The FSE system for crop simulation, version 2.1

D.W.G. van Kraalingen

ab-dlo

AB-TPE rapport, April 1995

Simulation Reports AB-TT

Simulation Reports AB-TPE is a series giving supplementary information on agricultural simulation models that have been published elsewhere. Knowledge of those publications will generally be necessary in order to be able to study this material.

Simulation Reports AB-TPE describe improvements of simulation models, new applications or translations of the programs into other computer languages. Manuscripts or suggestions should be submitted to: H. van Keulen (AB) or J. Goudriaan (TPE-WAU).

Simulation Reports AB-TPE are issued by AB-DLO and TPE-WAU and they are available on request. Announcements of new reports will be issued regularly. Addresses of those who are interested in the announcements will be put on a mailing list on request.

The DLO Centre for Agrobiological and Soil Fertility Research (AB-DLO) falls under the Agricultural Research Department (DLO) of the Dutch Ministry of Agriculture, Nature Management and Fisheries.

The aim of DLO is to generate knowledge and develop expertise for implementing the agricultural policies of the Dutch government, for strengthening the agricultural industry, for planning and management of rural areas and for the protection of the environment. At AB-DLO experiments and computer models are used in fundamental and strategic research on plants. The results are used to:

- achieve optimal and sustainable plant production systems;
- find new agricultural products and improve product quality;
- enhance nature and environmental quality in the countryside.

Address:
AB-DLO
P.O. Box 14
6700 AA Wageningen
The Netherlands

tel. 31.8370.75700
fax. 31.8370.23110
e-mail postkamer@ab.agro.nl

Table of Contents

Samenvatting

Dit rapport beschrijft een FORTRAN 77 programma dat een omgeving vormt voor het ontwikkelen van continue simulatie modellen. Deze omgeving wordt FSE (FORTRAN Simulation Environment) genoemd. De omgeving bestaat uit een hoofdprogramma, weersgegevens en verscheidene hulp programma's voor het uitvoeren van specifieke taken. De feitelijke modelvergelijkingen worden ondergebracht in één of meer subroutines, die bestuurd worden door het hoofdprogramma. De FSE omgeving is flexibel van opzet, voert de tijdsbesturing uit, haalt weersgegevens op uit datafiles en voorziet in de mogelijkheid van eenvoudige invoer van parameters en initiële toestanden en het maken van reruns hierop. Tevens zijn voorzieningen aanwezig voor het op een eenvoudige manier maken van uitvoertabellen en grafieken. De FSE omgeving kan zonder wijzigingen op zeer uiteenlopende computers worden gebruikt.

De FSE omgeving biedt oplossingen voor veel problemen die ondervonden worden door onderzoekers die in FORTRAN programmeren. Door gebruik te maken van de FSE omgeving kan de onderzoeker zich beter op de wetenschappelijke aspecten van het model richten zonder geconfronteerd te worden met de technische problemen van het modelleren in FORTRAN.

Recentelijk is door AB-DLO een vertaler ontwikkeld die programma's geschreven in de simulatietaal FST (Fortran Simulation Translator) kan vertalen naar complete FSE model routines. De FST taal is afgeleidt van CSMP en biedt het voordeel van eenvoudigheid voor de beginnende modelleerder, en de mogelijkheid tot het overgaan naar de meer flexibele FSE omgeving.

Summary

A FORTRAN 77 programming environment for continuous simulation of agro-ecological processes, such as crop growth and calculation of water balances is presented. This system, called FSE (FORTRAN Simulation Environment), consists of a main program, weather data and utilities for performing specific tasks. The model equations have to be defined in one or more subroutines that are called by the main program. Both simple and complex crop growth models can be written as model subroutines, driven by FSE. The FSE environment is flexible, retrieves weather data from file, enables easy input of parameters and initial states and has the capability to carry out reruns on these parameter values. Facilities are provided for the output of simulation results in the form of tables or graphs, and time control. The FSE program is highly portable to different computer platforms.

The FSE program overcomes many programming problems that model developers face when programming in FORTRAN. By using this environment, crop modellers can concentrate more on the scientific aspects of modelling than on the technical ones.

Recently a translator program was developed at AB-DLO that is able to translate programs written in the FST simulation language into FSE model routines. The FST language is derived from the CSMP simulation language. FST provides easy programming of simulation models and at the same time allows the user to easily switch to the more flexible FSE environment.

Acknowledgements

Many people are acknowledged for their contribution to the FSE program and this manual: Frits Penning de Vries for creating the possibility to translate the CSMP MACROS programs into FORTRAN. FSE and this manual is based heavily on this work. Gon van Laar for suggesting the FSE name and making constructive comments, Martin Kropff for helpful suggestions, Peter Kooman and Willem Stol for testing the FSE program and making suggestions for improvement, among others.

1 Introduction

This report presents version 2.1 of the FSE simulation environment for crop growth models in FORTRAN. The version 2.1 is an improved version of FSE 1 which was documented in Van Kraalingen (1991).

After discussing the principles of simulation in FORTRAN, a full working program is presented. Much of this report is based on work done within the SARP project, notably the conversion of the CSMP MACROS programs into FORTRAN (Van Kraalingen & Penning de Vries, 1990). It is intended to meet the need expressed by crop modellers at AB-DLO/TPE-WAU to have this approach further refined and documented, without special reference to the SARP simulation models, but as a general documentation to the FSE simulation environment.

In this report, no specific crop models are discussed or described. The aim of this report is to describe the FSE standard for crop simulation as used by AB-DLO/TPE-WAU. In Appendix I and on the corresponding floppy disk, the SUCROS version as described by Goudriaan and Van Laar (1994) is given as an example model programmed in FSE. This is by no means the standard SUCROS version of AB-DLO or TPE-WAU.

In the past, crop simulation models often used CSMP as the simulation language. Some time ago, however, many crop modellers have switched to FORTRAN. Several reasons have been the cause of this development, largely because most of the scientific community uses FORTRAN and therefore it is very difficult to exchange models written in CSMP. Furthermore, CSMP is no longer available commercially. In fact, CSMP is kept 'alive' by the computer centre at Wageningen Agricultural University. It is available on only a few computers and requires a considerable programming effort for maintenance. In contrast, good FORTRAN compilers are widely available and are easy to purchase. Consequently, crop models in FORTRAN can be exchanged more easily and can be run on more computers with less maintenance effort. There are also technical reasons for preferring FORTRAN to CSMP. One is that larger, more flexible and more sophisticated models can be developed that can run on themselves providing more flexible output, or can be incorporated into a larger structure, e.g. for parameter optimization, Geographical Information Systems (GIS), Crop Management systems or Educational software.

A new development at AB-DLO and TPE-WAU is the development of the FST simulation language (Van Kraalingen, Rappoldt & Van Laar, 1994). This language is based on CSMP but a new translator has been developed that translates FST programs into FSE-FORTRAN. This provides modellers the possibility to start writing their scientific problems in FST. When necessary they can easily switch to FSE-FORTRAN and introduce more complexity if required. FST is also available on request.

This report begins by describing some principles of simulation in FORTRAN and then goes on with explaining the FSE simulation environment for crop growth models. This environment consists of a main model that provides the control structure for reruns, weather data and timing, and a collection of utilities that perform specific tasks such as parameter reading from files and model output. This system of main model and utilities is called FSE (FORTRAN Simulation Environment). The principles of simulation and the simulation environment itself will provide a sound basis for modellers who are working in FORTRAN. It will save them of having to find out the correct sequence of calculations, model structure, subprocess communication, etc..

This report contains Technical Documentation of how FSE works internally and how a crop modeller should write his routines to make them compatible with FSE and it contains a Users Guide, which describes how to operate a working FSE model.

Utility routines from the TTUTIL utility library will be used frequently in this report and in the FSE program. For a full documentation of TTUTIL, including examples, see Rappoldt & Van Kraalingen (1990). The same holds for the WEATHER system used within FSE (Kraalingen *et al.*, 1991).

The FSE source program is distributed on floppy disk with the necessary libraries to compile, link and run the program if you are working with Microsoft Fortran 5.1. (See Chapter 8 for what is present on the floppy disk).

Technical Documentation

2 Technical principles of FSE

In this Chapter the technical principles adhered to in the FORTRAN Simulation Environment other than the general structure of Euler simulation in FORTRAN will be discussed (this is further outlined in Chapter 4). Pointwise the main FSE principles are:

- The scientific part of the model is separated from the non-scientific overhead

In general, the contents of any simulation model can be divided into the scientific process equations (e.g. those for photosynthesis in crop growth) and the non-scientific part for tasks that are not model specific, e.g. read statements, data outputting, check on weather data. In FSE, these tasks have been separated rather rigidly so that model developers can concentrate as much as possible on the scientific content of their model. This way, they are not bothered by solving 'scientifically irrelevant' programming problems. The scientific contents (actual model) are programmed as a separate subroutine that is linked with the so-called FSE-driver. This driver takes care of task sequencing (e.g. initialization, rate and state calculation, output timing), retrieves and checks weather data from file, controls the time update for dynamic simulation, and takes care of correct integration of various scientific submodels that may be present (e.g. crop growth and water balance). To do so, the FSE driver makes calls to the WEATHER system for weather data control, and to the utility library TTUTIL. In addition, calls to TTUTIL may also be made in the scientific model subroutine for tasks such as input data reading and output data writing. The general structure of the FSE system is schematically depicted in Fig. 1, illustrating the separation between the scientific part (model) and the non-scientific 'overhead' (FSE driver and utility routines).

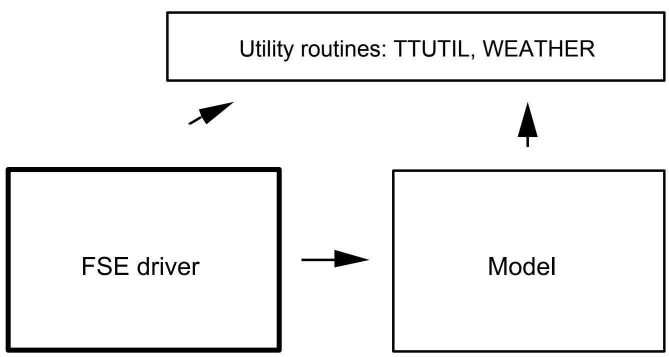

Figure 1 Simplified structure of FSE

- Complex functionality has been hidden in utility routines

Sometimes complex algorithms are used in the FSE program, for example, the set of routines that is used to read parameter values from data files, the routines to generate output tables and graphs and the `TIMER2` routine. These routines have a clearly defined task that is easy to understand, but their implementation in FORTRAN program can be very complex. For example, it is easy to understand that with the following statement: `CALL RDSREA ('WLVI', WLVI)` you get the value of the parameter `WLVI` from a data file. Rather complicated FORTRAN code underlies this subroutine and the user of the simulation program should not be bothered by it when using FSE. These routines are stored in a separate library, TTUTIL, the utility library of the Research Institute for Agrobiology and Soil Fertility (AB-DLO) and the Department of Theoretical Production Ecology of the Wageningen Agricultural University (TPE-WAU) (Rappoldt & Van Kraalingen, 1990).

- Straightforward program flow

It has been tried to make the program flow straightforward, by minimizing the number of `GOTO` statements. In general, the liberal use of `GOTO` statements is considered a bad programming practice, because the `GOTO` and the corresponding `CONTINUE` labels tend to be confusing. The problem is actually caused by the `CONTINUE` statement, because it represents a label to which any section of the program can jump to. In other words, with the statement `GOTO 10`, you will know where the program resumes execution, but at the line `10 CONTINUE` you can never be sure from where in the program a jump is made to that particular `CONTINUE` statement.

- Calculations are done in the correct sequence

Care has been taken to ensure that a model programmed in FSE is structured such that the different types of calculations, such as initialization, integration, rate calculation, time update and output are all done in the right order. Experience learns that this is often not the case in FORTRAN simulation models. Sometimes the rates and states in the model output do not pertain to the same TIME, or rate and state calculations are not performed separately; as a result, rates may be derived partly from state variables at the current time and partly from state variables one time step earlier. The results produced by a simulation model correctly implemented in the FSE program differ by not more than a rounding error from the results produced by the same model implemented in a continuous simulation language such as CSMP.

- Standard FORTRAN 77 is used, and transfer to new FORTRAN language definitions is easy

The FSE program has been written entirely in FORTRAN 77. This language is well defined (better than Pascal or C) and good compilers are available on many computers and operating systems. The definition of the language is published in ANSI document X3.9-1978. There are many good textbooks from which programming in FORTRAN 77 can be learned. Some of these have been listed in the reference section. For a definition of the language see Ter Haar (1983) among others.

The portability of the program is greatly improved by adhering to the definition of standard FORTRAN 77, and avoiding extensions that many compilers provide. To further improve portability among compilers, we have deliberately not used certain features that are part of the standard of the language (such as nested character operations) but that, in our experience, have sometimes been wrongly implemented in compilers.

At the time of writing of this document, a new FORTRAN standard has been defined: Fortran-90. It includes several features that were already defined in other languages or that were sometimes provided as FORTRAN compiler extensions. For example, advanced control structures such as `DO-WHILE` and volatile local variables in subroutines and functions. In the FSE program we have anticipated on these improvements by following guidelines from Ter Haar (1983, see Listing 1) a `DO-WHILE` control structure was emulated with `IF-ENDIF` statements, and by including a `SAVE` statement in all the subroutines and functions to prevent disappearance of local variables upon return to the calling program. The switch to Fortran-90 as a general programming language is, however, only worthwhile when good compilers on several computers are widely available. Until then, we will continue to use FORTRAN 77 as the language for the FSE program. If the `DO-WHILE` construct becomes part of the language, the emulated `DO-WHILE` structure can easily be modified:

Listing 1 The standard FORTRAN structure to emulate a DO-WHILE loop

Emulated DO-WHILE

```
10     IF (logical expression) THEN
          ...
          ...
       GOTO 10
       END IF
```

True DO-WHILE

```
DO WHILE (logical expression)
   ...
   ...
END DO
```

- Portability has been increased by not using large amounts of RAM memory

Large arrays are not used, because these increase RAM memory requirements. Although programming is often much easier, and program execution much faster, when arrays are used to solve specific problems, the use of arrays limits the number of computers on which the program can be run and often also the size of the problem that can be handled. Disk memory is often much larger than RAM memory, and therefore information is stored in temporary files whenever possible.

- The program is safeguarded against inaccurate floating point operations

The definition of standard FORTRAN 77 (like that of most programming languages) does not specify the algorithms to be used for floating point calculations. Consequently, the results of floating point operations can differ among compilers. The portability of a program in general is improved if these problems are anticipated and solved.

This inaccuracy is important in the `TIMER2` routine, which should trigger output whenever `TIME` has increased by a multiple of `PRDEL` (`PRDEL` is the time between successive outputs). Due to floating point inaccuracy, it is not correct simply to test if `TIME` is a multiple of `PRDEL` by using a `MOD` function. This problem has been solved by using integer variables (see `TIMER2` routine).

3 Implementation of Euler integration in FSE: Task Sequencing

This Chapter introduces the principles of Euler integration and the method adopted to couple different subprocesses without transgressing the rules of Euler integration. We assume here a basic knowledge of the state and rate variable approach as it is used in continuous simulation (see e.g. de Wit & Goudriaan, 1978; Penning de Vries & Van Laar, 1982; Leffelaar, 1993).

Various integration methods can be used in the simulation of continuous systems, ranging from simple rectangular integration (Euler) to higher order integration algorithms (trapezoidal, Runge-Kutta, etc.), possibly with a variable time step. From the point of view of program structure, a program that accommodates only Euler integration is less complicated and easier to understand than one accommodating higher order methods of integration. Because simulation models of crop growth in CSMP and FST often use Euler integration with a fixed time step of one day, and because the program structure is less complicated, this integration method is adopted in the FSE program.

3.1 Order of execution

Fig. 2 shows the correct order in which calculations should be executed when Euler integration is used:

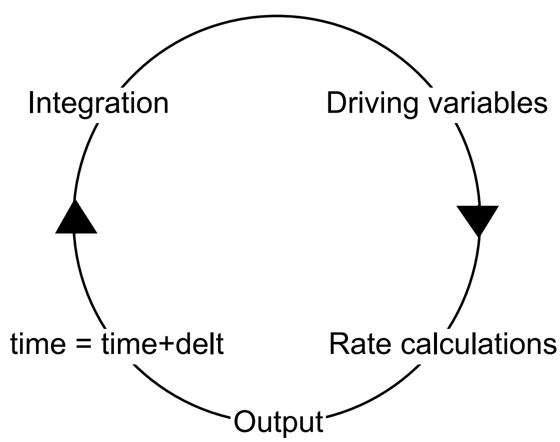

Figure 2 The order in which calculations should be executed when simulating continuous systems using Euler integration

Note that in this sequence, at the point where output is generated, state variables and rates of change correspond to the time for which they were calculated. Evidence that this sequence of calculations gives results in FORTRAN and CSMP that are identical, is shown for a simple simulation of exponential growth in Listing 2 and Listing 3.

Listing 2 CSMP program of exponential growth and output (only the relevant output is reproduced)

<program>

```
INCON IH=1.
PARAMETER RGR=0.1
```

```
H  = INTGRL (IH, GR)
GR = RGR*H
METHOD RECT
TIMER TIME=0.0, FINTIM=10., DELT=1.0, PRDEL=1.0
PRINT H, GR
END
STOP
ENDJOB
```

<output>

```
0TIMER VARIABLES     RECT    INTEGRATION      START TIME  = .00000
    DELT         DELMIN       FINTIM       PRDEL        OUTDEL     DELT
   1.0000     1.00000E-06  10.000      1.000        .00000     1.0000
1  DEMONSTRATION
0  TIME          H            GR
   .000000       1.0000       .10000
   1.00000       1.1000       .11000
   2.00000       1.2100       .12100
   3.00000       1.3310       .13310
   4.00000       1.4641       .14641
   5.00000       1.6105       .16105
   6.00000       1.7716       .17716
   7.00000       1.9487       .19487
   8.00000       2.1436       .21436
   9.00000       2.3579       .23579
   10.0000       2.5937       .25937
1$$$ SIMULATION HALTED FOR FINISH CONDITION  TIME     10.000
1$$$ CONTINUOUS SYSTEM MODELING PROGRAM III   V2.0    EXECUTION OUTPUT
```

Listing 3 FORTRAN program of exponential growth and output

<program>

```
      PROGRAM DEMO
      IMPLICIT NONE
      REAL RGR,FINTIM,DELT,H,GR,TIME
      PARAMETER (RGR=0.1, FINTIM=10., DELT=1.0)

      H    = 1.0
      GR   = 0.0
      TIME = 0.0

      OPEN (20, FILE='RES.OUT', STATUS='NEW')
      WRITE (20,'(A9,2A13)') 'TIME','H','GR'

10    IF (TIME.LE.FINTIM) THEN
         H  = H+GR*DELT                   <--integration
                                          <--driving variables (none)
         GR = RGR*H                       <--rate calculation
         WRITE (20,'(3G13.5)') TIME, H, GR  <--output
         TIME = TIME+DELT                 <--time=time+delt
      GOTO 10
```

```
      END IF

      STOP
      END
```

<output>

```
     TIME         H          GR
    .00000     1.0000      .10000
    1.0000     1.1000      .11000
    2.0000     1.2100      .12100
    3.0000     1.3310      .13310
    4.0000     1.4641      .14641
    5.0000     1.6105      .16105
    6.0000     1.7716      .17716
    7.0000     1.9487      .19487
    8.0000     2.1436      .21436
    9.0000     2.3579      .23579
    10.000     2.5937      .25937
```

To ensure that the results of the simulation are correct, the different types of calculations (integration, driving variables and rate calculations) should be strictly separated. In other words, first all states should be updated, then all driving variables should be calculated, after which all rates of change should be calculated. If this rule is not applied rigorously, there is a risk that some rates will pertain to states at the current time whereas others will pertain to states from the previous time step.

Since the calculations of rates and states cannot be mixed during a time step but should be executed separately, all the state calculations have to be grouped into one block as do all the rate calculations. Often, different subprocesses are interacting (e.g. a plant extracting water from the soil). In many cases these interactions among subprocesses involve only a few state variables. The water content at different depths in the soil is needed for the plant/soil system in the plant submodel. This is then used to determine water uptake for transpiration in dependence of rooting depth. The submodels for the plant and soil water thus exchange a limited amount of information, but they may contain very detailed descriptions of plant growth and soil moisture redistribution with many different rate and state calculations.

In view of this, it is not a good solution to combine all the state calculations from the different subprocesses into one large section and all the rate calculations in another. But it is feasible to separate the state and rate calculations within the subprocess descriptions (such as the plant) and have a calling program (what will be called the FSE-driver hereafter) to decide which of the two (rate or state section) to execute. With this method, the states can be calculated separately from the rates, whereas rates and states pertaining to the same subprocess are within the same subprogram. This technique is also discussed by Van Kraalingen and Rappoldt (1989). This concept of 'task-controlled execution' is illustrated in Fig. 3. The program lines of the plant and soil water subprocesses are separated into rate and state sections and only one of these is executed during a single call. Note that this program structure performs the calculations in exactly the same order as the circle given in Fig. 2.

So far, we have not discussed how to initialize the states, or where to enter the simulation circle and where to leave it (see Fig. 2).

It is convenient to leave the circle somewhere between time update and integration, because there the time and corresponding rates have been written to the output device and after the time update it seems logical to check whether the finish time (`FINTIM`) has been exceeded or whether further simulation is required. The most convenient way to initialize the subprocesses is to have this operation controlled by the FSE driver. This makes reruns possible, because in the main program the whole model can be reset to its initial state and be run again, with different weather data for instance. After initialization, it is most convenient to proceed with "Driving variables" and "Rate calculations" instead of entering the circle with "Integration". Entering the circle with "Driving variables" has clear advantages because in that case the rate variables do not have to be set to zero in the "Initialization" section to avoid that values from the last rate call are used in the first integration of the next run. These refinements to Fig. 2, among others, are shown in Fig. 4.

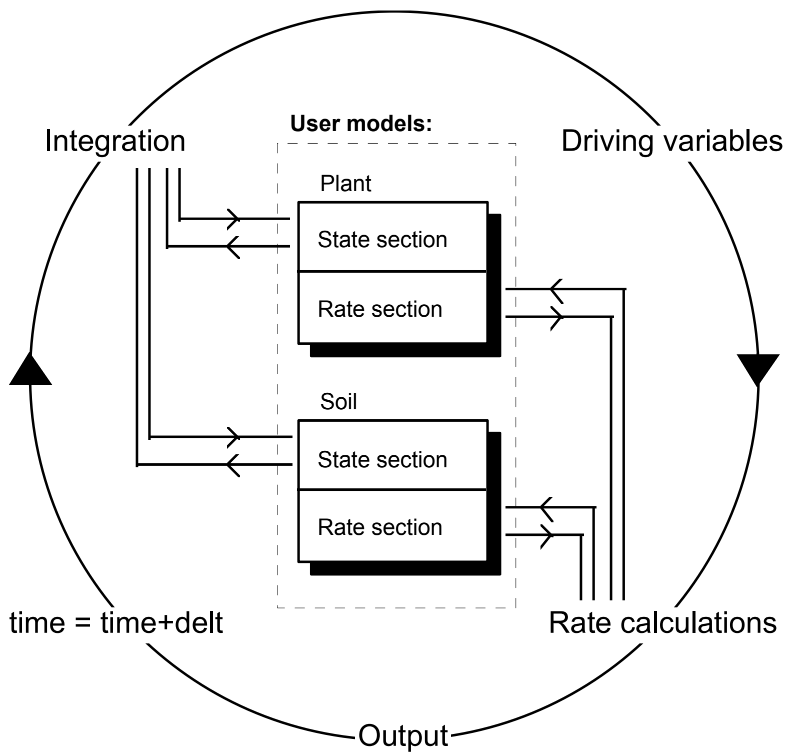

Figure 3 General structure for incorporating several subprocesses illustrated for a plant and a soil routine containing integration and rate calculation into a single simulation model

The question mark between "time = time+delt" and integration indicates the point at which it is decided whether or not to execute another time step. If the decision is "no", (possibly because time has reached the finish time or when another finish condition has been met) the model proceeds to the terminal section; if it is "yes" the circle is run once again. After proceeding to the terminal section, it must be decided whether a rerun is required. If the decision is "yes" the model has to be re-initialized after which a new simulation run is started.

Often simulation has to be terminated because of crop ripeness or when some other criterion has been met instead of a finish time that is exceeded. This test of finish conditions is positioned before "Output" in the circle, because, sometimes output is not done each time that the circle is run and it is convenient to have output from the last time that the circle was executed.

As shown in Fig. 3, the modularity of the subprocess descriptions is preserved by introducing the concept of task-controlled execution. To enable reruns, the various subprocess descriptions have to be initialized externally, and sometimes terminal calculations (e.g. harvest index) have to be done. Consequently, a subprocess description in the FSE program should recognize four different tasks: *initialization*, *integration*, *rate calculation* and *terminal calculation* (driving variables are calculated in the rate section).

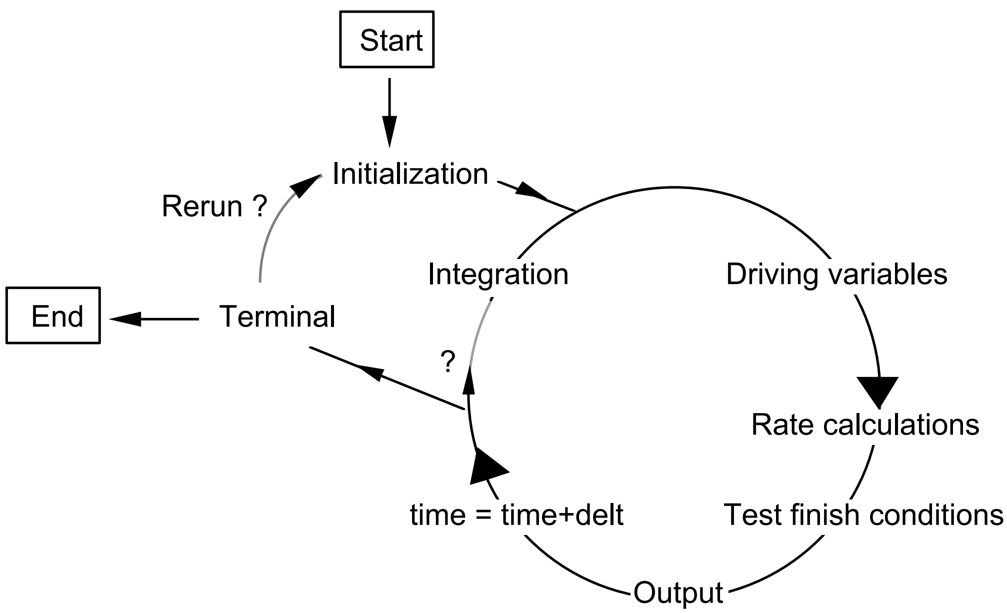

Figure 4 The order in which calculations are executed when simulating continuous systems using Euler integration, illustrating where to enter and leave the circle and how reruns are implemented

In the next Chapter we will discuss how this theory of continuous simulation using Euler integration has been implemented in FSE.

4 Outline of FSE-driver and utility system

In this Chapter the principles of Fig. 3 and Fig. 4, discussed in the previous Chapter, are implemented into a simplified FSE-driver and model routine. Specific aspects of the FSE-driver will be explained such as parameter input from file, reruns and output of results. For a full definition of the utility subroutines and functions, see Rappoldt & Van Kraalingen (1990), and for a full description of the weather subprogram and its corresponding data files, see Van Kraalingen *et al*. (1990).

By the end of this Chapter the reader should have a fair understanding of the FSE program (both the driver and underlying models). More technical details on the use of subprograms in simulation models can also be found in Van Kraalingen & Rappoldt (1989).

4.1 A simplified FSE driver

In Listing 4 a simplified FSE-driver is shown 'driving' the exponential growth program from Listing 3. This is not yet the full version of the FSE-driver but this version illustrates several of the important principles and features. These will be discussed below.

4.1.1 Loop control

After each time step it must be decided whether another time step is required or whether the simulation should proceed to the terminal section. One of the criteria to stop the simulation is that the finish time (`FINTIM`) has been exceeded. In crop growth simulation however, simulation is more often terminated because the crop is mature or some other criterion has been met. In other words, it should be possible to terminate the simulation loop from within each of the subprocesses. This is most conveniently done with a global variable called `TERMNL` of type `LOGICAL,` that indicates whether the loop should be terminated. The simulation loop should continue as long as `TERMNL = .FALSE.` This criterion is programmed as an emulated `DO-WHILE` loop. This is shown in the example program in Listing 4. Note that this program is conceptually similar to the program in Listing 3. (The implementation of the rerun facility is not shown here.)

The task-controlled execution concept discussed in the previous Chapter is implemented using an `INTEGER` variable `ITASK` that can have four different values, indicating the action required of the subroutines: 1=*initialization*, 2=*rate calculation*, 3=*integration* and 4=*terminal*. Within the loop, rate calculation and integration calls are done before and after the loop initial and terminal calls are done.

4.1.2 Rate calculation after initialization

As discussed in the previous Chapter, an integration call after initialization requires all rates to have been set at zero during initialization. With large models containing many rate variables this would require a long list of assignments to zero. We consider this an inelegant solution that is also error-prone (if the list is incomplete). A better solution is to perform rate calculations as the first step directly after initialization. The states have been initialized in the initial section, so it is permissible to compute rates of change from the states directly after initialization. In the dynamic section of the

FSE-driver therefore, an `IF-ENDIF` has been put around the integration section. The integration is now done only if a rate calculation has been carried out previously. This is also shown in Listing 4.

4.1.3 Time control

The control of time in a simulation program is more complicated than simply the increase of `TIME` with `DELT` and therefore it has been hidden in a subroutine called `TIMER2`, together with the control of output, updating of other time variables such as day and year, and the setting of the `TERMNL` flag when `FINTIM` is reached. Leap years are also recognized by this routine. Note that in FSE 1.0 a slightly different subroutine (`TIMER`) was used for time control in the simulation, the difference mainly being that with `TIMER`, `TIME` always begins at zero, while with `TIMER2` it always begins at the start time `STTIME`.

The output of the example program of Listing 4 shows how `TIMER2` works on the time control variables. For clarity, the output flag is ignored so that output is done at every time step to demonstrate that the output flag is switched on and off.

Listing 4 Simplified FSE-driver 'driving' a model routine that performs output only

```
      PROGRAM SMALL
      IMPLICIT NONE
      LOGICAL TERMNL, OUTPUT
      INTEGER ITASK , IDOY, IYEAR
      REAL STTIME,DELT,PRDEL,FINTIM,TIME,DOY

*     initialization of time variables
      ITASK  = 1
      STTIME = 360.
      DELT   = 1.0
      PRDEL  = 5.0
      FINTIM = 372.
      IYEAR  = 1984

*     initialization of TIMER2 and MODEL subroutines
      CALL TIMER2 (ITASK, STTIME, DELT, PRDEL, FINTIM,
     &             IYEAR, TIME  , DOY , IDOY , TERMNL, OUTPUT)
      CALL MODEL (ITASK , DELT  ,TIME, IYEAR, IDOY, DOY,
     &            OUTPUT, TERMNL)

*     run loop as long as TERMNL is .FALSE.
10    IF (.NOT.TERMNL) THEN
         IF (ITASK.EQ.2) THEN
*           integration
            ITASK = 3
            CALL MODEL (ITASK , DELT  ,TIME, IYEAR, IDOY, DOY,
     &                  OUTPUT, TERMNL)
         END IF

*        driving variables (none)

*        rate calculation and output
```

```
         ITASK = 2
         CALL MODEL (ITASK , DELT  ,TIME, IYEAR, IDOY, DOY,
     &               OUTPUT, TERMNL)
*        time update, update output flag, finish time reached ?
         CALL TIMER2 (ITASK, STTIME, DELT, PRDEL, FINTIM,
     &                IYEAR, TIME  , DOY , IDOY , TERMNL, OUTPUT)
      GOTO 10
      END IF

*     terminal calculations
      ITASK = 4
      CALL MODEL (ITASK , DELT  , TIME, IYEAR, IDOY, DOY,
     &            OUTPUT, TERMNL)
      END

*----------------------------------------------------------------
      SUBROUTINE MODEL (ITASK , DELT  ,TIME, IYEAR, IDOY, DOY,
     &                  OUTPUT, TERMNL)
      IMPLICIT NONE
*     Formal parameters
      LOGICAL TERMNL, OUTPUT
      REAL DELT,TIME, DOY
      INTEGER ITASK, IYEAR, IDOY
*     Local variables
      SAVE

      IF (ITASK.EQ.1) THEN

*        initialization of states and parameters
         WRITE (*,'(T11,6A7,/)')
     &     'TIME','IYEAR','IDOY','DOY','OUTPUT','TERMNL'
         WRITE (*,'(A,F7.0,2I7,F7.0,2L7)')
     &     ' Initial :',TIME,IYEAR, IDOY, DOY, OUTPUT, TERMNL

      ELSE IF (ITASK.EQ.2) THEN

*        rate calculation, finish conditions and output
         WRITE (*,'(A,F7.0,2I7,F7.0,2L7)')
     &     ' Rate    :',TIME,IYEAR, IDOY, DOY, OUTPUT, TERMNL

      ELSE IF (ITASK.EQ.3) THEN

*        integration
         WRITE (*,'(A,F7.0,2I7,F7.0,2L7)')
     &     ' State   :',TIME,IYEAR, IDOY, DOY, OUTPUT, TERMNL

      ELSE IF (ITASK.EQ.4) THEN

*        terminal calculations
         WRITE (*,'(A,F7.0,2I7,F7.0,2L7)')
     &     ' Terminal:',TIME,IYEAR, IDOY, DOY, OUTPUT, TERMNL
```

```
      END IF

      RETURN
      END
```

Listing 5	Output of the program from Listing 4

```
            TIME  IYEAR   IDOY    DOY OUTPUT TERMNL

Initial :   360.   1984    360   360.      T      F
Rate    :   360.   1984    360   360.      T      F
State   :   361.   1984    361   361.      F      F
Rate    :   361.   1984    361   361.      F      F
State   :   362.   1984    362   362.      F      F
Rate    :   362.   1984    362   362.      F      F
State   :   363.   1984    363   363.      F      F
Rate    :   363.   1984    363   363.      F      F
State   :   364.   1984    364   364.      F      F
Rate    :   364.   1984    364   364.      F      F
State   :   365.   1984    365   365.      T      F
Rate    :   365.   1984    365   365.      T      F
State   :   366.   1984    366   366.      F      F
Rate    :   366.   1984    366   366.      F      F
State   :   367.   1985      1     1.      F      F
Rate    :   367.   1985      1     1.      F      F
State   :   368.   1985      2     2.      F      F
Rate    :   368.   1985      2     2.      F      F
State   :   369.   1985      3     3.      F      F
Rate    :   369.   1985      3     3.      F      F
State   :   370.   1985      4     4.      T      F
Rate    :   370.   1985      4     4.      T      F
State   :   371.   1985      5     5.      F      F
Rate    :   371.   1985      5     5.      F      F
State   :   372.   1985      6     6.      T      F
Rate    :   372.   1985      6     6.      T      F
Terminal:   372.   1985      6     6.      T      T
```

4.2 Initialization of state variables and parameters from external data files

Three of the four sections distinguished in the plant submodel (integration, rate calculation and terminal) usually consist of relatively straightforward calculations. The initalization section, however, requires a separate explanation.

As explained in Chapter 3, model parameters have to be given values and states have to be initialized. As shown in Listing 3, this can be done by simple assignments such as `RGR = 0.1`. Any change in the value of one of the parameters or initial states, however, would then require compilation and linking of the model, a serious drawback compared with for instance CSMP and FST. In CSMP the user can run the model with different parameter sets automatically (after the `END` statement) or after parameter valueshave been changed in the CONTRO.SYS file. To introduce that option in the FSE program too, values of parameters and initial states are read from data files.

The values are extracted from the data files using a set of subroutines whose names all begin with RD (e.g. `RDSREA` means 'read a single real value'). With these routines the user can request the value from the datafile by supplying the name of the requested variable (of course after having defined which data file is used). The statement:

```
CALL RDSREA ('WLVI', WLVI)
```

requests the subroutine `RDSREA` to extract the value of `WLVI` from the data file and assign it to the variable `WLVI`. It does so by searching for the line: `WLVI =` <value> in the data file (in fact, the procedure is slightly different but that does not affect the understanding of the concept of the RD routines: the values are actually read from a temporary file which is created after syntax check and analysis of the data file). Consequently, the data file consists of the names and values of variables. An example datafile is given in Listing 6.

Listing 6 Example datafile. The syntax of data files is explained in more detail in Chapter 7

```
WLVI  = 10.; PLMXP = 38.
PLMTT = 0.,0., 10.,1., 30.,1., 50.,0.
ILEAF = 218
```

Listing 7 shows part of the initialization section of a plant model reading the datafile from Listing 6. An explanation is given below the listing.

Listing 7 Example illustrating the use of some RD routines

```
      ...
      IF (ITASK.EQ.1) THEN
         CALL RDINIT (IUNITD, IUNITL, FILEP)
         CALL RDSREA ('WLVI', WLVI)
         WLV = WLVI
         CALL RDSINT ('ILEAF', ILEAF)
         CALL RDAREA ('PLMTT', PLMTT, ILAR, IPLMTN)
         ...
         CLOSE (IUNITD)
      ELSE IF (ITASK.EQ.2) THEN
         ...
```

The statement:

```
CALL RDINIT (IUNITD, IUNITL, FILEP)
```

calls a subroutine that 1) opens the file with variable name `FILEP` using unit=`IUNITD+1` (`FILEP` is a character string that has been assigned the string `PLANT.DAT` in the calling program), 2) analyses the data file, 3) creates a temporary file from the data file using unit=`IUNITD`, 4) closes the data file (leaving `IUNITD` used for the temporary file !!), and 5) sends all error messages that have been created to a log file (with unit=`IUNITL`).

After this `RDINIT` call, the plant subroutine can retrieve the numerical values (including arrays) through several RD routines available in the library TTUTIL. These are given in Table 1. The `CLOSE` statement simply closes the temporary file that is created by the RD routines.

Table 1 Available input routines in the TTUTIL library for parameter input

Subroutine name	Meaning
RDINQR	Test if variable is in datafile (LOGICAL result !)
RDSREA	Read single real
RDSINT	Read single integer
RDSCHA	Read single character
RDSDOU	Read single double precision real
RDAREA	Read a not previously known number of reals
RDAINT	Read a not previously known number of integers
RDACHA	Read a not previously known number of characters
RDADOU	Read a not previously known number of double precision reals
RDFREA	Read a previously known number of reals
RDFINT	Read a previously known number of integers
RDFCHA	Read a previously known number of characters
RDFDOU	Read a previously known number of double precision reals

N.B. For details, see the TTUTIL documentation (Rappoldt & Van Kraalingen, 1990).

4.3 Implementation of reruns

Often, several runs with a crop growth simulation model are required. Examples are the study of crop yields for a number of years, or analysis of the effect of a different value of an input parameter. In CSMP and FST this can be done by repeating the parameter that is to be changed after an `END` statement. In Listing 8, weather data from 1984 are used in the first run; additional runs are made using weather data from 1985 and 1986. This facility is called the rerun facility. The output of the different runs is merged in the same output file, for easy comparison.

Listing 8 Example of the rerun facility in CSMP

```
TITLE DEMONSTRATION
PARAM YEAR=1984.
< model description etc. >
END
PARAM YEAR=1985.
END
PARAM YEAR=1986.
STOP
ENDJOB
```

We have included a similar rerun facility in the FSE system. By doing so, we prevent the user from making changes in the data files and run the model again (but, without compiling and linking). Each new run would also have deleted existing output files. This would have been an inconvenient way to do multiple runs.

The general idea behind the rerun facility in FSE is that the data files remain unchanged and that the changes in the data are specified in a separate file e.g. `RERUNS.DAT`, which may contain the names and values of variables from any of the 'standard' data files that are read by the program. Thus, the file `RERUNS.DAT` may contain parameters from soil, plant and timer data files. In the first run using FSE, the values from the standard data files will be used. In subsequent runs those values are then automatically replaced by the values from the rerun file. Execution will continue until all the combination sets from `RERUNS.DAT` have been used. The output of the different runs is merged into one output file. An example rerun file is:

```
WLVI = 8.0; DSI = 0.18
WLVI = 6.8; DSI = 0.25
WLVI = 8.0; DSI = 0.25
```

This specifies three reruns with different values of `WLVI` and `DSI`. Unlike in CSMP and FST, variables have to be repeated even if their value does not need to change (like with `DSI`). (This is explained in more detail in Chapter 7.

It may be deduced from Fig. 4 that the control structure for the reruns should be programmed as a loop around the actual model. In Listing 9 the principle of the reruns is illustrated, using the main program of Listing 4 as a basis. To shorten the text, the contents of the main loop (IF... until END IF) have not been repeated.

Listing 9 Program skeleton showing the implementation of reruns

```
      PROGRAM RERUN
      IMPLICIT NONE
      LOGICAL TERMNL,OUTPUT
      INTEGER I1,ITASK,INSETS,IYEAR,IDOY
      REAL STTIME,DELT,PRDEL,FINTIM,DOY,TIME

      CALL RDSETS (...,'RERUNS.DAT', INSETS)

      DO 5 I1=0,INSETS
         CALL RDFROM (I1, ...)
*        initialization of time variables
         TERMNL = .FALSE.
         ITASK  = 1
         STTIME = 10.0
         DELT   = 1.0
         PRDEL  = 5.0
         FINTIM = 100.
         IYEAR  = 1984
         CALL TIMER2 (ITASK, STTIME, DELT, PRDEL, FINTIM,
     &                IYEAR, TIME  , DOY , IDOY , TERMNL, OUTPUT)
         CALL MODELS (ITASK, OUTPUT, TERMNL, TIME, DOY, DELT, ...)

*        run loop as long as TERMNL is not .TRUE.
10       IF (.NOT.TERMNL) THEN
         < main loop contents not repeated here, see Listing 4 >
```

```
         GOTO 10
         END IF

         ITASK = 4
         CALL MODELS (ITASK, OUTPUT, TERMNL, TIME, DOY, DELT, ...)
5     CONTINUE

      STOP
      END
```

The call to RDSETS detects the possible presence of the RERUNS.DAT file and analyses this data file if it exists. The return variable INSETS contains the number of rerun sets present in the rerun file; its value is zero if the rerun file is absent or empty. The subsequent DO-loop runs INSETS+1 times, because there is always one more than the number of sets in the rerun file (one run with standard data files + INSETS reruns). The value of the DO-loop counter (I1, the set number) is then used in the call to RDFROM to select a parameter set for the simulation. For I1 is zero, the standard data files will be used by the RD routines, for I1 larger than zero, the RD routines will automatically replace values with values from the rerun file. No changes are necessary in the subprocess descriptions, as these replacements are made internally in the RD routines. To the plant or the soil water balance routines it appears as if the values that are returned by the RD call originate from the standard data files !!! Therefore no changes are necessary in the calls to make reruns possible.

Before a rerun is started, a check is done to see if all the variables of the preceding set were used. If this is not the case, it is assumed that there is a typing error in the reruns file and the simulation is stopped.

4.4 Output of simulation results

As shown in Listing 4, output is organized from each subroutine separately. This avoids large argument lists to communicate output variables to the main program and limits the number of changes in the main program when, for instance, another plant model with different output variables is used.

By using a set of special subroutines (the OUT routines), output from different models running under the FSE-driver, can be written to the same output file in the form of tables. It is also possible to add print plots of selected variables to that output file. The use of the OUT routines considerably simplifies the generation of output files. The available routines are OUTDAT for output of single real variables, OUTARR for one-dimensional arrays of real variables and OUTPLT for print plots of selected variables. Note that OUTPLT can only be used for variables that have been 'dumped' with either the OUTDAT or OUTARR routines. The basic operations are shown in Listing 10. In this example, a table and a print plot of the function y = sin (x), y = cos (x), for x=0, π (with steps of π/20) are created. Also, both values are stored in an array of two elements which is written to the output table with a single OUTARR call.

Listing 10 Example program showing calling conventions of the OUT routines

Program

```
      PROGRAM SINE
      IMPLICIT NONE
      REAL PI,X,SINX,COSX,A(2)
```

```
      INTEGER I1
      PARAMETER (PI=3.141597)

      CALL OUTDAT (1, 20, 'X', 0.)          <- define X as independent variable,
      DO 10 I1=0,20                            use unit=20 for output file
         X = REAL (I1)*PI/20.
         SINX = SIN (X)
         COSX = COS (X)
         A(1) = SIN (X)
         A(2) = COS (X)
         CALL OUTDAT (2, 0, 'X', X)         <- store value of X
         CALL OUTDAT (2, 0, 'SINX', SINX)   <- store value of SINX
         CALL OUTDAT (2, 0, 'COSX', COSX)   <- store value of COSX
         CALL OUTARR (A, 'A', 1, 2)         <- store array A from 1st to 2nd element
10    CONTINUE

      CALL OUTDAT ( 4, 0, 'sin + cos', 0.)  <- create normal output table
      CALL OUTPLT ( 1, 'SINX')              <- define SINX to be plotted
      CALL OUTPLT ( 1, 'A(2)')              <- define A(2) to be plotted
      CALL OUTPLT ( 6, 'sin + cos')         <- create printplot with title
      CALL OUTDAT (99, 0, ' ', 0.)          <- delete temporary file
      STOP
      END
```

Output

```
*-----------------------------------------------------------------------------
* Run no.:  1, (Table output)
* sin + cos

      X          SINX          COSX          A(1)          A(2)
   .00000       .00000       1.0000        .00000       1.0000
   .15708       .15643        .98769       .15643        .98769
   .31416       .30902        .95106       .30902        .95106
   .47124       .45399        .89101       .45399        .89101
   .62832       .58779        .80902       .58779        .80902
   .78540       .70711        .70711       .70711        .70711
   .94248       .80902        .58778       .80902        .58778
   1.0996       .89101        .45399       .89101        .45399
   1.2566       .95106        .30902       .95106        .30902
   1.4137       .98769        .15643       .98769        .15643
   1.5708       1.0000     -0.21895E-05    1.0000     -0.21895E-05
   1.7279       .98769       -.15644       .98769       -.15644
   1.8850       .95106       -.30902       .95106       -.30902
   2.0420       .89101       -.45399       .89101       -.45399
   2.1991       .80902       -.58779       .80902       -.58779
   2.3562       .70710       -.70711       .70710       -.70711
   2.5133       .58778       -.80902       .58778       -.80902
   2.6704       .45399       -.89101       .45399       -.89101
   2.8274       .30901       -.95106       .30901       -.95106
   2.9845       .15643       -.98769       .15643       -.98769
   3.1416     -0.43790E-05  -1.0000     -0.43790E-05  -1.0000
```

```
            sin + cos
            Variable  Marker  Minimum value  Maximum value
            --------  ------  -------------  -------------
            SINX         1      -0.4379E-05      1.000
            A(2)         2       -1.000          1.000

            Scaling: Individual

X
 .00000     1---------------------------------------------------------------2
 .15708     I         1     I               I               I               2
 .31416     I               I   1           I               I             2 I
 .47124     I               I            1  I               I            2  I
 .62832     I               I               I     1         I         2     I
 .78540     I               I               I            1  I      2        I
 .94248     I               I               I               I  21           I
 1.0996     I               I               I              2I        1      I
 1.2566     I               I               I         2     I            1  I
 1.4137     I               I               I    2          I              1I
 1.5708     I-------------------------------2-------------------------------1
 1.7279     I               I          2    I               I              1I
 1.8850     I               I     2         I               I            1  I
 2.0420     I               I2              I               I        1      I
 2.1991     I            2  I               I               I   1           I
 2.3562     I        2      I               I            1  I               I
 2.5133     I     2         I               I     1         I               I
 2.6704     I  2            I            1  I               I               I
 2.8274     I 2             I   1           I               I               I
 2.9845     2         1     I               I               I               I
 3.1416     *---------------------------------------------------------------I
```

The `OUTDAT` and `OUTPLT` routines also have a task parameter as input (the first argument in the call statement), similar to the subprocess descriptions. The first call (with `ITASK=1`) to `OUTDAT` specifies that `X` will be the independent variable and that unit=20 can be used for the output file. Subsequent calls with `ITASK=2` instruct the subroutine `OUTDAT` to store the incoming names and numerical values in a temporary file (with unit=21). The number of name-value combinations that can be stored depends solely on free disk space and not on RAM memory. The first call to `OUTDAT` below the `DO`-loop (with `ITASK=4`) instructs the routine to create an output table using the data values stored in the temporary file. Different output formats may be chosen, dependent on the value of the task variable. Tab-delimited format (e.g. for spreadsheet programs such as EXCEL) can be generated with `ITASK=5`, two-column format (for TTPLOT) with `ITASK=6`. With any of these `ITASK` values, the string between quotes is written above the output.

The `OUTARR` routine (see Listing 10) is actually an 'interface' routine to `OUTDAT`. The routine internally generates names (like `A(1)` and `A(2)`) and calls `OUTDAT` repeatedly for each of these name-value combinations. The range of subscripts that should be generated by `OUTARR` is specified by the third and the fourth (last) subroutine arguments.

The first and second calls to `OUTPLT` define that `SINX` and `A(2)` should be plotted (up to 25 variables can be plotted per graph). The third call to `OUTPLT` (`ITASK=6`) instructs the routine to create a graph using the variable(s) that were defined with `ITASK=1`. Two different options for the width of the plot are available, 80 and 132 columns, and two different scalings, a common scale for all variables or individual scaling for each of the variables (see Table 2). This procedure can be repeated several times. Separate print plots can be made of dry weights, weather data etc.

Table 2 Task variable options that should be supplied to `OUTPLT` to generate the different print plot types

Scaling	Width	
	132	80
Individual	4	6
Common	5	7

4.5 Weather data

The weather data used in FSE are read from external files. The weather data file definition, however, is different from those for the RD routines. The weather data system used (called WEATHER), has been developed jointly by AB-DLO and TPE-WAU. It is especially suited for use in crop growth simulation models and has been documented in a separate report that can be obtained from the same sources as this documentation (Van Kraalingen *et al.*, 1991). It is an easy system to understand and it will outlined briefly using some introductory paragraphs from Van Kraalingen *et al.* (1991). A list of weather data from all around the world that are currently available on request is given in Stol (1994).

The weather data system basically consists of two parts: the weather data files and a reading program to retrieve data from those files. A single data file can contain, at the most, the daily weather data from one meteorological station for one particular year. The country name (abbreviated), station number and year to which the data refer are reflected in the name of the data file. An example weather data file can be found in Appendix I.

The reading program consists of a set of subroutines and functions, only two of which are intended to be called by the user (Listing 11, `STINFO` and `WEATHR`). The others are internal to the reading program.

A call to the first subroutine (`STINFO`) defines the country (`CNTR`), station code (`ISTN`), year number (`IYEAR`) and the name of the directory containing the weather data (`WTRDIR`). A control parameter (`IFLAG`) should also be supplied to indicate where possible messages of the system should be directed to (screen and/or log file), in addition a name must be given to the log file if that name should differ from the default name `WEATHER.LOG`. The subroutine `STINFO` returns the location parameters (longitude, `LONG`, latitude, `LAT` and altitude, `ELEV`) of the selected meteorological station, and two coefficients of the Ångström formula (`ANGA` and `ANGB`) pertaining to the selected station, the latter only if the irradiation data are derived from sunshine hours. The value of a status variable (`ISTAT`) indicates a possible error or warning (e.g. the requested data file does not exist). The location parameters can later be used to calculate e.g. daylength (from latitude) or average air pressure (from altitude).

After this initialization, weather data for specific days can be obtained by calls to the second subroutine (`WEATHR`) with day numbers starting from January 1st as 1, as input parameter. The output of `WEATHR` consists of six weather variables for that day and the value of the status variable `ISTAT` indicating a possible error or warning (e.g. missing data, data obtained by interpolation, requested day is out of range, etc.). The six weather variables are daily shortwave irradiation (`RDD`), minimum and maximum air temperature (`TMMN` and `TMMX`), vapour pressure (`VP`), wind speed (`WN`) and rainfall (`RAIN`). In Listing 11, the weather data for 1985 from the meteorological station in Wageningen are extracted from the file NL1.985.

Listing 11 Example of use of the weather data system

```
      PROGRAM EXTR
      IMPLICIT NONE
      INTEGER IFLAG, ISTN,  IYEAR, ISTAT, IDOY
      REAL LONG,LAT,ELEV,A,B,RDD,TMMN,TMMX,VP,WN,RAIN
      CHARACTER WTRDIR*80, CNTR*6

      IFLAG  = 1101               <-  errors to screen and log file, warnings to log  file
      WTRDIR = ' '                <-  weather files stored on current directory
      CNTR   = 'NLD'              <-  country code of The Netherlands
      ISTN   = 1                  <-  station number of met station in Wageningen
      IYEAR  = 1985               <-  year number

      CALL STINFO (IFLAG, WTRDIR, ' ', CNTR, ISTN, IYEAR,
     &             ISTAT, LONG  , LAT, ELEV, A, B)
      < location parameters of station are now available to the program >

      DO 10 IDOY=1,365
        CALL WEATHR (IDOY, ISTAT, RDD, TMMN, TMMX, VP, WN, RAIN)
        < weather for day= IDOY is known, calculations can be done >
10    CONTINUE

      STOP
      END
```

Subroutine `STINFO` can be called again at any time during program execution to change any of its input parameters. A call to subroutine `STINFO` with identical input parameters is also permitted (in fact this is done regularly in the FSE-driver). Similarly, subroutine `WEATHR` can be called repeatedly with any day number between 1 and 365 (or 366 in the case of a leap year and the data file indeed contains 366 records).

5 Simulation models under the FSE-driver

5.1 Communication between FSE-driver and user's model(s)

Chapter 4 dealt with a simplified FSE-driver and descriptions of some of the utility routines that the FSE-driver and submodels can use. Part of that Chapter was written to increase the understanding of how the FSE-driver works. A modeller, however, does not need to know the FORTRAN details of what is going on in the FSE-driver. He or she normally should only be bothered with how the FSE-driver drives one or more models. This Chapter therefore describes the information that is passed from the FSE-driver to the model and vice versa. Unlike the strongly simplified FSE driver of Chapter 4 we will discuss here the communication between the full FSE-driver and the model.

The procedure with the older FSE 1.0 version to drive a specific model, was to include CALL statements in four different places in the FSE 1.0 driver. Experience has learnt however, that this method was error-prone. In FSE 2.1 this has been changed. The FSE 2.1 driver now calls a MODELS interface routine and transfers relevant 'environment' variables (such as TIME, OUTPUT etc.) to this routine. The user now has to include only one CALL statement to his specific model in this MODELS routine instead of four times in the FSE 1.0 driver itself. This greatly simplifies coupling of different models to each other. This principle is shown in Fig. 5 in a somewhat more elaborated scheme as in Fig. 1:

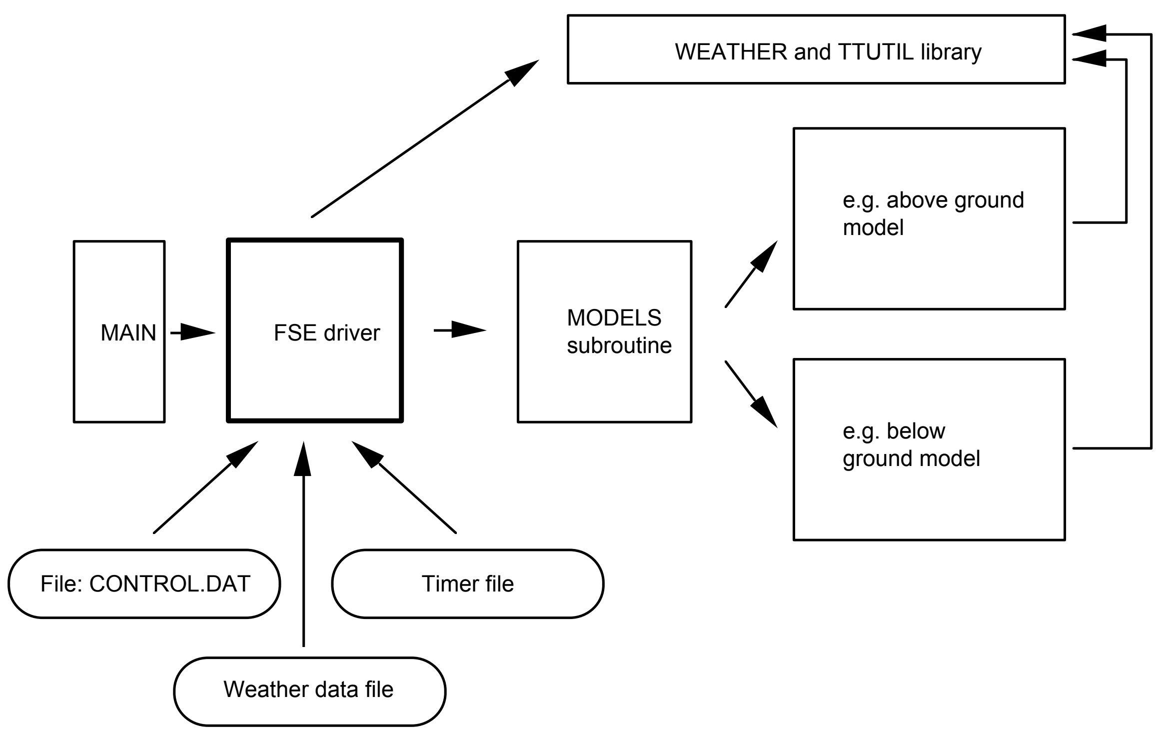

Figure 5 Schematic layout of the components of the FSE system

All execution starts with the MAIN program, this is, however, a one-line program with nothing more than a single call to the FSE-driver. The FSE-driver takes care of the following actions:

- reading of input and output file names from input file CONTROL.DAT
- reading of start and end rerun numbers from input file CONTROL.DAT
- running a loop across each rerun if a reruns file was found, the name of the reruns file is specified in input file CONTROL.DAT
- reading of time control, output control and weather control variables from the timer file, the name of the timer file is specified in input file CONTROL.DAT
- initialization of all models (through MODELS call with ITASK=1)
- running the dynamic simulation of all models while providing updated time variables (year, day etc.) and weather variables (through MODELS call with ITASK=2 in case of rate calculation, ITASK=3 in case of integration)
- termination of all models (through MODELS call ITASK=4), either when FINTIM is reached or when a model termination condition is fulfulled
- creation of output tables in the right format in an output file, the name of the output file is specified in input file CONTROL.DAT

The contents of the standard MODELS routine is given in Listing 12. The dummy arguments of this subroutine summarize the communication between the user's model and the FSE-driver. In Table 3 the dummy arguments of Listing 12 are explained.

Listing 12 Contents of the MODELS routine

```
      SUBROUTINE MODELS (ITASK , IUNITD, IUNITO, IUNITL,
     &                   FILEIT, FILEI1, FILEI2, FILEI3, FILEI4, FILEI5,
     &                   OUTPUT, TERMNL,
     &                   DOY   , IDOY  , YEAR  , IYEAR,      <- dummy arguments
     &                   TIME  , STTIME, FINTIM, DELT ,         (explained in Table 3)
     &                   LAT   , LONG  , ELEV  , WSTAT, WTRTER,
     &                   RDD   , TMMN  , TMMX  , VP   , WN, RAIN)
      IMPLICIT NONE

*     Formal parameters
      INTEGER   ITASK, IUNITD, IUNITO, IUNITL, IDOY, IYEAR
      CHARACTER FILEIT*(*), FILEI1*(*), FILEI2*(*)
      CHARACTER FILEI3*(*), FILEI4*(*), FILEI5*(*)
      LOGICAL   OUTPUT, TERMNL, WTRTER
      CHARACTER WSTAT*6
      REAL DOY,YEAR,TIME,STTIME,FINTIM,DELT
      REAL LAT,LONG,ELEV,RDD,TMMN,TMMX,VP,WN,RAIN

*     Local variables
*     <none>
      SAVE

      CALL MODEL (ITASK , IUNITD, IUNITO, IUNITL, <-  here calls can be put to the specific
     &            FILEIN,                             (user-made) simulation models, e.g.
     &            OUTPUT, TERMNL,                     of crop growth or soil water balance.
     &            DOY   , IDOY  , YEAR  , IYEAR,
     &            TIME  , STTIME, FINTIM, DELT ,
     &            LAT   , LONG  , ELEV  , WSTAT, WTRTER,
```

```
     &             RDD    , TMMN   , TMMX   , VP    , WN, RAIN)

      RETURN
      END
```

Table 3 Meaning of the variables that are communicated between the FSE-driver and the MODELS subroutine

Variable	Type	Meaning	Unit	Input or Output
ITASK	I4	Task that subroutine should perform (1 = initialization, 2 = rate calculation, output and finish conditions, 3 = integration, 4 = model termination)	-	I
IUNITD	I4	Unit number that can be used for reading of input files through call to RDINIT routine of TTUTIL	-	I
IUNITO	I4	Unit number used for output file	-	I
IUNITL	I4	Unit number used for log file	-	I
FILEIT	C*	Name of timer file (name originates from CONTROL.DAT)	-	I
FILEI1	C*	Name of input file no. 1 (if present in CONTROL.DAT)	-	I
FILEI2	C*	Name of input file no. 2 (if present in CONTROL.DAT)	-	I
FILEI3	C*	Name of input file no. 3 (if present in CONTROL.DAT)	-	I
FILEI4	C*	Name of input file no. 4 (if present in CONTROL.DAT)	-	I
FILEI5	C*	Name of input file no. 5 (if present in CONTROL.DAT)	-	I
OUTPUT	L4	Flag to indicate if output should be done (either .TRUE. or .FALSE), should not be set by user's model(s)	-	I
TERMNL	L4	Flag to indicate if simulation should stop (either .TRUE. or .FALSE), can be set by user's model(s)	-	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, six digit string variable, each digit corresponds with weather variable	-	I
WTRTER	L4	Flag whether weather can be used by model, is used by the FSE-driver to determine whether a model terminated itself because of missing weather	-	O
RDD	R4	Daily global shortwave radiation	$J/m^2/d$	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

Most of the variables are relevant at specific tasks only. For instance, names of the input files are only relevant at initialization. See Listing 14 and Appendix I for examples of this communication.

5.2 Use of input files by the FSE-driver

The FSE-driver needs three inputfiles. The most essential file is CONTROL.DAT. This file contains the names of the remaining input files for the FSE-driver (the timer file and the reruns file,see Appendix I for examples). This file also contains names of input files for the user's model(s) and the names of the output files. These names however are passed only to the MODELS routine. From the weather control variables of the timer file, the weather system determines which weather data file is required.

5.3 Program skeleton of empty FSE model

It will have become clear from the previous Chapters, that if a new subprocess is implemented within the FSE program, the new subroutine should distinguish the four different tasks. Listing 13 shows an empty subroutine that can be used as a starting point for a new subprocess description. This routine is the one that is called from the MODELS routine in Listing 12. Listing 13 is available on the floppy as the file SKELETON.FOR. The call to the new subroutine should be inserted in the MODELS subroutine once.

Listing 13 Empty subroutine that can be used to write a new model

```
      SUBROUTINE MODEL (ITASK , IUNITD, IUNITO, IUNITL,   <- Standard input and output
     &                  FILEIN,                              variables. The name FILEIN is
     &                  OUTPUT, TERMNL,                      used here instead of FILEI1,
     &                  DOY   , IDOY  , YEAR  , IYEAR,       because there is only "the"
     &                  TIME  , STTIME, FINTIM, DELT ,       input file.
     &                  LAT   , LONG  , ELEV  , WSTAT,
     &                  WTRTER, RDD   , TMMN  , TMMX ,
     &                  VP    , WN    , RAIN)
*     Title of the program                                <- Write the program's title here
*     <fill in your title here>                              as comment

      IMPLICIT NONE                                       <- Use this to avoid errors
*     Formal parameters                                   <- Declaration of every variable
      INTEGER ITASK , IUNITD, IUNITO, IUNITL                 in call definition
      INTEGER IDOY, IYEAR
      LOGICAL OUTPUT, TERMNL, WTRTER
      CHARACTER*(*) FILEIN, WSTAT
      REAL DOY, YEAR, TIME, STTIME, FINTIM, DELT
      REAL LAT, LONG, ELEV
      REAL RDD, TMMN, TMMX, VP, WN, RAIN
```

```
*     Standard local declarations                          <- Fill in model variable names
      INTEGER IWVAR                                           here
      CHARACTER WUSED*6
      REAL INTGRL

*     State variables, initial values and rates

*     Model parameters

*     Auxiliary variables                                  <- Declare arrays here for
                                                              interpolation with e.g. LINT.
*     LINT functions                                          Also declare actual and
                                                              maximum length of arrays
*     Declarations and values of constants                    here.

*     Used functions                                       <- Write here declarations of
                                                              functions such as LINT,
                                                              INTGRL etc.
      SAVE                                                 <- Avoid disappearance of
                                                              variable values
*     Code for the use of RDD, TMMN, TMMX, VP, WN, RAIN    <- Definition of which weather
*     (in that order) a letter 'U' indicates that the         variables are required by this
*     variable is Used in calculations                        model, in this example none
      DATA WUSED/'------'/

*     Check weather data availability                      <- The weather status code
      IF (ITASK.EQ.1.OR.ITASK.EQ.2.OR.ITASK.EQ.4) THEN        supplied by the FSE-driver is
         DO 10 IWVAR=1,6                                      compared with the weather
*           is there an error in the IWVAR-th weather         data this model requires. To
*           variable ?                                        achieve this, positions 1 to 6 in
            IF (WUSED(IWVAR:IWVAR).EQ.'U' .AND.               WUSED string are compared
     &          WSTAT(IWVAR:IWVAR).EQ.'4') THEN               with positions 1-6 in WSTAT
               WTRTER = .TRUE.                                string. These six positions
               TERMNL = .TRUE.                                correspond with the six
               RETURN                                         weather variables. If in
            END IF                                            WSTAT a particular position
10       CONTINUE                                             equals '4', then this weather
      END IF                                                  variable is missing. If that
                                                              position equals 'U' in WUSED,
                                                              than this weather variable is
                                                              essential for this model and
                                                              the run should terminate.
      IF (ITASK.EQ.1) THEN                                 <- Begin of initialization section
*        Initial
*        =======

*        Open input file                                   <- Open the input data file. Use
         CALL RDINIT (IUNITD, IUNITL, FILEIN)                 FSE-supplied units and
                                                              filename.
```

`*        Read initial states`	<- Reading of initial state variables can be done with: CALL RDSREA ('state',state), e.g.: CALL RDSREA ('WLVI',WLVI)
`*        Read model parameters`	<- Idem model parameters
`*        Read LINT functions`	<- Reading of AFGEN functions can be done with RDAREA calls
`         CLOSE (IUNITD)`	<- Close datafile
`*        Initial calculations`	<- Calculation of variables that have to be carried out only one
`*        Initially known variables to output`	<- Write variables to output file that do not change during simulation, use subroutine OUTDAT
`*        Send title(s) to OUTCOM`	<- You can send titles to the output file by: CALL OUTCOM ('title')
`*        Initialize state variables`	<- Assign state variables their initial value. E.g. WLV = WLVI (weight of leaves equals initial leaf weight)
`      ELSE IF (ITASK.EQ.2) THEN`	<- Begin of rate section
`*        Rates of change`	
`*        ===============`	<- Write here the rate and driving variable equations, make sure that the order of calculations is correct.
`*        Finish conditions`	<- Add finish conditions here
`*        IF (DVS.GT.2.) TERMNL = .TRUE.`	such as the one shown here as a comment line.
`*        Output section`	<- Send output variables to
`         IF (OUTPUT) THEN`	subroutine OUTDAT, use the
`*           CALL OUTDAT (2, 0, 'variable',variable)`	call that is shown here as a
`         END IF`	comment line.
`      ELSE IF (ITASK.EQ.3) THEN`	<- Beginning of integration
`*        Integration`	section.
`*        ===========`	
`*        state_var = INTGRL (state_var,rate_var,DELT)`	<- Use INTGRL function calls as shown here as a comment line.
`      ELSE IF (ITASK.EQ.4) THEN`	<- Begin of terminal section
`*        Terminal`	
`*        ========`	
`*        Terminal calculations`	<- Carry out terminal calculations such as harvest index etc.

```
*        Terminal output

         CONTINUE
      END IF

      RETURN
      END
```

<- Send terminal calculations to output, use subroutine OUTDAT calls as shown above

5.4 Adaption of an empty FSE model or existing FSE model

In this section possible modifications will be discussed to a FSE skeleton routine or an existing FSE routine. Changes in model behaviour caused by changes in one of the datafiles are discussed in Chapter 7.

You are strongly advised not to make changes in the subroutine structure and FSE-driver unless you are well acquainted with the theory underlying that structure. Changes will more often be made in the description of the plant or soil subprocesses. In general, you have to be more careful changing a FORTRAN program than one in CSMP or FST. If you understand the principles of the program, however, it is not difficult to implement modifications correctly and FORTRAN provides a much greater flexibility. A list of possible modifications will be discussed here. We assume that you know the syntax rules of FORTRAN.

5.4.1 Adaptation of subprocess calculations

- Begin by defining or modifying the integration section;
- initialize the state variables in the initial section;
- define the driving variables and the rate variables in the rate section;
- define the parameters in the initial section;
- check thoroughly that each of the rates that is used in the integral section appears to the left of an '=' sign in the rate section;
- check thoroughly from the top to the bottom that the sequence of the rate assignments in the rate section is correct. Each variable appearing to the right of an '=' sign must have been defined earlier in the subroutine (either in the rate section or in the initial section), or defined through the formal parameters of the subroutine.

If new variables are local to the subroutine, determine the exact position in the program where each variable should be assigned a value. Parameters and initial values of states are likely to be given their values in the initialization section, using one of the RD routines (see Chapter 4). These routines can extract the values of variables by their name from a data file. Different RD routines can be used depending on the data type of the variable (see Table 1). The routines `RDSREA`, `RDSINT`, and `RDAREA`, enable single reals, single integers and arrays of reals to be read. Driving variables should be assigned a value at the top of the rate section, rates should be defined in the proper order in the rate section. States should be integrated in the integration section.

It is especially important that rate calculations appear in the correct order. So any variable that appears to the right of the '=' sign of the modification you are making, and that is assigned a value in the rate section, should have been assigned a value *above* the line of the modification, i.e.:

Wrong

```
   ...
ELSE IF (ITASK.EQ.2) THEN
   ...
   A = B*...
   B = ...
ELSE IF (ITASK.EQ.3) THEN
   ...
```

Correct

```
   ...
ELSE IF (ITASK.EQ.2) THEN
   ...
   B = ...
   A = B*...
ELSE IF (ITASK.EQ.3) THEN
   ...
```

If variables are to be communicated among subprocesses, include them in the list of formal parameters too.

5.4.2 Adaptation of variables for output

In general the output list should appear at the end of the `ITASK=2` section. Output however, should only be generated if the output flag (`OUTPUT`) is on. Rates, states, and driving variables should be output here. A variable can be added to the output list by simply adding another call to `OUTDAT` (or `OUTARR`, if you want to output an array) in the output list, between the `IF-END-IF` lines. The names of variables in the output file will be in the same order as in the output list.

To obtain values of rate and state variables at every `PRDEL` and to have both pertain to the same time, it is essential thatthe location where output is generated is not changed.

5.4.3 Adaptation of finish conditions

Each subprocess can terminate the run by setting logical variable `TERMNL` to `.TRUE.`. However, in each subprocess description there is only one place where this can be done in such a way that corresponding states, driving variables, and rates are all output to file. This place is at the end of the `ITASK=2` section, as shown in Listing 4. Any additional finish condition should be added there, similar to the existing ones.

5.4.4 Adaptation of output titles

As shown in the listings of the modules, in the initial section the subroutine, `OUTCOM` is called that accepts a text string. This string is handled as a title by the output routines. Several subprocesses can send their title to the output routines and these titles are printed above each output table. There is no objection to several titles from a subprocess. The call to `OUTCOM` can be repeated several times with different text strings. A total of 25 titles can be handled by the output routines (identical ones are discarded).

5.4.5 Adaptation of print plotted variables

In the terminal sections of the subprocesses, calls to subroutine OUTPLT can be given. The calls with a '1' as the first argument define the list of variables to be print plotted. The call to OUTPLT with '4', '5', '6', or '7' as the first argument, actually creates the print plot for the selected variables. This has been described in more detail in Chapter 4.4. Variables can simply be modified or added in this section. Up to 25 variables can be print plotted in the same graph. More than one print plot can be made immediately after the first print plot by specifying a new set of variables to be plotted (with

ITASK=1 calls). Variables that have been written through OUTARR can be print plotted with the names under which they appear in the output table (e.g. RDF(1), RDF(2)). Print plots appear above output tables in the output file.

5.4.6 Adaptation of check on weather variables

The FSE 2.1 driver supplies to the model a single character string (WSTAT*6) from which each element represents the status code for a weather variable. Which weather variable must be available and which not is coded within the model subroutine with the WUSED character variable. Each position (1-6) corresponds with one of the six weather variables (irradiation (1), minimum temperature (2), maximum temperature (3), windspeed (4), vapour pressure (5) and rainfall (6)). A 'U' on a particular location means that the corresponding variable is Used and therefore may not be missing, a '-' allows that variable to be missing.

5.4.7 Adaptation of input file naming

The user can have the FSE driver provide the name(s) of the data file(s) to be used by the model(s) in the RDINIT calls. The names of five different data files are passed to the MODELS routine by the FSE driver through the character variables FILEI1, FILEI2, FILEI3, FILEI4 and FILEI5. These variables are read from the CONTROL.DAT file. It is however not essential to use the FILEI* variables. A data file names between quotes is also valid, it disables reruns on input file names however.

6 Main differences between FSE 1.0 and FSE 2.1

6.1 Improvements

- Reruns continue when errors have occurred in a particular run.
- Check on availability of weather data is now carried out in the user routine instead of in the driver program.
- All datafiles are now opened using one unit number, supplied by the driver.
- Output is now dependent only on the OUTPUT flag (in FSE 1.0, output was also dependent on the TERMNL flag).
- A real variable representing the year of simulation is available (YEAR), next to an integer variable (IYEAR).
- A file is opened to which routines can write logging information.
- Calls to user subroutines now don't have to be done from within the driver. A special subroutine (MODELS) is supplied from where this can be done.
- Initial and terminal output is possible.
- Names of weather variables have now been standardized.
- After reading a datafile with the RD routines, the CLOSE statement does not have to have a DELETE keyword. Leaving out the DELETE speeds up reruns because datafiles that have been parsed in the default run are not parsed again in reruns.
- Several output control variables have been added.
- Start and end run number of a series of runs can now be defined in the CONTROL.DAT file. For example one can omit the default run if required.
- All file names for input and output are now read from a control file (CONTROL.DAT). This increases flexibility.
- Input data can now be organized in tables.
- Variable names of input files can now be 31 characters long.

6.2 Changes

- A new timer subroutine with a different functionality is used (TIMER2). As a consequence of this, the variable DAYB is renamed to STTIME (STart TIME). The variable TIME itself now doesn't start at zero anymore but at STTIME. The following replacements have to be made in any model subroutine that is converted from FSE 1.0 to FSE 2.1:

occurrence:		must be replaced by:
DAYB	->	STTIME
TIME	->	TIME-STTIME
FINTIM	->	FINTIM+STTIME

- Finish conditions now have to be put at the end of the rate sections.
- The variable DOY is no longer sent to OUTDAT by the FSE-driver.

User Guide

7 How to operate FSE and its data files

This Chapter describes how to operate a ready to use model and explains the syntax of the corresponding data files. Chapter 5 explained how to modify the source code of the program. We assume here that you have successfully compiled and linked the FSE program and that you know how to start the execution of the model and are able to use an editor to create and modify data files.

As illustration, a complete FSE model, SUCROS , is given in Appendix I together with a listing of all input files.

7.1 Modification of data files

Most of the parameters and initial values of the state variables of the various subprocesses are read from data files. This has the advantage that the model does not have to be recompiled and linked if changes are applied only to data .

The data files that are needed to run the standard version of FSE are shown in Table 4.

Table 4 Data files necessary for using FSE

Name	Read by	Contents
CONTROL.DAT	FSE-driver (RDINIT)	Names of input and output files to be used.
TIMER.DAT	FSE-driver (RDINIT)	Time variables (year, time step, etc.), weather station, country, output control variables.
PLANT.DAT	MODEL (RDINIT)	Plant parameters and initial values of state variables.
Weather data files	FSE-driver (STINFO, WEATHR)	Daily weather data.
RERUNS.DAT (optional)	FSE-driver (RDSETS)	Defines reruns: either TIMER and plant parameters or name(s) of plant and timer data file.

Note: When models other than the ones supplied on the floppy disk are used with the FSE program it is possible that more data files are used with different names; see Paragraph 7.2.

The data files CONTROL.DAT, TIMER.DAT and PLANT.DAT have identical formats, and each variable in them may appear only once. The file RERUNS.DAT has basically the same syntax, except that it should consist of sets of identical variable names. Each such a set defines a rerun.

Syntax rules of CONTROL.DAT, TIMER.DAT and PLANT.DAT files:

- the file consists of variable names and one or more integer, real, double precision or string values, separated by an '=' sign. So: `PLMX = 20.`, is a valid specification, as is: `WTRDIR = 'NLD'`;
- the name of a variable cannot exceed 31 characters;
- for array variables, more than one value may follow the equal sign, separated by commas or spaces;
- identical numerical array values may be given as `n*`<numerical value>;

- variables may appear in the file in any order as long as this name is unique;
- comment lines start with '`*`' in the first column, or '`!`' in any column (the rest of the line is ignored);
- continuation character is '`,`' on preceding line, applies to arrays only;
- names of variables and numerical values can be given on the same line if separated by a single semicolon '`;`';
- Only the first 80 characters of each line on the data file are read;
- Supported data types are REAL, INTEGER and CHARACTER;
- Arrays may be organised in tables;
- No tabs or other control and extended ASCII characters are allowed in the file.

The syntax rules are illustrated in Listing 14.

Listing 14 Example data file

```
* example data file
A = 10.                    ! single real value
B = 0., 2., 3., 4.         ! array of four real values
C = 10., 20.,              ! array continued on next line
    30., 40.
D = 100*10.                ! array of 100 real values
E = 10.; F = 20.; G = 30.  ! more than one parameter on single line
H = 'PIET'                 ! string value
I = 5*'KLAAS'              ! array of string values
VOLGN   OBS   CALC      REM ! table
  1     10.4   10.1  'prachtig'
  2     7.8    8.    'te hoog'
  3     2.3    2.4   'ook mooi'
```

7.2 The CONTROL.DAT file

The CONTROL.DAT file contains the file names that are used during the execution of an FSE model. Also start and end rerun number can be defined in the rerun file. The CONTROL.DAT file was not present in the 1.0 version of FSE which made it impossible to do reruns on the names of input files. In FSE 2.1 reruns can now also be done on the names of all *input* files specified in the CONTROL.DAT except the name of the reruns file itself. In this file a distinction is made between names of input files (beginning with `FILEI`) and names of output files (beginning with `FILEO`). An example CONTROL.DAT is given in Appendix I.

FILEON and FILEOL
The `FILEON` and `FILEOL` variables are assigned the names of 1) the normal output file (containing the output table) and 2) the log file, respectively. Both `FILEON` and `FILEOL` may be set to the same file name.

FILEIT, FILEIR and FILEIn
The `FILEIT` and `FILEIR` variables are assigned the names of the timer file and the reruns file respectively. The file names mentioned above are used by the FSE-driver. The file names that can be used by the model(s) are specified through the variable names `FILEI1`, `FILEI2`, `FILEI3`, `FILEI4`, `FILEI5`. These names are optional and can be specified only when they are needed.

STRUN and ENDRUN
`STRUN` is an integer variable and specifies the run number of the first run. If `STRUN` is set to zero, execution will start with the default run, if set to 1, execution will start with the first rerun. The `ENDRUN` variable specifies the number of the last run. These names are optional and are specified only when they are needed.

7.3 The timer file

This data file specifies variables for:

- Weather control
  - directory in which the weather data are stored
  - country code
  - station number
- Time control
  - start time and finish time
  - time step of integration
  - start year
- Output control
  - time between different outputs
  - format of the output file
  - selection of output variables

An example file is given in Appendix I.

7.3.1 Weather control variables

Unlike in FSE 1.0, strings such as the weather directory and the country name can now be read by the RD routines. Consequently these strings can now be used in reruns, unlike in FSE 1.0. For a complete list of available weather data files, their corresponding country codes and station numbers, see Van Kraalingen *et al*. (1990) and Stol (1994).

WTRDIR
This line contains the weather directory. Very often, many weather data files will be used. For this reason it is convenient to store these data in a separate data directory. By supplying a directory name for the `WTRDIR` variable, you can direct the weather system to read weather data from that directory. Examples are:

```
WTRDIR = ' '                    <- use current directory
WTRDIR = 'WEATHER_DATA:'        <- example directory for AB-DLO/VMS system
WTRDIR = 'C:\SYS\WEATHER\'      <- example directory for IBM-PC and compatibles
WTRDIR = 'HD40:WEATHER:'        <- example directory for Apple Macintosh
```

CNTR
This line contains the abbreviated country code which is standardized as a 3 character ISO code. Examples are:

```
CNTR = 'NLD'                    <- country code for The Netherlands
CNTR = 'GBR'                    <- country code for United Kingdom
CNTR = 'ITA'                    <- country code for Italy
```

ISTN
This variable indicates the station number that should be used from the specified country. For example, when the country code is `NLD` (The Netherlands), `ISTN=1` and `IYEAR=1984` (from the timer control variables below), the daily weather data from Wageningen 1984 will be used by the model. During execution, the weather system will try to open a file by the name of NLD1.984 on the directory specified by `WTRDIR`.

IFLAG
This variable specifies what should be done with errors and warnings from the weather system. The `IFLAG` variable is an integer consisting of four digits:

	Digit value	
	0	1
first digit	warnings not to log file	warnings to log file
second digit	errors not to log file	errors to log file
third digit	warnings not to screen	warnings to screen
fourt digit	errors not to screen	errors to screen

For example when IFLAG = 1101 (the default value), it means that warnings and errors go to the log file, warnings are not sent to the screen and errors are sent to the screen.

7.3.2 Time control variables

STTIME, FINTIM, DELT and IYEAR
These variables represent the time parameters of the model. `STTIME` is the start time of the simulation; its value should be in between 1 and 365. In FSE 1.0 this variable was called `DAYB`. `FINTIM` is the finish time of the simulation (not the duration of the simulation). For example when `STTIME = 93.`, and `FINTIM = 103.`, `TIME` will start at 93 and the simulation will continue until `TIME = 103`. Note that in the FSE-driver various derived time variables are available such as the the day of the year (`DOY`). When a year boundary is crossed, `IYEAR` in the model is automatically increased (update of `TIME` and related variables is carried out by subroutine `TIMER2`). `DELT` is the time step of integration. The value of `DELT` cannot be chosen freely. Its value should be either a multiple of 1 (e.g. 2 or 10) or a multiple of `DELT` should equal 1 (e.g. 0.25, 0.10, 0.5). Often `DELT` is determined by the model that you are using. For SUCROS a value of one day is required.

7.3.3 Output control variables

PRDEL
The variable PRDEL indicates the time between consecutive outputs to file (the output interval). For example, when `PRDEL = 5.`, output is given each time that `TIME` has increased by the value of `PRDEL` (when `STTIME = 93.` output is at `TIME = 93.,98.,103.` etc.). Output can be fully suppressed by giving `PRDEL` the value `0.` When `PRDEL > 0` output is always given at the start of the simulation (`TIME` = `STTIME`) and when the simulation is terminated (either when `TIME` = `FINTIM` or some other finish criterion). So, by giving `PRDEL` a high value (e.g. 1000) intermediate outputs are suppressed and only the initial and terminal rates and states will be output.

IPFORM
The variable IPFORM defines whether an output table is required (no output table: IPFORM = 0) and if so in which format. A multiple column table (IPFORM = 4) is sufficient for normal printing and viewing. The normal table format is not very suitable to be imported in spreadsheet or graphics programs. Using IPFORM = 5, a tab-delimited multiple column table which is easily imported in programs such as Excel is generated.

DELTMP
The variable DELTMP defines whether the file with temporary output data (RES.BIN) should be deleted at termination of the simulation (DELTMP = 'N', do not delete, DELTMP = 'Y', delete). This file is built during the dynamic phase of the simulation and is read during the terminal phase of the simulation to generate the output file from. The temporary file is not of great value for normal purposes and can be deleted. However, there is the option of generating graphs directly from the RES.BIN file after termination of the simulation with the TTSELECT program. (TTSELECT is a graphical visualization tool for IBM-PC's and compatibles, available on request). For this special purpose the temporary file should not be deleted.

COPINF
The variable COPINF determines whether the input files mentioned in the CONTROL.DAT file must be copied to the output file. In FSE 1.0, input files were always copied to the output file before simulation started. In FSE 2.1, when copying is choosen (COPINF = 'Y'), input files are copied to the output file *after* writing the simulation results.

7.3.4 Optional output control variables

The above mentioned variables must all be present in the timer file, two variables, however, are optional.

PRSEL
The variabel PRSEL can be used to select a subset of the normal output variables without having to change the model. With PRSEL, e.g. several tables can be generated below each other. For example:

```
PRSEL = 'WSO','TADRW','<TABLE>','DVS','<TABLE>'
```

generates a table with WSO and TADRW after which a separate table with DVS is printed.

IOBSD
The variable IOBSD can be used to force output at days on which experimental observations were made. In many cases these observation data will not coincide with output intervals in the model unless PRDEL is set to unity (which may cause large output files to be generated). The IOBSD variable should be specified as a list of <observation_year>, <observation_day> combinations. A maximum of 50 <year, day> combinations can be defined here. Examples are:

```
IOBSD = 1984, 11, 1985, 117        <- Output is forced on day 11 in 1984 and day 117 in 1985
```

7.4 Other data files

The name and definition of other data files depend on the model used in conjunction with the FSE program. If you are working with a model like SUCROS, you are likely to be using a data file

PLANT.DAT. When simulating how lack of water limits growth, by adding a water balance subroutine, a file named SOIL.DAT containing soil parameters and initial values, may have to be present in the appropriate format. Normally, the general syntax rules as discussed above will apply to these data files.

7.5 The reruns file

If the reruns file is absent or empty, the model will execute only one single run, using the data from the standard data files. By creating a rerun file, the model will execute additional runs with different parameter combinations and/or initial values for the state variables (or even different input files). Therefore, the total number of runs made by the model is always one more than the number of rerun sets. Names of variables originating from different data files can be redefined in the same rerun file (see example). The format of the rerun files is identical to that of the other data files, except that the names of variables may appear in the file more than once. Arrays can also be redefined in a rerun file. The order and number of the variables should be the same in each set. A new set starts when the first variable is repeated. This is shown in the following example:

```
* example rerun file redefining the single variable DAYB from file
* TIMER.DAT and NPL from file PLANT.DAT

DAYB =  90.;  NPL = 250.             ! 1st rerun set
DAYB = 110.;  NPL = 210.             ! 2nd rerun set
DAYB = 110.;  NPL = 250.             ! etc.
DAYB = 130.;  NPL = 210.
DAYB = 130.;  NPL = 250.
```

Unlike reruns in the simulation languages CSMP or FST, each variable whose value is changed somewhere in the rerun file should be assigned a value in each set, even if that value is identical to the value in the previous set. An advantage of the method of defining reruns in FSE 2.1 is that it is much easier to identify the values of the parameters used by the model in a certain rerun (the parameters from the standard datafiles, modified by the parameters specified at that particular rerun). In CSMP or FST one has to inspect also previous reruns to see if parameters were modified there. The method of CSMP or FST clearly is a drawback when many reruns are required. Note that we discussed the implementation of reruns in the source code of the program in Chapter 4.

A feature of FSE 2.1 is that also the names of all input files (except the name of the reruns file and the output file) can be used in reruns. In this case not simply a few parameters are changed but whole parameter sets are swapped between reruns. This is shown in the following example:

```
FILEI1 = 'MODEL2.DAT'; FILEI2 = 'SOIL2.DAT'              ! 1st rerun set
FILEI1 = 'MODEL3.DAT'; FILEI2 = 'SOIL3.DAT'              ! 2nd rerun set
FILEI1 = 'MODEL4.DAT'; FILEI2 = 'SOIL4.DAT'              ! etc.
```

7.6 Running the model

The model does not require interactive input during execution. The runs have been specified completely in the data files. During execution, the model will display run number, year number and day number on the screen every time output to file is done. During execution, errors and warnings may occur from the weather system and/or from the other modules of the model. They generally consist of one line of text. If simulation is terminated by an error during the dynamic section of the run, the outputs generated before the error in that particular run occurred, are written to the temporary file but are not yet written to the output file until the terminal section of the model. If this occurs, data can be recovered from the temporary file, using the OUTREC program (OUTput RECovery, see the section on Error Recovery in this Chapter).

7.7 Examination of output

The standard model typically creates three output files: RES.DAT, MODEL.LOG and WEATHER.LOG.

RES.DAT
The RES.DAT file contains the output of the model with the output of reruns merged below each other in the file. The internal format of the output file RES.DAT depends on the value of the variable `IPFORM` from the timer file. If printplots were made with the OUTPLT routine, they appear before the output tables. If the `COPINF` variable was set to `'Y'` in the timer file, also copies of the input files mentioned in CONTROL.DAT will be present in the RES.DAT output file.

MODEL.LOG
This file may contain error and warning messages from routines used during the simulation. Messages about input variables whose values have been replaced by the rerun facility can be particularly useful. To make sure the execution of the model was without errors one has to inspect this file.

WEATHER.LOG
This file contains all the messages generated by the weather system. By default, all the comment headers of the data files, all warnings and all errors from the weather system are written to this log file. If shortly before termination of the model a message is displayed about possible errors and warnings from the weather system one has to look into this file and interpret the messages. Messages may be as unimportant as rainfall not being available when it was not used by the model but they can also be of a much more severe type.

It is also possible to view the output graphically on IBM-PC's and compatibles. This can be done with the TTSELECT program, provided `DELTMP` is set to `'N'` in the timer file. TTSELECT, however, is not part of the FSE standard distribution software.

7.8 Errors and warnings from the FSE program

Errors are defined as conditions that make it impossible to continue simulation. Examples are: a parameter value not found in a data file, or weather data not available for the year requested. A warning occurs in the case of unlikely events that do not, however, prevent continuation. Examples are: an attempt to search outside the range of the independent variable in a LINT function table, or one or more weather data that are not available for the requested day but are provided by interpolation.

All errors terminate model execution and a message to that effect is displayed on the screen. In some cases the error is also written to the output file. Warnings are displayed on the screen and are sometimes also written to the output file (remember, warnings allow simulation to continue).

The weather system can also generate errors and warnings. Unlike errors from other sections of the model, the weather system itself never terminates execution of the model. It is the FSE-driver that subsequently terminates the simulation run. The default is that errors from the weather system are written to the screen and the log file WEATHER.LOG. Warnings are written to the log file only.

The general syntax of errors and warnings is similar:

```
ERROR in <module name>: <error text>

WARNING from <module name>: <warning text>
```

for example:

```
ERROR in LIMIT: argument error, MIN = 10.5, MAX = 8.3

WARNING from OUTDAT: zero length variable name
```

7.9 Error recovery

If a run is terminated by some error from the model, the output file RES.DAT will not contain the results of that specific run. But the results up till the error occurred are written to the temporary file RES.BIN. This file can be converted into an output table by running the output recovery program OUTREC. This program requests an integer number from the user. A standard output table of every run stored in RES.BIN is generated by '14' (the default), '15' generates a tab-delimited table (meant to be imported in Excel), '16' generates an output of only two columns at a time. The output table will be written to the file RECOVER.DAT so that any existing RES.DAT file is not deleted.

The listing of the OUTREC program is given in Appendix I.

8 Installing the FSE program

8.1 Requirements for running the FSE program

There are few requirements for running the FSE program. Any standard FORTRAN-77 compiler on any computer should be able to compile the program successfully, because it has been developed and tested using DEC Fortran on VAX and AXP systems using Open VMS for VAX and Open VMS for AXP respectively, Apple Macintosh using Language Systems Fortran, IBM compatible PC's using Microsoft Fortran 5.1 and Microsoft Professional Powerstation 1.0 and Atari 520ST+ using PROFortran.

The minimum RAM memory requirement and the necessity of a separate floating point processor depends on the computer and the compiler. Free RAM memory should be at least 512 kb. A mathematical coprocessor is in general not required but will often speed up calculations considerably. A free hard disk space of about 1 Mb is required.

If you intend to do serious development work with the FSE program or any other FORTRAN program, we recommend you to use the FORCHECK program to check your source code for errors (see references). In FORCHECK the syntax, variable declaration, argument passing and standard FORTRAN checking capability is much better than in most compilers and this will save you much time instead of debugging any FORTRAN program.

The FSE 2.1 program relies heavily on the TTUTIL and WEATHER utility libraries. On the floppy you find ready-made versions of the TTUTIL and WEATHER object libraries for Microsoft FORTRAN 5.1 on IBM-PC or compatible computers. For each library there is a separate floppy disk available with source files that you can use to build your own TTUTIL or WEATHER libraries. Send in a request to the suppliers of FSE to obtain them (mentioned in the Summary).

If you are working on an IBM-PC or compatible computer, you are advised to use Microsoft FORTRAN 5.1 or any later version. Two utility programs FORTRAN.EXE and LINK.EXE are available on the floppy disk to be used with this compiler (to simplify compilation and linking). Any other standard FORTRAN 77 compiler on any machine with at least 512 kb RAM can also be used, but this requires renewed compilation of the TTUTIL and WEATHER source files, also you will not be able to use the userfriendly FORTRAN and LINK programs.

8.2 Contents of the disk

The disk you receive is a 3.5" high density disk and has been formatted for IBM-PC's and compatibles. If you are working on another machine and have no way to transfer the source files to your machine, send a request to one of the addresses mentioned in the introduction to obtain the programs in another disk format (do not forget to specify your hardware configuration).

In the following we assume that you have received the FSE program with the spring wheat version of SUCROS as a plant routine. Any other model that is programmed using FSE will have a comparable directory structure on the floppy disk. The contents of the disk is:

Directory	Filename	Contents of the file
A:\	MODEL.FOR MODEL.EXE FSE.FOR CONTROL.DAT MODEL.DAT TIMER.DAT RERUNS.DAT SKELETON.FOR	Spring wheat SUCROS subroutines Executable file Source of FSE-driver program as subroutine Data file with names of input and output files Data file with plant parameters and initial state variables Data file with weather, time and control variables Data file with example rerun file Empty FSE model, to be adapted by the user
A:\TOOLS	FORTRAN.EXE LINK.EXE OUTREC.EXE OUTREC.FOR	Tool for easy compilation (for MS-Fortran 5.1) Tool for easy linking (for MS-Fortran 5.1) (warning: not identical to LINK.EXE from MS-Fortran 5.1) Tool for output recovery after a program crash Source of OUTREC program
A:\LIBS	TTUTIL.LIB DRIVERS.LIB WEATHER.LIB	TTUTIL object library (for MS-Fortran 5.1) Drivers library containing FSE 2.1 driver (for MS-Fortran 5.1) WEATHER object library (for MS-Fortran 5.1)
A:\WEATHER	NLD1....	Weather data, Netherlands, Wageningen, ...1970-1990

N.B.:

- The SUCROS crop growth model is put on the floppy only for illustration purposes. This is not, by definition, the latest release of SUCROS. To obtain the latest version of SUCROS send in a written request to AB-DLO or TPE-WAU.
- The source file of the FSE-driver is supplied as a separate file.
- The FORTRAN.EXE and LINK.EXE tools are meant for easy compilation and linking (provided the compiler is installed following the requirements below). It is, however, not necessary to use them if you are an experienced user of the Microsoft compiler.

8.3 General installation of FSE on IBM-compatibles using Microsoft FORTRAN 5.1

This section explains the installation of FSE *and* the required installation of Microsoft FORTRAN 5.1.

1) Install the compiler on the directory C:\SYS\F77. Make sure you have also installed the compiler's library on the directory C:\SYS\F77 as follows: large memory model, floating point emulator, no C and no MS FORTRAN 3.30 compatibility. This library will have the name: LLIBFORE.LIB. Make sure that the directory of the compiler files (FL.EXE, etc.) is not 'in' the PATH, contrary to what is suggested by the Microsoft installation procedure.

2) Create a new directory on your hard disk and move the files to that directory. For example:
```
MD FSE2_1 <Enter>
CD FSE2_1 <Enter>
```

3) Now install FSE 2.1 by typing:
```
XCOPY A:\*.* C: /S <Enter>
```
(The `XCOPY /S` command copies the files *and* directory structure to the hard disk.)

4) Move the files from the A:\LIBS directory to the directory C:\SYS\F77.

5) After installation move the files from the TOOLS directory to a directory that is in the PATH or add the TOOLS directory to the PATH (not necessary if you will use your own utilities to compile and link).

6) Create a directory C:\TMP.

7) Add to your C:\AUTOEXEC.BAT file the following statements:
```
SET LIB=C:\SYS\F77
SET TMP=C:\TMP
```

8) Restart your PC.

8.4 Using the FORTRAN.EXE and LINK.EXE tools to compile and link FSE

If you would like to use the FORTRAN.EXE and LINK.EXE tools it is important to follow the installation instructions of the previous Chapter carefully. After successfull installation you can compile the FORTRAN files, link with the object libraries, and run the program using the following commands (in these examples "MODEL" stands for the name of the actual simulation model):

`FORTRAN MODEL <Enter>`	<- normal compilation of MODEL
`LINK MODEL,DRIVERS/L,TTUTIL/L,WEATHER/L <Enter>`	<- linking of MODEL.OBJ with DRIVERS.LIB, TTUTIL.LIB and WEATHER.LIB
`MODEL`	<- running of MODEL.EXE

Preparation for debugging with CodeView (the debugger of the Microsoft programming languages) can be done with:

`FORTRAN MODEL /DEBUG <Enter>`	<- debug compilation of MODEL
`LINK MODEL,DRIVERS/L,TTUTIL/L,WEATHER/L /DEBUG<Enter>`	<- linking of MODEL.OBJ with DRIVERS.LIB, TTUTIL.LIB and WEATHER.LIB with debug options
`CV MODEL`	<- debugging of MODEL.EXE

After successful compilation and linking, repeated execution can be invoked by typing the name of the program (e.g. `MODEL`).

The executable files, created by these tools, will run on any IBM-compatible computer provided the proper amount of RAM memory is present. They do not require a coprocessor but will use one if it is present.

The FORTRAN.EXE and LINK.EXE tools have many more features that cannot be described in this short Chapter. Additional documentation about these tools can be requested from the suppliers of FSE (address given in the Introduction).

8.5 Working with other FORTRAN compilers on IBM PC's and compatibles

We have no experience with other compilers on IBM-PC's and compatibles. You will probably have to obtain the source files from the TTUTIL and WEATHER library and create object libraries from them. Also, you will have to figure out how to link the FSE program with these newly created object libraries.

8.6 Working on a VAX/VMS or AXP/VMS computer of AB-DLO or TPE-WAU

The first step is to transfer the source files from the root of the floppy disk to your work directory on the VMS machine. Files from other directories on the floppy are not useful on these systems.

Simple compilation, linking and execution can be done with :

`$ FORTRAN/CHECK=BOUNDS/STANDARD MODEL,FSE`	<-	compilation of MODEL.FOR and FSE.FOR
`$ LINK MODEL,FSE,TTUTIL/L,WEATHER/L`	<-	linking of MODEL.OBJ with FSE.OBJ, TTUTIL.LIB and WEATHER.LIB
`$ RUN MODEL`	<-	running of MODEL.EXE

(If necessary consult somebody from the computer department of AB-DLO or TPE-WAU to find out how to define the TTUTIL and WEATHER logicals).

The TTUTIL and WEATHER object libraries can be linked automatically after the following commands are given (include these in your LOGIN.COM for permanent use. Be aware, however, that from then on these two libraries are always used during linking even if you don't need them):

```
$ DEFINE LNK$LIBRARY TTUTIL:
$ DEFINE LNK$LIBRARY_1 WEATHER:
```

The link command is now shorter:

```
$ LINK file
```

Weather data can be accessed directly by specifying the logical name WEATHER_DATA: as the value of the WTRDIR directory variable in the TIMER.DAT file.

8.7 Working on another VAX or AXP computer

Before you can start work with the FSE program you should obtain the source files of TTUTIL and WEATHER. These can be sent to you either by ordinary mail or by e-mail. Refer to the Introduction of this manual for the ordinary and e-mail addresses.

8.8 Working on an Apple Macintosh using MacFortran/020 v2.3

Generally the easiest solution is to obtain the two object libraries compiled for MacFortran/020 v2.3 from the suppliers of FSE (address is mentioned in the Introduction) and link these to the main program (it is difficult to create a library yourself because the routines have to be inserted in a specific order as the linker cannot resolve backward references). Another solution is to add INCLUDE statements to the required routines at the end of your program, although this may give problems with the debugger.

The object library and/or source files of TTUTIL and WEATHER can be obtained by submitting a request to the AB-DLO address mentioned in the Introduction (specify your processor and coprocessor type !).

8.9 Working on an Apple Macintosh using Language Systems Fortran

The object library and/or source files of TTUTIL and WEATHER for this compiler can be obtained by submitting a request to the AB-DLO address mentioned in the introduction (specify your processor and coprocessor type !).

References and further reading

IBM, 1975.
Continuous System Modeling Program III. General system information manual (GH19-7000) and users manual (SH19-7001-2). IBM Data Processing Division, White Plains, New York.

FORCHECK (no year).
A FORTRAN-77 Verifier and Programming aid, E.W. Kruyt, Dept. of Physiology. Leiden University, PO Box 9604, 2300 RC Leiden, The Netherlands.

Goudriaan, J. & H.H. van Laar, 1994.
Simulation of crop growth processes. J. Goudriaan & H.H. van Laar (Eds), Kluwer Academic Publishers, Dordrecht, The Netherlands, 238 pp.

Haar, L.G.J. ter, 1983.
FORTRAN 77, programmers pocket guide. Nederlands Normalisatie Instituut, 47 pp.

Kraalingen, D.W.G. van, 1991.
The FSE system for crop simulation. Simulation Report CABO-TT nr. 23. Centre for Agrobiological Research and Dept. of Theoretical Production Ecology. Wageningen, The Netherlands, 77 pp. (available on request).

Kraalingen, D.W.G. van, C. Rappoldt & H.H. van Laar, 1994.
The Fortran Simulation Translator (FST), a simulation language. In: J. Goudriaan & H.H. Van Laar (Eds), Simulation of crop growth processes, Kluwer Academic Publishers, Dordrecht, The Netherlands, 238 pp.

Kraalingen, D.W.G. van & F.W.T. Penning de Vries, 1990.
The FORTRAN version of CSMP MACROS. Simulation Report CABO-TT nr. 21. Centre for Agrobiological Research and Dept. of Theoretical Production Ecology, Wageningen, The Netherlands, 145 pp. (available on request).

Kraalingen, D.W.G. van & C. Rappoldt, 1989.
Subprograms in simulation models. Simulation Report CABO-TT nr. 18. Centre for Agrobiological Research and Dept. of Theoretical Production Ecology, Wageningen, The Netherlands, 54 pp. (available on request).

Kraalingen, D.W.G. van, W. Stol, P.W.J. Uithol & M. Verbeek, 1991.
User Manual of CABO/TPE Weather System, CABO/TPE internal communication, 27 pp. (available on request).

Leffelaar, P.A.L., 1993.
On systems analysis and simulation of ecological processes, with examples in CSMP and FORTRAN, Kluwer Academic Publishers, Dordrecht, The Netherlands, 294 pp.

Meissner, L.P. & E.I. Organick, 1984.
FORTRAN 77, featuring structured programming. Addison-Wesley publishing company, 500 pp.

Penning de Vries, F.W.T. & H.H. van Laar, 1982.
Simulation of plant growth and crop production. Simulation Monograph, PUDOC, Wageningen, 308 pp.

Rappoldt, C. & D.W.G. van Kraalingen, 1990.
FORTRAN utility library TTUTIL. Simulation Report CABO-TT no. 20. Centre for Agrobiological Research and Dept. of Theoretical Production Ecology, Wageningen, The Netherlands, 54 pp. (available on request).

Stol, W., 1994.
Synoptic and climatic data for agro-ecological research. The AB-MET database. Simulation

Report CABO-TT, no. 37. Centre for Agrobiological Research and Dept. of Theoretical Production Ecology, Wageningen, The Netherlands, 103 pp. (available on request).
Wagener, J.L., 1980.
FORTRAN 77, principles of programming. John Wiley & Sons, New York, 370 pp.
Wit, C.T. de & J. Goudriaan, 1978.
Simulation of ecological processes. Simulation Monograph, PUDOC, Wageningen, 175 pp.

Appendix I: Program and data file listings

The file order in this Appendix is as follows:

MODEL.FOR	Main program and SUCROS subroutines,
FSE.FOR	FSE-2.1-driver,
CONTROL.DAT	Data file containing names of input and output files,
TIMER.DAT	Data file containing time, weather and output control variables,
MODEL.DAT	Data file containing parameters and initial values of the states for a crop,
NLD1.980	Weather data file
RERUNS.DAT	Data file with some reruns,
OUTREC.FOR	Program to recover output after an unexpected model crash.

File: MODEL.FOR

```
      PROGRAM MAIN
      IMPLICIT NONE
      CALL FSE
      END

*----------------------------------------------------------------------*
* SUBROUTINE MODELS                                                    *
* Authors: Daniel van Kraalingen                                       *
* Date   : 5-Jul-1993                                                  *
* 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)
      IMPLICIT NONE

*     Formal parameters
      INTEGER   ITASK, IUNITD, IUNITO, IUNITL, IDOY, IYEAR
      CHARACTER FILEIT*(*), FILEI1*(*), FILEI2*(*)
      CHARACTER FILEI3*(*), FILEI4*(*), FILEI5*(*)
      LOGICAL   OUTPUT, TERMNL, WTRTER
      CHARACTER 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
      CALL MODEL (ITASK , IUNITD, IUNITO, IUNITL,
     &            FILEI1,
     &            OUTPUT, TERMNL,
     &            DOY   , IDOY  , YEAR  , IYEAR,
     &            TIME  , STTIME, FINTIM, DELT ,
     &            LAT   , LONG  , ELEV  , WSTAT, WTRTER,
     &            RDD   , TMMN  , TMMX  , VP   , WN, RAIN)
      RETURN
      END

*----------------------------------------------------------------------*
* SUBROUTINE MODEL                                                     *
* Authors:                                                             *
* Date   :                                                             *
* Purpose:                                                             *
*                                                                      *
* 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  *
* 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           *
*----------------------------------------------------------------------*

      SUBROUTINE MODEL (ITASK , IUNITD, IUNITO, IUNITL,
     &                  FILEIN,
     &                  OUTPUT, TERMNL,
     &                  DOY   , IDOY  , YEAR  , IYEAR,
     &                  TIME  , STTIME, FINTIM, DELT ,
     &                  LAT   , LONG  , ELEV  , WSTAT, WTRTER,
```

```
     &                   RDD   , TMMN  , TMMX  , VP   , WN, RAIN)

*     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*(*) FILEIN, WSTAT
      REAL DOY, YEAR, TIME, STTIME, FINTIM, DELT
      REAL LAT, LONG, ELEV, RDD, TMMN, TMMX, VP, WN, RAIN

*     Standard local declarations
      INTEGER IWVAR
      CHARACTER WUSED*6

*     State variables, initial values and rates
      REAL DVS   , IDVS  , DVR
      REAL LAI   , ILAI  , RLAI
      REAL EAI   , IEAI  , REAI
      REAL WRT   , WRTI  , GRT
      REAL WLVG  , WLVI  , RWLVG
      REAL WLVD  , WLVDI , DLV
      REAL WST   , WSTI  , GST
      REAL WSO   , WSOI  , GSO
      REAL TNASS , ZERO  , RTNASS

*     Model parameters
      REAL AMX   , ASRQLV, ASRQRT, ASRQSO, ASRQST
      REAL CFLV  , CFRT  , CFSO  , CFST  , DOYEM
      REAL EAR   , EFF   , FRTRL , KDF   , LAICR
      REAL LATT  , MAINLV, MAINRT, MAINSO, MAINST
      REAL Q10   , RGRL  , SCP   , SLA   , TBASE
      REAL TREF

*     Auxiliary variables
      REAL AMAX  , AMDVS , AMTMP , ASRQ  , CHKDIF
      REAL CHKFL , CHKIN , CO2LV , CO2RT , CO2SO
      REAL CO2ST , DAVTMP, DAYL  , DDTMP , DLAI
      REAL DS0   , DTEFF , DTGA  , DTMAX , DTMIN
      REAL DTR   , EMERG , FLV   , FRT   , FSH
      REAL FSO   , FST   , GLAI  , GLV   , GPHOT
      REAL GTW   , HI    , MAINT , MAINTS, MNDVS
      REAL RDR   , RDRDV , RDRSH , TADRW , TAI
      REAL TDRW  , TEFF  , TRANSL, WLV   , LA0, NPL

*     LINT functions
      REAL AMDVST
      INTEGER IMAMDV, ILAMDV
      PARAMETER (IMAMDV =  40)
      DIMENSION AMDVST(IMAMDV)
      REAL AMTMPT
      INTEGER IMAMTM, ILAMTM
      PARAMETER (IMAMTM =  40)
      DIMENSION AMTMPT(IMAMTM)
      REAL DVRRT
      INTEGER IMDVRR, ILDVRR
      PARAMETER (IMDVRR =  40)
      DIMENSION DVRRT (IMDVRR)
      REAL DVRVT
      INTEGER IMDVRV, ILDVRV
      PARAMETER (IMDVRV =  40)
      DIMENSION DVRVT (IMDVRV)
      REAL FLVTB
      INTEGER IMFLVT, ILFLVT
      PARAMETER (IMFLVT =  40)
      DIMENSION FLVTB (IMFLVT)
```

```
      REAL FSHTB
      INTEGER IMFSHT, ILFSHT
      PARAMETER (IMFSHT =  60)
      DIMENSION FSHTB (IMFSHT)
      REAL FSTTB
      INTEGER IMFSTT, ILFSTT
      PARAMETER (IMFSTT =  40)
      DIMENSION FSTTB (IMFSTT)
      REAL RDRT
      INTEGER IMRDRT, ILRDRT
      PARAMETER (IMRDRT =  40)
      DIMENSION RDRT  (IMRDRT)

*     Declarations and values of constants
*     none

*     Used functions
      REAL LINT  , INSW  , LIMIT , NOTNUL, INTGRL
      SAVE

*     Code for the use of RDD, TMMN, TMMX, VP, WN, RAIN  (in that order)
*     a letter 'U' indicates that the variable is Used in calculations
      DATA WUSED/'UUU---'/

*     Check weather data availability
      IF (ITASK.EQ.1.OR.ITASK.EQ.2.OR.ITASK.EQ.4) THEN
         DO 10 IWVAR=1,6
*           is there an error in the IWVAR-th weather variable ?
            IF (WUSED(IWVAR:IWVAR).EQ.'U' .AND.
     &          WSTAT(IWVAR:IWVAR).EQ.'4') THEN
               WTRTER = .TRUE.
               TERMNL = .TRUE.
               RETURN
            END IF
10       CONTINUE
      END IF

      IF (ITASK.EQ.1) THEN
*        Initial
*        =======

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

*        Read initial states
         CALL RDSREA ('IDVS  ',IDVS  )
         CALL RDSREA ('IEAI  ',IEAI  )
         CALL RDSREA ('ILAI  ',ILAI  )
         CALL RDSREA ('LA0   ',LA0   )
         CALL RDSREA ('NPL   ',NPL   )
         CALL RDSREA ('WLVDI ',WLVDI )
         CALL RDSREA ('WLVI  ',WLVI  )
         CALL RDSREA ('WRTI  ',WRTI  )
         CALL RDSREA ('WSOI  ',WSOI  )
         CALL RDSREA ('WSTI  ',WSTI  )
         CALL RDSREA ('ZERO  ',ZERO  )

*        Read model parameters
         CALL RDSREA ('AMX   ',AMX   )
         CALL RDSREA ('ASRQLV',ASRQLV)
         CALL RDSREA ('ASRQRT',ASRQRT)
         CALL RDSREA ('ASRQSO',ASRQSO)
         CALL RDSREA ('ASRQST',ASRQST)
         CALL RDSREA ('CFLV  ',CFLV  )
         CALL RDSREA ('CFRT  ',CFRT  )
         CALL RDSREA ('CFSO  ',CFSO  )
         CALL RDSREA ('CFST  ',CFST  )
         CALL RDSREA ('DOYEM ',DOYEM )
         CALL RDSREA ('EAR   ',EAR   )
         CALL RDSREA ('EFF   ',EFF   )
```

```
      CALL RDSREA ('FRTRL ',FRTRL )
      CALL RDSREA ('KDF   ',KDF   )
      CALL RDSREA ('LAICR ',LAICR )
      CALL RDSREA ('LATT  ',LATT  )
      CALL RDSREA ('MAINLV',MAINLV)
      CALL RDSREA ('MAINRT',MAINRT)
      CALL RDSREA ('MAINSO',MAINSO)
      CALL RDSREA ('MAINST',MAINST)
      CALL RDSREA ('Q10   ',Q10   )
      CALL RDSREA ('RGRL  ',RGRL  )
      CALL RDSREA ('SCP   ',SCP   )
      CALL RDSREA ('SLA   ',SLA   )
      CALL RDSREA ('TBASE ',TBASE )
      CALL RDSREA ('TREF  ',TREF  )

*     Read LINT functions
      CALL RDAREA ('DVRVT ',DVRVT ,IMDVRV,ILDVRV)
      CALL RDAREA ('DVRRT ',DVRRT ,IMDVRR,ILDVRR)
      CALL RDAREA ('AMDVST',AMDVST,IMAMDV,ILAMDV)
      CALL RDAREA ('AMTMPT',AMTMPT,IMAMTM,ILAMTM)
      CALL RDAREA ('FSHTB ',FSHTB ,IMFSHT,ILFSHT)
      CALL RDAREA ('FLVTB ',FLVTB ,IMFLVT,ILFLVT)
      CALL RDAREA ('FSTTB ',FSTTB ,IMFSTT,ILFSTT)
      CALL RDAREA ('RDRT  ',RDRT  ,IMRDRT,ILRDRT)
      CLOSE (IUNITD)

*     Initial calculations

*     Initially known variables to output

*     Send title(s) to OUTCOM
      CALL OUTCOM ('Crop growth for potential production (SUCROS1)')

*     Initialize state variables
      DVS    = IDVS
      LAI    = ILAI
      EAI    = IEAI
      WRT    = WRTI
      WLVG   = WLVI
      WLVD   = WLVDI
      WST    = WSTI
      WSO    = WSOI
      TNASS  = ZERO

   ELSE IF (ITASK.EQ.2) THEN
*     Rates of change
*     ===============

      EMERG     = INSW(DOY-DOYEM, 0., 1.)
      AMDVS     = LINT(AMDVST,ILAMDV, DVS)
      MAINTS    = MAINLV*WLVG + MAINST*WST + MAINRT*WRT + MAINSO*WSO

*     1.8 DRY MATTER PARTITIONING

      FSH       = LINT(FSHTB,ILFSHT, DVS)

      FLV       = LINT(FLVTB,ILFLVT, DVS)

      FST       = LINT(FSTTB,ILFSTT, DVS)

*     1.10 LEAF AND EAR DEVELOPMENT

      TAI       = 0.5 * EAI + LAI
      RDRSH     = LIMIT(0., 0.03, 0.03 * (LAI-LAICR) / LAICR)

      WLV       = WLVG  + WLVD

*     DTR       = AFGEN(DTRT,  DOY) * 1.E06
      DTR       = RDD
*     DTMAX     = AFGEN(TMAXT, DOY)
      DTMAX     = TMMX

*     DTMIN     = AFGEN(TMINT, DOY)
      DTMIN     = TMMN
      CHKFL     = TNASS * (12./44.)
      CO2RT     = 44./12. * (ASRQRT*12./30. - CFRT)
      CO2LV     = 44./12. * (ASRQLV*12./30. - CFLV)
      CO2ST     = 44./12. * (ASRQST*12./30. - CFST)
      CO2SO     = 44./12. * (ASRQSO*12./30. - CFSO)
      TADRW     = WLV   + WST  + WSO
      MNDVS     = WLVG / NOTNUL(WLV)

      DAVTMP    = 0.5 * (DTMAX + DTMIN)
      DDTMP     = DTMAX - 0.25 * (DTMAX-DTMIN)

*     1.13 CARBON BALANCE CHECK

      CHKIN     = (WLV - WLVI) * CFLV + (WST - WSTI) * CFST + (WRT -
     $   WRTI) * CFRT + WSO * CFSO
      FRT       = 1. - FSH

      FSO       = 1. - FLV - FST
      RDRDV     = INSW(DVS-1.0, 0., LINT(RDRT,ILRDRT, DAVTMP))
      TDRW      = TADRW + WRT

      HI        = WSO / NOTNUL(TADRW)
      TEFF      = Q10**((DAVTMP-TREF)/10.)
      DVR       = INSW(DVS-1., LINT(DVRVT,ILDVRV, DAVTMP)
     $  ,LINT(DVRRT,ILDVRR, DAVTMP)) * EMERG

      DTEFF     = MAX(0., DAVTMP-TBASE)
      AMTMP     = LINT(AMTMPT,ILAMTM, DDTMP)

*     1.9 GROWTH OF PLANT ORGANS AND TRANSLOCATION

      ASRQ      = FSH * (ASRQLV*FLV + ASRQST*FST + ASRQSO*FSO) +
     $  ASRQRT*FRT

      CHKDIF    = (CHKIN-CHKFL)/NOTNUL(CHKIN)

*     1.4 LEAF CO2 ASSIMILATION

      AMAX      = AMX * AMDVS * AMTMP

      TRANSL    = INSW(DVS-1., 0., WST * DVR * FRTRL)
      CALL SUBEAI(DELT,DVS,EAR,TADRW,RDRDV,EAI,  REAI)
      RDR       = MAX(RDRDV, RDRSH)

*     1.7 MAINTENANCE

      MAINT     = MAINTS * TEFF * MNDVS

      DLAI      = LAI * RDR


*     1.5 DAILY GROSS CO2 ASSIMILATION

      CALL TOTASS(DOY,LATT,DTR,SCP,AMAX,EFF,KDF,TAI,  DAYL,DTGA,DS0)


*     1.6 CARBOHYDRATE PRODUCTION

      GPHOT     = DTGA * 30./44.

      DLV       = WLVG * DLAI/NOTNUL(LAI)

      GTW       = (GPHOT - MAINT + 0.947*TRANSL*CFST*30./12.) / ASRQ
      GST       = FST * FSH * GTW - TRANSL
      GSO       = FSO * FSH * GTW
```

```
      GRT        = FRT * GTW
      GLV        = FLV * FSH * GTW
      CALL GLA(DOY, DOYEM, DTEFF, DVS, NPL, LA0, RGRL, DELT, SLA  ,
  $    LAI, GLV,      GLAI)
      RTNASS     = ((GPHOT - MAINT)*44./30.) -  (GRT*CO2RT + GLV*
  $    CO2LV +  (GST+TRANSL)*CO2ST + GSO*CO2SO +  (1.-0.947)* TRANSL*
  $    CFST*44./12.)
      RWLVG      = GLV - DLV
      RLAI       = GLAI - DLAI

*        Finish conditions
      IF (DVS.GT.2.) TERMNL = .TRUE.

*        Output section
      IF (OUTPUT) THEN
         CALL OUTDAT (2,0,'DOY   ',DOY   )
         CALL OUTDAT (2,0,'DTR   ',DTR   )
         CALL OUTDAT (2,0,'DVS   ',DVS   )
         CALL OUTDAT (2,0,'TDRW  ',TDRW  )
         CALL OUTDAT (2,0,'TADRW ',TADRW )
         CALL OUTDAT (2,0,'WLVG  ',WLVG  )
         CALL OUTDAT (2,0,'WLVD  ',WLVD  )
         CALL OUTDAT (2,0,'WLV   ',WLV   )
         CALL OUTDAT (2,0,'WST   ',WST   )
         CALL OUTDAT (2,0,'WSO   ',WSO   )
         CALL OUTDAT (2,0,'WRT   ',WRT   )
         CALL OUTDAT (2,0,'LAI   ',LAI   )
         CALL OUTDAT (2,0,'EAI   ',EAI   )
         CALL OUTDAT (2,0,'HI    ',HI    )
         CALL OUTDAT (2,0,'DTMAX ',DTMAX )
         CALL OUTDAT (2,0,'DTMIN ',DTMIN )
         CALL OUTDAT (2,0,'GPHOT ',GPHOT )
         CALL OUTDAT (2,0,'DAYL  ',DAYL  )
         CALL OUTDAT (2,0,'DS0   ',DS0   )
         CALL OUTDAT (2,0,'TRANSL',TRANSL)
         CALL OUTDAT (2,0,'CHKIN ',CHKIN )
         CALL OUTDAT (2,0,'CHKFL ',CHKFL )
         CALL OUTDAT (2,0,'CHKDIF',CHKDIF)
      END IF

   ELSE IF (ITASK.EQ.3) THEN
*        Integration
*        ===========
      DVS    = INTGRL (DVS    ,DVR    ,DELT)
      LAI    = INTGRL (LAI    ,RLAI   ,DELT)
      EAI    = INTGRL (EAI    ,REAI   ,DELT)
      WRT    = INTGRL (WRT    ,GRT    ,DELT)
      WLVG   = INTGRL (WLVG   ,RWLVG  ,DELT)
      WLVD   = INTGRL (WLVD   ,DLV    ,DELT)
      WST    = INTGRL (WST    ,GST    ,DELT)
      WSO    = INTGRL (WSO    ,GSO    ,DELT)
      TNASS  = INTGRL (TNASS  ,RTNASS,DELT)

   ELSE IF (ITASK.EQ.4) THEN
*        Terminal
*        ========

*        Terminal calculations

*        Terminal output
      CONTINUE
   END IF

   RETURN
   END


* 1.15 SUBROUTINES

* ---------------------------------------------------------------------*
*  SUBROUTINE GLA                                                      *
*  Purpose: This subroutine computes daily increase of leaf area index *
*           (ha leaf/ ha ground/ d)                                    *
*                                                                      *
*  FORMAL PARAMETERS:  (I=input,O=output,C=control,IN=init,T=time)     *
*  name   type meaning                                    units  class *
*  ----   ---- -------                                    -----  ----- *
*  DOY     R4  Day number (Jan 1st = 1)                     -       T  *
*  DOYEM   R4  Day number of crop emergence                 -       T  *
*  DTEFF   R4  Daily effective temperature                 oC       I  *
*  DVS     R4  Development stage of the crop                -       I  *
*  NPL     R4  Plant density                          plants m-2 I  *
*  LA0     R4  Extrapolated leaf area at emergence    cm2 plant-1 I  *
*  RGRL    R4  Relative leaf growth rate                 ha ha-1   I  *
*  DELT    R4  Time step of integration                    d       T  *
*  SLA     R4  Specific leaf area                        ha kg-1   I  *
*  LAI     R4  Leaf area index                           ha ha-1   I  *
*  GLV     R4  Growth rate of the leaves               kg ha-1 d-1 I  *
*  GLAI    R4  Growth rate of leaf area index          ha ha-1 d-1 O  *
* ---------------------------------------------------------------------*

      SUBROUTINE GLA (DOY,DOYEM,DTEFF,DVS,NPL,LA0,RGRL,DELT,SLA,
     $                   LAI,GLV, GLAI)
      IMPLICIT REAL (A-Z)

*-----growth during maturation stage
      GLAI  = SLA * GLV

*-----growth during juvenile stage
      IF ((DVS.LT.0.3).AND.(LAI.LT.0.75)) THEN
      GLAI  = (LAI * (EXP(RGRL*DTEFF*DELT)-1.))/DELT
      ENDIF

*-----growth at day of seedling emergence
      IF ((DOY.GE.DOYEM).AND.(LAI.EQ.0.)) GLAI = (NPL * LA0)/DELT

*-----growth before seedling emergence
      IF (DOY.LT.DOYEM) GLAI = 0.

      RETURN
      END


*----------------------------------------------------------------------*
*  SUBROUTINE SUBEAI                                                   *
*  Purpose: This subroutine calculates ear area index                  *
*                                                                      *
*  FORMAL PARAMETERS:  (I=input,O=output,C=control,IN=init,T=time)     *
*  name   type meaning                                    units  class *
*  ----   ---- -------                                    -----  ----- *
*  DELT    R4  Time step of integration                    d       T  #
*  DVS     R4  Development stage of the crop               -       I  *
*  EAR     R4  Ear area/weight ratio                     kg ha-1   I  *
*  TADRW   R4  Total above-ground dry weight             kg ha-1   I  *
*  RDR     R4  Relative death rate                         d-1     I  *
*  EAI     R4  Ear area index                            ha ha-1   I  *
*  REAI    R4  Growth rate ear area index              ha ha-1 d-1 O  *
*----------------------------------------------------------------------*

      SUBROUTINE SUBEAI(DELT,DVS,EAR,TADRW,RDRDV,EAI, REAI)
      IMPLICIT REAL(A-Z)

      IF (DVS.LT.0.8) REAI = 0.
      IF (DVS.GE.0.8 .AND. EAI.EQ.0.) THEN
                REAI = (EAR * TADRW)/DELT
      ELSE
                REAI = 0.
      ENDIF
      IF (DVS.GE.1.3) REAI = -RDRDV * EAI
      RETURN
```

```
      END


*----------------------------------------------------------------------*
*  SUBROUTINE ASTRO                                                    *
*  Purpose: This subroutine calculates astronomic daylength,           *
*           diurnal radiation characteristics such as the daily        *
*           integral of sine of solar elevation and solar constant.    *
*                                                                      *
*  FORMAL PARAMETERS:  (I=input,O=output,C=control,IN=init,T=time)     *
*  name   type meaning                                    units  class *
*  ----   ---- -------                                    -----  ----- *
*  DOY     R4  Daynumber (Jan 1st = 1)                      -      T  *
*  LAT     R4  Latitude of the site                       degrees   I  *
*  SC      R4  Solar constant                             J m-2 s-1 O  *
*  DS0     R4  Daily extraterrestrial radiation           J m-2 d-1 O  *
*  SINLD   R4  Seasonal offset of sine of solar height       -      O  *
*  COSLD   R4  Amplitude of sine of solar height             -      O  *
*  DAYL    R4  Astronomic daylength (base = 0 degrees)       h      O  *
*  DSINB   R4  Daily total of sine of solar height           s      O  *
*  DSINBE  R4  Daily total of effective solar height         s      O  *
*                                                                      *
*  FATAL ERROR CHECKS (execution terminated, message)                  *
*  condition: LAT > 67, LAT < -67                                      *
*----------------------------------------------------------------------*

      SUBROUTINE ASTRO (DOY, LAT,
     &                  SC , DS0, SINLD, COSLD, DAYL, DSINB, DSINBE)
      IMPLICIT REAL (A-Z)

*-----PI and conversion factor from degrees to radians
      PI    = 3.141592654
      RAD   = PI/180.

*-----check on input range of parameters
      IF (LAT.GT.67.)  STOP 'ERROR IN ASTRO: LAT> 67'
      IF (LAT.LT.-67.) STOP 'ERROR IN ASTRO: LAT>-67'

*-----declination of the sun as function of daynumber (DOY)
      DEC   = -ASIN (SIN (23.45*RAD)*COS (2.*PI*(DOY+10.)/365.))

*-----SINLD, COSLD and AOB are intermediate variables

      SINLD = SIN (RAD*LAT)*SIN (DEC)
      COSLD = COS (RAD*LAT)*COS (DEC)
      AOB   = SINLD/COSLD

*-----daylength (DAYL)
      DAYL   = 12.0*(1.+2.*ASIN (AOB)/PI)

      DSINB  = 3600.*(DAYL*SINLD+24.*COSLD*SQRT (1.-AOB*AOB)/PI)
      DSINBE = 3600.*(DAYL*(SINLD+0.4*(SINLD*SINLD+COSLD*COSLD*0.5))+
     &         12.0*COSLD*(2.0+3.0*0.4*SINLD)*SQRT (1.-AOB*AOB)/PI)

*-----solar constant (SC) and daily extraterrestrial radiation (DS0)
      SC  = 1370.*(1.+0.033*COS (2.*PI*DOY/365.))
      DS0 = SC*DSINB

      RETURN
      END


*----------------------------------------------------------------------*
*  SUBROUTINE TOTASS                                                   *
*  Purpose: This subroutine calculates daily total gross               *
*           assimilation (DTGA) by performing a Gaussian integration   *
*           over time. At three different times of the day,            *
*           radiation is computed and used to determine assimilation   *
*           whereafter integration takes place.                        *
*                                                                      *
*  FORMAL PARAMETERS:  (I=input,O=output,C=control,IN=init,T=time)     *
*  name   type meaning                                    units  class *
*  ----   ---- -------                                    -----  ----- *
*  DOY     R4  Day number (January 1 = 1)                   -      T  *
*  LAT     R4  Latitude of the site                       degrees   I  *
*  DTR     R4  Daily total of global radiation            J/m2/d    I  *
*  SCP     R4  Scattering coefficient of leaves for visible            *
*              radiation (PAR)                              -      I  *
*  AMAX    R4  Assimilation rate at light saturation      kg CO2/   I  *
*                                                        ha leaf/h     *
*  EFF     R4  Initial light conversion factor           kg CO2/J/  I  *
*                                                        ha/h m2 s     *
*  KDF     R4  Extinction coefficient for diffuse light     -      I  *
*  LAI     R4  Leaf area index as used for photosynthesis  ha/ha    I  *
*              Note: This can involve stem, flower or                  *
*                    ear area index!!                                  *
*  DAYL    R4  Astronomic daylength (base = 0 degrees)       h      O  *
*  DTGA    R4  Daily total gross assimilation          kg CO2/ha/d O  *
*  DS0     R4  Daily extraterrestrial radiation           J/m2/s    O  *
*                                                                      *
*                                                                      *
*  SUBROUTINES and FUNCTIONS called : ASTRO, ASSIM                     *
*----------------------------------------------------------------------*

      SUBROUTINE TOTASS (DOY,  LAT , DTR, SCP, AMAX, EFF, KDF, LAI,
     &                   DAYL, DTGA, DS0)
      IMPLICIT REAL(A-Z)
      REAL XGAUSS(3), WGAUSS(3)
      INTEGER I1, IGAUSS

      DATA IGAUSS /3/
      DATA XGAUSS /0.112702, 0.500000, 0.887298/
      DATA WGAUSS /0.277778, 0.444444, 0.277778/

      PI   = 3.141592654

      CALL ASTRO(DOY,LAT,SC,DS0,SINLD,COSLD,DAYL,DSINB,DSINBE)

*-----assimilation set to zero and three different times of the day (HOU
      DTGA = 0.

      DO 10 I1=1,IGAUSS

*--------at the specified HOUR, radiation is computed and used to comput
*        assimilation
         HOUR = 12.0+DAYL*0.5*XGAUSS(I1)

*--------sine of solar elevation
         SINB  = MAX (0., SINLD+COSLD*COS (2.*PI*(HOUR+12.)/24.))

*--------diffuse light fraction (FRDF) from atmospheric
*        transmission (ATMTR)
         PAR   = 0.5*DTR*SINB*(1.+0.4*SINB)/DSINBE
         ATMTR = PAR/(0.5*SC*SINB)

         IF (ATMTR.LE.0.22) THEN
            FRDF = 1.
         ELSE IF (ATMTR.GT.0.22 .AND. ATMTR.LE.0.35) THEN
            FRDF = 1.-6.4*(ATMTR-0.22)**2
         ELSE
            FRDF = 1.47-1.66*ATMTR
         END IF

         FRDF  = MAX (FRDF, 0.15+0.85*(1.-EXP (-0.1/SINB)))

*--------diffuse PAR (PARDF) and direct PAR (PARDR)
         PARDF = PAR * FRDF
         PARDR = PAR - PARDF

         CALL ASSIM (SCP,AMAX,EFF,KDF,LAI,SINB,PARDR,PARDF,FGROS)
```

```
*--------integration of assimilation rate to a daily total (DTGA)
         DTGA  = DTGA+FGROS*WGAUSS(I1)

10    CONTINUE

      DTGA = DTGA * DAYL

      RETURN
      END


*----------------------------------------------------------------------*
*  SUBROUTINE ASSIM                                                    *
*  Purpose: This subroutine performs a Gaussian integration over       *
*           depth of canopy by selecting five different LAI's and      *
*           computing assimilation at these LAI levels. The            *
*           integrated variable is FGROS.                              *
*                                                                      *
*  FORMAL PARAMETERS:  (I=input,O=output,C=control,IN=init,T=time)     *
*  name   type meaning                                    units  class *
*  ----   ---- -------                                    -----  ----- *
*  SCP     R4  Scattering coefficient of leaves for visible            *
*              radiation (PAR)                              -       I  *
*  AMAX    R4  Assimilation rate at light saturation      kg CO2/   I  *
*                                                        ha leaf/h     *
*  EFF     R4  Initial light conversion factor           kg CO2/J/  I  *
*                                                        ha/h m2 s     *
*  KDF     R4  Extinction coefficient for diffuse light             I  *
*  LAI     R4  Leaf area index as used for photosynthesis  ha/ha    I  *
*              Note: This can involve stem, flower or                  *
*                    ear area index!!                                  *
*  SINB    R4  Sine of solar height                          -      I  *
*  PARDR   R4  Instantaneous flux of direct radiation (PAR) W/m2    I  *
*  PARDF   R4  Instantaneous flux of diffuse radiation(PAR) W/m2    I  *
*  FGROS   R4  Instantaneous assimilation rate of         kg CO2/   O  *
*              whole canopy                              ha soil/h     *
*                                                                      *
*----------------------------------------------------------------------*

      SUBROUTINE ASSIM (SCP, AMAX, EFF, KDF, LAI, SINB, PARDR, PARDF,
     &                  FGROS)
      IMPLICIT REAL(A-Z)
      REAL XGAUSS(5), WGAUSS(5)
      INTEGER I1, I2, IGAUSS

*-----Gauss weights for five point Gauss
      DATA IGAUSS /5/
      DATA XGAUSS /0.0469101,0.2307534,0.5      ,0.7692465,0.9530899/
      DATA WGAUSS /0.1184635,0.2393144,0.2844444,0.2393144,0.1184635/

*-----reflection of horizontal and spherical leaf angle distribution
      SQV  = SQRT(1.-SCP)
      REFH = (1.-SQV)/(1.+SQV)
      REFS = REFH*2./(1.+2.*SINB)

*-----extinction coefficient for direct radiation and total direct flux
      CLUSTF = KDF / (0.8*SQV)
      KBL    = (0.5/SINB) * CLUSTF
      KDRT   = KBL * SQV

*-----selection of depth of canopy, canopy assimilation is set to zero
      FGROS = 0.

      DO 10 I1=1,IGAUSS
         LAIC = LAI * XGAUSS(I1)

*--------absorbed fluxes per unit leaf area: diffuse flux, total direct
*        flux, direct component of direct flux.
         VISDF = (1.-REFH)*PARDF*KDF  *EXP (-KDF  *LAIC)
         VIST  = (1.-REFS)*PARDR*KDRT *EXP (-KDRT *LAIC)
         VISD  = (1.-SCP) *PARDR*KBL  *EXP (-KBL  *LAIC)

*--------absorbed flux (J/M2 leaf/s) for shaded leaves and assimilation
*        shaded leaves
         VISSHD = VISDF + VIST - VISD
         IF (AMAX.GT.0.) THEN
            FGRSH  = AMAX * (1.-EXP(-VISSHD*EFF/AMAX))
         ELSE
            FGRSH = 0.
         END IF

*--------direct flux absorbed by leaves perpendicular on direct beam and
*        assimilation of sunlit leaf area

         VISPP  = (1.-SCP) * PARDR / SINB
         FGRSUN = 0.
         DO 20 I2=1,IGAUSS
            VISSUN = VISSHD + VISPP * XGAUSS(I2)
            IF (AMAX.GT.0.) THEN
               FGRS = AMAX * (1.-EXP(-VISSUN*EFF/AMAX))
            ELSE
               FGRS = 0.
            END IF
            FGRSUN = FGRSUN + FGRS * WGAUSS(I2)
20       CONTINUE

*--------fraction sunlit leaf area (FSLLA) and local assimilation
*        rate (FGL)
         FSLLA = CLUSTF * EXP(-KBL*LAIC)
         FGL   = FSLLA  * FGRSUN + (1.-FSLLA) * FGRSH

*--------integration of local assimilation rate to canopy
*        assimilation (FGROS)
         FGROS = FGROS + FGL * WGAUSS(I1)

10    CONTINUE
      FGROS = FGROS * LAI

      RETURN
      END
```

File: FSE.FOR

```
*----------------------------------------------------------------------*
*                                                                      *
*                                                                      *
*               FORTRAN Simulation Environment (FSE 2.1a)              *
*                              September, 1995                         *
*                                                                      *
*                                                                      *
*     FSE 2.1 is a simulation environment suited for simulation of     *
*     biological processes in time, such as crop and vegetation growth,*
*     insect population development etc.                               *
*                                                                      *
*     The MAIN program, subroutine FSE and subroutine MODELS are       *
*     programmed by D.W.G. van Kraalingen, DLO Institute for           *
*     Agrobiological and Soil Fertility Research (AB-DLO),             *
*     PO Box 14, 6700 AA, Wageningen, The Netherlands (e-mail:         *
*     d.w.g.van.kraalingen@ab.agro.nl).                                *
*                                                                      *
*     FSE version 2.1 is described in:                                 *
*        Kraalingen, D.W.G. van, 1995. The FSE system for crop         *
*        simulation, version 2.1. Quantitative Approaches in Systems   *
*        Analysis; no.1. DLO Research Institute for Agrobiology and    *
*        Soil Fertility, Wageningen. The C.T. de Wit Graduate School   *
*        for Production Ecology. 70 pp.                                *
*                                                                      *
```

```
*     Data files needed for FSE 2.1:                                   *
*          (excluding data files used by models called from MODELS):   *
*        - CONTROL.DAT (contains file names to be used),               *
*        - timer file whose name is specified in CONTROL.DAT,          *
*        - optionally, a rerun file whose name is specified in         *
*          CONTROL.DAT,                                                *
*        - weather data files as specified in timer file               *
*     Object libraries needed for FSE 2.1:                             *
*        - TTUTIL (at least version 3.2)                               *
*        - WEATHER (at least version from 17-Jan-1990)                 *
*----------------------------------------------------------------------*

      SUBROUTINE FSE

c     IMPLICIT NONE

*-----Standard declarations for simulation and output control

      INTEGER   ITASK   , INSETS, ISET  , IPFORM, IL, ILEN
      LOGICAL   OUTPUT  , TERMNL, RDINQR, STRUNF, ENDRNF
      CHARACTER COPINF*1, DELTMP*1
      INTEGER   INPRS   , STRUN , ENDRUN

      INTEGER   IMNPRS
      PARAMETER (IMNPRS=100)
      CHARACTER PRSEL(IMNPRS)*11

*-----Declarations for time control
      INTEGER   IDOY, IYEAR
      REAL      DELT, DOY, FINTIM, PRDEL, STTIME, TIME, YEAR

*-----Declarations for weather system
      INTEGER   IFLAG    , ISTAT1, ISTAT2 , ISTN
      REAL      ANGA     , ANGB  , ELEV   , LAT , LONG, RDD
      REAL      TMMN     , TMMX  , VP     , WN  , RAIN
      LOGICAL   WTRMES   , WTRTER
      CHARACTER WTRDIR*80, CNTR*7, WSTAT*6, DUMMY*1

*-----Declarations for file names and units
      INTEGER   IUNITR   , IUNITD   , IUNITO   , IUNITL   , IUNITC
      CHARACTER FILEON*80, FILEOL*80
      CHARACTER FILEIC*80, FILEIR*80, FILEIT*80
      CHARACTER FILEI1*80, FILEI2*80, FILEI3*80, FILEI4*80, FILEI5*80

*-----Declarations for observation data facility
      INTEGER   INOD , IOD

      INTEGER   IMNOD
      PARAMETER (IMNOD=100)
      INTEGER   IOBSD(IMNOD)

*-----For communication with OBSSYS routine
      COMMON /FSECM1/ YEAR,DOY,IUNITD,IUNITL,TERMNL

      SAVE

*-----File name for control file and empty strings for input
*     files 1-5. WTRMES flags any messages from the weather system

      DATA FILEIC /'CONTROL.DAT'/
      DATA FILEI1 /' '/, FILEI2 /' '/, FILEI3 /' '/
      DATA FILEI4 /' '/, FILEI5 /' '/
      DATA WTRMES /.FALSE./

      DATA STRUNF /.FALSE./, ENDRNF /.FALSE./

*-----Unit numbers for control file (C), data files (D),
*     output file (O), log file (L) and rerun file (R).
      IUNITC = 10
      IUNITD = 20
      IUNITO = 30
      IUNITL = 40
      IUNITR = 50

*-----Open control file and read names of normal output file, log file
*     and rerun file (these files cannot be used in reruns)
      CALL RDINIT (IUNITC,0, FILEIC)
      CALL RDSCHA ('FILEON', FILEON)
      CALL RDSCHA ('FILEOL', FILEOL)
      CALL RDSCHA ('FILEIR', FILEIR)

*     check if start run number was found, if there, read it
      IF (RDINQR('STRUN'))  THEN
         CALL RDSINT ('STRUN',STRUN)
         STRUNF = .TRUE.
      END IF
*     check if end run number was found, if there, read it
      IF (RDINQR('ENDRUN')) THEN
         CALL RDSINT ('ENDRUN',ENDRUN)
         ENDRNF = .TRUE.
      END IF
      CLOSE (IUNITC)

*-----Open output file and possibly a log file
      CALL FOPENS  (IUNITO, FILEON, 'NEW', 'DEL')
      IF (FILEOL.NE.FILEON) THEN
         CALL FOPENS  (IUNITL, FILEOL, 'NEW', 'DEL')
      ELSE
         IUNITL = IUNITO
      END IF

c*     initialization of logfile for processing of end_of_run values
c      CALL OPINIT

*-----See if rerun file is present, and if so read the number of rerun
*     sets from rerun file

      CALL RDSETS (IUNITR, IUNITL, FILEIR, INSETS)

*======================================================================*
*======================================================================*
*                                                                      *
*                    Main loop and reruns begin here                   *
*                                                                      *
*======================================================================*
*======================================================================*

      IF (.NOT.ENDRNF) THEN
*        no end run was found in control.dat file
         ENDRUN = INSETS
      ELSE
         ENDRUN = MAX (ENDRUN, 0)
         ENDRUN = MIN (ENDRUN, INSETS)
      END IF

      IF (.NOT.STRUNF) THEN
*        no start run was found in control.dat file
         STRUN = 0
      ELSE
         STRUN = MAX (STRUN, 0)
         STRUN = MIN (STRUN, ENDRUN)
      END IF

      DO 10 ISET=STRUN,ENDRUN

      WRITE (*,'(A)') '   FSE 2.1: Initialize model'

*-----Select data set
      CALL RDFROM (ISET, .TRUE.)
```

```
*======================================================================*
*                                                                      *
*                        Initialization section                        *
*                                                                      *
*======================================================================*

      ITASK  = 1
      TERMNL = .FALSE.
      WTRTER = .FALSE.

*-----Read names of timer file and input files 1-5 from control
*     file (these files can be used in reruns)
      CALL RDINIT (IUNITC,IUNITL,FILEIC)
      CALL RDSCHA ('FILEIT', FILEIT)
      IF (RDINQR ('FILEI1')) CALL RDSCHA ('FILEI1', FILEI1)
      IF (RDINQR ('FILEI2')) CALL RDSCHA ('FILEI2', FILEI2)
      IF (RDINQR ('FILEI3')) CALL RDSCHA ('FILEI3', FILEI3)
      IF (RDINQR ('FILEI4')) CALL RDSCHA ('FILEI4', FILEI4)
      IF (RDINQR ('FILEI5')) CALL RDSCHA ('FILEI5', FILEI5)
      CLOSE (IUNITC)

*-----Read time, control and weather variables from timer file
      CALL RDINIT (IUNITD  , IUNITL, FILEIT)
      CALL RDSREA ('STTIME', STTIME)
      CALL RDSREA ('FINTIM', FINTIM)
      CALL RDSREA ('PRDEL' , PRDEL )
      CALL RDSREA ('DELT'  , DELT  )
      CALL RDSINT ('IYEAR' , IYEAR )
      CALL RDSINT ('ISTN'  , ISTN  )
      CALL RDSINT ('IPFORM', IPFORM)
      CALL RDSCHA ('COPINF', COPINF)
      CALL RDSCHA ('DELTMP', DELTMP )
      CALL RDSCHA ('WTRDIR', WTRDIR)
      CALL RDSCHA ('CNTR'  , CNTR)
      CALL RDSINT ('IFLAG' , IFLAG)

*-----See if observation data variable exists, if so read it
      INOD = 0
      IF (RDINQR('IOBSD')) THEN
         CALL RDAINT ('IOBSD' , IOBSD, IMNOD, INOD)
         IF (IOBSD(1).EQ.0) INOD = 0
      END IF

*-----See if variable with print selection exists, if so read it
      INPRS = 0
      IF (RDINQR('PRSEL')) CALL RDACHA ('PRSEL',PRSEL,IMNPRS,INPRS)

      CLOSE (IUNITD)

*-----Initialize TIMER and OUTDAT routines
      CALL TIMER2 (ITASK, STTIME, DELT, PRDEL, FINTIM,
     &             IYEAR, TIME  , DOY , IDOY , TERMNL, OUTPUT)
      YEAR = REAL (IYEAR)
      CALL OUTDAT (ITASK, IUNITO, 'TIME', TIME)

*-----Open weather file and read station information and return
*     weather data for start day of simulation.
*     Check status of weather system, WTRMES flags if warnings or errors
*     have occurred during the whole simulation. WTRTER flags if the run
*     should be terminated because of missing weather

      CALL STINFO (IFLAG , WTRDIR, ' ', CNTR, ISTN, IYEAR,
     &             ISTAT1, LONG  , LAT, ELEV, ANGA, ANGB)
      CALL WEATHR (IDOY  , ISTAT2, RDD, TMMN, TMMX, VP, WN, RAIN)
      IF (ISTAT1.NE.0.OR.ISTAT2.NE.0) WTRMES = .TRUE.
      WSTAT  = '444444'
      IF (ABS (ISTAT2).GE.111111) THEN
         WRITE (WSTAT,'(I6)') ABS (ISTAT2)
      ELSE IF (ISTAT2.EQ.0) THEN
         WSTAT = '111111'
      END IF

c*-----initialize OBSSYS routine
c      IF (ITASK.EQ.1) CALL OBSINI

*-----Conversion of total daily radiation from kJ/m2/d to J/m2/d
      RDD = RDD*1000.

*-----Call routine that handles the different models
      CALL 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)


*======================================================================*
*                                                                      *
*                       Dynamic simulation section                     *
*                                                                      *
*======================================================================*


      WRITE (*,'(A)') '   FSE 2.1: DYNAMIC loop'

20    IF (.NOT.TERMNL) THEN

*----------------------------------------------------------------------*
*                      Integration of rates section                    *
*----------------------------------------------------------------------*

      IF (ITASK.EQ.2) THEN

*--------Carry out integration only when previous task was rate
*        calculation

         ITASK = 3

*--------Call routine that handles the different models
         CALL 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)

*--------Turn on output when TERMNL logical is set to .TRUE.
         IF (TERMNL.AND.PRDEL.GT.0.) OUTPUT = .TRUE.

      END IF

*----------------------------------------------------------------------*
*               Calculation of driving variables section               *
*----------------------------------------------------------------------*

      ITASK = 2

*-----Write time of output to screen and file
      CALL OUTDAT (2, 0, 'TIME', TIME)
      IF (OUTPUT) THEN
         IF (ISET.EQ.0) THEN
            WRITE (*,'(13X,A,I5,A,F7.2)')
     &        'Default set, Year:', IYEAR, ', Day:', DOY
         ELSE
            WRITE (*,'(13X,A,I3,A,I5,A,F7.2)')
     &        'Rerun set:', ISET, ', Year:', IYEAR, ', Day:', DOY
         END IF
```

```
      END IF

*-----Get weather data for new day and flag messages
      CALL STINFO (IFLAG , WTRDIR, ' ', CNTR, ISTN, IYEAR,
     &             ISTAT1, LONG  , LAT, ELEV, ANGA, ANGB)
      CALL WEATHR (IDOY, ISTAT2, RDD, TMMN, TMMX, VP, WN, RAIN)
      IF (ISTAT1.NE.0.OR.ISTAT2.NE.0) WTRMES = .TRUE.
      WSTAT  = '444444'
      IF (ABS (ISTAT2).GE.111111) THEN
         WRITE (WSTAT,'(I6)') ABS (ISTAT2)
      ELSE IF (ISTAT2.EQ.0) THEN
         WSTAT = '111111'
      END IF

*-----Conversion of total daily radiation from kJ/m2/d to J/m2/d
      RDD = RDD*1000.

*----------------------------------------------------------------------*
*              Calculation of rates and output section               *
*----------------------------------------------------------------------*

*-----Call routine that handles the different models
      CALL 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)

      IF (TERMNL.AND..NOT.OUTPUT.AND.PRDEL.GT.0.) THEN
*--------Call model routine again if TERMNL is switched on while
*        OUTPUT was off (this call is necessary to get output to file
*        when a finish condition was reached and output generation
*        was off)
         IF (ISET.EQ.0) THEN
            WRITE (*,'(13X,A,I5,A,F7.2)')
     &        'Default set, Year:', IYEAR, ', Day:', DOY
         ELSE
            WRITE (*,'(13X,A,I3,A,I5,A,F7.2)')
     &        'Rerun set:', ISET, ', Year:', IYEAR, ', Day:', DOY
         END IF
         OUTPUT = .TRUE.
         CALL OUTDAT (2, 0, 'TIME', TIME)
         CALL 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)
      END IF

*----------------------------------------------------------------------*
*                            Time update                               *
*----------------------------------------------------------------------*

*-----Check for FINTIM, OUTPUT and observation days
      CALL TIMER2 (ITASK, STTIME, DELT, PRDEL, FINTIM,
     &             IYEAR, TIME  , DOY , IDOY , TERMNL, OUTPUT)
      YEAR = REAL (IYEAR)
      DO 30 IOD=1,INOD,2
         IF (IYEAR.EQ.IOBSD(IOD).AND.IDOY.EQ.IOBSD(IOD+1))
     &       OUTPUT = .TRUE.
30    CONTINUE

      GOTO 20
      END IF

*======================================================================*
```

```
*                                                                      *
*                           Terminal section                           *
*                                                                      *
*======================================================================*

      ITASK = 4

      WRITE (*,'(A)') '   FSE 2.1: Terminate model'

*-----Call routine that handles the different models
      CALL 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)

*-----Generate output file dependent on option from timer file
      IF (IPFORM.GE.4) THEN
         IF (INPRS.EQ.0) THEN
            CALL OUTDAT (IPFORM, 0, 'Simulation results',0.)
         ELSE
*           Selection of output variables was in timer file
*           write tables according to output selection array PRSEL
            CALL OUTSEL (PRSEL,IMNPRS,INPRS,IPFORM,'Simulation results')
         END IF
      END IF

      IF (WTRTER) THEN
         WRITE (*,'(/,A,/,/,/)')
     &     ' The run was terminated due to missing weather'
         WRITE (IUNITO,'(/,A,/,/,/)')
     &     ' The run was terminated due to missing weather'
         IF (IUNITO.NE.IUNITL) WRITE (IUNITL,'(/,A,/,/,/)')
     &     ' The run was terminated due to missing weather'
      END IF

*-----Delete temporary output file dependent on switch from timer file
      IF (DELTMP.EQ.'Y'.OR.DELTMP.EQ.'y') CALL OUTDAT (99, 0, ' ', 0.)

10    CONTINUE

      IF (INSETS.GT.0) CLOSE (IUNITR)

*-----If input files should be copied to the output file,
*     copy rerun file (if present) and timer file and if there, input
*     files 1-5

      IF (COPINF.EQ.'Y'.OR.COPINF.EQ.'y') THEN
         IF (INSETS.GT.0) CALL COPFL2 (IUNITR, FILEIR, IUNITO, .TRUE.)
         CALL COPFL2 (IUNITD, FILEIT, IUNITO, .TRUE.)
         IF (FILEI1.NE.' ') CALL COPFL2 (IUNITD, FILEI1, IUNITO, .TRUE.)
         IF (FILEI2.NE.' ') CALL COPFL2 (IUNITD, FILEI2, IUNITO, .TRUE.)
         IF (FILEI3.NE.' ') CALL COPFL2 (IUNITD, FILEI3, IUNITO, .TRUE.)
         IF (FILEI4.NE.' ') CALL COPFL2 (IUNITD, FILEI4, IUNITO, .TRUE.)
         IF (FILEI5.NE.' ') CALL COPFL2 (IUNITD, FILEI5, IUNITO, .TRUE.)
      END IF

*-----Delete all .TMP files that were created by the RD* routines
*     during simulation
      CALL RDDTMP (IUNITD)

*-----Write to screen which files contain what
      IL = ILEN (FILEON)
      WRITE (*,'(/,3A)') ' File: ',FILEON(1:IL),
     &  ' contains simulation results'
      WRITE (*,'(2A)') ' File: WEATHER.LOG',
     &  ' contains messages from the weather system'
      IL = ILEN (FILEOL)
```

```
      WRITE (*,'(3A,/)') ' File: ',FILEOL(1:IL),
     &  ' contains messages from the rest of the model'

*-----Write message to screen and output file if warnings and/or errors
*     have occurred from the weather system, pause and wait for return
*     from user to make sure he has seen this message

      IF (WTRMES) THEN

         WRITE (*,'(/,A,/,A,/,A)') ' WARNING from FSE:',
     &     ' There have been errors and/or warnings from',
     &     ' the weather system, check file WEATHER.LOG'
         WRITE (IUNITO,'(A,/,A,/,A)') ' WARNING from FSE:',
     &     ' There have been errors and/or warnings from',
     &     ' the weather system, check file WEATHER.LOG'

         WRITE (*,'(A)') ' Press <Enter>'
         READ  (*,'(A)') DUMMY

      END IF

*-----Close output file and temporary file of OUTDAT
      CLOSE (IUNITO)
      CLOSE (IUNITO+1)

*-----Close log file (if used)
      IF (FILEOL.NE.FILEON) CLOSE (IUNITL)

*-----Close log file of weather system
      CLOSE (91)

c*-----Write end_of_run values to file
c      CALL OPWRIT (IUNITO)

      RETURN
      END
```

File: CONTROL.DAT

```
*----------------------------------------------------------------------*
* File names to be used by FSE 2.1                                     *
*                                                                      *
* The input files (except FILEIR) may may used in reruns.              *
* Up to five input data files may be used (FILEI1-5)                   *
* Also begin and end run numbers can be given here                     *
*----------------------------------------------------------------------*

  FILEON = 'RES.DAT'       ! Normal output file
  FILEOL = 'MODEL.LOG'     ! Log file
  FILEIR = 'RERUNS.DAT'    ! Reruns file
  FILEIT = 'TIMER.DAT'     ! File with timer data
  FILEI1 = 'MODEL.DAT'     ! First input data file

* FILEI2 = ' '             ! Second input data file (not used)
* FILEI3 = ' '             ! Third input data file (not used)
* FILEI4 = ' '             ! Fourth input data file (not used)
* FILEI5 = ' '             ! Fifth input data file (not used)

* STRUN  = 0               ! Run number where execution should start
* ENDRUN = x               ! Run number where execution should end
```

File: TIMER.DAT

```
*
* TIMER variables used in GENERAL and FSE translation modes
* ---------------------------------------------------------
STTIME = 80.                 ! start time
FINTIM = 300.                ! finish time
DELT   = 1.                  ! time step (for Runge-Kutta first guess)
PRDEL  = 5.                  ! output time step
IPFORM = 4                   ! code for output table format:
                             ! 4 = spaces between columns
                             ! 5 = TAB's between columns (spreadsheet
                             !     output)
                             ! 6 = two column output
COPINF = 'N'                 ! Switch variable whether to copy the
                             ! input files to the output file ('N' =
                             ! do not copy, 'Y' = copy)
DELTMP = 'N'                 ! Switch variable what should be done
                             ! with the temporary output file ('N' =
                             ! do not delete, 'Y' = delete)
IFLAG  = 1101                ! Indicates where weather error and
                             ! warnings go (1101 means errors and
                             ! warnings to log file, errors to screen,
                             ! see FSE manual)
*IOBSD = 1991,182            ! List of observation data for which
                             ! output is required. The list should
                             ! consist of pairs <year>,<day>
                             ! combinations


* WEATHER control variables
* -------------------------
WTRDIR = 'WEATHER\'
CNTR   = 'NLD'               ! Country code
ISTN   = 1                   ! Station code
IYEAR  = 1980                ! Year
```

File: MODEL.DAT

```
* Initial constants
* -----------------
 ZERO   = 0.
 NPL    = 210.
 LA0    = 5.7E-5
 WSTI   = 0.
 WRTI   = 0.
 WLVI   = 0.
 WSOI   = 0.
 WLVDI  = 0.
 IDVS   = 0.
 ILAI   = 0.
 IEAI   = 0.


* Model parameters
* ----------------
 DOYEM  = 90.
 AMX    = 40.
 EFF    = 0.45
 KDF    = 0.60
 SCP    = 0.20
 LATT   = 52.
 TREF   = 25.
 Q10    = 2.
 MAINLV = 0.03
 MAINST = 0.015
 MAINSO = 0.01
 MAINRT = 0.015
 ASRQRT = 1.444
 ASRQLV = 1.463
 ASRQSO = 1.415
```

```
 ASRQST = 1.513
 FRTRL  = 0.20
 RGRL   = 0.009
 SLA    = 0.0022
 EAR    = 6.3E-5
 LAICR  = 4.0
 TBASE  = 0.
 CFSO   = 0.471
 CFRT   = 0.467
 CFLV   = 0.459
 CFST   = 0.494


* Interpolation functions
* -----------------------
 DVRVT  =
    -10., 0.,
    0., 0.,
    30., 0.027
 DVRRT  =
    -10., 0.,
    0., 0.,
    30., 0.031
 AMDVST =
    0.0, 1.0,
    1.0, 1.0,
    2.0, 0.5,
    2.5, 0.0
 AMTMPT =
    -10., 0.,
    0., 0.,
    10., 1.,
    25., 1.,
    35., 0.,
    50., 0.
 FSHTB  =
    0.00, 0.50,
    0.10, 0.50,
    0.20, 0.60,
    0.35, 0.78,
    0.40, 0.83,
    0.50, 0.87,
    0.60, 0.90,
    0.70, 0.93,
    0.80, 0.95,
    0.90, 0.97,
    1.00, 0.98,
    1.10, 0.99,
    1.20, 1.00,
    2.50, 1.00
 FLVTB  =
    0.00, 0.65,
    0.10, 0.65,
    0.25, 0.70,
    0.50, 0.50,
    0.70, 0.15,
    0.95, 0.00,
    2.50, 0.00
 FSTTB  =
    0.00, 0.35,
    0.10, 0.35,
    0.25, 0.30,
    0.50, 0.50,
    0.70, 0.85,
    0.95, 1.00,
    1.05, 0.00,
    2.50, 0.00
 RDRT   =
    -10., 0.03,
    10., 0.03,
    15., 0.04,
    30., 0.09
```

File: NLD1.980

```
*----------------------------------------------------------*
*   Country: Netherlands
*   Station: Wageningen
*      Year: 1980
*    Source: Dep. of Meteorology, Wageningen Agricultural
*            University.
*    Author: Peter Uithol
* Longitude: 05 40 E
*  Latitude: 51 58 N
* Elevation: 7 m.
*  Comments: Location Haarweg.
*
*  Columns:
*  ========
*  station number
*  year
*  day
*  irradiation (kJ m-2 d-1)
*  minimum temperature (degrees Celsius)
*  maximum temperature (degrees Celsius)
*  vapour pressure (kPa)
*  mean wind speed (m s-1)
*  precipitation (mm d-1)
*----------------------------------------------------------*
   5.67  51.97     7. 0.00 0.00
   1 1980   1  2540.  -1.2   1.4   0.620   3.5   6.2
   1 1980   2  3520.  -6.5   1.4   0.530   1.7   0.0
   1 1980   3  1510.  -8.2   0.1   0.490   2.2   0.2
```

<continued>

```
   1 1980 362  3220.  -3.3   4.3   0.600   1.1   0.7
   1 1980 363   870.  -2.7   3.4   0.620   2.8   0.0
   1 1980 364   350.   3.3   7.2   0.870   3.5   0.0
   1 1980 365   320.   6.4   8.2   0.920   4.3   0.0
   1 1980 366   570.   5.7   8.6   0.820   7.2   2.0
```

File: RERUNS.DAT

```
* Example rerun file

NPL = 210.; AMX=30.
NPL = 250.; AMX=40.
NPL = 250.; AMX=30.
```

File: OUTREC.FOR

```
      PROGRAM OUTREC

*     Creates output tables from RES.BIN files. Use for instance after
*     a program crash, leaving you without a formatted output table.
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ILEN, FOPENG, ERROR, OUTCOM, UPPERC,
*                            IFINDC, AMBUSY
*
*     Author: Daniel van Kraalingen
*     Date  : April 1995
*     TTUTIL Version 3.4
```

```
      IMPLICIT NONE
      INTEGER ICH

      WRITE (*,'(A,5(/,A))')
     & ' Normal output table of last set:  4',
     & ' Tab-delimited table of last set:  5',
     & '   Two column output of last set:  6',
     & ' Normal output table of all sets: 14',
     & ' Tab-delimited table of all sets: 15',
     & '   Two column output of all sets: 16'


      CALL ENTDIN ('Your choice please',14,ICH)

      CALL OUTDAT (ICH,20,'Recovered file',0.)

      WRITE (*,'(A)') ' Output successfully recovered'

      END
```

Overgebleven tekst

The complexity of the routine that drives the models under FSE (the FSE driver) has also been hidden by using an interface routine with the introduction of FSE 2.1. This interface routine receives data from FSE such as day of the year, year etc. and makes these available to the different models that are called by the interface routine. Also the interface routine can be used to organize interactions among models more conveniently than with FSE 1.0.