Pascal and Fortran Systems in Vax/VMS Simnon - A Cookbook

Mårtensson, Bengt

1987

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Mårtensson, B. (1987). *Pascal and Fortran Systems in Vax/VMS Simnon - A Cookbook*. (Technical Reports TFRT-7351). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

CODEN: LUTFD2/(TFRT-7351)/1-011/(1987)

Pascal and Fortran Systems in VAX/VMS Simnon—A Cookbook

Bengt Mårtensson

Department of Automatic Control
Lund Institute of Technology
March 1987

Department of Automatic Control **Lund Institute of Technology** P.O. Box 118 S-221 00 Lund Sweden	*Document name* Report
	Date of issue March 23, 1987
	Document Number CODEN: LUTFD2/(TFRT-7351)/1-011/(1987)
Author(s) Bengt Mårtensson	*Supervisor*
	Sponsoring organisation

Title and subtitle

Pascal and Fortran Systems in VAX/VMS Simnon—A Cookbook

Abstract

It is demonstrated how to write Pascal and Fortran systems in the VAX-VMS version of Simnon. Examples are given. Some hopefully useful hints and comments are given along the way.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title		*ISBN*
Language English	*Number of pages* 11	*Recipient's notes*
Security classification		

Pascal and Fortran Systems in VAX/VMS-Simnon—A Cookbook

Bengt Mårtensson, March 23, 1987

Abstract: It is demonstrated how to write Pascal and Fortran systems in the VAX-VMS version of Simnon. Examples are given. Some hopefully useful hints and comments are given along the way.

1. Introduction

The simulation program Simnon [Åström], [Elmqvist], has its own language for describing systems governed by differential- or difference equations. This is most often a very natural way of describing a dynamical system with no special structure, and mildly complicated equations. It is *not* a programming language, but a language for describing a dynamical system governed by a system of first order differential- and/or difference equations. A Simnon file is not a program but a set of *declarations* of states, variables etc. together with equations for updating these. Note that the concept of *equation* is different from the programming language concept *assignment*. For example: equations commute, assignments do not.

However, there are situations in which you would like the full power of a general purpose programming language for describing a dynamical system. Data structures, in particular arrays of different sorts, can be natural. Possibly file input/output is necessary. It can be desirable to use libraries of Fortran- or Pascal code. Functions and procedures can be needed. It might be necessary to control program flow in conditionals or repetition statements. Etc. etc.

Another situation is when you want to hide the code describing the system, or some variables, from the user. This might be desirable for either technical, commercial, or educational purposes.

All this can be achieved by using external systems in Simnon. Originally, these were Fortran systems. On VAX/VMS however, the parameter passing is similar in different programming languages, which makes it possible to implement external Simnon systems in e.g. Pascal.

This is a report on Pascal- and Fortran systems in VAX/VMS Simnon. It is a "cookbook", intended as a guide, not as a definite reference manual.

It should be stressed out that these possibilities do not necessarily generalize to other versions of Simnon. For example, standard Pascal does not support some elements (such as separate compilation) which are necessary. Simnon for IBM-PC does not support external systems. On the other hand, using VMS it should be possible to write external systems for Simnon using any compiled language that runs under VMS.

In what follow it is assumed that the reader is familiar with Simnon, as described e.g. in [Åström]. The formal, "definite" description of Fortran systems in Simnon, the official Simnon manual [Elmqvist], is painfully out of date, but can be consulted for some details left out here.

Writing external systems for Simnon leaves the user with much more freedom than writing ordinary Simnon systems. The safety-net of checks performed by Simnon is bypassed by necessity. (Simnon stops gracefully if you e.g. try to divide by 0 in a usual system, but (of course) just crashes if you do so in an external system.) Therefore, a more qualified user is required. This report reflects this fact. However, a wizard is by no means required. The user is strongly advised to first get a simple system, e.g. the one used in this report, to work, before attacking the real problem.

Section 2 describes in detail the structure of Pascal systems. This uses a short interface `simdefs.pas` intended to streamline the system. In Section 3 Fortran systems are briefly described. In Section 4 some comments are given. Finally, in Section 5 we summarize "the recipe of the cookbook".

My thanks go to Leif Andersson and Tomas Schönthal who have been very helpful when I found out many of the things which are documented here. I have also got helpful feedback from Ulf Holmberg, Mats Andersson, and Per Olof Olsson.

2. The Structure of a Pascal System

This section will be a demonstration of the template system *template*. We are going to demonstrate a Pascal system equivalent to the Simnon system `template`, shown in Figure 1.

```
continuous system template
state  x  y
der    dx dy
input u

dx = y
dy = -x - 0.1*y + 2*y*u*(1 - x*x - y*y) + 0.5*u

end
```

Figure 1. The Simnon System `template.t`

The file `template.pas` is shown in Figure 2. There is a short interface routine `simdefs.pas` intended to streamline the use of Pascal systems, and to make the Pascal file look more like modern programming. (And to prevent "Fortran programmers write Fortran in any language" as much as possible...) This file, listed in the appendix, should be considered as a glue to be used "but not examined".

The Pascal system is compiled as a `module`, and henceforth no `begin` of a main block is permitted. The file is ended by an odd `end.` as can be seen from the example.

The constant `maxindex` is assigned a positive integer. This corresponds to the maximal index in vectors. (Note that this must be done even if you are not

```
Module Template;

{ This is a template for writing Pascal system in Simnon

Bengt Martensson, 86-10-15 }

const
   maxindex = 10;        { e.g...}

%include 'simdefs'

const
   a = 0.1;
   b = 2;
   c = 0.5;

var
   x,y,dx,dy,x0,y0,u : real;

[Global] procedure Template;

begin
   case destin.ipart of
identification           : Ident('CONT','Template');
declaration              : begin
                              States(x,'x');
                              States(y,'y');
                              Ders(dx,'dx');
                              Ders(dy,'dy');
                              Inits(x0,'x0');
                              Inits(y0,'y0');
                              Inputs(u,'u');
                           end;
constant_assignment      : ;
initial_section          : ;
output_section           : ;
dynamic_section          : begin
                              dx := y;
                              dy := -x - a*y + b*y*u*(1 - x*x - y*y) + c*u;
                           end;
accepted_values_computation
                         : ;
final_computations       : ;
   end;
end;

end.
```

Figure 2. The Pascal file `template.pas`.

using any arrays.) Then the line `%include simdefs` follows, which inserts the file `simdefs.pas` into the compilation unit.

Types, variables and external procedures for Simnon are then declared in the usual manner. The user also has access to the data types

`vector = array [1..maxindex] of real`

`matrix = array [1..maxindex,1..maxindex] of real`

The dynamic memory allocation in standard Pascal is inhibited by declaring these variables outside all procedures. Otherwise, all variables will be lost between successive calls.

The main procedure is of course declared global, and consists of a case statment. The structure follows from the example. The part `identification` is executed when Simnon is started, and also when the `syst` command is given. This section should consist of a call of the procedure `Ident`. The first argument should be either `'CONT'` as in the example, indicating a continuous time system, or `'DISC'`, indicating a discrete time system. The statement shown are equivalent to the Simnon statement `continuous system template`. The part `declaration` is executed when the `syst` command is given. This part ties the Pascal system's variable with Simnon "variables", with the Simnon name contained in the string in the second argument. The available declaration functions are shown in Figure 3.

```
Procedure Ident(stype,sysid: string);
Procedure Tsamp(var v:real; vid: string);
Procedure Inputs(var v:real; vid: string);
Procedure Outputs(var v:real; vid: string);
Procedure States(var v:real; vid: string);
Procedure Inits(var v:real; vid: string);
Procedure Ders(var v:real; vid: string);
Procedure News(var v:real; vid: string);
Procedure Pars(var v:real; vid: string);
Procedure Auxvars(var v:real; vid: string);
Procedure Inputv(var v:vector; n: integer; vid: string);
Procedure Outputv(var v:vector; n: integer; vid: string);
Procedure Initv(var v:vector; n: integer; vid: string);
Procedure Statev(var v:vector; n: integer; vid: string);
Procedure Derv(var v : vector; n : integer; vid: string);
Procedure Newv(var v : vector; n : integer; vid: string);
Procedure Parv(var v : vector; n : integer; vid: string);
Procedure Auxvarv(var v : vector; n : integer; vid: string);
```

Figure 3. Available Simnon-declaration functions. Procedures with names ending with "s" declares scalars, while the ones ending with "v" declares vectors.

`Ident` is used in part `identification`, described above. All the rest are used in part `declaration`. The procedure `Tsamp` declares the sampling variable, corresponding to `tsamp` in the Simnon language. The rest the of procedures declares states, inputs, outputs, initial value variables, derivatives, "new"-variables, parameters and auxiliary variables. Note that all these are available for both scalars (`real`) and `vector`'s. The procedures dealing with scalars have names ending with "s", while the corresponding vector version ends with "v". The use of the vector version follows from the example: Say that `maxindex` = 10, and that `a` is declared as a variable of type `vector`. Then the call `derv(a,'a')` will declare the Simnon state variables `a1`, `a2`,. . .,`a10`.

In all cases, the first argument is the Pascal variable name, and the second argument a string containing the Simnon name of it. This string will be truncated to a length of at most 8 characters. Note that for reasons unknown to this author (probably a bug/feature), the same identifier must not be used for a state variable and its corresponding init-variable. This "feature" is slightly annoying.

Section `constant_assignment` is also executed during the `syst` command. Assignments of `par` and `init` variables should be made here. These are not mandatory, however. The value 0 is automatically assigned to all variables before the call of the section `constant_assignment`.

The part `initial_section` corresponds to the `initial` in the Simnon language. In here should go computations to be performed once before every simulation.

The parts `output_section` and `dynamic_section` are called at every step of the simulation. `output_section` consists of computations of auxiliary variables, output variables and derivatives/new's. Note, however, that inputs to the system are undefined during the execution of `output_section`. The section `dynamic_section` consists of computations of auxiliary variables and derivatives/new's.

The part `accepted_values_computation` consists of computations on "accepted values", i.e. computations in points to be plotted. It is called before every instant Simnon stores anything in the store-file. Time-consuming calculation, not needed for the dynamics- or output-sections, can be put here. (For example, if you want to compute the logarithm of something for plotting purposes, this should preferably go in here, just as probably file i/o.)

The part `final_computations` is called after the simulation is completed. This can be used e.g. for closing external files.

Gluing it together

A Fortran source file `systs.for` is required. See Figure 4 for the concrete example. The variable `NSYSTS` should be put equal to the number of included systems. Adjust the computed `GOTO` statement if necessary. A call to the Pascal system is done as in the `8 CALL TEMPLATE` line.

The Pascal system and `SYSTS` are then linked together with Simnon's object libraries into your own executable version of Simnon. The best way to do this is to create a command file, say `make.com`, which will compile the Pascal file, link it all together using all relevant object libraries, and possibly doing some clean-up. A command file running at the Department of Automatic Control's computer BODE is shown in Figure 5. At another site, consult the person responsible for the installation of Simnon.

```
$ if f$search("simnon.exe") .eqs. "" then goto cont
$ delete simnon.exe;*
$ cont:
$ on error then goto delobj
$ pascal template
$ link/nomap/executable=simnon -
  paclib:simlib/include=(simnon$main,extsub),-
  build:unimpl,-
  use:[]template, -
  use:[]systs, -
  paclib:simlib/library,-
  build:every/options
$ delobj:
$ delete template.obj;*
```

Figure 5. The command file `make.com`

To run your own Simnon version use `invoke simnon`, not `run simnon`. Otherwise, the program will not start.

```
      SUBROUTINE SYSTS
C
      DIMENSION SNAM1(2),SIFIL(2),FUNC1(2)
      COMMON/DESTIN/ISYST,IDUM
      COMMON/NSYSTS/NSYST
      COMMON/NALLOC/NS
      COMMON /SYSAV/ S(10000)
      COMMON/SAVEAR/IS(42)
      COMMON/DEVICE/LKB,LTP,LLP,LDIS,LTO,LPLOT,LXXX,LDK1,LDK2,LDK3,LDK4
C
      DATA SNAM1,FUNC1/4HNOIS,4HE1  ,4HFUNC,4H1   /
      DATA SIFIL/4HIFIL,4HE   /
C
C     SIZE OF INTEGER SAVE AREAS (IS)
C
C     OPTA   14
C     SNOISE  6
C     SDELAY  9
C     SIFILE  5
C     SFUNC   4
C     LOGGER  2
C     STIME   1
C
C      ** VAX ERROR HANDLER FOR FORTRAN SYSTEM IS DECLARED HERE:
      EXTERNAL FORSYSHDL
      CALL LIB$ESTABLISH(FORSYSHDL)
C
C *************** Adjust the line below ************************
      NSYST=8
      NS=10000
C
C **************** Adjust the line below *************************
      GO TO (1,2,3,4,5,6,7,8),ISYST
C
 1    CALL SOPTA(IS,S)
      RETURN
C
 2    CALL SNOISE(SNAM1,IS(15),S)
      RETURN
C
 3    CALL STIME(IS(21),S)
      RETURN
C
 4    CALL SDELAY(IS(22),S)
      RETURN
C
 5    CALL SIFILE(SIFIL,LDK2,IS(31),S)
      RETURN
C
 6    CALL LOGGER(IS(36),S)
      RETURN
C
 7    CALL SFUNC(FUNC1,IS(38),S)
      RETURN
C ***************** Add calls to your systems here as indicated ******
 8    CALL TEMPLATE
      RETURN
C
      END
```

Figure 4. The Fortran file `systs.for`

```
      SUBROUTINE TEMPLATE
C
      COMMON /DESTIN/ IDUM, IPART
C
      GO TO(1,2,3,4,5,6,7,8), IPART
C
 1    CALL IDENT2('CONT    ',8hTEMPLATE)
      RETURN
C
 2    CALL STATE2(X,8hX       )
      CALL STATE2(Y,8hY       )
      CALL DER2(DX,8hDX      )
      CALL DER2(DY,8hDY      )
      CALL INIT2(X0,8hX0      )
      CALL INIT2(Y0,8hY0      )
      CALL INPUT2(U,8hU       )
      RETURN
C
 3    RETURN
C
 4    RETURN
C
 5    RETURN
C
 6    DX = Y
      DY = -X - 0.1*Y + 2*Y*U*(1 - X*X - Y*Y) + 0.5*U
      RETURN
C
 7    RETURN
 8    RETURN
C
      END
```

Figure 6. The Fortran file `template.for`

3. Fortran Systems

This section is more briefly written than the other sections. Figure 6 shows the Fortran System file `triode.for`. This Fortran system differs from the previous Pascal system in the following ways:

1 Note the general structure, the `SUBROUTINE` statement, the `COMMON` block and the computed `GOTO` statement.

2 The subroutines for declaring (scalar) variables for Simnon has names such as `STATE2`. The full list of names can be found in Figure 7. Note that the identifiers are required to be exactly 8 characters long. We do not comment further upon this, but hope that the example should be enough for the reader familiar with Fortran, together with the description of Pascal systems. For a fuller description, the reader is referred to [Elmqvist]. Also compare the call in `simdefs.pas`, listed in Appendix.

```
IDENT2
TSAMP2
INPUT2
OUTPU2
STATE2
INIT2
DER2
NEW2
PAR2
VAR2
INPUV2
OUTPV2
STATV2
INITV2
DERV2
NEWV2
PARV2
VARV2
```

Figure 7. List of the Fortran Declaration Calls.

4. Some Comments

There is available a global boolean variable `lstop`. Setting this `true` will interrupt the simulation. (This is how the standard system `CTERM` works.)

It is not possible to write a Pascal (or Fortran) system and "control" it with a "controller" with "direct term". In this case, Simnon will complain about "algebraic loop detected", regardless if it is true or not. This is due to a limitation in Simnon's equation sorter. If the problem occurs, a possible solution can be to incorporate other systems in one external system.

It is tempting to call the executable code something else than `simnon.exe`. This will require a more elaborate `invoke` command.

Since the executable code is fairly large ($>$ 500 blocks) and it is fairly easy to generate, it is probably a good idea to let it reside on a non-backed-up area, if you have access to it.

5. Summary: The Recipe

1. Create Pascal-file `system.pas`.
2. Modify `systs.for`.
3. Fortran-compile `systs.for`.
4. Create appropriate command procedure `make.com`, compiling `system.pas`, linking it, cleaning up etc.
5. `$ @make`
6. `$ invoke simnon` (*not* `$ run simnon`.)

References

ÅSTRÖM, K. J. (1985): "A Simnon Tutorial," Report CODEN: LUTFD2/(TFRT-3176)/1–87/(1985), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1975): “SIMNON, An Interactive Simulation Program For Nonlinear Systems,” Report CODEN: LUTFD2/(TFRT-3091)/1–???/(1975), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Appendix. Simdefs.pas

```
{ Simdefs }
{ Interface for Pascal systems in VMS-Simnon }
{ Bengt Martensson, 87-03-19 }

{ The constant MAXINDEX has to be assigned a positive integer before
  %include simdefs}

const
   identification = 1;
   declaration = 2;
   constant_assignment = 3;
   initial_section = 4;
   output_section = 5;
   dynamic_section = 6;
   accepted_values_computation = 7;
   final_computations = 8;

type
   simnonid = packed array [1..8] of char;
   destinrec = record
                  idum,ipart: integer
               end;
   index = 1..maxindex;
   vector = array [index] of real;
   matrix = array [index,index] of real;
   string = varying [255] of char;

var
   destin: [common] destinrec;
   t: [common(time)] real;
   lstop: [common(user)] boolean;

procedure nameinterface(procedure
                        Simnon_declaration(var y : real;s : simnonid);
                        var x : real; the_name : string);
var
   identifier : Simnonid;
begin
   if the_name.length > 8 then the_name.length := 8;
   identifier := the_name;
   Simnon_declaration(x,identifier);
end;

procedure nameinterface_vector(procedure
Simnon_declaration(var vec : vector; m : integer; systname : simnonid);
                   var v : vector; n : integer; the_name : string);
var
   identifier : Simnonid;
begin
   if the_name.length > 8 then the_name.length := 8;
   identifier := the_name;
   Simnon_declaration(v,n,identifier);
end;

[external(ident2)]Procedure Ident$simnon(stype,sysid: simnonid); external;
```

```
[external(tsamp2)]Procedure Tsamp$simnon(var v:real; vid: simnonid); external;
[external(input2)]Procedure Input$simnon(var v:real; vid: simnonid); external;
[external(outpu2)]Procedure Outpu2$simnon(var v:real; vid: simnonid); external;
[external(state2)]Procedure State$simnon(var v:real; vid: simnonid); external;
[external(init2)]Procedure Init$simnon(var v:real; vid: simnonid); external;
[external(Der2)]Procedure Der$simnon(var v:real; vid: simnonid); external;
[external(new2)]Procedure New$simnon(var v:real; vid: simnonid); external;
[external(par2)]Procedure Par$simnon(var v:real; vid: simnonid); external;
[external(var2)] Procedure auxvar$simnon(var v:real; vid: simnonid);external;
[external(inpuv2)]Procedure Inputv$simnon(var v:vector; n: integer;
                                           vid: simnonid); external;
[external(outpv2)]Procedure Outputv$simnon(var v:vector; n: integer;
                                            vid: simnonid); external;
[external(statv2)]Procedure Statev$simnon(var v:vector; n: integer;
                                           vid: simnonid); external;
[external(initv2)]Procedure Initv$simnon(var v:vector; n: integer;
                                          vid: simnonid); external;
[external(Derv2)]Procedure Derv$simnon(var v:vector; n: integer;
                                        vid: simnonid); external;
[external(newv2)]Procedure Newv$simnon(var v:vector; n: integer;
                                        vid: simnonid); external;
[external(parv2)]Procedure Parv$simnon(var v:vector; n: integer;
                                        vid: simnonid); external;
[external(varv2)] Procedure Auxvarv$simnon(var v:vector; n: integer;
                                            vid: simnonid); external;


Procedure Ident(stype,sysid: string);
var
   timeid,nameid : simnonid;
begin
   timeid := stype;
   nameid := sysid;
   Ident$simnon(timeid,sysid);
end;

Procedure Tsamp(var v:real; vid: string);
begin
   nameinterface(Tsamp$simnon,v,vid);
end;

Procedure Inputs(var v:real; vid: string);
begin
   nameinterface(Input$simnon,v,vid);
end;

Procedure Outputs(var v:real; vid: string);
begin
   nameinterface(Outpu2$simnon,v,vid);
end;

Procedure States(var v:real; vid: string);
begin
   nameinterface(State$simnon,v,vid);
end;

Procedure Inits(var v:real; vid: string);
begin
   nameinterface(Init$simnon,v,vid);
end;

Procedure Ders(var v:real; vid: string);
begin
```

```
   nameinterface(Der$simnon,v,vid);
end;

Procedure News(var v:real; vid: string);
begin
   nameinterface(New$simnon,v,vid);
end;

Procedure Pars(var v:real; vid: string);
begin
   nameinterface(Par$simnon,v,vid);
end;

Procedure Auxvars(var v:real; vid: string);
begin
   nameinterface(Auxvar$simnon,v,vid);
end;

Procedure Inputv(var v:vector; n: integer; vid: string);
begin
   nameinterface_vector(Inputv$simnon,v,n,vid);
end;

Procedure Outputv(var v:vector; n: integer; vid: string);
begin
   nameinterface_vector(Outputv$simnon,v,n,vid);
end;

Procedure Initv(var v:vector; n: integer; vid: string);
begin
   nameinterface_vector(Initv$simnon,v,n,vid);
end;

Procedure Statev(var v:vector; n: integer; vid: string);
begin
   nameinterface_vector(Statev$simnon,v,n,vid);
end;

Procedure Derv(var v : vector; n : integer; vid: string);
begin
   nameinterface_vector(Derv$simnon,v,n,vid);
end;

Procedure Newv(var v : vector; n : integer; vid: string);
begin
   nameinterface_vector(Newv$simnon,v,n,vid);
end;

Procedure Parv(var v : vector; n : integer; vid: string);
begin
   nameinterface_vector(Parv$simnon,v,n,vid);
end;

Procedure Auxvarv(var v : vector; n : integer; vid: string);
begin
   nameinterface_vector(Auxvarv$simnon,v,n,vid);
end;
```