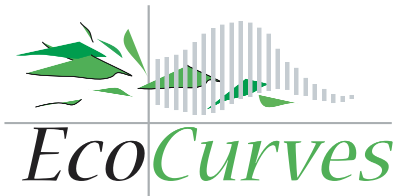
The Fortran Simulation Translator

Version 3: description of new features

C. Rappoldt, D.W.G. van Kraalingen



DRAFT of EcoCurves rapport 7, ISSN 1872-5449

The Fortran Simulation Translator

In opdracht van de leerstoelgroepen Gewas- en Onkruidecologie en Plantaardige Productiesystemen van Wageningen Universiteit en van het International Rice Research Institute in Los Baños, Philippijnen.

The Fortran Simulation Translator

Version 3: description of new features

C. Rappoldt[1], D.W.G. van Kraalingen[2]

[1]EcoCurves, Kamperfoelieweg 17, 9753 ER Haren, Nederland
[2]Alterra, P.O. Box 47, 6700 AA Wageningen, The Netherlands

E-mail: kees.rappoldt@ecocurves.nl

EcoCurves rapport 7

EcoCurves, Haren, 2008

REFERAAT

C. Rappoldt, D.W.G. van Kraalingen, 2008. *The Fortran Simulation Translator* ; *Version 3: description of new features.* EcoCurves rapport 7, EcoCurves, Haren. 56 blz. ; 3 ref.

The FST translator has been updated to allow long variable names and longer program lines. User defined functions are supported and events. Events interrupt the normal simulation process and allow the user to change conditions or system status when some time is reached (time events) or when some situation occurs (state events). After these changes the simulation continues. An example program with events is discussed.

Keywords: model, language, event, crop growth

Dit rapport is beschikbaar als PDF file op www.ecocurves.nl.

© 2008 C. Rappoldt, EcoCurves
Kamperfoelieweg 17, 9753 ER Haren (gn), Nederland
Tel.: (050) 5370979; e–mail: kees.rappoldt@ecocurves.nl

Voorplaat: “Collisions”

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van EcoCurves.

EcoCurves aanvaardt geen aansprakelijkheid voor eventuele schade voortvloeiend uit het gebruik van de FST vertaler en de documentatie in dit rapport.

[EcoCurves April 2008]

Contents

Preface 7

1 Introduction 9

2 FST version 3 11
- 2.1 Syntax . . . 11
- 2.2 Calendar connection in GENERAL mode . . . 12
  - 2.2.1 A problem of the GENERAL mode of FST 2 . . . 12
  - 2.2.2 The solution . . . 12
  - 2.2.3 Calendar connection in combination with WEATHER . . . 13
  - 2.2.4 The available calendar variables . . . 13
  - 2.2.5 Referring to StartYear, StartDOY and OneDay . . . 14
- 2.3 New intrinsic functions . . . 14
  - 2.3.1 The intrinsic function SimulationTime . . . 14
  - 2.3.2 The intrinsic functions SUM and DOT_PRODUCT . . . 15
  - 2.3.3 Other new intrinsic functions . . . 15
- 2.4 String arguments of subroutines and functions . . . 15
- 2.5 User defined functions . . . 16
- 2.6 Appended Fortran subprograms . . . 16
  - 2.6.1 Number of subroutine and function arguments . . . 17
  - 2.6.2 What does the translator do with Fortran? . . . 18
  - 2.6.3 The Fixed/Free form . . . 18
- 2.7 Some other changes . . . 19

3 Time and state events 21
- 3.1 Setting variables . . . 21
- 3.2 Events by example . . . 22
- 3.3 Event sections: the rules . . . 24
- 3.4 Reaching a state event . . . 25
  - 3.4.1 General mode . . . 26
  - 3.4.2 FSE mode . . . 26
  - 3.4.3 Scaling the value of the event function . . . 27
  - 3.4.4 Missed state events . . . 27
- 3.5 Simultaneous events . . . 27

4 Example 29
- 4.1 FST model Particle1.fst . . . 29
  - 4.1.1 A nice surprise! . . . 31
- 4.2 Explanation of the model . . . 31
  - 4.2.1 INITIAL . . . 31
  - 4.2.2 DYNAMIC . . . 31
  - 4.2.3 TERMINAL . . . 32
- 4.3 Inspecting the logfile . . . 32

4.4 Connecting the calendar: Particle2.fst 33
4.5 In FSE mode: Particle3.fst 34
4.6 Comments on this program 35
4.7 The generated Fortran code 36
4.8 Adding events to existing FSE models 36

5 Installation **37**
5.1 FSTwin on a PC 37
5.2 The FST object libraries 37
5.2.1 Drivers 38
5.2.2 Weather 38
5.2.3 TTutil 38

Bibliography **39**

Appendix A General mode Fortran **42**

Appendix B FSE mode Fortran **49**

Preface

We wrote FST, the Fortran Simulation Translator, about 20 years ago as a tool for our teacher in Theoretical Production Ecology, the late professor C.T de Wit. It translates the statements of a simulation language into Fortran. The translator quickly gained popularity among Wageningen crop growth modellers.

The translator is most often used in combination with a modelling environment called FSTwin that provides the user with an editor, calls version 2 of the translator, calls the Compaq Visual Fortran compiler, calls the linker, executes model.exe and finally allows the user to see results through a charting tool or by inspecting tables. This environment now exists about 13 years and has not been changed since.

The update described in this report addresses two problems. The first problem is that the Compaq compiler is not sold anymore, which has been solved by adapting the FSTwin environment to the Intel Visual Fortran compiler. The second problem is that the FST language itself looks a bit archaic with its 6 character acronyms and its maximum line length of 72 characters. This has been solved by a number of adaptions and improvements in the FST translator.

We were also able to finally fulfill an old request: the addition of state and time events. An event interrupts the simulation and takes place when a specified condition is reached. Events allow the user to reset integrals, to change a process setting, to harvest a crop, to change a direction, etc. Events allow more elegant modelling in many situations and we expect that FST users will use them al lot.

The generated Fortran is in free source form Fortran-95. Fortran users may like the new translator for its capability to generate code for event handling, which is not an entirely trivial thing. The source code of the modernized simulation drivers is freely available.

We have done our best to add the new features "in style". This means that a model which passes the translator without error messages is a correct model, at least technically. The translator will remain to be freely available and we intend to maintain it as good as we can and as long as there is a need to do so. We thank Jan Goudriaan, Peter Leffelaar, Gon van Laar and Herman van Keulen for their enthusiasm and support.

Haren / Wageningen, October 2008
Kees Rappoldt, Daniel van Kraalingen

Chapter 1

Introduction

The FST translator translates a completely specified simulation model into a Fortran-90 program with datafiles. The datafiles contain the values for the model parameters and the Fortran program contains a file input section for reading the model parameters from file. There is also a so called rerun file, which specifies parameter values for which the model run has to be repeated.

This report does not contain a full description of FST, you can find that in the existing documentation of Rappoldt & van Kraalingen (1996). Here we describe extensions and new features introduced with version 3 of the FST simulation language. In Chapter 2 we describe the new and more generous syntax rules, and a number of smaller additions, the most important of these probably being the concept of a calendar connection.

In Chapter 3 we introduce the use of state and time events. A time event is a change in the system which takes place at a specified moment in time. Something is added to the system, for instance, a process is activated or the value of a parameter suddenly changes. Another useful application of time events is a simple one: a new day begins and we want to reset an integral which contains a daily total.

A state event allows the same kind of sudden changes but it does not happen at a preset moment in time, but whenever a certain condition is met. Harvest takes place, for instance, when a crop reaches a certain stage. Or the velocity of an object reverses when its coordinate reaches a certain value (the object hits a wall and bounces back).

In Chapter 4 an example program with state events is extensively discussed. The form and style of the generated Fortran-90 can be inspected in the Appendices. In Chapter 5 technical information can be found with respect to supported platforms and Fortran compilers.

Chapter 2

FST version 3

2.1 Syntax

The following changes have been made to the syntax checking procedures of the translator:

> The length of variable names has been increased to 31 characters, in accordance with the Fortran standard (Metcalf *et al.*, 2004; Chapman, 2008).
>
> Warnings on the use of lowercase characters have been removed from the translator. Character case is not significant, however. This means that the same variable may occur in the program in various combinations of lowercase and uppercase characters.
>
> A program line may be up to 132 characters long, including the continuation code "`...`", which has been left unchanged.
>
> Statements with a asterisk "`*`" at the first position or statements beginning with an exclamation mark "`!`" in any position are treated as comment statements. Note that inline comments following an FST statement are still not possible.

The use of lowercase characters is a matter of taste and style. Lowercase characters allow names as "VelocityX" or "MolarVolume" which are usually considered to be more readable than names like MOLARVOLUME. It may be a good idea to keep the FST keywords themselves in uppercase, which leads to statements like

```
! example of the use of lowercase variable names
  CONSTANT MolarVolume = 22.4
  PARAMETER Height = 2.0 ; Width = 3.0
  INCON InitialVelocityX = 3.4 ; InitialVelocityY = 5.6
```

Assignments in the generated Fortran will contain the variable names as they appear in the FST calculation statements. The first occurrence of a variable in FST is used in the generation of declarations, datafiles and variable listings. An exception is the use of FUNCTION names in AFGEN function calls. For technical reasons, these names appear in uppercase in the generated Fortran.

2.2 Calendar connection in GENERAL mode

2.2.1 A problem of the GENERAL mode of FST 2

The FSE mode of the translator is most often used for crop simulation. The FSE mode requires the day as unit of time and all rates of change to be expressed as amounts per day. The FSE mode also requires the specification of WEATHER data.

The TRANSLATION_GENERAL mode of FST does not imply a certain unit of time. Equations may be dimensionless or in convenient time units, depending on the problem.

The addition of WEATHER is possible, also in GENERAL mode, by just specifying in a WEATHER statement a country, a station and a year. *By doing just that*, however, the start time STTIME suddenly becomes a Day-Of-Year value (between 1.000 and 366.000 for a non-leap year), and the unit of simulated time must be a day.

Hence, in TRANSLATION_GENERAL mode, the possibility of choosing a convenient unit of simulated time and convenient values of STTIME and FINTIM disappears as soon as a WEATHER statement is used.

2.2.2 The solution

In FST 3, a better method exists for coupling a calendar to the simulation in GENERAL mode. Three additional TRANSLATION_GENERAL variables are introduced:

> **StartYear**. The year at which the simulation starts.
>
> **StartDOY**. The Day-Of-Year (between 1.0000 and 366.0000 for a non-leap year) at which the simulation starts.
>
> **OneDay**. The length of a day in model time units. For example, "OneDay=86400.0" means the model is in seconds, "OneDay=1440.0" means the model in in minutes, "OneDay=24.0" means hours and "OneDay=0.1" means decades as the unit of time of the model equations.

The combination of StartYear and StartDOY defines the start time as a calendar time. The FST translator identifies the value of STTIME with this calendar time and from that moment on, the simulated Time (starting at STTIME and ending at FINTIM), is connected to the calendar using the value of OneDay. Hence, by defining these three TRANSLATION_GENERAL variables, the *simulated time becomes connected to the calendar time.*

The advantage of this method over the old method of FSE 2 is that the user is free to choose convenient values of STTIME and FINTIM, needed for other aspects of the model. The choice is independent of the simulated calendar interval and there is no implied unit of time.

Another advantage is that a calendar connection does not require a WEATHER statement anymore. Using subroutines, other types of input data become possible. A WEATHER statement is still valid, however.

... DRAFT ...

2.2.3 Calendar connection in combination with WEATHER

Like in FST 2, weather data are made available through a WEATHER statement. There is just one problem: The start year StartYear is also defined by the WEATHER variable IYEAR. Therefore, a program defining a calendar connection with StartYear, StartDOY and OneDay, must not contain IYEAR anymore.

Without the three new variables, however, the old method still works. A WEATHER statement implies “StartYear=IYEAR”, “StartDOY=STTIME” and “OneDay=1.0” and the generated datafile TIMER.DAT contains just that.

2.2.4 The available calendar variables

A calendar connection in GENERAL mode, either explicitly (by means of the new TRANSLATION_GENERAL variables), or implicitly (by WEATHER use) makes available the following variables as “driver-supplied variables”:

iDOY The current Day-Of-Year as an integer variable, in the range $[1, \ldots, 365]$ for a non-leap year and $[1, \ldots, 366]$ for a leap year.

DOY The current Day-Of-Year as a real number (with a fractional part).

Year The current year number as a real variable.

ClockTime The clocktime as a real value between 0.0 and 23.99999.

SimDays Simulated time sofar, a real value in days (with a fractional part).

iHourOfDay The hour number of the day, as an integer value in the range $[1, \ldots, 24]$.

FractSec Fractional seconds reading of a digital clock as a real number in $[0.00, \ldots, 1.00)$.

ClockSec The integer seconds reading of a digital clock, in $[0, \ldots, 59]$.

ClockMin The integer minutes reading of a digital clock, in $[0, \ldots, 59]$.

ClockHour The integer hours reading of a digital clock, in $[0, \ldots, 23]$.

ClockDay The calendar day as an integer, in $[0, \ldots, 31]$.

ClockMonth The calendar month as an integer value, in $[1, \ldots, 12]$.

ClockYear The integer year reading of a digital clock, in value equal to the variable Year.

Although the variables DOY, Year, ClockTime, SimDays and FractSec are single precision real values, the internal timing calculations of the simulation driver take place in double precision arithmetic. The driver-supplied clock variables are therefore accurate, also in case of a simulated time interval of thousands of days.

From these variables only iDOY, DOY and Year are available in FSE mode[1]. In FSE mode the calendar is connected by means of the required WEATHER statement.

[1] The WEATHER variable iYear is the start year of the simulation and cannot be referenced in an FST model. The current year of the simulation is available as the REAL variable Year, which may be converted to an integer by NINT(Year). In general mode the integer variable ClockYear can be used.

... DRAFT ...

Note that the integer calendar variables can directly be used to select an array element by means of the ELEMNT function, like in

```
! example of the use of an integer calendar variable
  DECLARATIONS
  ARRAY Values
  ...
  MODEL
  PARAMETER MonthValues(1:6) = 2.0 ; MonthValues(6:N) = 3.0
  INITIAL
  ARRAY_SIZE N=12
  X = ELEMNT (Values, ClockMonth)
  ...
  END
```

The ELEMNT function is an intrinsic FST array function described in Rappoldt & van Kraalingen (1996, section 4.3.4.2).

2.2.5 Referring to StartYear, StartDOY and OneDay

The new control variables StartYear, StartDOY and OneDay can be referenced in calculations. The first two are integer variables and OneDay is a real variable. The use of these variables requires an explicit definition in a TRANSLATION_GENERAL statement, however. An implicit calendar connection with just a WEATHER statement, does not allow references to StartYear, StartDOY and OneDay.

This limitation should not be a problem in practice since a calendar connection with WEATHER is most likely to occur in older FST models, in which the new variables are not used anyway. For newly written programs, an explicit connection is the preferred method.

2.3 New intrinsic functions

2.3.1 The intrinsic function SimulationTime

The intrinsic function SimulationTime converts a Date/Time specification into a value of the simulation time between STTIME and FINTIM. This clearly requires a calendar connection, either explicitly by setting StartYear, StartDOY and OneDay in TRANSLATION_GENERAL mode, or implicitly by means of the WEATHER statement in FSE mode[2].

The arguments of SimulationTime are six *integer* values, variables or expressions, Year, Month, Day, Hour, Minute, Second, in this order. For example, if the time interval between Jan-15 and June-1 of the start year is needed in a simulation as variable DeltaT, this variable can be calculated in the INITIAL section as

```
! example of SimulationTime function call
  deltaT = SimulationTime(iYear,6,1,0,0,0) - SimulationTime(iYear,1,15,0,0,0)
```

[2] In TRANSLATION_FSE mode the start year is given as IYEAR in the required WEATHER statement, the value of STTIME is the start value of the calendar DOY (Day Of Year) and the unit of time is always one day.

... DRAFT ...

The value of DeltaT will be in days in TRANSLATION_FSE mode, but DeltaT will be in other time units in TRANSLATION_GENERAL mode, depending on the value of OneDay in the calendar connection. And of course, in leap years the period will be one day longer since February, 29 lies within the specified period.

A more interesting application of the function SimulationTime is the calculation of the time of a time event (see Section 3). The statement below calculates the first event time as 10-May-2008 13:14:17 plus an additional X time units.

```
! a calculated time in a FirstTime statement
  EVENT
    FirstTime SimulationTime(2008,5,15,13,14,17) + X
    ...
  ENDEVENT
```

By means of a trick an event date may be made a parameter of the model. For just a date (and a fixed time 00:00:00, say) we need three numbers. A parameter array Date is declared with array size 3. The three elements are converted into integer numbers in the SimulationTime call. Here is the code:

```
! a date as a model parameter
ARRAY Date(1:ND)
...
ARRAY_SIZE ND=3
PARAMETER Date(1:2)=2008.0, 5.0 ; Date(3:ND)=15.0
...
EVENT
  FirstTime SimulationTime(NINT(Date(1)),NINT(Date(2)),NINT(Date(3)), 0,0,0)
  ...
```

This construction allows reruns on calendar dates at which an event takes place!

2.3.2 The intrinsic functions SUM and DOT_PRODUCT

The Fortran-95 intrinsic functions SUM and DOT_PRODUCT (Metcalf *et al.*, 2004; Chapman, 2008) have been added to the list of supported Fortran intrinsics. SUM accepts a single array argument and DOT_PRODUCT requires two arguments declared as arrays with identical upper and lower bounds.

Explicit array bounds in calls to SUM and DOT_PRODUCT are not supported by FST, however. If you want that, you have to use the FST intrinsic functions ARSUMM and ARIMPR respectively. SUM and DOT_PRODUCT have been added for increased speed when the entire arrays need to be summed or multiplied.

2.3.3 Other new intrinsic functions

Other additions to the list of supported Fortran intrinsic functions are CEILING, FLOOR, AMAX0, AMAX1, AMIN0, AMIN1 and FLOAT. The definition of these functions can be found in descriptions of the Fortran language.

2.4 String arguments of subroutines and functions

Fortran subroutines could always be called from FST for doing standardized calculations or for calculations which are impossible, inefficient or clumsy in the FST

. . . DRAFT . . .

language itself. The arguments of a Fortran subroutine or function (see next section) should be declared using a declaration statement in the DECLARATIONS section of the program.

The declaration allows the translator to determine which of the actual arguments in the call(s) are defined in the subroutine and which ones are merely used. This is important for determining the order of the calculations[3].

In FST 3, the string constant STRING is introduced as an input argument type, in addition to integer, real scalar, and real array input. Only string constants are allowed, e.g. `'For calculating A'`. String expressions or string variables are not allowed in FST. String arguments allow subprograms to generate meaningful messages if something goes wrong. An example is given in the next section 2.5.

2.5 User defined functions

User-defined functions are treated in the same way as subroutines. Function calls can be written directly in calculation statements, which may lead to more elegant programs than the use of subroutines in combination with intermediate variables. Limitations, however, are that functions must return a single precision, scalar, real variable and that all function arguments must be input arguments.

An example is the declaration of a function MichaelisMenten in the following way:

```
! example of the use of lowercase variable names
  DECLARATIONS
  DEFINE_FUNCTION MichaelisMenten (STRING, INPUT, INPUT, INPUT)
  ...
  MODEL
  Xloss = MichaelisMenten ('Reaction 1', Vmax1, K1, X) + ...
          MichaelisMenten ('Reaction 2', Vmax2, K2, X)
  ...
  END
```

The DEFINE_FUNCTION statement declares four input arguments, a string constant and three real values, variables or function calls. In the calls to MichaelisMenten you see that the real arguments are the Vmax, the "half rate concentration" K and the substrate concentration X. Note that the same Michaelis-Menten expression (in the actual Fortran function) is used for two different reactions.

The string constant in the calls can be used by the function itself for meaningful messages in case there is something wrong. In case of negative concentrations X, for instance, the function might force the program to stop, but without further information the user does not even know which function call *and which concentration* caused the problem. This is solved by including the string in an error message written from the function.

2.6 Appended Fortran subprograms

Fortran subroutines and functions can be appended to the FST model. In principle, the translator copies them to the generated Fortran file and leaves it to the compiler to check the Fortran code. However, the appended Fortran must be either in free

[3] In the generated Fortran (not in FST), a variable must be defined before it can be used.

... DRAFT ...

source form or in fixed source form (Metcalf *et al.*, 2004; Chapman, 2008). A mixture of the two is not allowed and would not make sense since Fortran compilers do not like a mixed form file either.

This implies that the translator has to decide in which of the two forms the Fortran code was written. The translator may fail to do this properly, unless the FST user knows the rules of the game. So what are the rules?

- The Fortran source form is determined using the first non-empty line after the STOP statement.
- If this line begins with an exclamation mark '`!`' (at any position), the source form is set to free. The line is considered as a Fortran-95 style comment statement.
- If the line begins with a asterisk '`*`', a '`c`' or a '`C`' at the first position, the source form is set to fixed. The line is considered as a Fortran-77 style comment statement.
- Otherwise, if the first character is at position 7 or later, the source form is assumed to be fixed. If the position of the first character is in [1,6], the source form is assumed to be free.

For the FST part of the model, the use of an asterisk or exclamation mark in comment lines does not matter. After the STOP statement, however, you should be aware of the difference.

Sometimes these simple rules may lead to a surprise, for instance, if your first Fortran-95 subroutine happens to start at position 7, the source form is classified as fixed. Problems are easily solved by adding a comment in the proper source form, on top of the Fortran code.

Once the source form is determined, all statements, continuation lines and comment statements should conform to *either the fixed or the free* source form. If the subprograms are in fixed source form, for instance, exclamation marks as comment characters are not allowed. If your subprograms are in free source form, the Fortran-77 method of statement continuation or the asterisk "`*`" at the beginning of a comment line is not allowed.

Note that in the Fortran 95/2003 standard, the fixed source is an obsolescent feature of the language, a candidate for deletion in future versions of the standard.

2.6.1 Number of subroutine and function arguments

The translator checks the number of arguments in SUBROUTINE and FUNCTION statements against the FST declaration of the subprogram. It does so by finding the first words of non-comment statements. If the first word happens to be SUBROUTINE or FUNCTION, the arguments are counted.

Note that this will not always work for functions, since functions may begin in Fortran like "REAL FUNCTION MichaelisMenten". In this case, the function statement will not be found and the number of arguments will not be verified. Verification is also impossible if the called Fortran programs are separately compiled and then linked with the model, or if they are part of a precompiled object library. Hence, the user must always be aware of possible problems with the argument list of linked subprograms. Such problems usually lead to a runtime crash.

... DRAFT ...

What is always verified, however, is the consistency between the DEFINE_CALL or DEFINE_FUNCTION declaration in FST and the actual calls.

We further remark that FST does not support the use of keyword arguments or optional arguments in subprogram calls[4]. This implies that Fortran-95 subprograms requiring such a call cannot be used directly from FST. In such a situation the user will have to write a trivial interface routine that translates the simple Fortran-77 style call from FST into the desired Fortran-95 call. The interface routine then contains a USE statement for a Fortran-95 module or an explicit interface description for the Fortran-95 subprogram.

2.6.2 What does the translator do with Fortran?

Fortran in free source form, is copied to the generated source file completely unchanged. This implies that modern Fortran-95 modules can be appended to FST code without any problem[5].

Fortran in fixed source form, is converted to free source form by changing the continuation and the comment lines. All other characteristics of the old Fortran-77 routines are left untouched as, for instance, the statement begin, and numbers as statement labels. So what you basically get, is free source form Fortran beginning at position 7.

The conversion is necessary since the model itself is generated by the translator in free source form and compilers tend to complain about source form mixtures in the same file.

The user is advised to replace the original Fortran by the converted Fortran copied from Model.f90[6]. Alternatively, Michael Metcalf's program[7] can be used for a more complete conversion.

2.6.3 The Fixed/Free form

There is a source form which can be combined with both free and fixed source form Fortran source files. This is the Fixed/Free source form (Metcalf *et al.*, 2004; Chapman, 2008) with continued lines coded with an ampersand (&) at position 73 *and* any character at position 6 of the continuation line. If your Fortran is in this form, the addition of an exclamation mark comment line on top of the Fortran section will cause the translator to switch to free form. It will otherwise complain about a line length above 72.

The Fixed/Free source form may lead to error messages on continued SUBROUTINE or FUNCTION statements for which the translator attempts to determine the number of arguments. In such a case, the fixed/free form continuation should

[4] In Fortran 95 a subroutine SUB with a dummy argument A can be called using 'call SUB(A=myvar)', where 'myvar' is the actual argument in the calling program. Keyword arguments in calls are especially useful in case of optional arguments, which do not *have to* be there. All this, however, requires the subroutine interface to be known at compile time in the calling program, either through a USE statement or through an explicit interface declaration. Subprogram calls from FST are Fortran-77 style calls without keyword arguments.

[5] But they cannot be called directly from FST, see section 2.6.1.

[6] A Fortran section beginning with a subroutine statement at position 7 may then be incorrectly classified as fixed form. This can be repaired by adding a comment line beginning with an exclamation mark on top of the Fortran code.

[7] Available from many websites, e.g. ftp.numerical.rl.ac.uk/pub/MandR/convert.f90 or http://www.nag.co.uk/nagware/Examples/convert.f90.

... DRAFT ...

be removed from these statements. For other statements the Fixed/Free source form should not be a problem.

2.7 Some other changes

A warning used to be given on the use of a scalar variable as an array argument in a function or subroutine call. The warning told the user that the array length appearing in the subroutine is 1. This situation has been changed into an error condition since (1) In Fortran-95 there is a distinction between degenerate arrays[8] and scalars, (2) Usually it *was* an error (a forgotten array declaration) and (3) Arrays with length 1 can be declared if necessary.

The OUTPUT statement for generating matrix printer plots is no longer maintained. We do not test it anymore, it may not function properly and we intend to completely remove it from the FST language. If there are users who cannot do without, please let us know.

A few subprograms linked with the generated Fortran, have been moved from the utility library TTUTIL to the drivers library. The moved subprograms are TIMER2, INTGRL, INSW, FCNSW, LIMIT, LINT2 and CHKTSK. The files have been converted to Fortran-90 free format and CHKTSK has been adapted to the new event handling ITASK=5 section of FSE models.

Finally, numerous small improvements in the text of error messages and warnings have been made.

[8] A degenerate array is an array with length 1.

... DRAFT ...

CHAPTER 3

Time and state events

Events interrupt the normal simulation cycle of rate calculations and status updates. The simulation is interrupted in order to change something in parameter values or even the system status. After the event, the simulation continues, until possibly another event takes place.

This chapter describes how events can be initiated and what sort of things can be done during an event. There are two types of event. A time event simply takes place at a prescribed moment in simulated time. When the event time is reached the instructions belonging to the event are executed, a new event time may be set, and the simulation continues.

State events are more complicated. A state event takes place when a certain condition is reached, for instance if the state variable A reaches the value 5.0, a state event must happen. In FST, this takes the form of a zero condition equal to $A-5.0$. If this expression becomes (almost) zero, the event takes place. Then, after the "event function" has moved away from zero and becomes zero once more, the event takes place again.

Below a detailed description is given for both event types. At first, however, a new type of variable needs to be introduced, the setting variable.

3.1 Setting variables

A setting variable is defined by means of a SET statement in the following way

```
! example of the use of a setting variable
  DECLARATIONS
  ...
  INITIAL
  SET CumulativeAmount = 0.0
  SET NitrogenContent = 20.0 * MAX(Ncon, 2.0)
  ...
```

The setting variables `CumulativeAmount` and `NitrogenContent` are defined by means of a SET statement. The second example shows that the SET statement is a calculation. It is *not* a value assignment like PARAMETER, INCON or CONSTANT statements, but expressions can be used as if the setting variable was an ordinary calculated variable. Also array expressions can be used if the setting variable is declared as an array before. In fact, any initially calculated variable

can be made into a setting variable by just putting the keyword SETTING (or just SET) in front of the calculation[1].

The following rules apply to setting variables (the keyword SETTING may be used instead of SET).

- The SET or SETTING statement may occur only in the INITIAL section of the FST program. The defining expression may refer to PARAMETERs, CONSTANTs, driver supplied variables, or other initially calculated variables.
- A model setting variable can be defined and used as any other initially calculated variable. This implies that the calculations in the SETTING statement are sorted (put into a computable order) together with the other initial calculations.

There are, however, three differences between a setting variable and an ordinary, initially calculated variable.

1. If a setting variable is listed in a PRINT statement, it is sent to dynamic output, together with the dynamic variables that change during the simulation.
2. A setting variable may change value during an event.
3. A setting variable may act as a rate of change in an INTGRL statement.

In a model without events, setting variables are just initially calculated variables, behaving like dynamic variables with respect to output, but with no other special function.

3.2 Events by example

An event section contains everything which has to be specified about an event, when it takes place and what should happen. The simplest event section is

```
! example of and event section
  DYNAMIC
  ...
  EVENT
     FIRSTTIME StTime + 2.0
  ENDEVENT
```

This initiates a time event at 2.0 time units after start time. During the event, however, nothing happens, it just interrupts the simulation and no new event time is specified. A periodic time event can be initiated by

```
! example of and event section
  DYNAMIC
  PARAMETER Period = 4.0
  ...
  EVENT
     FIRSTTIME StTime + 2.0
     NEXTTIME Time + Period
  ENDEVENT
```

[1]With the exception of variables calculated in a subroutine call. If such a variable, say A, needs to become a setting variable, a help variable (say Help) is first calculated in the subroutine and then assigned to the setting variable by SET A = HELP.

... DRAFT ...

This initiates a series of time events, beginning at 2.0 time units after start time and returning every Period time units thereafter. Still, however, the event does not change anything in the system status.

In the next example this is different. Each time event changes the setting variable named `SetPoint` and resets a state variable StateA to zero.

```
! example of event section
  INCON Aini = 1.2345
  INITIAL
  SET SetPoint = 10.0
  ...
  DYNAMIC
  RateA = ...
  StateA = INTGRL(Aini, RateA)
  ...
  EVENT
     FIRSTTIME StTime + 2.0
     PARAMETER Period = 4.0
     NEXTTIME Time + Period
     NEWVALUE StateA = 0.0
     NEWVALUE SetPoint = -SetPoint
  ENDEVENT
```

The first NEWVALUE statement redefines the state variable StateA at zero. SetPoint is redefined as the opposite of the old SetPoint. Hence, the variable SetPoint is initially +10.0 and then, beginning at 2.0 time units after starttime, changes all the time from +10.0 to −10.0 and back, every Period time units.

State events lack a FIRSTTIME and NEXTTIME statement and instead contain a ZEROCONDITION statement in which an expression is specified that triggers a state event when it becomes (almost) zero. So suppose we want to change the SetPoint variable from the previous example each time the integral StateA reaches the value TOP. This is done with

```
! example of and event section
  INCON Aini = 1.2345
  INITIAL
  SET SetPoint = 10.0
  ...
  DYNAMIC
  RateA = ...
  StateA = INTGRL(Aini, RateA)
  ...
  EVENT
     ZEROCONDITION StateA - Top
     PARAMETER Top = 200.0
     NEWVALUE StateA = 0.0
     NEWVALUE SetPoint = -SetPoint
  ENDEVENT
```

Like in the previous example, the state StateA is the integral of RateA over time, beginning at Aini. But now, if StateA ever reaches 200.0, the event takes place. The setpoint changes into its opposite value and StateA is reset.

Note that events not necessarily take place. In the last example, if StateA never reaches the value Top, the event will never happen.

. . . **DRAFT** . . .

3.3 Event sections: the rules

Most rules are about the references that can be made in event sections to other variables of the model. A FIRSTTIME statement, for instance, should not refer to dynamic variables, since the first time event time must be calculated during the initial phase of the simulation.

The rules of the game:

1. An event section begins with an EVENT statement and it ends with an ENDEVENT statement.
2. An FST program may contain several event sections (actually about 50).
3. An event section is contained in the DYNAMIC section of the model. Its position within the DYNAMIC section is not significant. It does not have any consequences for the way in which the dynamic calculations are sorted and written to the generated Fortran code. Hence, an event section may be put close to the dynamic calculations to which it is naturally related, or event sections may be grouped at the beginning or end of the DYNAMIC section. This is a matter of style and taste.
4. However, when during simulation two or more events occur simultaneously, the order of execution depends on the order of the event sections in the FST model. Details on this can be found in section 3.5.
5. A time event section must contain one and only one FIRSTTIME statement.
6. A FIRSTTIME statement contains a constant or *expression* specifying the first event time as function of initially known variables (PARAMETERs, CONSTANTs), driver supplied variables and initially calculated variables including setting variables.
7. A time event section may further contain a single NEXTTIME statement.
8. A NEXTTIME statement specifies the next time event time as a constant or *expression*. The expression may refer to initially known variables, initially calculated variables, dynamically calculated variables, driver supplied variables and to variables calculated in the same event section where the NEXTTIME statement is in.
9. A state event section must contain one and only one ZEROCONDITION statement.
10. A ZEROCONDITION statement contains a scalar expression which may refer to initially known variables, driver supplied variables and all initially and dynamically calculated variables, including state and setting variables.
11. An event section may contain one or more calculation statements, defining variables which are not defined elsewhere in the model. The expressions, subroutine calls and function calls used may refer to initially known variables, driver-supplied variables and initially or dynamically calculated variables, *and to other calculated variables defined in other calculation statements in the same event section*. Just like initial, dynamic and terminal calculations, the calculations in each event section are sorted by the FST translator.
12. An event section may contain one or more NEWVALUE statements.

. . . DRAFT . . .

13. Each NEWVALUE statement redefines a state variable or a setting variable, which may be either a scalar or array variable.

14. A NEWVALUE definition of a *scalar* (non-array) state or setting may refer to itself (the old value of the state or setting).

15. The NEWVALUE definition of a *array* state or setting cannot refer to itself. Such a calculation requires a help variable, an array with the same length, which is calculated in the event section as a copy of the (old value of) state ot setting array to be changed.

16. A NEWVALUE definition *must not refer to any other state or setting variable which is redefined in the same event section.*

17. NEWVALUE statement may refer to all *other* initially known or dynamically calculated variables, to initially calculated variables, driver supplied variables, or to calculated variables defined in calculation statements in the same event section.

18. Just like the INITIAL, DYNAMIC or TERMINAL section of an FST program an EVENT section may contain statements like PARAMETER, CONSTANT, TIMER. The function of such statements does not depend on their position anywhere between INITIAL and STOP. These statements do not interfere with the functionality of the event section.

Rule number 16 probably requires some clarification. The reason for this rule is that, by allowing such references, the order in which the NEWVALUE instructions are executed would make a difference. The use of “old values” can always be realized by calculating a help variable as a copy of a state variable or setting, and then using the calculated help variable in a NEWVALUE expression.

The order in which the operations specified are carried out is as follows:

1. The sorted calculation statements are executed.

2. The NEWVALUE assignments are executed. Their order is arbitrary (see the rules above).

3. In case of a time event, the NEXTTIME is calculated. This implies that state and setting variables possibly occurring in the NEXTTIME expression refer to *new* values for those states and settings which were just redefined.

4. Finally output variables defined in the event section (by means of calculation statements) are sent to output, accompanied by the time at which the event took place.

This order does not depend on the order of the statements in the event section.

Last but not least we should mention that many different state and time event sections may occur in an FST model. This clearly may lead to complicated model behavior.

3.4 Reaching a state event

Sofar, the way in which the ZEROCONDITION expression is treated has remained a bit vague. The reason is that this depends on the type of simulation carried out.

... DRAFT ...

3.4.1 General mode

In TRANSLATION_GENERAL mode the expression specified in the ZEROCONDITION statement is called the event function.

This function is monitored during the simulation and as soon as it crosses zero (from either side) the time at which the zero crossing occurs is found in an iterative procedure by means of a number of bisection steps.

There clearly is some tolerance involved here. This is the value of SEVTOL (State Event Tolerance), which may be specified in a TRANSLATION_GENERAL statement, but which has a default of value of 1.0E−5. As soon as the event function is within SEVTOL from zero, the event is triggered[2].

If the TRANSLATION_GENERAL control variable TRACE is set to 4, the iterative search is reported to the logfile. Such a report looks like

```
    +  1.04167E-02 -->   658.62       (  0.10417E-01)
    +  1.04167E-02 -->   658.63       (  0.10417E-01)
    +  1.04167E-02 -->   658.64       (  0.10417E-01)

 Step      Time         Event  Event function
 ----      ----         -----  --------------
   0    658.6512044       2      0.45776E-04
   1    658.6459961       2    -0.16153E-03
   2    658.6486003       2    -0.57936E-04
   3    658.6499023       2    -0.60797E-05

    +  9.11458E-03 -->   658.65       (  0.10417E-01)
 Output flag set =====   658.65    (preparing for event)
 Output flag set =====   658.65
 State event  2 at ===   658.65    ( 4 iterations, remaining error -0.60797E-05)
    +  1.04167E-02 -->   658.66       (  0.10417E-01)
    +  1.04167E-02 -->   658.67       (  0.10417E-01)
```

The first lines here show a few regular integration steps. Then the second event function seems to cross zero (in the generated Fortran code, the events get numbers). In four iteration steps the zero is reached with a sufficient accuracy. Then an event-preparing rate call *with output* takes place, sending variables to output at their pre-event values. The actual event call to the model is a rate call with output enabled and with the proper event flags set. The model takes care of the event handling and then does a regular rate calculation with output. After completion, the event is reported once more and the normal simulation cycle is resumed.

A state event cannot occur without zero *crossing* of the event function. This implies that the function must first be at least SEVTOL away from zero and then it may cross zero (again).

3.4.2 FSE mode

The FSE mode of the translator leads to simulations with a fixed time step, often set to one day for crop growth models. In fact there is no continuous time in FSE mode but there are just discrete time steps. In this situation an adapted time step in order to reach precisely an event time would be inconsistent with the approach.

Therefore, in FSE mode, events take place as soon as the event time is reached or passed, or as soon as a zero condition is reached or passed. There is no iterative

[2] SEVTOL may be referenced in expressions.

. . . DRAFT . . .

search for a precise state event time nor a calculated time step in order to precisely reach the preset time of a time event.

This approach is a bit crude. It is the only approach, however, which seems consistent with the fixed steps of the process simulation. Crop harvest, for instance, takes place at a certain day and not at twenty minutes and 10 seconds past four in the afternoon. The same holds for fertilization, weeding or other events that may occur in a simulation of crop growth with a one day time step.

3.4.3 Scaling the value of the event function

The value of SEVTOL is an absolute tolerance. For an event function with values in the order of, say, one million it does not make sense to require an accuracy of 10^{-5}. Then the tolerance SEVTOL may be set to a larger value. Increasing the SEVTOL value, however, will also affect other state events. So, a larger value of SEVTOL is an option only in models containing a single, or a few similar state events.

A better method is to scale the event function, i.e. divide it by a constant in such a way that its values lie at a reasonable distance from zero, for instance in $[-1, +1]$. An example is an event that takes place when some coordinate X reaches the value of (parameter) A. The event function would then be $(X - A)$. If this function becomes very large it is better to write $(X - A)/A$ for the event function or to use the size of the system as a scaling constant, as in $(X - A)/\text{SystemSize}$.

3.4.4 Missed state events

In principle state events can be missed if, during a single time step, an event function crosses zero multiple times. Therefore the time step should be prevented to become too large, especially if the variable time step Runge-Kutta method is used. This can be done using the TRANSLATION_GENERAL variable DELMAX.

3.5 Simultaneous events

This section is relevant only for models with several event sections from which two or more may occur simultaneously. Even in GENERAL mode the time at which a certain state event takes place (found iteratively) may sometimes coincide with a time event or another state event. Here we explain how the generated Fortran program deals with such a situation.

There are two ways of dealing with two simultaneous events. The first method ("Update Once") is to execute the code for both events (event calculations, change state or setting) and then recalculate once all dynamic variables in order to have new and updated rates of change. The second method is "handle event 1", "recalculate dynamic variables", "handle event 2" and "recalculate dynamic variables". This method is referred to as "Insert Updates".

If an update after event 1 does not have any consequences for the calculations or the NEWVALUE statement(s) in event 2, the two methods lead to the same result. The FST translator, however, does not verify such an event independence and it is possible to write an FST model for which the results depend on the way in which simultaneous events are handled. This is different for the two translation modes.

. . . DRAFT . . .

In FSE mode the “Update Once” method is used for all events (state or time). Hence, there is no dynamic update in between the execution of simultaneous events. The order of events is the order of their respective event sections in the FST model.

In GENERAL mode the simulation driver first handles all pending *state events* in a single call to the generated Fortran model. The state events are handled in the order of their respective event sections in the FST model. After execution of all pending state events, the dynamic variables are updated once. Hence, for simultaneous *state events* the “Update Once” method is used[3], just like in FSE mode.

Pending *time events* in GENERAL mode, however, are handled *after all pending state events*. For *time events* the GENERAL simulation driver uses the “Insert Updates” method. Each time event is followed by an update of the dynamic variables[4] and simultaneous time events are handled in the order of the (time) event sections in the FST model.

[3]Given the SEVTOL tolerance, the event functions of the pending state events all cross zero for the current system status. A dynamic update after handling just one of the events (an inserted update) could have implications for the other event functions, for which it was just decided they cross zero. Hence, when several event functions cross zero simultaneously, the driver must assume these events are indeed simultaneous and the events take place without update of the dynamic status in between.

[4]This can be seen as inserting a zero length time step between simultaneous time events.

... DRAFT ...

Chapter 4

Example

Three example FST programs are included in the installation of FST 3:

- **Particle1.FST** implements a model of a “particle“ bouncing in a 2D box. The model uses the TRANSLATION_GENERAL mode of the translator and the times at which the particle hits the walls are iteratively found.
- **Particle2.FST** is the same as Particle1.fst, but a calendar connection is added, just for illustrative purposes. This allows the setting of a time event at a prescribed date and time of the calendar.
- **Particle3.FST** is again the same model, but now translated in FSE mode.

The model is physically described as follows:

- A particle moves around in a two-dimensional box. There is no energy loss through friction, there is no energy stored in spin (rotation of the particle) and collisions with the walls are fully elastic.
- The box itself does not move.
- The motion of the particle is described by the laws of classical mechanics, i.e. Newtons law.
- There is a constant gravity field with a fixed direction.

The independence of the motion in the two directions means that there is a constant time interval between any two collisions in the x direction and another constant interval between the collisions in the y direction. The example model simulates the motion and “measures” the time between two collisions in the x direction (the other direction is left as an exercise).

4.1 FST model Particle1.fst

The model Particle1.fst is given in the following listing. Besides simulating the movement, the program also “measures” the time between two hits in the X direction.

```
0001   DECLARATIONS
0002      DEFINE_FUNCTION TimeBetweenHits (INPUT, INPUT, INPUT, INPUT)
```

```
0003   MODEL
0004      TRANSLATION_GENERAL DRIVER='RKDRIV' ; EPS = 1.0E-6 ; SEVTOL = 1.0E-5 ; TRACE = 4

0005      TIMER STTIME = 0.0 ; FINTIM = 200.0 ; DELT = 0.01 ; PRDEL = 0.5 ; IPFORM = 4
0006      PRINT X, Y, VX, VY, TheoryPeriodX, TheoryPeriodY, Ratio, MeasuredPeriodX

0007   INITIAL
       !  position, velocity and acceleration
0008      INCON  IX = -4.0 ;  IY = -0.5
0009      INCON IVX =  2.0 ; IVY =  1.0
0010      PARAM  AX = +0.1 ;  AY = -0.1

       !  box size
0011      PARAM HalfSizeX = 5.0 ; HalfSizeY = 2.5

       !  initialize measurement
0012      Set Counter      = 0.0
0013      Set ClockRunning = 0.0
0014      Set ClockedTotal = 0.0

0015   DYNAMIC
       !  place
0016      RX = VX
0017      RY = VY
0018      X = INTGRL (IX, RX)
0019      Y = INTGRL (IY, RY)

       !  velocity
0020      VXR = AX
0021      VYR = AY
0022      VX = INTGRL (IVX, VXR)
0023      VY = INTGRL (IVY, VYR)

       !  stop watch
0024      MeasuredTime = INTGRL(zero, ClockRunning)
0025      INCON zero = 0.0

       !  left and right boundary
0026      EVENT
0027         ZeroCondition abs(X) - HalfSizeX
0028         NewValue VX = -VX

       !     switch on the stopwatch
0029         NewValue ClockRunning = 1.0

       !     read the stopwatch ; count the hits
0030         NewValue ClockedTotal = MeasuredTime
0031         NewValue Counter = Counter + 1.0
0032      ENDEVENT

       !  bottom and top
0033      EVENT
0034         ZeroCondition abs(Y) - HalfSizeY
0035         NewValue VY = -VY
0036      ENDEVENT

0037   TERMINAL
0038      MeasuredPeriodX = ClockedTotal / NOTNUL(Counter-1.0)

       !  theoretical time between hits
```

... DRAFT ...

```
0039      TheoryPeriodX = TimeBetweenHits(IX,IVX,AX,HalfSizeX)
0040      TheoryPeriodY = TimeBetweenHits(IY,IVY,AY,HalfSizeY)
0041      Ratio = TheoryPeriodY / TheoryPeriodX
0042   END
0043   STOP

0044   REAL FUNCTION TimeBetweenHits (X, V, A, D)
0045     IMPLICIT NONE
0046     REAL, INTENT(IN) :: X, V, A, D
0047     TimeBetweenHits = (SQRT(V**2+2.0*A*(D-X)) - SQRT(V**2-2.0*A*(D+X))) / A
0048     Return
0049   END FUNCTION TimeBetweenHits
```

4.1.1 A nice surprise!

Execution of this model and plotting the orbit (Y as function of X) reveals unexpected behavior. By chance? Figure it out and have some fun!

4.2 Explanation of the model

4.2.1 INITIAL

At first the initial positions and velocities are set, the box size in two directions[1] and the constant accelerations in the gravity field. Finally, the measurement of the time interval for the x direction is set up.

The "measurement" method is most easily understood by thinking about a stopwatch. The stopwatch is switched on when the first hit takes place and remains on after that. Then, during hits the stopwatch is read and the number of hits is counted. In the terminal section, the stopwatch time and the number of collisions are used to calculate a average measured period MeasuredPeriodX.

The stop watch is switched on by means of the setting variable ClockRunning, which is SET in initial at 0.0 and redefined in the first event (and all events thereafter) as 1.0.

4.2.2 DYNAMIC

Particle position and velocity are calculated by integrating velocity and acceleration in both dimensions. Then the value of the switch ClockRunning is used as a rate of change. This rate of change will be 0.0 before the first event and will be 1.0 after. This implies that MeasuredTime will be equal to the time passed since the first event (in the x direction) took place.

The EVENT sections are easy. When the walls are hit, the velocity in the appropriate direction is reversed. For the "x direction walls", the stopwatch is switched on, the present value of the stopwatch is stored as ClockedTotal and the counter is updated.

[1] PARAMETER and INCON statement need not to be in the INITIAL section.

... DRAFT ...

4.2.3 TERMINAL

The average time interval between two collisions can now be calculated from the last stopwatch reading ClockedTotal and the number of collisions. Note that the number of intervals is one less than the number of collisions.

The time interval between collisions can also be calculated theoretically. For elastic and frictionless bouncing between two walls at positions $-D$ and $+D$ from the origin, for an initial position X, an initial velocity V and a constant acceleration of gravity a, the time ΔT between two collisions can be derived with classical mechanics as

$$\Delta T = \frac{\sqrt{V^2 + 2a(D-X)} - \sqrt{V^2 - 2a(D+X)}}{a}$$

This expression has been used to create the Fortran function TimeBetweenHits which is declared on top of the model and is used to calculate the time intervals for both directions.

4.3 Inspecting the logfile

During development of a model containing events inspection of the logfile[2] is important. The model in section 4.1 contains TRACE=1 (statement 4) and produces a brief initialization report in which the events are mentioned:

```
   Initialization of RKDRIV and user MODEL completed:
         ---------------------------------------
                 Number of states:    5
            Integration starts at:  0.00000
            Output interval PRDEL:  0.50000
               Finish time FINTIM:   200.0
             State event accuracy:  0.10000E-04
                Relative accuracy:  0.10000E-05
                Initial time step:  0.10000E-01
                     Maximum step:   200.0
            Requested state event:   1
            Requested state event:   2
         ---------------------------------------
```

The most complete information on the progress of the simulation is obtained by setting TRACE=4. Every time step is reported and detailed information is given on the state event iteration. With this setting of TRACE the first part of the logfile produced by the model in section 4.1 becomes

```
   RKDRIV: DYNAMIC loop
   ==========
      time step           time
      ---------------------------------
 Output flag set =====   0.00000
    +  1.00000E-02 -->   1.00000E-02 try next   1.58193E-02
    +  1.58193E-02 -->   2.58193E-02 try next   2.17876E-02
    +  2.17876E-02 -->   4.76069E-02 try next   8.71506E-02
```

[2] During model execution a logfile MODEL.LOG is created. In GENERAL mode the content of this file depends on the value of the TRANSLATION_GENERAL variable TRACE, as documented in Rappoldt & van Kraalingen (1996, section 7.5.5).

... DRAFT ...

```
   +  8.71506E-02 -->   0.13476 try next   0.34860
   +  0.34860     -->   0.48336 try next    1.3944
   +  1.66401E-02 -->   0.50000 try next    1.3944
Output flag set =====   0.50000
   +  0.50000     -->   1.0000 try next    1.3944
Output flag set =====   1.0000
   +  0.50000     -->   1.5000 try next    1.3944
Output flag set =====   1.5000
   +  0.50000     -->   2.0000 try next    1.3944
Output flag set =====   2.0000
   +  0.50000     -->   2.5000 try next    1.3944
Output flag set =====   2.5000
   +  0.50000     -->   3.0000 try next    1.3944
Output flag set =====   3.0000
   +  0.50000     -->   3.5000 try next    1.3944
Output flag set =====   3.5000

Step      Time      Event  Event function
----      ----      -----  --------------
  0      4.0000       2     0.20000
  1      3.7500       2     4.68752E-02
  2      3.6250       2    -3.20308E-02
  3      3.6875       2     7.61771E-03
  4      3.6563       2    -1.21577E-02
  5      3.6719       2    -2.25782E-03
  6      3.6797       2     2.68292E-03
  7      3.6758       2     2.13385E-04
  8      3.6738       2    -1.02210E-03
  9      3.6748       2    -4.04358E-04
 10      3.6753       2    -9.56059E-05
 11      3.6755       2     5.88894E-05
 12      3.6754       2    -1.83582E-05
 13      3.6755       2     2.02656E-05
 14      3.6754       2     9.53674E-07

   +  0.17545     -->   3.6754 try next    1.3944
Output flag set =====   3.6754 preparing for event
Output flag set =====   3.6754
```

This a detailed reporting may slow down the execution of the model, but in case of problems or unexpected results it is often useful to inspect for instance the order in which various events take place.

4.4 Connecting the calendar: Particle2.fst

The process simulated by the model in section 4.1 clearly does not depend on calendar times. As an illustration, however, we will connect the calendar by adding the statement

```
!  calendar connection
   TRANSLATION_GENERAL StartYear=1985 ; StartDOY=100.5 ; OneDay=24.0
```

which sets the start time[3] of the model as 10-Apr-1985 12:00:00 and sets the the unit of time at one hour. More details about calendar use can be found in section 2.2. After adding the above calendar connection to our example model, the first part of the logfile changes into (cf. section 4.3)

[3]Note that StartDOY is based on day numbers and starts at 1.0000 for January,1 at 00:00:00.

... DRAFT ...

```
RKDRIV: DYNAMIC loop
==========
   time step          time
   ---------------------------------
Output flag set =====   0.00000 (10-Apr-1985 12:00:00)
   +  1.00000E-02 -->   1.00000E-02 (10-Apr-1985 12:00:36) try next   1.58193E-02
   +  1.58193E-02 -->   2.58193E-02 (10-Apr-1985 12:01:33) try next   2.17876E-02
   +  2.17876E-02 -->   4.76069E-02 (10-Apr-1985 12:02:51) try next   8.71506E-02
   +  8.71506E-02 -->   0.13476 (10-Apr-1985 12:08:05) try next   0.34860
   +  0.34860     -->   0.48336 (10-Apr-1985 12:29:00) try next    1.3944
   +  1.66401E-02 -->   0.50000 (10-Apr-1985 12:30:00) try next    1.3944
Output flag set =====   0.50000 (10-Apr-1985 12:30:00)
   +  0.50000     -->   1.0000 (10-Apr-1985 13:00:00) try next    1.3944
Output flag set =====   1.0000 (10-Apr-1985 13:00:00)
   +  0.50000     -->   1.5000 (10-Apr-1985 13:30:00) try next    1.3944
Output flag set =====   1.5000 (10-Apr-1985 13:30:00)
   +  0.50000     -->   2.0000 (10-Apr-1985 14:00:00) try next    1.3944
Output flag set =====   2.0000 (10-Apr-1985 14:00:00)
   +  0.50000     -->   2.5000 (10-Apr-1985 14:30:00) try next    1.3944
Output flag set =====   2.5000 (10-Apr-1985 14:30:00)
   +  0.50000     -->   3.0000 (10-Apr-1985 15:00:00) try next    1.3944
Output flag set =====   3.0000 (10-Apr-1985 15:00:00)
   +  0.50000     -->   3.5000 (10-Apr-1985 15:30:00) try next    1.3944
Output flag set =====   3.5000 (10-Apr-1985 15:30:00)

Step         Time                           Event  Event function
----         ----                           -----  --------------
  0     4.0000 (10-Apr-1985 16:00:00)        2      0.20000
  1     3.7500 (10-Apr-1985 15:45:00)        2      4.68752E-02
  2     3.6250 (10-Apr-1985 15:37:30)        2     -3.20308E-02
  3     3.6875 (10-Apr-1985 15:41:15)        2      7.61771E-03
  4     3.6563 (10-Apr-1985 15:39:23)        2     -1.21577E-02
  5     3.6719 (10-Apr-1985 15:40:19)        2     -2.25782E-03
  6     3.6797 (10-Apr-1985 15:40:47)        2      2.68292E-03
  7     3.6758 (10-Apr-1985 15:40:33)        2      2.13385E-04
  8     3.6738 (10-Apr-1985 15:40:26)        2     -1.02210E-03
  9     3.6748 (10-Apr-1985 15:40:29)        2     -4.04358E-04
 10     3.6753 (10-Apr-1985 15:40:31)        2     -9.56059E-05
 11     3.6755 (10-Apr-1985 15:40:32)        2      5.88894E-05
 12     3.6754 (10-Apr-1985 15:40:31)        2     -1.83582E-05
 13     3.6755 (10-Apr-1985 15:40:32)        2      2.02656E-05
 14     3.6754 (10-Apr-1985 15:40:32)        2      9.53674E-07

   +  0.17545     -->   3.6754 (10-Apr-1985 15:40:32) try next    1.3944
Output flag set =====   3.6754 (10-Apr-1985 15:40:32) preparing for event
Output flag set =====   3.6754 (10-Apr-1985 15:40:32)
```

The simulation times are precisely the same as in section 4.3, but each value of time now corresponds to a real clock time which is mentioned in the logfile.

4.5 In FSE mode: Particle3.fst

The TRANSLATION_GENERAL statements are deleted or "commented out" and replaced by the following

```
!  FSE mode translation with weather
   TRANSLATION_FSE
   WEATHER CNTR = 'NLD' ;  ISTN = 1 ; IYEAR = 1985
   TIMER STTIME = 1.0 ; FINTIM = 201.0 ; DELT = 0.01 ; PRDEL = 0.5 ; IPFORM = 4
```

. . . DRAFT . . .

Note that a WEATHER statement is required in FSE mode, even if no weather variables are used. The unit of time is fixed in FSE mode (a day) and cannot be changed into an hour. The FST model does not really imply a unit of time, however. The choice of one hour in Particle2.fst was an arbitrary one as well. Therefore we let the program run here 200 days, starting at day 1.0 in 1985.

In FSE mode there is no logfile with a precise event description. In the output file RES.DAT, however, we can find back the events. Just before and just after each event additional output calls are made to the model which result in two successive lines in RES.DAT with the same time value but with different values for variables changed at the event. For the first event of the simulation we find:

```
 TIME          X            Y             VX          VY      THEORYPERIODX ...
1.00000     -4.0000     -0.50000       2.0000      1.0000      -  ....
1.50000     -2.9878     -1.22499E-02   2.0500     0.95000      -  ....
2.00000     -1.9505      0.45050       2.1000     0.90000      -  ....
2.50000    -0.88826      0.88825       2.1500     0.85000      -  ....
3.00000     0.19899       1.3010       2.2000     0.80000      -  ....
3.50000      1.3112       1.6888       2.2500     0.75000      -  ....
4.00000      2.4485       2.0515       2.3000     0.70000      -  ....
4.50000      3.6107       2.3893       2.3500     0.65000      -  ....
4.68000      4.0352       2.5047       2.3680     0.63200      -  ....
4.68000      4.0352       2.5047       2.3680    -0.63200      -  ....
5.00000      4.7979       2.2975       2.4000    -0.66400      -  ....
5.09000      5.0143       2.2374       2.4090    -0.67300      -  ....
5.09000      5.0143       2.2374      -2.4090    -0.67300      -  ....
5.50000      4.0348       1.9533      -2.3680    -0.71400      -  ....
6.00000      2.8631       1.5840      -2.3180    -0.76400      -  ....
6.50000      1.7163       1.1898      -2.2680    -0.81400      -  ....
```

The particle hits a wall for the first time at 3.68 units of time after the start (Time value 4.68, velocity VY reverses). In TRANSLATION_GENERAL mode the first hit was simulated at 3.6754 units of time after the start.

This illustrates the difference between the two simulation modes. In the GENERAL mode the time of the hit is approximated numerically. In FSE mode it is just the first time encountered at which the event function changes sign (the particle is "through the wall"). Hence, in FSE mode the event times will be more accurate for smaller values of DELT. In GENERAL mode this is not the case.

For this particular model we may expect that the time between two successive hits will be slightly too large in FSE mode since the particle always passes the wall before the model notices the event and reverses the velocity. This is indeed the case. The FSE model calculates an average time between hits of 4.5941 time units. In GENERAL mode the output variable MeasuredPeriodX is 4.5896, in precise accordance with the theoretical value TheoryPeriodX which is 4.5896 as well[4]

4.6 Comments on this program

The event sections in this example do not contain calculation statements. As explained in section 3.3, event sections may contain calculations in precisely the same way as the initial or terminal section of the program, including subroutine and function calls. The calculations may refer to each other (and to initial and dynamic variables) and are brought in computable order by the translator.

[4] For very large simulation times the time measured in FSE mode decreases, probably due to integration errors and a slow energy gain of the particle.

... DRAFT ...

The function TimeBetweenHits is not very robust since the arguments of both square root calls may become negative. This happens if the particle has not enough kinetic energy to reach the wall from which it is drawn away by gravity. A better program would test for this condition before calculating the square roots. If just one wall is hit other expressions can be derived.

The function call could further be extended with a string argument, for instance 'X' or 'Y', as explained in section 2.4. Messages from the function could then be accompanied by this identification string.

4.7 The generated Fortran code

Appendix A contains the GENERAL mode Fortran-90 code generated by the translator for the Particle1.fst. Appendix B contains the FSE mode Fortran for Particle3.fst.

The general style of the generated Fortran was left unchanged. In FSE mode the SUBROUTINE MODEL statement is the same as in FST 2, which implies that older models, possibly edited "by hand", can be linked with the new drivers library.

The idea of the GENERAL mode Fortran is that the simulation driver knows all, time, states and event function values, and the model is called only for calculating rates of change. In FSE mode the situation is more or less reversed. The model(s) store all information locally, and the FSE driver just organizes a meaningful execution of the various sections for initial, dynamic, etc.

The addition of events did not change this approach. This implies that the code for handling events *in the model* is more complex in FSE mode than in GENERAL mode. Interested users may inspect the Fortran code in Appendix B.

4.8 Adding events to existing FSE models

The FSE simulation driver is not used only in combination with generated Fortran code. There are also "handwritten" FSE models. Such models can be linked to the same FSE driver used for generated Fortran under FST 3. The model structure described in van Kraalingen (1995) was left unchanged, including the argument list of the model subroutine.

Sudden changes in a simulated system have sometimes been implemented as complex "constructs", just because events could not be handled. Hence, there may be a need for adding "by hand" code required for state or time events.

The best strategy is probably to inspect at first Fortran code generated by the FST translator. If your model requires state events only, the Fortran code in Appendix B can be used as a reference. Additions are required in the declarations, at the end of the "ITASK=1" section, at the end of the "ITASK=2" section and at the beginning of the "ITASK=3" section. Finally, a new "ITASK=5" section has to be written in which the events actually take place.

It may be a good idea to write a sort of dummy FST model containing just the desired events together with the settings and states involved. The Fortran generated by the FST translator then contains sections which can be copied to the handwritten FSE model.

. . . DRAFT . . .

CHAPTER 5

Installation

The FST translator translates an FST model into a Fortran-90 program with datafiles. If the generated Fortran program is compiled by a Fortran compiler and linked with a few object libraries belonging to FST, the resulting application or "exe file" executes the model runs specified in the FST model.

This obviously requires an installed Fortran compiler with FST object libraries precompiled for the installed compiler. Since also the translator itself is a Fortran program, FST is highly portable between platforms. It can be readily adapted to any new hardware platform or operating system for which a Fortran compiler is available[1].

The translator and the FST object libraries are currently available for the Intel Fortran compiler and the Compaq Visual Fortran compiler on a PC under Windows. On a Macintosh the translator runs as a PowerPC application under OS X, the FST object libraries are available for Absoft Pro Fortran 9.2 (for PowerPC). A port to intel macs will be available in 2009.

5.1 FSTwin on a PC

On a windows machine, the FSTwin environment provides an integrated modelling environment from which the FST model is edited, the translator is called, the compiler and linker are called and output can be inspected or plotted. The FSTwin environment requires the Compaq Visual Fortran compiler or the Intel Fortran compiler. If both compilers are installed the user can switch between them.

=== Here some more details of the installation have to be added ===

5.2 The FST object libraries

A Fortran program generated by the FST translator has to be compiled and linked with the object libraries Drivers, Weather and TTutil.

[1] The use of Fortran in parallel computing, the huge amount of existing Fortran code and the recent updates of the Fortran standard (Fortran-90, Fortran-95 and Fortran-2003) guarantee that the language will be available for decades to come.

5.2.1 Drivers

The Drivers library contains the GENERAL and FSE "simulation drivers" which drive the simulation through time. It is the driver that contains all the logic for calling the initial, dynamic and terminal section of the model, for event execution and periodic output. The Drivers library further contains the Fortran implementation of the FST intrinsic functions.

For the FST 3 update, the GENERAL driver has been largely rewritten in order to implement events and to merge the code of the FST 2 drivers EUDRIV and RKDRIV. These two drivers implemented the two TRANSLATION_GENERAL integration methods, but were in fact largely identical.

The FSE driver is still similar to the one documented by (van Kraalingen, 1995). Some logic for event execution has been added, but the argument list of the model subroutines was not touched (cf. section 4.8). This implies that existing FSE models (either generated or "handwritten") can be combined with the updated drivers library.

5.2.2 Weather

The Weather library (Weather 2005 version 1.2) contains the interface between weather datafiles and the weather variables used in an FST model. The original library (van Kraalingen *et al.*, 1991) has been updated several times. The Fortran code has been improved and the datafile format has been adapted to year numbers above 1999. Simple user calls to the library, however, did not change.

5.2.3 TTutil

The TTutil[2] library (van Kraalingen & Rappoldt, 2000) contains a bunch of utilities for file i/o and string handling.

A minor difficulty is that the subroutine FatalERR which is used in the FSTwin environment differs from the FatalERR subroutine in TTutil. The FSTwin environment generates its own FatalERR by means of a translator option and the subroutine is absent from the TTutil library that comes with FSTwin. In a future maintenance update of TTutil this inconsistency will be resolved.

[2] The letters "TT" come from "Theoretische Teeltkunde", the Dutch name of the department of Theoretical Production Ecology founded by the late prof. C.T. de Wit.

... DRAFT ...

References

Chapman, S. J., 2008. Fortran 95/2003 for scientists and engineers. MaxGraw-Hill, New York.

Metcalf, M., Reid, J., Cohen, M., 2004. Fortran 95/2003 explained. Oxford University Press, Oxford.

Rappoldt, C., van Kraalingen, D. W. G., 1996. The Fortran Simulation Translator FST version 2.0. Technical report, DLO Research Inistitute for Agrobiology and Soil fertility; The C.T.de Wit graduate school for Production Ecology, Wageningen, the Netherlands. Quantitative Approaches in Systems Analysis No. 5.

van Kraalingen, D. W. G., 1995. The FSE system for crop simulation, version 2.1. Technical report, DLO Research Inistitute for Agrobiology and Soil fertility; The C.T.de Wit graduate school for Production Ecology, Wageningen, the Netherlands.

van Kraalingen, D. W. G., Rappoldt, C., 2000. Reference manual of the fortran utility library ttutil v. 4. Technical report, Plant Research International (Report 5), Wageningen, the Netherlands. Updated PDF file available from kees.rappoldt@ecocurves.nl.

van Kraalingen, D. W. G., Stol, W., Uithol, P. W. J., Verbeek, M. G. M., 1991. User manual of CABO/TPE Weather System. Technical report, Centre for Agrobiological Research, Department of Theoretical Production Ecology of the Agricultural University, Wageningen, the Netherlands.

Appendices

Appendix A

General mode Fortran

The Fortran code generated by the translator from the example model Particle1.fst in section 4 is listed in this Appendix. The EUDRIV and RKDRIV drivers are contained in Module GeneralDrivers. This module also contains the driver supplied variables as PUBLIC copies of private variables and the subroutines called by the model to manage the events.

```
!----------------------------------------------------------------------!
! General info about this file                                         !
!                                                                      !
! Contents      : Generated Fortran program                            !
! Creator       : FST translator version 3.00                          !
! Creation date : 13-Oct-2008, 10:55:19                                !
! Source file   : PARTICLE1.FST                                        !
!----------------------------------------------------------------------!

!----------------------------------------------------------------------!
!                       STANDARD MAIN PROGRAM                          !
!----------------------------------------------------------------------!
  PROGRAM MAIN

! Calls subroutine RERUNS, from where EUDRIV or RKDRIV are
! called for all reruns. These drivers run the sumulation.
!
! Libraries used: DRIVERS and TTUTIL

! module use
  USE GeneralDrivers, ONLY: GeneralDriversVERSION

  IMPLICIT NONE

! administration
  CHARACTER(LEN=*), PARAMETER :: PrName    = 'MAIN'
  CHARACTER(LEN=*), PARAMETER :: PrVersion = 'for FST 3.00'
  CHARACTER(LEN=*), PARAMETER :: PrAuthor  = 'Kees Rappoldt 1995 2008'

! local
  INTEGER :: OutdatUnit, LogUnit, ITMP, Getun2
  LOGICAL :: TOSCR, TOLOG

  INTERFACE
!    interface to MODEL subroutine
     SUBROUTINE Model (ITASK,OUTPUT,TIME,STATE,RATE,SCALE,NDECL,NEQ)
     INTEGER :: ITASK, NDECL, NEQ
```

```
      REAL    :: TIME,STATE(NDECL),RATE(NDECL),SCALE(NDECL)
      LOGICAL :: OUTPUT
      END SUBROUTINE Model
   END INTERFACE

! open logfile
   call OpenLogF (.false., 'model', PrName, PrVersion, PrAuthor, .true.)
   call GeneralDriversVERSION
   call MESSINQ (TOSCR, TOLOG, LogUnit) ; TOSCR = .true.
   call MESSINI (TOSCR, TOLOG, LogUnit)

! open results file
   OutdatUnit = Getun2 (30,39,2)
   call FOPENS (OutdatUnit,'RES.DAT','NEW','DEL')
! all model runs
   call RERUNS (OutdatUnit,'TIMER.DAT','MODEL.DAT',MODEL)
! close results file .and. temporary output (res.bin)
   close (OutdatUnit) ; close (OutdatUnit+1)

! close all RD* temporary files using the free unit number OutdatUnit
   call RDDTMP (OutdatUnit)

! close logfile
   close (LogUnit)
! check for used units (can be switched on for debugging purposes)
! call USEDUN(10,99)
   END PROGRAM MAIN
!----------------------------------------------------------------------!
!                  TRANSLATED SIMULATION MODEL                         !
!----------------------------------------------------------------------!
   SUBROUTINE Model (ITASK,OUTPUT,TIME,STATE,RATE,SCALE,NDECL,NEQ)

! Model subroutine for use with driver EUDRIV or RKDRIV
! This subroutine should be linked with all SUBROUTINES used
! (as far as they are not included in the FST source file), with
! the library containing the simulation DRIVERS and with TTUTIL.
!
! ================ TITLE of the FST model ================
! none
!
! The STANDARD (!!) parameter list of this model subroutine:
!
! ITASK  - task of model routine                    I
! OUTPUT = .TRUE. output request                    I
! TIME   - time                                     I
! STATE  - state array of model                    I/O
! RATE   - rates of change belonging to STATE      I/O
! SCALE  - size scale of state variables           I/O
! NDECL  - declared size of arrays                  I
! NEQ    - Number of state variables, for ITASK=1   O
!                                       otherwise   I

   USE CHART
   USE GeneralDrivers

   IMPLICIT NONE
! formal
   INTEGER :: ITASK, NDECL, NEQ
   REAL    :: TIME, STATE(NDECL), RATE(NDECL), SCALE(NDECL)
   LOGICAL :: OUTPUT
```

. . . DRAFT . . .

```
! Number of state variables NSV
  INTEGER, PARAMETER :: NSV = 5

! Array size variables
! none

! State variables, initial values and rates
  REAL X, IX, RX
  REAL Y, IY, RY
  REAL VX, IVX, VXR
  REAL VY, IVY, VYR
  REAL MeasuredTime, zero, ClockRunning

! Model parameters
  REAL AX, AY, HalfSizeX, HalfSizeY

! Setting variables (Redefinable in events)
  REAL ClockedTotal, Counter

! Calculated variables
  REAL MeasuredPeriodX, Ratio, TheoryPeriodX, TheoryPeriodY

! Interpolation functions used in AFGEN en CSPLIN functions
! none

! Declarations and values of constants
! none

! other
  INTEGER IUMOD
  INTEGER Getun2
  REAL NOTNUL, TimeBetweenHits
  SAVE

  if (ITASK == 1) then

!    Initial section
!    ===============

!    Check size of NDEC against number of states NSV
     if (NDEC.LT.NSV) then
!       driver capacity too low ; stop program
        WRITE (*,'(A,I4,/,A,I4)') '  The number of state variables is',NSV, &
         ' and the capacity of the driver is',NDEC
        call FatalERR ('MODEL','Driver capacity too low')
     end if

!    Open input file
     IUMOD = Getun2 (10,29,2)
     call RDINIT (IUMOD,IULOG, ModelFile)

!    Read initial states
     call RDSREA ('IVX', IVX)
     call RDSREA ('IVY', IVY)
     call RDSREA ('IX', IX)
     call RDSREA ('IY', IY)
     call RDSREA ('zero', zero)

!    Read model parameters
     call RDSREA ('AX', AX)
```

. . . DRAFT . . .

```
     call RDSREA ('AY', AY)
     call RDSREA ('HalfSizeX', HalfSizeX)
     call RDSREA ('HalfSizeY', HalfSizeY)

!    Read LINT/CSPLIN interpolation functions
!    none

!    Read SCALE array and close datafile
     call RDFREA ('SCALExxx',SCALE,NDEC,NSV)
     close (IUMOD)

!    Set number of state variables
     NEQ = NSV

!    initial calculations

!    initialize measurement
     Counter      = 0.0
     ClockRunning = 0.0
     ClockedTotal = 0.0

!    initially known variables to output
!    none

!    send title(s) to OUTCOM
!    none

!    Initialize state variables
     X = IX
     Y = IY
     VX = IVX
     VY = IVY
     MeasuredTime = zero
!
!    request events
     call SetStateEventRequest (1)
     call SetStateEventRequest (2)

!    assign local variable names to state array
     STATE(1) = X
     STATE(2) = Y
     STATE(3) = VX
     STATE(4) = VY
     STATE(5) = MeasuredTime

  else if (ITASK == 2) then

!    rates of change section
!    =======================
!    Assign state array to local variable names
     X = STATE(1)
     Y = STATE(2)
     VX = STATE(3)
     VY = STATE(4)
     MeasuredTime = STATE(5)

!    event handling
     if (HandleEvent) then
        if (StateEvent(1)) then
!          event calculations, based on last rate call
```

... DRAFT ...

```
!          event NewValue assignments
           VX = -VX

!          switch on the stopwatch
           ClockRunning = 1.0

!          read the stopwatch ; count the hits
           ClockedTotal = MeasuredTime
           Counter = Counter + 1.0
        end if
        if (StateEvent(2)) then
!          event calculations, based on last rate call
!          event NewValue assignments
           VY = -VY
        end if
     end if

!    dynamic calculations
!    place
     RX = VX
     RY = VY

!    velocity
     VXR = AX
     VYR = AY

!    finish conditions
     if (KEEP.EQ.1) then
        continue
     end if

!    output
     if (OUTPUT) then
        call ChartNewGroup
        call ChartOutputRealScalar ('TIME',TIME)
        call OUTDAT (2, 0, 'TIME  ', TIME  )
        call OUTDAT (2, 0, 'X', X)
        call ChartOutputRealScalar('X', X)
        call OUTDAT (2, 0, 'Y', Y)
        call ChartOutputRealScalar('Y', Y)
        call OUTDAT (2, 0, 'VX', VX)
        call ChartOutputRealScalar('VX', VX)
        call OUTDAT (2, 0, 'VY', VY)
        call ChartOutputRealScalar('VY', VY)
     end if

!    assign calculated rates to rate array
     RATE(1) = RX
     RATE(2) = RY
     RATE(3) = VXR
     RATE(4) = VYR
     RATE(5) = ClockRunning

!    assign local state variable names to state array
     if (HandleEvent) then
        STATE(1) = X
        STATE(2) = Y
        STATE(3) = VX
        STATE(4) = VY
        STATE(5) = MeasuredTime
     end if
```

. . . DRAFT . . .

```
  else if (ITASK == 3) then

!    calculate state event functions
!    ===============================
     call StateEventFunction (1, abs(X) - HalfSizeX)
     call StateEventFunction (2, abs(Y) - HalfSizeY)

  else if (ITASK == 4) then

!    terminal section
!    ================
!    assign terminal states and rates to local variable names
     X = STATE(1)
     Y = STATE(2)
     VX = STATE(3)
     VY = STATE(4)
     MeasuredTime = STATE(5)
     RX = RATE(1)
     RY = RATE(2)
     VXR = RATE(3)
     VYR = RATE(4)
     ClockRunning = RATE(5)

!    terminal calculations
     MeasuredPeriodX = ClockedTotal / NOTNUL(Counter-1.0)

!    theoretical time between hits
     TheoryPeriodX = TimeBetweenHits(IX,IVX,AX,HalfSizeX)
     TheoryPeriodY = TimeBetweenHits(IY,IVY,AY,HalfSizeY)
     Ratio = TheoryPeriodY / TheoryPeriodX

!    terminal output
     call ChartTerminalGroup
     call ChartOutputRealScalar ('TIME',TIME)
     call OUTDAT (2, 0, 'TIME  ', TIME  )
     call OUTDAT (2, 0, 'TheoryPeriodX', TheoryPeriodX)
     call ChartOutputRealScalar('TheoryPeriodX', TheoryPeriodX)
     call OUTDAT (2, 0, 'TheoryPeriodY', TheoryPeriodY)
     call ChartOutputRealScalar('TheoryPeriodY', TheoryPeriodY)
     call OUTDAT (2, 0, 'Ratio', Ratio)
     call ChartOutputRealScalar('Ratio', Ratio)
     call OUTDAT (2, 0, 'MeasuredPeriodX', MeasuredPeriodX)
     call ChartOutputRealScalar('MeasuredPeriodX', MeasuredPeriodX)

!    printplot output
!    none
  end if

  Return
  END SUBROUTINE Model

REAL FUNCTION TimeBetweenHits (X, V, A, D)
  IMPLICIT NONE
  REAL, INTENT(IN) :: X, V, A, D
  TimeBetweenHits = (SQRT(V**2+2.0*A*(D-X)) - SQRT(V**2-2.0*A*(D+X))) / A
  Return
END FUNCTION TimeBetweenHits
```

... DRAFT ...

Appendix B

FSE mode Fortran

The Fortran code generated by the translator from the example model Particle3.fst discussed in section 4 is listed in this Appendix. The general idea of the generated Fortran has not changed. The model still contains the model status as local variables, which are updated during integration calls from the FSE driver. And still there is the possibility of coupling several FSE models by adding additional model calls in the MODELS routine.

Even the list of formal parameters of the model subroutine has not been extended. Communication on event handling between the FSE driver and the model(s) is organized via a Fortran-90 Module EventFlagsFSE containing 4 logical variables. The MODELS routine contains instructions for dealing with these event flags when more models are added by hand.

```
!----------------------------------------------------------------------!
! General info about this file                                         !
!                                                                      !
! Contents      : Generated FSE module                                 !
! Creator       : FST translator version 3.00                          !
! Creation date : 13-Oct-2008, 11:09:18                                !
! Source file   : PARTICLE3.FST                                        !
!----------------------------------------------------------------------!

  PROGRAM MAIN
  IMPLICIT NONE
  CALL FSE
  END PROGRAM MAIN

!----------------------------------------------------------------------!
! SUBROUTINE MODELS                                                    !
! Authors: Daniel van Kraalingen                                       !
! Date   : 5-Jul-1993                                                  !
!        : May-2008: (KR) adapted to events as generated by FST 3      !
! Purpose: This subroutine is the interface routine between the FSE-   !
!          driver and the simulation models. This routine is called    !
!          by the FSE-driver at each new task at each time step. It    !
!          can be used by the user to specify calls to the different   !
!          models that have to be simulated                            !
!                                                                      !
! FORMAL PARAMETERS:  (I=input,O=output,C=control,IN=init,T=time)      !
! name   type meaning                                    units  class !
! ----   ---- -------                                    -----  ----- !
! ITASK   I4  Task that subroutine should perform          -      I  !
! IUNITD  I4  Unit that can be used for input files        -      I  !
```

```
! IUNITO  I4  Unit used for output file                         -      I  !
! IUNITL  I4  Unit used for log file                            -      I  !
! FILEIT  C*  Name of timer input file                          -      I  !
! FILEI1  C*  Name of input file no. 1                          -      I  !
! FILEI2  C*  Name of input file no. 2                          -      I  !
! FILEI3  C*  Name of input file no. 3                          -      I  !
! FILEI4  C*  Name of input file no. 4                          -      I  !
! FILEI5  C*  Name of input file no. 5                          -      I  !
! OUTPUT  L4  Flag to indicate if output should be done         -      I  !
! TERMNL  L4  Flag to indicate if simulation is to stop         -     I/O !
! DOY     R4  Day number within year of simulation (REAL)       d      I  !
! IDOY    I4  Day number within year of simulation (INTEGER)    d      I  !
! YEAR    R4  Year of simulation (REAL)                         y      I  !
! IYEAR   I4  Year of simulation (INTEGER)                      y      I  !
! TIME    R4  Time of simulation                                d      I  !
! STTIME  R4  Start time of simulation                          d      I  !
! FINTIM  R4  Finish time of simulation                         d      I  !
! DELT    R4  Time step of integration                          d      I  !
! LAT     R4  Latitude of site                            dec.degr.   I  !
! LONG    R4  Longitude of site                           dec.degr.   I  !
! ELEV    R4  Elevation of site                                 m      I  !
! WSTAT   C6  Status code from weather system                   -      I  !
! WTRTER  L4  Flag whether weather can be used by model         -      O  !
! RDD     R4  Daily shortwave radiation                      J/m2/d    I  !
! TMMN    R4  Daily minimum temperature                    degrees C  I  !
! TMMX    R4  Daily maximum temperature                    degrees C  I  !
! VP      R4  Early morning vapour pressure                    kPa     I  !
! WN      R4  Average wind speed                               m/s     I  !
! RAIN    R4  Daily amount of rainfall                         mm/d    I  !
!                                                                        !
! Fatal error checks: none                                               !
! Warnings          : none                                               !
! Subprograms called: models as specified by the user                    !
! File usage        : none                                               !
!------------------------------------------------------------------------!

  SUBROUTINE MODELS (ITASK , IUNITD, IUNITO, IUNITL,                 &
                     FILEIT, FILEI1, FILEI2, FILEI3, FILEI4, FILEI5, &
                     OUTPUT, TERMNL,                                 &
                     DOY   , IDOY  , YEAR  , IYEAR,                  &
                     TIME  , STTIME, FINTIM, DELT ,                  &
                     LAT   , LONG  , ELEV  , WSTAT, WTRTER,          &
                     RDD   , TMMN  , TMMX  , VP   , WN, RAIN)

  USE EventFlagsFSE

  IMPLICIT NONE
! Formal parameters
  INTEGER          ::  ITASK, IUNITD, IUNITO, IUNITL, IDOY, IYEAR
  CHARACTER(LEN=*) :: FILEIT, FILEI1, FILEI2, FILEI3, FILEI4, FILEI5
  LOGICAL          :: OUTPUT, TERMNL, WTRTER
  CHARACTER(LEN=6) :: WSTAT*6
  REAL             :: DOY,YEAR,TIME,STTIME,FINTIM,DELT
  REAL             :: LAT,LONG,ELEV,RDD,TMMN,TMMX,VP,WN,RAIN

! Local variables
! none
  SAVE

! Only one model used here
! When more models are added the event flags should be handled
```

. . . DRAFT . . .

```
! before and after each model call in the same way as below.

! copy prepare flag for this model
  EventReq    = .false.

  CALL MODEL (ITASK , IUNITD, IUNITO, IUNITL,         &
              FILEIT, FILEI1,                         &
              OUTPUT, TERMNL,                         &
              DOY   , IDOY  , YEAR  , IYEAR,          &
              TIME  , STTIME, FINTIM, DELT ,          &
              LAT   , LONG  , ELEV  , WSTAT, WTRTER,  &
              RDD   , TMMN  , TMMX  , VP   , WN, RAIN)

! communicate request flag to driver
  AnyModelReq = AnyModelReq .or. EventReq

  RETURN
  END SUBROUTINE MODELS

!----------------------------------------------------------------------!
! SUBROUTINE MODEL                                                     !
! Authors: FST translator                                              !
! Date   :                                                             !
! Purpose: This subroutine is the translated FST file                  !
!                                                                      !
! FORMAL PARAMETERS:  (I=input,O=output,C=control,IN=init,T=time)      !
! name   type meaning                                    units  class !
! ----   ---- -------                                    -----  ----- !
! ITASK   I4  Task that subroutine should perform          -      I   !
! IUNITD  I4  Unit of input file with model data           -      I   !
! IUNITO  I4  Unit of output file                          -      I   !
! IUNITL  I4  Unit number for log file messages            -      I   !
! FILEIT  C*  Name of timer input file                     -      I   !
! FILEIN  C*  Name of file with input model data           -      I   !
! OUTPUT  L4  Flag to indicate if output should be done    -      I   !
! TERMNL  L4  Flag to indicate if simulation is to stop    -     I/O  !
! DOY     R4  Day number within year of simulation (REAL)    d      I   !
! IDOY    I4  Day number within year of simulation (INTEGER) d      I   !
! YEAR    R4  Year of simulation (REAL)                    y      I   !
! IYEAR   I4  Year of simulation (INTEGER)                 y      I   !
! STTIME  R4  Start time of simulation (=day number)       d      I   !
! FINTIM  R4  Finish time of simulation (=day number)      d      I   !
! DELT    R4  Time step of integration                     d      I   !
! LAT     R4  Latitude of site                          dec.degr. I   !
! LONG    R4  Longitude of site                         dec.degr. I   !
! ELEV    R4  Elevation of site                            m      I   !
! WSTAT   C6  Status code from weather system              -      I   !
! WTRTER  L4  Flag whether weather can be used by model    -      O   !
! RDD     R4  Daily shortwave radiation                 J/m2/d    I   !
! TMMN    R4  Daily minimum temperature                degrees C  I   !
! TMMX    R4  Daily maximum temperature                degrees C  I   !
! VP      R4  Early morning vapour pressure               kPa     I   !
! WN      R4  Daily average windspeed                     m/s     I   !
! RAIN    R4  Daily amount of rainfall                    mm/d    I   !
!                                                                      !
! Fatal error checks: if one of the characters of WSTAT = '4',         !
!                     indicates missing weather                        !
! Warnings          : none                                             !
! Subprograms called: models as specified by the user                  !
! File usage        : IUNITD,IUNITD+1,IUNITO,IUNITO+1,IUNITL           !
!----------------------------------------------------------------------!
```

... **DRAFT** ...

```
  SUBROUTINE MODEL (ITASK , IUNITD, IUNITO, IUNITL,        &
                    FILEIT, FILEIN,                        &
                    OUTPUT, TERMNL,                        &
                    DOY   , IDOY  , YEAR  , IYEAR,         &
                    TIME  , STTIME, FINTIM, DELT ,         &
                    LAT   , LONG  , ELEV  , WSTAT, WTRTER, &
                    RDD   , TMMN  , TMMX  , VP   , WN, RAIN)
  USE CHART
  USE EventFlagsFSE, ONLY: EventReq

! Title of the program
! <fill in your title here>

  IMPLICIT NONE

! Formal parameters
  INTEGER          :: ITASK , IUNITD, IUNITO, IUNITL, IDOY, IYEAR
  LOGICAL          :: OUTPUT, TERMNL, WTRTER
  CHARACTER(LEN=*) :: FILEIT, FILEIN, WSTAT
  REAL             :: DOY, YEAR, TIME, STTIME, FINTIM, DELT
  REAL             :: LAT, LONG, ELEV, RDD, TMMN, TMMX, VP, WN, RAIN

! Standard local variables
  INTEGER          :: IWVAR

! Array size variables
! none

! State variables, initial values and rates
  REAL X, IX, RX
  REAL Y, IY, RY
  REAL VX, IVX, VXR
  REAL VY, IVY, VYR
  REAL MeasuredTime, zero, ClockRunning

! Model parameters
  REAL AX, AY, HalfSizeX, HalfSizeY

! Setting variables
  REAL ClockedTotal, Counter

! Other calculated variables
  REAL MeasuredPeriodX, Ratio, TheoryPeriodX, TheoryPeriodY

! Interpolation functions used in AFGEN en CSPLIN functions
! none

! Declarations and values of constants
! none

! state event control
  INTEGER :: ivnt
  INTEGER, PARAMETER :: SEVcnt = 2
  REAL, DIMENSION(SEVcnt), SAVE    :: SEVfun, SEVfunSIGN
  LOGICAL, DIMENSION(SEVcnt), SAVE :: StateEvent, AtEventState

! Used functions
  REAL INTGRL, NOTNUL, TimeBetweenHits

! Code for the use of RDD, TMMN, TMMX, VP, WN, RAIN  (in that order)
```

... DRAFT ...

```
! a letter 'U' indicates that the variable is Used in calculations
  CHARACTER(LEN=6), PARAMETER :: WUSED = '------'
  SAVE

! Check weather data availability
  if (ITASK==1 .or. ITASK==2 .or. ITASK==4) then
     do IWVAR=1,6
!       is there an error in the IWVAR-th weather variable ?
        if (WUSED(IWVAR:IWVAR)=='U' .and. WSTAT(IWVAR:IWVAR)=='4') then
           WTRTER = .TRUE.
           TERMNL = .TRUE.
           RETURN
        end if
     end do
  end if

  if (ITASK == 1) then

!    Initial section
!    ===============

!    Open input file
     CALL RDINIT (IUNITD, IUNITL, FILEIN)

!    Read initial states
     call RDSREA ('IVX', IVX)
     call RDSREA ('IVY', IVY)
     call RDSREA ('IX', IX)
     call RDSREA ('IY', IY)
     call RDSREA ('zero', zero)

!    Read model parameters
     call RDSREA ('AX', AX)
     call RDSREA ('AY', AY)
     call RDSREA ('HalfSizeX', HalfSizeX)
     call RDSREA ('HalfSizeY', HalfSizeY)

!    Read LINT functions
!    none

     close (IUNITD)

!    Initial calculations

!    initialize measurement
     Counter      = 0.0
     ClockRunning = 0.0
     ClockedTotal = 0.0

!    Initially known variables to output
!    none

!    Send titles to OUTCOM
!    none

!    Initialize state variables
     X = IX
     Y = IY
     VX = IVX
     VY = IVY
     MeasuredTime = zero
```

```
!    release all state events
     AtEventState = .false.

  else if (ITASK == 2) then

!    Rates of change section
!    =======================
!    place
     RX = VX
     RY = VY

!    velocity
     VXR = AX
     VYR = AY

!    Finish conditions
!    none

!    Output
     if (OUTPUT) then
        call OUTDAT (2, 0, 'X', X)
        call ChartOutputRealScalar('X', X)
        call OUTDAT (2, 0, 'Y', Y)
        call ChartOutputRealScalar('Y', Y)
        call OUTDAT (2, 0, 'VX', VX)
        call ChartOutputRealScalar('VX', VX)
        call OUTDAT (2, 0, 'VY', VY)
        call ChartOutputRealScalar('VY', VY)
     end if

!    event functions
     SEVfun(1) = abs(X) - HalfSizeX
     SEVfun(2) = abs(Y) - HalfSizeY

!    do not repeat same event at same time
     StateEvent = (SEVfunSIGN > 0.0 .and. SEVfun <= 0.0 .or. &
                   SEVfunSIGN < 0.0 .and. SEVfun >= 0.0) .and. .not.AtEventState

     EventReq = any(StateEvent)

  else if (ITASK == 3) then

!    Integration section
!    ===================
!    Before integration, save event function signs for the current state
!    However, disable events which are in event state now
     do ivnt=1,SEVcnt
        if (AtEventState(ivnt)) then
!          current state is event state ; disable this event
!          event function must first move away from zero
           SEVfunSIGN(ivnt) = 0.0
        else
           if (SEVfun(ivnt) < 0.0) then
!             event function negative
              SEVfunSIGN(ivnt) = -1.0
           else if (SEVfun(ivnt) > 0.0) then
!             event function positive
              SEVfunSIGN(ivnt) = +1.0
           else
!             event function zero ; no event can take place
```

... DRAFT ...

```
               SEVfunSIGN(ivnt) = 0.0
            end if
         end if
      end do

!     now, by integration, move away from all event states !!!
      AtEventState = .false.

      X = INTGRL (X, RX, DELT)
      Y = INTGRL (Y, RY, DELT)
      VX = INTGRL (VX, VXR, DELT)
      VY = INTGRL (VY, VYR, DELT)

!     stop watch
      MeasuredTime = INTGRL (MeasuredTime, ClockRunning, DELT)

   else if (ITASK == 4) then

!     Terminal section
!     ================

!     Terminal calculations
      MeasuredPeriodX = ClockedTotal / NOTNUL(Counter-1.0)

!     theoretical time between hits
      TheoryPeriodX = TimeBetweenHits(IX,IVX,AX,HalfSizeX)
      TheoryPeriodY = TimeBetweenHits(IY,IVY,AY,HalfSizeY)
      Ratio = TheoryPeriodY / TheoryPeriodX

!     Terminal output
      call OUTDAT (2, 0, 'TheoryPeriodX', TheoryPeriodX)
      call ChartOutputRealScalar('TheoryPeriodX', TheoryPeriodX)
      call OUTDAT (2, 0, 'TheoryPeriodY', TheoryPeriodY)
      call ChartOutputRealScalar('TheoryPeriodY', TheoryPeriodY)
      call OUTDAT (2, 0, 'Ratio', Ratio)
      call ChartOutputRealScalar('Ratio', Ratio)
      call OUTDAT (2, 0, 'MeasuredPeriodX', MeasuredPeriodX)
      call ChartOutputRealScalar('MeasuredPeriodX', MeasuredPeriodX)

!     Printplot output
!     none

   else if (ITASK == 5) then

!     Handle Events
!     =============
      if (StateEvent(1)) then
!        event calculations, based on last rate call
!        output of event-calculated variables ; added to previous group

!        event NewValue assignments
         VX = -VX

!        switch on the stopwatch
         ClockRunning = 1.0

!        read the stopwatch ; count the hits
         ClockedTotal = MeasuredTime
         Counter = Counter + 1.0
```

... DRAFT ...

```
!       this prevents event repetition
        AtEventState(1) = .true.
     end if
     if (StateEvent(2)) then
!       event calculations, based on last rate call
!       output of event-calculated variables ; added to previous group

!       event NewValue assignments
        VY = -VY

!       this prevents event repetition
        AtEventState(2) = .true.
     end if
  end if

  Return
  END SUBROUTINE MODEL

REAL FUNCTION TimeBetweenHits (X, V, A, D)
  IMPLICIT NONE
  REAL, INTENT(IN) :: X, V, A, D
  TimeBetweenHits = (SQRT(V**2+2.0*A*(D-X)) - SQRT(V**2-2.0*A*(D+X))) / A
  Return
END FUNCTION TimeBetweenHits
```

... DRAFT ...