main job of this class is to define insel_content. For example, for Block.sum(6,4) :
#     s 1 CONST
#     p 1
#             6
#     s 2 CONST
#     p 2
#             4
#     s 3 sum 1.1 2.1
#     s 4 SCREEN 3.1
#     p 4
#             '(6E15.7)'

class Block < Model
  attr_reader :name, :parameters, :inputs

  def initialize(name, parameters, *inputs)
    @name, @parameters, @inputs = name, parameters, inputs
    @outputs_number=1
  end

  # Method to access results from a block with :
  #   Block.launch(:do, [1,10,1]).inspect
  def self.launch(name, parameters, *inputs)
    new(name, parameters, *inputs).results
  end

  # Shortcut to access results from a block without parameters :
  #   Block.sum(6,4)
  def self.method_missing(sim, *inputs)
    launch(sim, [], *inputs)
  end

  private

  # Defines the model that will be fed to insel
  # Writes the needed CONST blocks, then the interesting block, then SCREEN block
  def insel_content
    tmp_content=[constants, s_part, p_part , screen].compact.join("\n")
    tmp_content.gsub(/i '(.*?)'/){File.read($1)}
  end

  # Writes a CONST block for every input
  def constants
    @i=0
    @c_ids = []
```

```
    inputs.map{|input|
      @c_ids << "#{@i+=1}.1 "
      "s #{@i} CONST\np #{@i}\n\t#{input}"
    }
  end

  # Defines the links between the block and its inputs
  def s_part
    "s #{@i+=1} #{@name} #{@c_ids}"
  end

  # Defines the screen block to show the output
  # The outputs_number is 1 by default, but can be defined to be more :
  #   Block.new(:mtm,['Strasbourg'], 12)[2]
  def screen
    input_ids = (1..@outputs_number).map{|o|
      "#{@i}.#{o}"
    }.join(" ")
    "s #{@i+1} SCREEN #{input_ids}\np #{@i+1}\n\t'(6E15.7)'"
  end

  # Writes the parameters for the block, if needed
  def p_part
    ps = parameters.map{|p|
      case p
        when String : "'#{p}'"
        else   p
        end
    }
    ["p #{@i}", ps].join("\n\t") unless parameters.empty?
  end
end

# Reads a template file present in 'templates' folder with template_name.insel name
# Replaces every placeholder with specified values and uses it as insel_content
#
# For example, templates/a_times_b.insel :
#########################
#   s 1 MUL  3.1 2.1
#   s 2 CONST
#   p 2
#              $a$
#   s 3 CONST
#   p 3
#              $b$
#   s 4 SCREEN  1.1
#   p 4
#       '*'
#
########################
#
#
# Template.a_times_b(:a=> 5, :b=>3)
# => 15.0
```

```
  class Template < Model
    attr_reader :name, :parameters, :filename

    def initialize(name, parameters)
      @name, @parameters = name.to_s, parameters.merge(:bp_folder => File.join(InselPath, 'data', 'bp'
      @filename = File.expand_path(File.join(File.dirname(__FILE__), '..', 'templates', @name+'.insel'
    end

    def self.method_missing(sim, *parameters)
      new(sim, *parameters).results
    end

    private

    # Replaces every placeholder with specified values and uses it as insel_content
    def insel_content
      tmp_content=File.read(@filename)
      parameters.each{|k,v|
        tmp_content.gsub!("$#{k}$",v.to_s)
      }
      tmp_content.gsub(/i '(.*?)'/){File.read($1)}
    end
  end
end
```

Approach One: Interactive Ruby Interpreter: Start irb in Terminal

Voraussetzung: "insel" executable muss im Pfad liegen!

```
irb
>> require 'lib/insel'
>> Insel::Block.pi
=> 3.141593
>> exit
```

or using namespace `Insel`

```
irb
>> require 'lib/insel'
>> include Insel
>> Block.pi
=> 3.141593
>> exit
```

Approach Two: Write Ruby file and run ruby "filename"

```
# Needed library in order to call Insel blocks and templates from Ruby
# Loads the content of lib/insel.rb
require 'lib/insel'

# Avoids writing 'Insel::Block' instead of just 'Block'
include Insel
```

```
# Returns the value of Pi block
#   Block.block_name
puts Block.pi

# Calculates sin((6+4)*9) = sin (90) = 1
#   Block.block_name(input1,input2,....,inputN)
puts Block.sin(Block.mul(Block.sum(6,4), 9))

# Creates an Array from 1 to 10
#   Block.launch(:block_name, [parameter1,parameter2,...,parameterM])
puts Block.launch(:do, [1,10,1]).inspect

# Gets average temperature in december in Strasbourg [C]
#   Block.new(:block_name, [parameter1,parameter2,...,parameterM],input1, input2, ...., inputN)[which_
puts Block.new(:mtm,['Strasbourg'], 12)[2]

# Calculates 5*7 with a template
#   Template.template_name(:variable1 => value1, ..., :variableN => valueN)
puts Template.a_times_b(:a => 5, :b => 7)

# Fill factor in % of SunPower SPR-305-WHT-I by STC [%]
# NOTE: The pv_id could be different on other systems
puts Template.fill_factor(:pv_id => '003281', :temperature=> 25, :irradiance => 1000)*100

# Isc of SunPower SPR-305-WHT-I by STC [A]
puts Template.i_sc(:pv_id => '003281', :temperature=> 25, :irradiance => 1000)

# Uoc of SunPower SPR-305-WHT-I by STC [A]
puts Template.u_oc(:pv_id => '003281', :temperature=> 25, :irradiance => 1000)


[1,2,3,4,5,6,7,8,9,10].each{|e| puts Insel::Template.a_times_b(:a=> e,:b=>5)}


(-25..75).step(25){|ta| puts Template.fill_factor(:pv_id=> '003281',:temperature => ta, :irradiance =>
```

13.4 Optimization with GenOpt

13.5 Direct calls of INSEL blocks

The open DLL concept of INSEL allows programmers to interact with exported functions of INSEL DLLs. In principle, there are two different methods how the interaction can be implemented.

As described earlier in Module 12, all INSEL blocks have a unique interface, which is

```
SUBROUTINE name(IN,OUT,IP,RP,DP,BP,SP)
```

in Fortran, or

```
#include "MyTypes.h"
extern "C" void name(REAL* IN, REAL* OUT, INT* IP, REAL* RP,
   DOUBLE* DP, REAL* BP, STRARRAY SP, unsigned int SPlen = FOR_STRLEN)
```

in C/C++. The include file `MyTypes.h` contains the definition of the data types `REAL*` etc. as shown in Module 12, page **??**.

The first method to access INSEL blocks programmatically is to directly call the subroutine or function. The second method makes use of the wrapper class `CinselBlock` which is exported by `inselDi.dll`.

We start with the first approach. Although it is slightly more complicated it has the advantage of showing the gift – and not just the wrapping paper.

Identification call

In any case, the calling program has to care for the allocation of the block specific memory, i. e. in particular the size of the input array `IN`, the output array `OUT`, the internal memory arrays `IP`, `RP`, and `DP`, the numerical block parameters `BP`, and the string parameters `SP`. The blocks memory requirements can be found by an Idenfication call, i. e. with `IP(2) = -1` in Fortran or `IP[1] = -1` in C/C++. The routine returns the information:

```
IP(1)  = OPM
IP(2)  = INMIN
IP(3)  = IPS
IP(4)  = BPMIN
IP(5)  = SPMIN
IP(6)  = SPS
IP(7)  = GROUP
IP(8)  = RPS
IP(9)  = DPS
IP(10) = BPS
SP     = BNAMES
IN     = FLOAT(INS)
OUT    = FLOAT(OUTS)
```

in Fortran notation – see Module 12, page 266f for the meaning of the variables. Most important, recall that if `OPM` is greater than one, the routine contains more

than one INSEL block and it depends on the value of `OPM` which block is executed by a call.

Exercise 13.3 Write a Fortran or C program, and find out which INSEL blocks are implemented in `SUBROUTINE fb0043`, for instance, and how much memory is required for the arrays.

Hints All INSEL blocks are usually exported in C calling convention. The subroutine `fb0043` is exported from `inselFB.dll`. If you link the DLL statically, link your program with `inselFB.lib`. A quick-and-dirty solution could use a `print,*` Fortran- or a `printf` C statement. Making use of the INSEL message system as explained in Module 12, page 261ff would be the much better solution in a professional environment.

Solution The Fortran solution based on Microsoft Fortran PowerStation 4.0 uses the interface statement. Other Fortran compilers can have a different syntax for the inclusion of code in C calling convention.

```
INTERFACE TO SUBROUTINE FB0043[C](IN,OUT,IP,RP,DP,BP,SP)
   INTEGER          IP  [REFERENCE]
   REAL             IN  [REFERENCE]
   REAL             OUT [REFERENCE]
   REAL             RP  [REFERENCE]
   REAL             BP  [REFERENCE]
   DOUBLE PRECISION DP  [REFERENCE]
   CHARACTER*80     SP  [REFERENCE]
END

PROGRAM IDCALL
IMPLICIT NONE
INTEGER          IP(10),i
REAL             IN
REAL             OUT
REAL             RP
REAL             BP
DOUBLE PRECISION DP
CHARACTER*80     SP

IP(2) = -1
CALL FB0043(IN,OUT,IP,RP,DP,BP,SP)
print*," "
print*,"   Blockname: ",SP
i = ANINT(IN)
print*,"   INs: ",i
i = ANINT(OUT)
print*,"   OUTs:",i
print*,"   IPs: ",IP(3)
print*,"   RPs: ",IP(8)
print*,"   DPs: ",IP(9)
print*,"   BPs: ",IP(10)
print*," "
STOP
```

```
END
```

Output This is the output of the program:

```
Blockname: FDIST

INs:           1
OUTs:          4
IPs:          18
RPs:        1002
DPs:        1002
BPs:           5
```

C/C++ code The C/C++ code is very similar:

```
#include <stdio.h>
#include "MyTypes.h"

extern "C" void fb0043(REAL* IN, REAL* OUT, INT* IP, REAL* RP,
   DOUBLE* DP, REAL* BP, STRARRAY SP, unsigned int SPlen = FOR_STRLEN);

void main()
{
   INT      IP[10];
   REAL     IN;
   REAL     OUT;
   REAL     RP;
   REAL     BP;
   DOUBLE   DP;
   STRARRAY SP;
   int      i;

   IP[1] = -1;
   fb0043(&IN, &OUT, IP, &RP, &DP, &BP, SP);
   printf("\n");
   printf("   Blockname: %s\n",SP);
   i = int(IN);
   printf("   INs:  %i\n",i);
   i = int(OUT);
   printf("   OUTs: %i\n",i);
   printf("   IPs:  %i\n",IP[2]);
   printf("   RPs:  %i\n",IP[7]);
   printf("   DPs:  %i\n",IP[8]);
   printf("   BPs:  %i\n",IP[9]);
   printf("\n");
}
```

C/C++ output The output, too:

```
Blockname: FDIST

INs:  1
OUTs: 4
IPs:  18
```

```
RPs:  1002
DPs:  1002
BPs:  5
```

Constructor call

Before an INSEL block can be used, it must be called in the Constructor call. This is accomplished by calling the block with `IP(2) = 1` in Fortran or `IP[1] = 1` in C/C++. Some blocks perform plausibility checks or initializations in the mode, some do nothing. Nevertheless, any INSEL block should be called in the Constructor call in order to avoid unwanted side effects.

Standard call

After these preparations the respective INSEL block is ready for use and can be called in Standard call mode, i. e. with `IP(2) = 0` in Fortran or `IP[1] = 0` in C/C++, as often as you like. Nearly all INSEL blocks allow for an unlimited number of instances. The calling program must take care for their memory management, however.

Destructor call

Some few INSEL blocks – like the fitting routines, for example – perform their main action during the Destructor call, most blocks do nothing in this call mode. Hence, every INSEL block instance should be called in this mode before the host program terminates.

INSEL message output

Before we look at some examples, notice that all INSEL blocks can generate textual output like error messages, warnings etc. What is the target of these message streams? In INSEL all output messages finally end in a call to the routine `os0txt` implemented in an inselText DLL.

Some default inselText DLLs are provided with INSEL, like `msgBox.dll` which generates a MessageBox with an OK button for each INSEL message output, or `noText.dll` which completely supresses the INSEL message output.

INSEL interface programmers can write their own DLLs for INSEL message output. The function `os0txt` takes two parameters: a handle to the window for the text output and a pointer to a string, which contains the message text. The prototype of the function being

```
void os0txt(int hwnd, char czMeldung[80]);
```

The DLL which provides the routine `os0txt` is specified in `inselDi.ini`.

CONST example

Let us start with a super-trivial example and define a constant 17 with the CONST block of INSEL and display its value on screen via a call to the SCREEN block. Although this is in fact not really a have-to-use-INSEL example, it shows the main principles of interacting with INSEL blocks.

Fortran code

```
C     CONST block -------------------------------------------
      INTERFACE TO SUBROUTINE FB0001[C](IN,OUT,IP,RP,DP,BP,SP)
         INTEGER          IP  [REFERENCE]
         REAL             IN  [REFERENCE]
```

```
         REAL             OUT [REFERENCE]
         REAL             RP  [REFERENCE]
         REAL             BP  [REFERENCE]
         DOUBLE PRECISION DP  [REFERENCE]
         CHARACTER*80     SP  [REFERENCE]
      END
C     SCREEN block -----------------------------------------
      INTERFACE TO SUBROUTINE FB0014[C](IN,OUT,IP,RP,DP,BP,SP)
         INTEGER          IP  [REFERENCE]
         REAL             IN  [REFERENCE]
         REAL             OUT [REFERENCE]
         REAL             RP  [REFERENCE]
         REAL             BP  [REFERENCE]
         DOUBLE PRECISION DP  [REFERENCE]
         CHARACTER*80     SP  [REFERENCE]
      END
C     -------------------------------------------------------
      PROGRAM TRIVIAL_BUT_

      IMPLICIT NONE
      INTEGER          IP1(10),IP2(11)
      REAL             IN1,    IN2(6)
      REAL             OUT1,   OUT2
      REAL             RP1,    RP2
      REAL             BP1,    BP2
      DOUBLE PRECISION DP1,    DP2
      CHARACTER*80     SP1,    SP2
      INTEGER          WINDOW / 0 /
      CHARACTER*80     TEXT   /" "/

C     Initialise INSEL message system
      CALL LOSOTXT(WINDOW,TEXT)

C     Constructor calls
      IP1(2) = 1
      BP1    = 17.0
      CALL FB0001(IN1,OUT1,IP1,RP1,DP1,BP1,SP1)

      IP2(2) = 1
      IP2(5) = 1 ! SCREEN block with one input
      SP2    = '('   SCREEN block: '',F7.1)'
      CALL FB0014(IN2,OUT2,IP2,RP2,DP2,BP2,SP2)

C     Standard calls
      IP1(2) = 0
      CALL FB0001(IN1,OUT1,IP1,RP1,DP1,BP1,SP1)

      IP2(2) = 0
      IN2    = OUT1
      CALL FB0014(IN2,OUT2,IP2,RP2,DP2,BP2,SP2)

C     Destructor calls
      IP1(2) = 2
      CALL FB0001(IN1,OUT1,IP1,RP1,DP1,BP1,SP1)
```

```
IP2(2) = 2
CALL FB0014(IN2,OUT2,IP2,RP2,DP2,BP2,SP2)

STOP
END
```

Output As expected, the output is:

```
SCREEN block:    17.0
```

Please, observe a few details in the Fortran code.

LOSOTXT First, before anything happens, the INSEL message system should be initialized by a call to `LOSOTXT`. The variable `WINDOW` contains the handle to the window, where the output messages go to. If this parameter is set to zero, `inselText.dll` writes into a DOS box. `TEXT` usually contains the complete message string and can be blank in the first call. Both parameters are handed over by reference, i. e. a C call could look like

```
extern "C" void __stdcall LOSOTXT
   (_int32* dummy, char Text[80], unsigned int len = 80);
_int32 Fenster;
char   Text[80];
LOSOTXT(&Fenster,Text);
```

`LOSOTXT` provides a quick solution to hand over a message to the INSEL message system.

IP(5) Second, the SCREEN block makes use of the parameter `IP(5)` which is reserved in INSEL for the number of currently connected block inputs. Usually, the inselEngine sets this parameter. However, in external programs, the calling program must set `IP(5)` to an appropriate value, i. e. one in the present case.

Third, the string parameter `SP(1)` of the SCREEN block should be set before the constructor call is made, since the constructor call performs plausibility checks on its value.

Fourth, all blocks should finally be called in the Destructor call. If you call the SCREEN block with an invalid format you will see one reason, why.

WARNING Do not initialize variables which don't exist, i. e. if a block has no `RP`, for instance, do not assign a value to it, otherwise the result is unpredictable.

We are now ready for a more complex application.

Exercise 13.4 Write a Fortran or C program which reads monthly mean values for any location from the inselWeather data base (MTM block in em0018), calculates a time series of global radiation on a horizontal plane for one year in daily resolution (GENGD block in em0016) and plots the data (PLOT block in fb0044). For the generation of the sequence of days and months use the CLOCK block in fb0024.

Hints Rather than explaining twenty details we show verbatim copies of the original block headers. They can also serve as further examples for the src2tex application, as described in Section ??.

CLOCK This is the header of the CLOCK block, as implemented in `fb0024.f`.

```
C------------------------------------------------------------------------
C #Begin
C #Block CLOCK
C #Description
C    The CLOCK block generates date and time of the actual
C    simulation time step with constant increment.
C #Layout
C    #Inputs      0 $\ldots$ [1]
C    #Outputs     6
C    #Parameters  13
C    #Strings     1
C    #Group       T
C #Details
C    #Inputs
C       #IN(1)   Output $t$ of a predecessor (optional). Should the
C                block defining the $t$ signal be executed again after
C                CLOCK has finished its operation, the
C                CLOCK block performs a reset and starts again.
C    #Outputs
C       #OUT(1)  Year   $a$
C       #OUT(2)  Month  $M$
C       #OUT(3)  Day    $d$
C       #OUT(4)  Hour   $h$
C       #OUT(5)  Minute $m$
C       #OUT(6)  Second $s$
C    #Parameters
C       #BP(1)   Start on year $a_1$
C       #BP(2)   Start on month $M_1$
C       #BP(3)   Start on day $d_1$
C       #BP(4)   Start on hour $h_1$
C       #BP(5)   Start on minute $m_1$
C       #BP(6)   Start on second $s_1$
C       #BP(7)   Stop  on year $a_2$
C       #BP(8)   Stop  on month $M_2$
C       #BP(9)   Stop  on day $d_2$
C       #BP(10)  Stop  on hour $h_2$
C       #BP(11)  Stop  on minute $m_2$
C       #BP(12)  Stop  on second $s_2$
C       #BP(13)  Increment $\Delta t$
C    #Strings
C       #SP(1)   Unit of the increment $\Delta t$,
C                case sensitive, ie 'm' $\neq$ 'M', for example
C                \begin{detaillist}
C                   \item['a'] Years
C                   \item['M'] Months
C                   \item['d'] Days
C                   \item['h'] Hours
C                   \item['m'] Minutes
```

```
C                  \item['s'] Seconds
C               \end{detaillist}
C #Internals
C    #Integers
C       #IP(1)  Return code
C       #IP(2)  Call mode
C               \begin{detaillist}
C                  \item[-1] Identification call
C                  \item[0]  Standard call
C                  \item[1]  Constructor call
C                  \item[2]  Destructor call
C               \end{detaillist}
C       #IP(3)  Operation mode
C       #IP(4)  User defined block number
C       #IP(5)  Number of current block inputs
C       #IP(6)  Jump parameter
C       #IP(7)  Debug level
C       #IP(8..10)  Reserved
C       #IP(11) Integer representation of BP(1)
C       #IP(12) Integer representation of BP(2)
C       #IP(13) Integer representation of BP(3)
C       #IP(14) Integer representation of BP(4)
C       #IP(15) Integer representation of BP(5)
C       #IP(16) Corresponding Julian day
C       #IP(17) Integer representation of BP(7)
C       #IP(18) Integer representation of BP(8)
C       #IP(19) Integer representation of BP(9)
C       #IP(20) Integer representation of BP(10)
C       #IP(21) Integer representation of BP(11)
C       #IP(22) Corresponding Julian day
C       #IP(23) First call to CLOCK block
C       #IP(24) Mode
C       #IP(25) Integer representation of BP(13)
C       #IP(26) Current year
C       #IP(27) Current month
C       #IP(28) Current day
C       #IP(29) Current hour
C       #IP(30) Current minute
C       #IP(31) Second of year when to start as defined thru BP(1) to
C               BP(6)
C       #IP(32) Second of year when to stop  as defined thru BP(7) to
C               BP(12)
C       #IP(33) Current Julian day
C       #IP(34) Counter for the number of calls with invalid date
C    #Reals
C       #RP(1)  Current second
C    #Doubles
C       #None
C #Dependencies
C    Subroutine CKDATE
C    Subroutine CKTIME
C    Subroutine GREGOR
C    Function   ID
C    Function   ISOY
```

```
C    Subroutine JULIAN
C    Subroutine MSG
C    Subroutine STRIP
C #Authors
C    Juergen Schumacher
C #End
C-----------------------------------------------------------------------
```

MTM This is the header of the MTM block, as implemented in `em0018.f`.

```
C-----------------------------------------------------------------------
C #Begin
C #Block MTM, MTMLALO
C #Description MTM
C    The MTM block returns monthly mean values of meteorological
C    data from the inselWeather database.
C #Description MTMLALO
C    The MTMLALO block returns monthly mean values of meteorological
C    data for a location specified by latitude and longitude
C    interpolated from the inselWeather database.
C #Layout MTM
C    #Inputs      1
C    #Outputs     9
C    #Parameters  0 $\ldots$ [6]
C    #Strings     1
C    #Group       S
C #Layout MTMLALO
C    #Inputs      1
C    #Outputs     9
C    #Parameters  2
C    #Strings     0
C    #Group       S
C #Details
C    #Inputs
C       #IN(1)  Month $M \in [1,12]$
C    #Outputs
C       #OUT(1) Global radiation $G_{\rm h}$ / W\,m$^{-2}$ on a horizontal
C               plane
C       #OUT(2) Wind speed $v_{\rm w}$ / m\,s$^{-1}$
C       #OUT(3) Ambient temperature $T$ / $^\circ$C
C       #OUT(4) Minimum ambient temperature $T_{\rm a,min}$
C               / $^\circ$C
C       #OUT(5) Maximum ambient temperature $T_{\rm a,max}$
C               / $^\circ$C
C               / $^\circ$C
C       #OUT(6) Rain / mm
C       #OUT(7) Annual mean ambient temperature
C               / $^\circ$C
C       #OUT(8) Maximum ambient temperature difference
C               / $^\circ$C
C       #OUT(9) Relative humidity
C    #Parameters MTM
C       #BP(1)  Latitude  $\varphi \in [-90^\circ,+90^\circ]$, northern
C               hemisphere positive
C       #BP(2)  Longitude  $\lambda \in [0^\circ,360^\circ)$,
```

```
C               west of Greenwich; values east of Greenwich may be used
C               with a minus sign
C       #BP(3)  Latitude range $\Delta\varphi$ / $^\circ$
C       #BP(4)  Longitude range $\Delta\lambda$ / $^\circ$
C       #BP(5)  Country code CC
C       #BP(6)  Continent code KC
C    #Parameters MTMLALO
C       #BP(1)  Latitude  $\varphi \in [-90^\circ,+90^\circ]$, northern
C               hemisphere positive
C       #BP(2)  Longitude  $\lambda \in [0^\circ,360^\circ)$,
C               west of Greenwich; values east of Greenwich may be used
C               with a minus sign
C    #Strings MTM
C       #SP(1)  Name of location
C    #Strings MTMLALO
C       #None
C #Internals
C    #Integers
C       #IP(1)  Return code
C       #IP(2)  Call mode
C               \begin{detaillist}
C                  \item[-1] Identification call
C                  \item[0]  Standard call
C                  \item[1]  Constructor call
C                  \item[2]  Destructor call
C               \end{detaillist}
C       #IP(3)  Operation mode
C       #IP(4)  User defined block number
C       #IP(5)  Number of current block inputs
C       #IP(6)  Jump parameter
C       #IP(7)  Debug level
C       #IP(8..10)  Reserved
C       #IP(11) Counter for the number of calls with invalid input
C       #IP(12) Continent code
C       #IP(13) Country code
C       #IP(14) REPORT PROVISORIUM
C    #Reals
C       #RP(1-12)  Global radiation / W\,m$^{-2}$
C       #RP(13-24) Wind speed / m\,s$^{-1}$
C       #RP(25-36) Ambient temperature / \degC
C       #RP(37-48) Minimum ambient temperature / \degC
C       #RP(49-60) Minimum ambient temperature / \degC
C       #RP(61-72) Precipitation / mm
C       #RP(73)    Latitude from data base
C       #RP(74)    Longitude from data base
C       #RP(75)    Estimated time zone
C       #RP(76)    Height from data base
C       #RP(77-88) Relative humidity
C       #RP(89)    Gmean
C       #RP(90)    vmean
C       #RP(91)    T1mean
C       #RP(92)    T2mean
C       #RP(93)    T3mean
C       #RP(94)    Rmean
```

```
C       #RP(95)    RHmean
C    #Doubles
C       #None
C #Dependencies
C    Subroutine MSG
C    Subroutine MTLOC
C    Subroutine MTMCLO
C    Subroutine MTMDAT
C    Subroutine MTMGET
C    Subroutine MTMLST
C    Subroutine MTMPTR
C    Subroutine STRIP
C #Authors
C    Christian Langer
C    Jibbo Mueller
C    Juergen Schumacher
C    Marc Esser
C #End
C-------------------------------------------------------------------------
```

GENGD This is the header of the GENGD block, as implemented in `em0016.f`.

```
C-------------------------------------------------------------------------
C #Begin
C #Block GENGD
C #Description
C    The GENGD block generates a series of daily global
C    radiation data from monthly mean values.
C #Layout
C    #Inputs      4
C    #Outputs     1
C    #Parameters  9
C    #Strings     0
C    #Group       S
C #Details
C    #Inputs
C       #IN(1)  Monthly mean value $G_{\rm h}(M)$ / W\,m$^{-2}$ of global
C               radiation on a horizontal plane
C       #IN(2)  Year $a$
C       #IN(3)  Month $M \in [1,12]$
C       #IN(4)  Day $d \in [1,31]$
C    #Outputs
C       #OUT(1) Daily mean value $G_{\rm h}(d)$ / W\,m$^{-2}$ of global
C               radiation on a horizontal plane
C    #Parameters
C       #BP(1)  Model
C               \begin{detaillist}
C                  \item[0] Gordon Reddy model
C                  \item[1] Aguiar Collares-Pereira model
C               \end{detaillist}
C       #BP(2)  Latitude  $\varphi \in [-90^\circ,+90^\circ]$, northern
C               hemisphere positive
C       #BP(3)  Longitude  $\lambda \in [0^\circ,360^\circ)$,
C               west of Greenwich; values east of Greenwich may be used
C               with a minus sign
```

```
C       #BP(4)  Time zone $Z \in [0,23]$, Greenwich Mean Time $Z=0$,
C               Central European Time $Z=23$.
C       #BP(5)  Variance factor $f_\sigma$ to the Gordon / Reddy
C               correlation, eq \ref{GR_sigma}; if unknown $f_\sigma =
C               1$ is recommended
C       #BP(6)  Coefficient $c_{\sigma}$ corresponding to the
C               year-to-year variability due to different climatic
C               conditions. When $c_{\sigma}$ is set to zero
C               the year-to-year variability is omitted.
C               $c_{\sigma} = 0.185$ approximates North American
C               variability, while $c_{\sigma} = 0.3$ approximates
C               European variability
C       #BP(7)  Autocorrelation coefficient $\rho(1)$ at a lag of one
C               day; if unknown $\rho(1) = 0.3$ is recommended
C       #BP(8)  Autocorrelation coefficient $\rho(2)$ at a lag of two
C               days; if unknown $\rho(2) = 0.57 \rho(1)$ is recommended
C       #BP(9)  Initialisation $I_{\rm seed}$ of random number generator
C    #Strings
C       #None
C #Internals
C    #Integers
C       #IP(1)  Return code
C       #IP(2)  Call mode
C               \begin{detaillist}
C                  \item[-1] Identification call
C                  \item[0]  Standard call
C                  \item[1]  Constructor call
C                  \item[2]  Destructor call
C               \end{detaillist}
C       #IP(3)  Operation mode
C       #IP(4)  User defined block number
C       #IP(5)  Number of current block inputs
C       #IP(6)  Jump parameter
C       #IP(7)  Debug level
C       #IP(8..10)  Reserved
C       #IP(11) Year for which data have already been generated
C       #IP(12) Month for which data have already been generated
C       #IP(13) Updated version of $I_{\rm seed}$ as manipulated by
C               ran1.for
C       #IP(14)..IP(16) Integer memory for Ran1
C       #IP(17) Mode
C       #IP(18) Last generated kt value (mode 2 only)
C    #Reals
C       #RP(1)..RP(31)   Memory for daily radiation data
C       #RP(32)..RP(128) Real memory for Ran1
C    #Doubles
C       #None
C #Dependencies
C    Subroutine GENGD
C    Subroutine MSG
C    Function   GASDEV
C #Authors
C    Juergen Schumacher
C #End
```

```
C-----------------------------------------------------------------------
```

PLOT This is the header of the PLOT block, as implemented in `fb0044.f`.

```
C-----------------------------------------------------------------------
C #Begin
C #Block PLOT, PLOTP, PLOTPMC, PLOTPM3D, PLOTG
C #Description PLOT
C    The PLOT block generates graphical output of its connected
C    input data via gnuplot.
C #Description PLOTP
C    The PLOTP block generates a parametric graphical output of
C    its connected input data via gnuplot.
C #Description PLOTPMC
C    The PLOTPMC block generates a palette-mapped carpet plot output
C    of its connected input data via gnuplot.
C #Description PLOTPM3D
C    The PLOTPM3D block generates a palette-mapped 3D plot output
C    of its connected input data via gnuplot.
C #Layout PLOT
C    #Inputs      $2 \ldots [20]$
C    #Outputs     0
C    #Parameters  0
C    #Strings     $0 \ldots [1]$
C    #Group       S
C #Layout PLOTP
C    #Inputs      $3 \ldots [20]$
C    #Outputs     0
C    #Parameters  0
C    #Strings     $0 \ldots [1]$
C    #Group       S
C #Layout PLOTPMC
C    #Inputs      3
C    #Outputs     0
C    #Parameters  0
C    #Strings     $0 \ldots [1]$
C    #Group       S
C #Layout PLOTPM3D
C    #Inputs      3
C    #Outputs     0
C    #Parameters  0
C    #Strings     $0 \ldots [1]$
C    #Group       S
C #Details
C    #Inputs PLOT
C       #IN(1)  Any signal $x$
C       #IN(2)  Any signal $y_1$
C       #IN(n)  Any signal $y_{n-1}$
C    #Inputs PLOTP
C       #IN(1)  Curve parameter $p$
C       #IN(2)  Any signal $x$
C       #IN(3)  Any signal $y_1$
C       #IN(n)  Any signal $y_{n-2}$
C    #Inputs PLOTPMC
C       #IN(1)  Any signal $x$
```

```
C       #IN(2)  Any signal $y$
C       #IN(3)  Any signal $z$
C    #Inputs PLOTPM3D
C       #IN(1)  Any signal $x$
C       #IN(2)  Any signal $y$
C       #IN(3)  Any signal $z$
C    #Outputs
C       #None
C    #Parameters
C       #None
C    #Strings
C       #SP(1) File name fn of a gnuplot command file. If
C              no file name is provided, a default file insel.gnu
C              is generated with default gnuplot commands.
C #Internals
C    #Integers
C       #IP(1)  Return code
C       #IP(2)  Call mode
C               \begin{detaillist}
C                  \item[-1] Identification call
C                  \item[0]  Standard call
C                  \item[1]  Constructor call
C                  \item[2]  Destructor call
C               \end{detaillist}
C       #IP(3)  Operation mode
C       #IP(4)  User defined block number
C       #IP(5)  Number of current block inputs
C       #IP(6)  Jump parameter
C       #IP(7)  Debug level
C       #IP(8..10)  Reserved
C       #IP(11) Unit number of data file insel.gpl
C       #IP(12) Unit number of gnuplot file *.gnu
C       #ICOLS  Number of columns in the gnuplot data file
C    #Reals
C       #None
C    #Doubles
C       #None
C #Dependencies
C    Subroutine MSG
C    Subroutine STRIP
C #Authors
C    Juergen Schumacher
C #End
C-------------------------------------------------------------------------
```

Solution

At first, we choose Stuttgart as location and do not solve the task in one step but check the access to the data base in a first step. There are many traps, so it seems to be advisable, to reduce their number and start with a smaller piece of cake.

This is the Fortran code which plots the twelve monthly mean values of the global irradiance – read from the inselWeather data base.

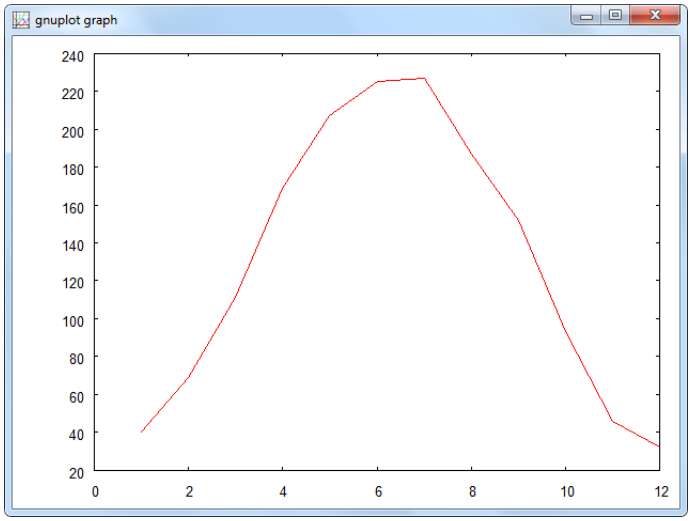
```
C     CLOCK block -------------------------------------------
      INTERFACE TO SUBROUTINE FB0024[C](IN,OUT,IP,RP,DP,BP,SP)
```

```
      INTEGER          IP  [REFERENCE]
      REAL             IN  [REFERENCE]
      REAL             OUT [REFERENCE]
      REAL             RP  [REFERENCE]
      REAL             BP  [REFERENCE]
      DOUBLE PRECISION DP  [REFERENCE]
      CHARACTER*80     SP  [REFERENCE]
   END
C  MTM block ------------------------------------------------
   INTERFACE TO SUBROUTINE EM0018[C](IN,OUT,IP,RP,DP,BP,SP)
      INTEGER          IP  [REFERENCE]
      REAL             IN  [REFERENCE]
      REAL             OUT [REFERENCE]
      REAL             RP  [REFERENCE]
      REAL             BP  [REFERENCE]
      DOUBLE PRECISION DP  [REFERENCE]
      CHARACTER*80     SP  [REFERENCE]
   END
C  PLOT block ------------------------------------------------
   INTERFACE TO SUBROUTINE FB0044[C](IN,OUT,IP,RP,DP,BP,SP)
      INTEGER          IP  [REFERENCE]
      REAL             IN  [REFERENCE]
      REAL             OUT [REFERENCE]
      REAL             RP  [REFERENCE]
      REAL             BP  [REFERENCE]
      DOUBLE PRECISION DP  [REFERENCE]
      CHARACTER*80     SP  [REFERENCE]
   END
C  ------------------------------------------------------------
   PROGRAM dailyRadiationData1

   IMPLICIT NONE  ! CLOCK   MTM     PLOT
   INTEGER          IP1(34),IP2(13),IP3(13)
   REAL             IN1,    IN2,    IN3(20)
   REAL             OUT1(6),OUT2(9),OUT3
   REAL             RP1,    RP2(95),RP3
   REAL             BP1(13),BP2(6), BP3
   DOUBLE PRECISION DP1,    DP2,    DP3
   CHARACTER*80     SP1,    SP2,    SP3
   INTEGER          WINDOW / 0 /
   CHARACTER*80     TEXT   /' '/

   INTEGER i

C  Initialise INSEL message system
   CALL LOSOTXT(WINDOW,TEXT)

C  Constructor calls
   BP1(1)  = 2006.0  ! Start year
   BP1(2)  = 1.0     ! Start month
   BP1(3)  = 1.0     ! Start day
   BP1(4)  = 0.0     ! Start hour
   BP1(5)  = 0.0     ! Start minute
   BP1(6)  = 0.0     ! Start second
```

```
      BP1(7)  = 2007.0  ! End year
      BP1(8)  = 1.0     ! End month
      BP1(9)  = 1.0     ! End day
      BP1(10) = 0.0     ! End hour
      BP1(11) = 0.0     ! End minute
      BP1(12) = 0.0     ! End second
      BP1(13) = 1.0     ! Increment
      SP1     = 'M'     ! Run in months
      DO i = 1,34
         IP1(i) = 0
      END DO
      IP1(2) = 1        ! Constructor call
      RP1    = 0.0
      CALL FB0024(IN1,OUT1,IP1,RP1,DP1,BP1,SP1)
      IF (IP1(1) .NE. 0) STOP 'CLOCK constructor call failed'

      DO i = 1,13
         IP2(i) = 0
      END DO
      DO i = 1,95
         RP2(i) = 0.0
      END DO
      DO i = 1,6
         BP2(i) = 0.0
      END DO
      SP2    = 'Stuttgart'
      IP2(2) = 1 ! Constructor call
      CALL EM0018(IN2,OUT2,IP2,RP2,DP2,BP2,SP2)
      IF (IP2(1) .NE. 0) STOP 'MTM constructor call failed'

      DO i = 1,13
         IP3(i) = 0
      END DO
      IP3(3) = 1    ! Operation mode OPM = 1, i.e. PLOT block (not
PLOTP)
      IP3(5) = 2    ! Two block inputs: (1) Month, (2) Radiation
      RP3    = 0.0
      BP3    = 1.0  ! Mode 1
      SP3    = ' '
      IP3(2) = 1    ! Constructor call
      CALL FB0044(IN3,OUT3,IP3,RP3,DP3,BP3,SP3)
      IF (IP3(1) .NE. 0) STOP 'PLOT constructor call failed'

C     Standard calls
      IP1(2) = 0
      IP2(2) = 0
      IP3(2) = 0
      DO i = 1,12
C        Call CLOCK to return month
         CALL FB0024(IN1,OUT1,IP1,RP1,DP1,BP1,SP1)
         IN2 = OUT1(2) ! = Current month
C        Call MTM to return monthly mean radiation value
         CALL EM0018(IN2,OUT2,IP2,RP2,DP2,BP2,SP2)
         !print*,"G = ",OUT2(1)
```

```
         IN3(1) = i
         IN3(2) = OUT2(1)
         CALL FB0044(IN3,OUT3,IP3,RP3,DP3,BP3,SP3)
      END DO

C     Destructor calls
      IP1(2) = 2
      IP2(2) = 2
      IP3(2) = 2
      CALL FB0024(IN1,OUT1,IP1,RP1,DP1,BP1,SP1)
      CALL EM0018(IN2,OUT2,IP2,RP2,DP2,BP2,SP2)
      CALL FB0044(IN3,OUT3,IP3,RP3,DP3,BP3,SP3)

      STOP
      END
```

The program generates this Gnuplot graph:



Now, it's only a small step to calculate and plot the daily radiation time series.

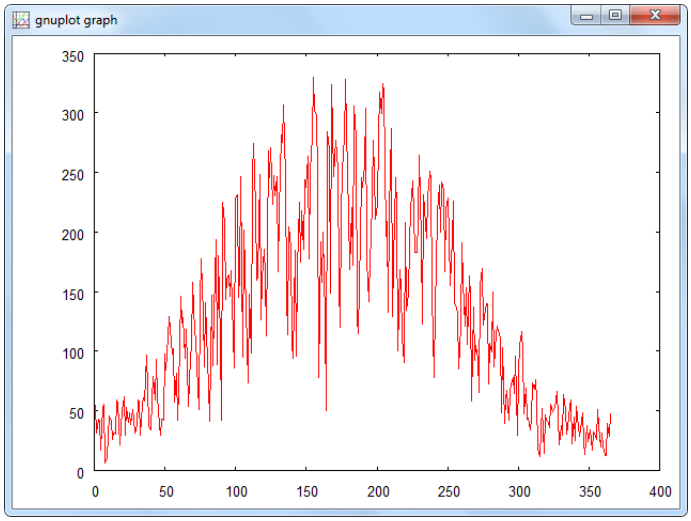
```
C     CLOCK block ------------------------------------------
      INTERFACE TO SUBROUTINE FB0024[C](IN,OUT,IP,RP,DP,BP,SP)
         INTEGER          IP  [REFERENCE]
         REAL             IN  [REFERENCE]
         REAL             OUT [REFERENCE]
         REAL             RP  [REFERENCE]
         REAL             BP  [REFERENCE]
         DOUBLE PRECISION DP  [REFERENCE]
         CHARACTER*80     SP  [REFERENCE]
      END
C     MTM block --------------------------------------------
      INTERFACE TO SUBROUTINE EM0018[C](IN,OUT,IP,RP,DP,BP,SP)
         INTEGER          IP  [REFERENCE]
```

```
      REAL             IN  [REFERENCE]
      REAL             OUT [REFERENCE]
      REAL             RP  [REFERENCE]
      REAL             BP  [REFERENCE]
      DOUBLE PRECISION DP  [REFERENCE]
      CHARACTER*80     SP  [REFERENCE]
   END
C  PLOT block ---------------------------------------------
   INTERFACE TO SUBROUTINE FB0044[C](IN,OUT,IP,RP,DP,BP,SP)
      INTEGER          IP  [REFERENCE]
      REAL             IN  [REFERENCE]
      REAL             OUT [REFERENCE]
      REAL             RP  [REFERENCE]
      REAL             BP  [REFERENCE]
      DOUBLE PRECISION DP  [REFERENCE]
      CHARACTER*80     SP  [REFERENCE]
   END
C  GENGD block ----------------------------------------------
   INTERFACE TO SUBROUTINE EM0016[C](IN,OUT,IP,RP,DP,BP,SP)
      INTEGER          IP  [REFERENCE]
      REAL             IN  [REFERENCE]
      REAL             OUT [REFERENCE]
      REAL             RP  [REFERENCE]
      REAL             BP  [REFERENCE]
      DOUBLE PRECISION DP  [REFERENCE]
      CHARACTER*80     SP  [REFERENCE]
   END
C  ---------------------------------------------------------
   PROGRAM dailyRadiationData2

   IMPLICIT NONE  ! CLOCK   MTM     PLOT    GENGD
   INTEGER          IP1(34),IP2(13),IP3(13),IP4(18)
   REAL             IN1,    IN2,    IN3(20),IN4(4)
   REAL             OUT1(6),OUT2(9),OUT3,   OUT4
   REAL             RP1,    RP2(95),RP3,    RP4(128)
   REAL             BP1(13),BP2(6), BP3,    BP4(9)
   DOUBLE PRECISION DP1,    DP2,    DP3,    DP4
   CHARACTER*80     SP1,    SP2,    SP3,    SP4
   INTEGER          WINDOW / 0 /
   CHARACTER*80     TEXT   /' '/

   INTEGER i

C  Initialise INSEL message system
   CALL LOSOTXT(WINDOW,TEXT)

C  Constructor calls
   BP1(1)  = 2006.0  ! Start year
   BP1(2)  = 1.0     ! Start month
   BP1(3)  = 1.0     ! Start day
   BP1(4)  = 0.0     ! Start hour
   BP1(5)  = 0.0     ! Start minute
   BP1(6)  = 0.0     ! Start second
   BP1(7)  = 2007.0  ! End year
```

```
BP1(8)  = 1.0     ! End month
BP1(9)  = 1.0     ! End day
BP1(10) = 0.0     ! End hour
BP1(11) = 0.0     ! End minute
BP1(12) = 0.0     ! End second
BP1(13) = 1.0     ! Increment
SP1     = 'd'     ! Run in days
DO i = 1,34
   IP1(i) = 0
END DO
IP1(2) = 1        ! Constructor call
RP1    = 0.0
DP1    = 0.0
CALL FB0024(IN1,OUT1,IP1,RP1,DP1,BP1,SP1)
IF (IP1(1) .NE. 0) STOP 'CLOCK constructor call failed'

DO i = 1,13
   IP2(i) = 0
END DO
DO i = 1,95
   RP2(i) = 0.0
END DO
DO i = 1,6
   BP2(i) = 0.0
END DO
DP2    = 0.0
SP2    = 'Stuttgart'
IP2(2) = 1 ! Constructor call
CALL EM0018(IN2,OUT2,IP2,RP2,DP2,BP2,SP2)
IF (IP2(1) .NE. 0) STOP 'MTM constructor call failed'

DO i = 1,13
   IP3(i) = 0
END DO
IP3(3) = 1    ! Operation mode OPM = 1, i.e. PLOT block (not PLOTP)
IP3(5) = 2    ! Two block inputs: (1) Month, (2) Radiation
RP3    = 0.0
DP3    = 0.0
BP3    = 1.0  ! Mode 1
SP3    = ' '
IP3(2) = 1    ! Constructor call
CALL FB0044(IN3,OUT3,IP3,RP3,DP3,BP3,SP3)
IF (IP3(1) .NE. 0) STOP 'PLOT constructor call failed'

DO i = 1,18
   IP4(i) = 0
END DO
DO i = 1,128
   RP4(i) = 0.0
END DO
BP4(1) = 1.0     ! Model:           Use the Gordon Reddy model
BP4(2) = RP2(73) ! Latitude:        Is available from MTM
BP4(3) = RP2(74) ! Longitude:       Dito
BP4(4) = 23.0    ! Time zone:       We know it, definitely
```

```
      BP4(5) = 1.0     ! Variance factor: Recommended default
      BP4(6) = 0.0     ! No year-to-year variability
      BP4(7) = 0.3     ! Recommended default
      BP4(8) = 0.171   ! Recommended default = 0.57 * BP4(7)
      BP4(9) = 4711    ! Any initialisation of the random number generator
      IP4(2) = 1       ! Constructor call
      CALL EM0016(IN4,OUT4,IP4,RP4,DP4,BP4,SP4)
      IF (IP4(1) .NE. 0) STOP 'GENGD constructor call failed'

C     Standard calls
      IP1(2) = 0
      IP2(2) = 0
      IP3(2) = 0
      IP4(2) = 0
      DO i = 1,365
C        Call CLOCK to return month
         CALL FB0024(IN1,OUT1,IP1,RP1,DP1,BP1,SP1)
         IN2 = OUT1(2)      ! = Current month
C        Call MTM to return monthly mean radiation value
         CALL EM0018(IN2,OUT2,IP2,RP2,DP2,BP2,SP2)
         IN4(1) = OUT2(1)   ! Monthly mean radiation
         IN4(2) = OUT1(1)   ! Year
         IN4(3) = OUT1(2)   ! Month
         IN4(4) = OUT1(3)   ! Day
         CALL EM0016(IN4,OUT4,IP4,RP4,DP4,BP4,SP4)
         IN3(1) = i
         IN3(2) = OUT4
         CALL FB0044(IN3,OUT3,IP3,RP3,DP3,BP3,SP3)
      END DO

C     Destructor calls
      IP1(2) = 2
      IP2(2) = 2
      IP3(2) = 2
      IP4(2) = 2
      CALL FB0024(IN1,OUT1,IP1,RP1,DP1,BP1,SP1)
      CALL EM0018(IN2,OUT2,IP2,RP2,DP2,BP2,SP2)
      CALL FB0044(IN3,OUT3,IP3,RP3,DP3,BP3,SP3)
      CALL EM0016(IN4,OUT4,IP4,RP4,DP4,BP4,SP4)

      STOP
      END
```

This is the plot of the daily radiation time series for Stuttgart, Germany.



13.6 The C++ class `CinselBlock`

C/C++ programmers may prefer to use the wrapper class `CinselBlock` instead of directly interfering with the INSEL blocks. The class is exported by `inselTools.dll`, hence the code needs to be linked with `inselTools.lib` as usual. We show an example, how the wrapper class can be used to call a single INSEL block – the attenuator block ATT.

```
#include <stdio.h>
#include <windows.h>
#include "CinselBlock.h"

extern "C" void __stdcall LOSOTXT(_int32* dummy,
   char Text[80], unsigned int len = 80);

void main()
{
   // Initialise INSEL message output
   _int32 Fenster = 0;
   char Text[80];
   LOSOTXT(&Fenster,Text);

   // Create INSEL block
   CinselBlock myATT("c:/Programme/inselDi/inselFB.dll",
                     "fb0006",  // (1) ROOT, (2) GAIN, (3) ATT
                     1,         // Inputs
                     1,         // Outputs
                     10,        // INTEGER          parameters
                     0,         // REAL             parameters
                     0,         // DOUBLE PRECISION parameters
                     1,         // Block            parameters
```

```
                        0          // String           parameters
                       );
   int iRC = 0;                    // Return code
   int i;
                                   // Set required variables
   myATT.setOperationMode(3);      // ATT block
 //myATT.setBP(1,0);               // BP(1) = 0 gives an error message
   myATT.setBP(1,2);               // BP(1) = 2 is allright

   myATT.callBlock(1);             // Constructor call
   iRC = myATT.getIP(1);           // Get return code
   if (iRC != 0)
   {
      sprintf(Text,"Return code IP(1) = %d\n",iRC);
      LOSOTXT(&Fenster,Text);
      return;
   }
   else
   {
      for (i = 0; i<= 10; i++)
      {
         myATT.setIN(1,i);         // IN(1)
         myATT.callBlock(0);       // Standard call
         sprintf(Text," %d / BP(1) =  %f\n",i,myATT.getOutput(1));
         LOSOTXT(&Fenster,Text);
      }
   }
   myATT.callBlock(2);             // Destructor call
}
```

The include file `CinselBlock.h` contains the prototypes of all members of the `CinselBlock` class and will be discussed in a minute.

C++ constructor

After the initialisation of the INSEL message system the constructor of `CinselBlock` creates an instance named `myATT` of the attenuator block. The call allocates the complete memory of the ATT block. Its parameters are the full path to the DLL which contains the routine with the ATT block. Please observe, that it is no longer necessary to statically link the library `inselFB.lib`. The constructor call loads the library dynamically.

One vs. zero

Since fb0006 contains more than one block (the source code of fb0006 has been presented on page **??**ff.) the operation mode has to be set to the value three for the ATT block. This is accomplished by the function `myATT.setOperationMode`. The next statement sets the first block parameter to a value of two before the block is called in Constructor call. Please notice, that the `CinselBlock` class uses the index one for the first block parameter – and not zero, as is the usual habit in C/C++.

The rest of the code should be self explaining. We used the `LOSOTXT` routine for textual output. This is better than just a printf output but in a professional project the MSG routine as discussed in section 11.3.2 is the better choice.

CinselBlock.h The complete header file is this:

```
typedef char (*STRARRAY)[80];
typedef UINT (__cdecl *LPFNDLLFUNC)
             (float*, float*, int*, float*,
              double*, float*, STRARRAY, unsigned int);
enum CallMode
{
   ConstructorCall = 1,
   DestructorCall  = 2,
   StandardCall    = 0
};
class __declspec(dllexport) CinselBlock
{
public:
   CinselBlock(char DLLName[], char FName[], int nIN, int nOUT,
      int nIP, int nRP, int nDP, int nBP, int nSP);
   ~CinselBlock(void);
   HINSTANCE   m_hDLL; // Handle to DLL
   int    setIN(int iIndex, float Value);
   int    setBP(int iIndex, float Value);
   int    setIP(int iIndex, int Value);
   int    setSP(int iIndex, char czText[80]);
   int    setINArray(float Value[]);
   int    setBPArray(float Value[]);
   int    setOperationMode(int Value);
   int    setNumberOfUserInput(int Value);
   float  getOutput(int iIndex);
   float  getBP(int iIndex);
   float  getRP(int iIndex);
   double getDP(int iIndex);
   int    getIP(int iIndex);
   int    getOutputArray(float Value[]);
   int    getRPArray(float Value[]);
   int    callBlock(void);
   int    callBlock(int iCallMode);
          // Parses a file for BP's and generates an appropiate array.
          // Return value is the array size.
   int    SetBPfromFile(char szFileName[]);
   int    SetBPfromFile(char szFileName[], int iStartPos);
private:
   LPFNDLLFUNC m_UserBlock; // Function pointer
   int      m_nIn;
   int      m_nOut;
   int      m_nRP;
   int      m_nBP;
   float*   m_pIN;
   float*   m_pOUT;
   int*     m_pIP;
   float*   m_pRP;
   double*  m_pDP;
   float*   m_pBP;
   STRARRAY m_pSP;
};
```

Exercise 13.5 It should now be clear, how the wrapper class `CinselBlock` can be used to solve more advanced applications. Maybe you like to realize the example with the daily radiation data generation with it.

Solution

```
#include <stdio.h>
#include <windows.h>
#include "CinselBlock.h"

extern "C" void __stdcall LOSOTXT(_int32* dummy,
   char Text[80], unsigned int len = 80);

void main()
{
   // Initialise INSEL message output
   _int32 Fenster = 0;
   char Text[80];
   LOSOTXT(&Fenster,Text);

   // Create INSEL blocks
   // CLOCK (fb0024), MTM (em0018), PLOT (fb0044), and GENGD (em0016)

   CinselBlock myCLOCK("c:/Programme/inselDi/inselFB.dll","fb0024",
                       0,         // Inputs
                       6,         // Outputs
                      34,         // INTEGER          parameters
                       1,         // REAL             parameters
                       0,         // DOUBLE PRECISION parameters
                      13,         // Block            parameters
                       1          // String           parameters
                      );
   CinselBlock myMTM("c:/Programme/inselDi/inselEM.dll","em0018",
                       1,         // Inputs
                       9,         // Outputs
                      13,         // INTEGER          parameters
                      95,         // REAL             parameters
                       0,         // DOUBLE PRECISION parameters
                       6,         // Block            parameters
                       1          // String           parameters
                      );
   CinselBlock myPLOT("c:/Programme/inselDi/inselFB.dll","fb0044",
                       2,         // Inputs
                       0,         // Outputs
                      13,         // INTEGER          parameters
                       1,         // REAL             parameters
                       0,         // DOUBLE PRECISION parameters
                       1,         // Block            parameters
                       0          // String           parameters
                      );
   CinselBlock myGENGD("c:/Programme/inselDi/inselEM.dll","em0016",
                       4,         // Inputs
                       1,         // Outputs
                      18,         // INTEGER          parameters
```

```
                  128,          // REAL             parameters
                    0,          // DOUBLE PRECISION parameters
                    9,          // Block            parameters
                    0           // String           parameters
                   );
int iRC = 0;                    // Return code
int i;

myCLOCK.setBP( 1,2006.0);       // BP( 1) = Start year
myCLOCK.setBP( 2,   1.0);       // BP( 2) = Start month
myCLOCK.setBP( 3,   1.0);       // BP( 3) = Start day
myCLOCK.setBP( 4,   0.0);       // BP( 4) = Start hour
myCLOCK.setBP( 5,   0.0);       // BP( 5) = Start minute
myCLOCK.setBP( 6,   0.0);       // BP( 6) = Start second
myCLOCK.setBP( 7,2007.0);       // BP( 7) = End year
myCLOCK.setBP( 8,   1.0);       // BP( 8) = End month
myCLOCK.setBP( 9,   1.0);       // BP( 9) = End day
myCLOCK.setBP(10,   0.0);       // BP(10) = End hour
myCLOCK.setBP(11,   0.0);       // BP(11) = End minute
myCLOCK.setBP(12,   0.0);       // BP(12) = End second
myCLOCK.setBP(13,   1.0);       // BP(13) = Increment
myCLOCK.setSP( 1,"d");          // SP(1)  = Unit of increment
myCLOCK.callBlock(1);           // Constructor call

iRC = myCLOCK.getIP(1);
if (iRC != 0)
{
   sprintf(Text,"CLOCK constructor call failed");
   LOS0TXT(&Fenster,Text);
   return;
}

myMTM.setSP( 1,"Stuttgart");    // SP(1)  = Location
myMTM.callBlock(1);             // Constructor call

iRC = myMTM.getIP(1);
if (iRC != 0)
{
   sprintf(Text,"MTM constructor call failed");
   LOS0TXT(&Fenster,Text);
   return;
}

myPLOT.setOperationMode(1);     // PLOT block (not PLOTP)
myPLOT.setNumberOfUserInput(2);
myPLOT.setBP(1,1.0);            // BP(1) = Mode
myPLOT.callBlock(1);            // Constructor call

iRC = myPLOT.getIP(1);
if (iRC != 0)
{
   sprintf(Text,"PLOT constructor call failed");
   LOS0TXT(&Fenster,Text);
   return;
```

```
   }

   myGENGD.setBP(1,1.0);          // Model:           Use the Gordon Reddy model
   myGENGD.setBP(2,myMTM.getRP(73)); // Latitude:     Is available from MTM
   myGENGD.setBP(3,myMTM.getRP(74)); // Longitude:    Dito
   myGENGD.setBP(4,23.0);         // Time zone:       We know it, definitely
   myGENGD.setBP(5,1.0);          // Variance factor: Recommended default
   myGENGD.setBP(6,0.0);          // No year-to-year variability
   myGENGD.setBP(7,0.3);          // Recommended default
   myGENGD.setBP(8,0.171);        // Recommended default = 0.57 * BP4(7)
   myGENGD.setBP(9,4711.0);       // Initialisation of the random number generator
   myGENGD.callBlock(1);          // Constructor call
   iRC = myGENGD.getIP(1);
   if (iRC != 0)
   {
      sprintf(Text,"GENGD constructor call failed");
      LOSOTXT(&Fenster,Text);
      return;
   }
   for (i = 0; i<= 365; i++)
   {
      myCLOCK.callBlock(0);      // Standard call
      myMTM.setIN(1,myCLOCK.getOutput(2));
      myMTM.callBlock(0);
      myGENGD.setIN(1,myMTM.getOutput(1));
      myGENGD.setIN(2,myCLOCK.getOutput(1));
      myGENGD.setIN(3,myCLOCK.getOutput(2));
      myGENGD.setIN(4,myCLOCK.getOutput(3));
      myGENGD.callBlock(0);
      myPLOT.setIN(1,(float)i);
      myPLOT.setIN(2,myGENGD.getOutput(1));
      myPLOT.callBlock(0);
   }

   myCLOCK.callBlock(2);          // Destructor call
   myMTM.callBlock(2);            // Destructor call
   myPLOT.callBlock(2);           // Destructor call
   myGENGD.callBlock(2);          // Destructor call
}
```

The graphical output is the same as the one on page 360, of course.

Summary

- You have seen how the Identification call either in Fortran or C/C++ to any INSEL block can be used to find information about the block's memory requirements.
- It has been shown how INSEL blocks can be accessed from scratch with a trivial Fortran program.
- The GENGD block has been used to generate a time series of daily radiation data with a Fortran program which is absolutely independent of the

inselEngine.

- The wrapper class `CinselBlock` has been introduced to program a second C/C++ version of the generation of daily radiation time series.