Simulation Reports CABO-TT

The FSE system for crop simulation

D.W.G. van Kraalingen

Simulation Report CABO-TT nr. 23

A joint publication of

Centre for Agrobiological Research (CABO)

and

Department of Theoretical Production Ecology (TPE), Agricultural University

Wageningen 1991

Simulation Reports CABO-TT

Simulation Reports CABO-TT 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 CABO-TT 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 (CABO) or J. Goudriaan (TPE).

Simulation Reports CABO-TT are issued by CABO and TPE 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.

Address

Simulation Reports CABO-TT
P.O. Box 14
6700 AA Wageningen
Netherlands

Authors affiliation

D.W.G. van Kraalingen:
Centre of Agrobiological Research
P.O. Box 14,
6700 AA Wageningen, The Netherlands

Contents **Page no.**

Samenvatting

Dit rapport beschrijft een FORTRAN 77 omgeving voor het ontwikkelen van continue simulatie modellen. Deze omgeving wordt FSE (FORTRAN Simulation Environment) genoemd. Het systeem bestaat uit een hoofdprogramma, weersgegevens, en verscheidene utility routines for het uitvoeren van specifieke taken. De feitelijke modelvergelijkingen worden in de vorm van één of meer subroutines gebracht die bestuurd worden door het hoofdprogramma. De FSE omgeving is flexibel, voert de tijdsbesturing uit, haalt weers-gegevens op uit datafiles en voorziet in de mogelijkheid van eenvoudige invoer van parameters en initiële toestanden. 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 draaien.

De FSE omgeving biedt oplossingen voor veel problemen die ondervonden worden door onderzoekers die in FORTRAN werken. Door gebruik te maken van deze 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.

Summary

An environment for continuous simulation of crop growth in FORTRAN 77 is discussed. 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. The environment is flexible, provides weather data, easy input of parameters and initial states, easy output in the form of tables or graphs, and time control. The FSE program is highly portable to other computers.

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

Acknowledgements

The following 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. The FSE program 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 and the idea of writing this manual, and Peter Kooman and Willem Stol for testing the FSE program and making suggestions for its improvement.

1 Introduction

This report presents a simulation environment for crop growth models in FORTRAN. After discussing the principles of simulation in FORTRAN these are translated into a full working program. Much of this report is based on work done for 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 in CABO/TPE to have this approach further refined and documented without special reference to the SARP programs but as a general documentation to the FSE simulation environment.

In the past, crop simulation often used CSMP as the simulation language. But recently, 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. Yet good FORTRAN compilers are widely available and are easy to purchase. Consequently, crop models in FORTRAN can be exchanged more easily and can 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, and Educational software.

In this report we begin by describing the principles of simulation in FORTRAN and then go on to discuss a 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 input 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 initial 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. In this report the SUCROS model is used as an example of a crop model that has been programmed using the FSE program.

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).

To obtain the FSE program, as well as the full TTUTIL library on floppy disk, write to:

Centre for Agrobiological Research,
P.O. Box 14,
6700 AA Wageningen,
The Netherlands.

or:

Department of Theoretical Production Ecology,
Agricultural University,
P.O. Box 430,
6700 AK Wageningen,
The Netherlands.

2 Principles of the FORTRAN Simulation Environment

In this Chapter we will discuss the principles adhered to in the development of the FORTRAN Simulation Environment other than the general structure of Euler simulation in FORTRAN (this is further outlined in Chapter 3).

• Calculations are done in the correct order

Care has been taken to ensure that the structure of the model is such that the different types of calculations, such as integration, rate calculation, time update, output and initialization, are all done in the right order. Experience learns that often in continuous simulation models this is not the case. Sometimes the rates and states in the model output do not pertain to the same TIME, or rate and state calculations are not done separately; as a result, rates may be derived partly from state variables at the current time and partly from state variables one DELT ago. 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.

• Efforts have been made to conceal program complexity that is irrelevant to the simulation

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 `TIMER` routine. These routines have a clearly defined task that is easy to understand, but their implementation into a 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. But the FORTRAN code is very complex, and the user of the simulation program does not need to know it. These routines are stored in a separate library, TTUTIL, the utility library of the Department of Theoretical Production Ecology of the Wageningen Agricultural University.

We have also tried to make the program flow straightforward, by minimizing the number of `GOTO` statements. In general, the liberal use of `GOTO` statements is considered bad programming, 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 done to the `CONTINUE` statement.

• 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 compiler 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 the compiler.

At the time of writing of this document, a new FORTRAN standard had been defined: Fortran-90 (note the lower case name). 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 these improvements to the language by following the guidelines in ter Haar (1983, see listing 1) and emulating a `DO-WHILE` control structure 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 progamming language is however, only worthwile when good compilers on several computers are widely available. Until then, we will presumably 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 or arrays limits the number of computers on which the program can be run. Disk memory is often much larger than RAM memory, and therefore information is stored in temporary files where possible. This practice limits the total array size of the whole program to about 42 kb of memory.

- **The program is safeguarded against inaccurate floating point operations**

The definition of standard FORTRAN 77 (like that of most 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.

The inaccuracy is important in the `TIMER` routine, which should trigger output whenever `TIME` is a multiple of `PRDEL` (consequently, `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 `TIMER` routine).

3 The structure of Euler integration in the FSE program

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 variable approach as it is used in continuous simulation (see e.g. de Wit & Goudriaan, 1978; Penning de Vries & van Laar, 1982).

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 Euler integration only, is less complicated and easier to understand than one accommodating higher order methods of integration. But this less complicated structure requires changes to be made in the integration section of the simulating subroutines to allow higher order integration methods to be used (see Rappoldt & van Kraalingen, 1990). The simulation of crop growth in CSMP often uses Euler integration with a fixed time step of one day and because the program structure is less complicated, we have adopted this integration method in the FSE program.

Order of execution

Fig. 1 shows the correct order in which calculations should be executed when Euler integration is used:

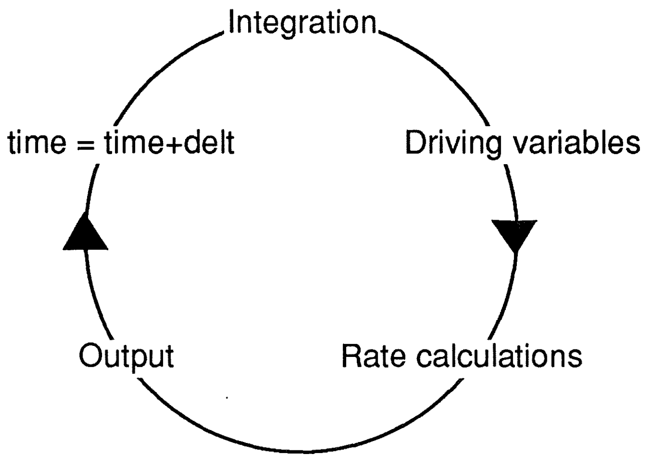

Fig. 1: The order in which calculations are executed when simulating continuous systems using Euler integration.

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 Listings 2 and 3.

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

```
TITLE DEMONSTRATION
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

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 DEMO
      IMPLICIT REAL (A-Z)
      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

    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
```

In theory, the sequence in which various state variables are updated is not important because their values should not depend on each other but should be fully determined by the rate variables. In practice, however, state variables may sometimes be derived from other state variables (e.g. root/shoot ratio or total weight of leaves equals weight of dead leaves plus weight of green leaves). It is therefore important to put the state calculations in the right order, as is also necessary for the rate calculations.

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, 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 merged into one block as do all the rate calculations. Often, different subprocesses are interacting (e.g. a plant transpiring water from the soil). In many cases these interactions among the subprocesses are only weak. 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 share 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 the above, it is not a good solution to combine all the state calculations from the different subprocesses into one large program 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 decide which of the two 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. 2. 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. 1.

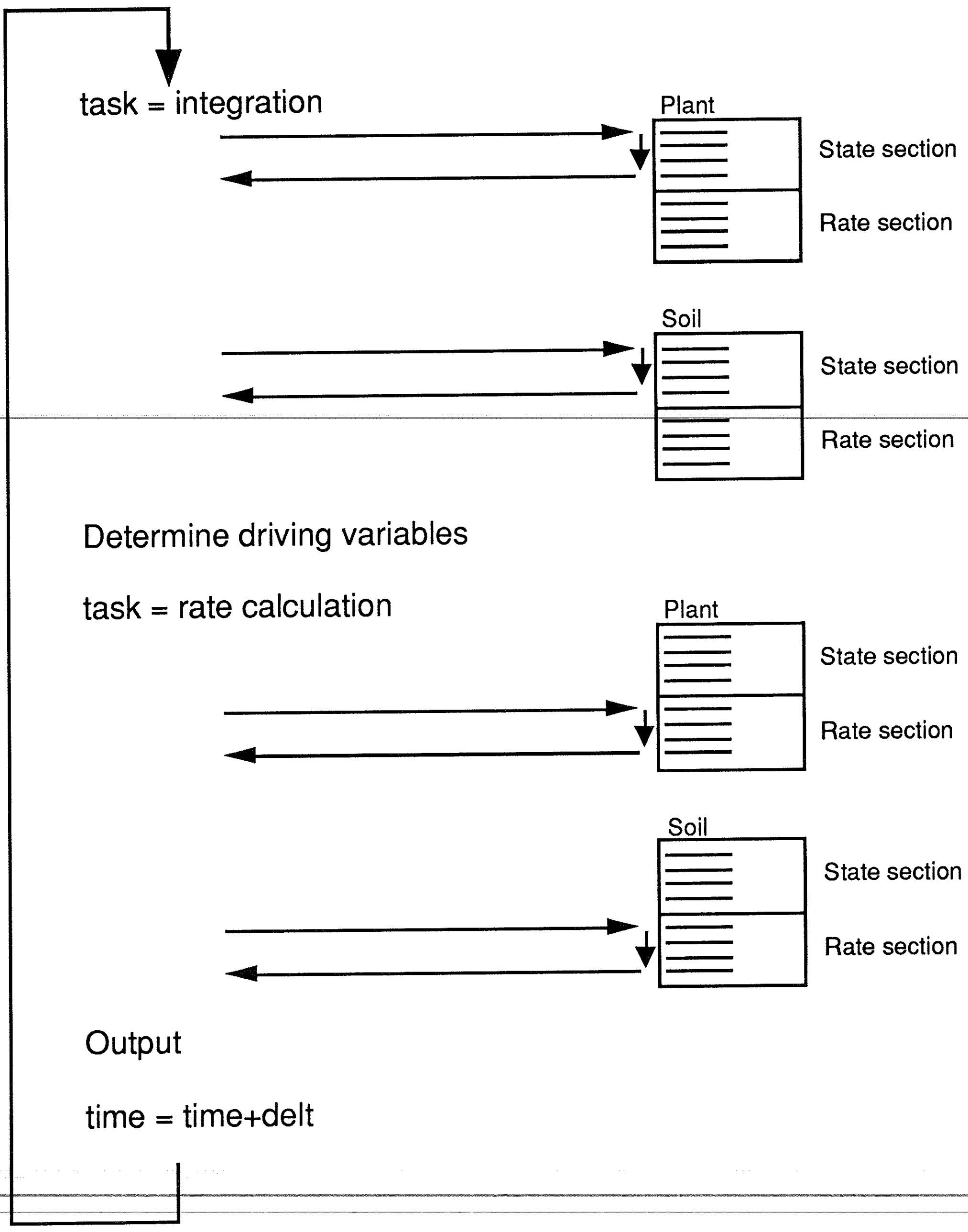

Fig. 2: General structure for incorporating several subprocesses containing integration and rate calculation into a single simulation model.

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. 1).

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. Consequently, the circle should also be entered between time update and integration. The most convenient way to initialize the subprocesses is to have this operation controlled by the main program. 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. These refinements to Fig. 1 are shown in Fig. 3.

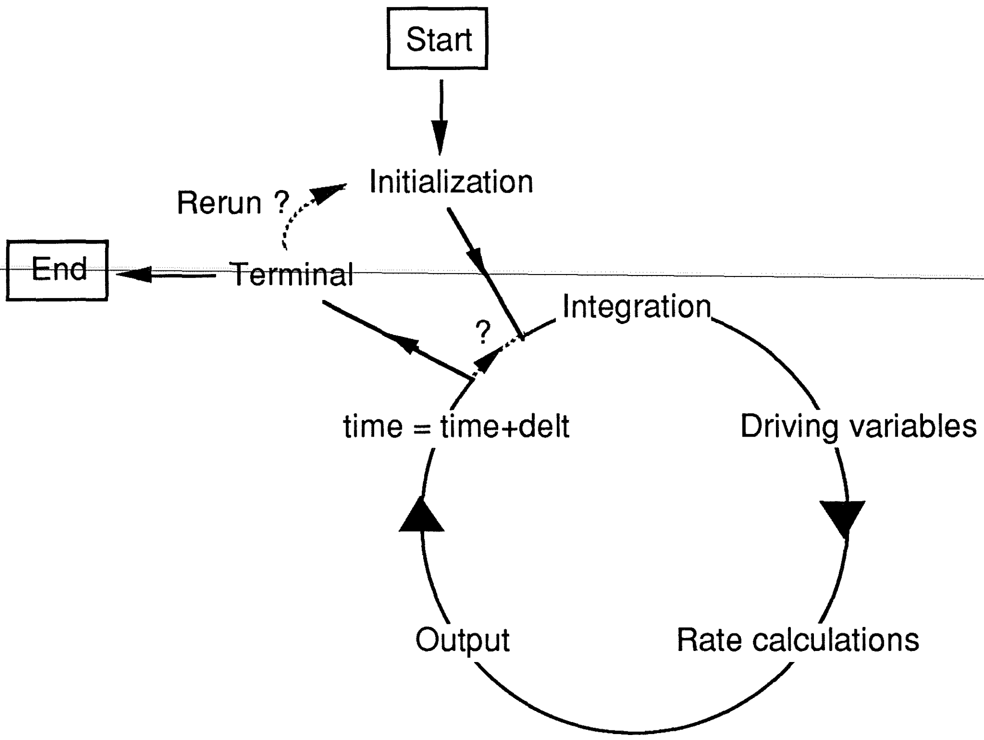

Fig. 3: The order in which calculations are executed when simulating continuous systems using Euler integration, illustrating where to enter and leave the circle and how reruns are implemented.

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", the model proceeds to the terminal section; if it is "yes" the circle is run once more. 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 and a new simulation run is started.

As shown in Fig. 2, the modularity of the subprocess descriptions is preserved by introducing the concept of task-controlled execution (the calling program decides what the subroutine should do: either integration or rate calculation). To be able to do reruns, the various subprocess descriptions that can also be driven by the task variable have to be initialized externally, and some terminal calculations (e.g. harvest index) have to be done. Thus, a subprocess description in the FSE program should recognize four different tasks: <u>initialization</u>, <u>integration</u>, <u>rate calculation</u> and <u>terminal calculation</u>.

A consequence of this structure is that the first step after initialization is integration. This does no harm if the rates have been set to zero explicitly, so that the first integration has no effect on the value of the states. In practice this means incorporating many rate assignments to zero into the model. To avoid this, integration can be skipped if the previous task was initialization (during which the states have been assigned values anyway). The subsequent rate calculation will then use the state variables to initialize the rates of change. This shortcut is implemented in the main program of FSE but has not been shown in Fig. 3.

In the next Chapter we will discuss how this theory of continuous simulation using Euler integration is implemented in FORTRAN.

4 Implementing Euler integration in FORTRAN

In this Chapter the principles of Fig. 2 and Fig. 3, discussed in the previous Chapter, are translated into a full program. Also, some more technical details will be discussed, to clarify how the program actually works. For a definition of the utility subroutines and functions, see Rappoldt & van Kraalingen (1990), and for a description of the weather subprogram and its corresponding datafiles, see van Kraalingen *et al.* (1990).

By the end of this Chapter you should have a fair understanding of the FSE program. More technical details of the use of subprograms in simulation models can also be found in van Kraalingen & Rappoldt (1989).

Control of the time loop

As shown in Fig. 3, 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.` and the 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.)

Some other features are also illustrated in this example program:

- **Implementing the concept of task-controlled execution**

The task 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.

- **Initializing the rates by skipping integration 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 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 model 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.

• Time control and output at multiples of PRDEL

The time control in a simulation program is more complicated than simply the increase of `TIME` with `DELT` and therefore it has been hidden in a `TIMER` subroutine, together with the setting of a flag when output is required (at the start time of the simulation, when `TIME` is a multiple of `PRDEL` and at the time that the simulation terminates). The basic actions of the subroutine `TIMER` are: `ITASK=1`: check values of `FINTIM`, `DELT`, `TIME`, etc. and copy these to variables local to the subroutine, turn output flag on, `ITASK=2`: check whether local time variables have the same value as the global time variables, add `DELT` to `TIME`, calculate day number (`DAY`), flag if `TIME` is a multiple of `PRDEL` using the variable `OUTPUT`, flag if `TIME` has exceeded `FINTIM` using the variable `TERMNL`.

The TIMER routine that is used in the FSE program has more features; for example, the year of simulation is automatically incremented and the day number is reset to 1 when the day number passes 365. Leap years are also recognized (DAY runs until 366), and the day number is available as an integer and as a real value (`IDAY` and `DAY`).

• An outline of a plant routine

Program listing 4 shows a 'stripped' version of a main program and of a plant routine. Note that the subroutine consists of four sections, corresponding to initialization, rate calculations, integration, and terminal. We have chosen for each subprocess to write its relevant output variables to an output device separately, instead of organizing the output from the main program. This has the advantage that fewer variables have to be communicated with the main program and fewer changes have to be made in the main program when a new plant routine is used.

Listing 4: Example program of simulation loop driving a hypothetical plant routine.

```
      PROGRAM SMALL
      IMPLICIT REAL (A-Z)
      LOGICAL TERMNL, OUTPUT
      INTEGER ITASK

*     initialization
      TERMNL = .FALSE.
      ITASK  = 1
      TIME   = 0.0
      FINTIM = 100.
      DELT   = 1.0

*     initialization of TIMER and PLANT subroutines
```

```
      CALL TIMER (ITASK, DELT, PRDEL, FINTIM, TIME, DAY, TERMNL, OUTPUT)
      CALL PLANT (ITASK, OUTPUT, TERMNL, TIME, DAY, DELT, ...)

*     run loop as long as TERMNL is not .TRUE.
10    IF (.NOT.TERMNL) THEN

         IF (ITASK.NE.1) THEN
*           integration
            ITASK = 3
            CALL PLANT (ITASK, OUTPUT, TERMNL, TIME, DAY, DELT, ...)
         END IF

*        driving variables
         CALL WEATHR (DAY, weather variables)

*        rate calculation
         ITASK = 2
         CALL PLANT (ITASK, OUTPUT, TERMNL, TIME, DAY, DELT,
     &               weather variables)

*        time update, TIME multiple of PRDEL ? => OUTPUT = .TRUE.
         CALL TIMER (ITASK, DELT, PRDEL, FINTIM, TIME, DAY, TERMNL, OUTPUT)
      GOTO 10
      END IF

*     terminal calculations
      ITASK = 4
      CALL PLANT (ITASK, OUTPUT, TERMNL, TIME, DAY, DELT, ...)

      STOP
      END

*-----------------------------------------------------------------

      SUBROUTINE PLANT (ITASK, OUTPUT, TERMNL, TIME, DAY, DELT,
     &                  weather variables)
      IMPLICIT REAL (A-Z)
      LOGICAL TERMNL, OUTPUT
      INTEGER ITASK
      SAVE

      IF (ITASK.EQ.1) THEN
*        initialization of states and parameters
         WLVI = 6.8
         WLV = WLVI
         ...
      ELSE IF (ITASK.EQ.2) THEN
*        rate calculation and output
         GLV = ...
         ...
         IF (OUTPUT .OR. TERMNL) THEN
*           output to file with unit=IUNITO
            WRITE (IUNITO, format) WLV, ...
         END IF
      ELSE IF (ITASK.EQ.3) THEN
*        state calculation and check if TERMNL must be .TRUE.
         WLV = WLV+GLV*DELT
```

```
            ...
            IF (DVS.GE.2.0) TERMNL = .TRUE.
         ELSE IF (ITASK.EQ.4) THEN
*           terminal calculations
            ...
         END IF

         RETURN
         END
```

The crop growth simulation program thus consists of a main program and one or more subroutines (similar to Listing 4) that represent the different subprocesses. Because of the complexity of the descriptions of the subprocesses (soil water and plant), and the limited number of interactions between plant and soil, the soil water and plant subprocesses will normally be separated into different subroutines.

Initializing the states and parameters from external data files

Three of the four sections distinguished in the plant submodel (integration, rate calculation and terminal) 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 4, this can be done by simple assignments such as `WLVI = 6.8`. 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 CSMP. In CSMP the user can run the model with different parameter sets automatically (after the `END` statement) or after parameter values in the CONTRO.SYS file have been changed. To introduce that option in the FSE program too, initial parameters values 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 by supplying the name of the requested variable (after having defined which data file to use). 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. The syntax of the data files is explained in more detail in Chapter 5.

As an example, Listing 5 shows part of the initialization section of a plant routine. An explanation is given below the example listing.

Listing 5: Example illustrating the use of some RD routines.

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

The statement:

```
CALL RDINIT (IUNITP, IUNITO, FILEP)
```

calls the routine that 1) opens the file with variable name `FILEP` using unit=`IUNITP+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=`IUNITP`, 4) closes the data file (leaving `IUNITP` used !!), and 5) sends all error messages that have been created to a log file (with unit=`IUNITO`; this log file must have been opened previously !).

After this call, the plant subroutine can acquire the numerical values (including arrays) through three different RD routines, `RDSREA` (read single real), `RDSINT` (read single integer), and `RDAREA` (read array of reals). The `CLOSE` statement deletes the temporary file that is created by the RD routines.

The corresponding data file PLANT.DAT pertaining to Listing 5 could be as follows:

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

The `TIMER` routine does not initialize itself from a data file like the plant and soil routines, but initial values are extracted from a data file in the main program, after which `TIMER` is initialized with these values.

Implementing 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 the analysis of the effect of a different value for an input parameter. In CSMP this can be done by repeating the parameter that is to be changed after an END statement. In Listing 6, weather data from 1984 are used in the first run; additional runs are made using weather data from 1985 and 1986. This facility in CSMP is called the rerun facility. The output of the different runs is merged in the same output file, for easy comparison.

Listing 6: 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 program. Had this not been implemented, the user would have had to make 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 a clumsy way to do multiple runs.

The general idea behind the rerun facility is that the data files remain identical and that the changes in data are specified in a separate file called 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, 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 rerun sets from RERUNS.DAT have been used. The output of the different runs is merged in 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, variables have to be repeated even if they do not change value. This is explained in more detail in Chapter 5.

It may be deduced from Fig. 3 that the control structure for the reruns should be programmed as a loop around the actual model. In Listing 7 the principle of the reruns is illustrated, using the main program of Listing 3 as a basis. To shorten the text, the contents of the main loop (IF... until END IF) have not been repeated.

Listing 7: Schematic program showing the implementation of reruns.

```
      PROGRAM RERUN
      IMPLICIT REAL (A-Z)
      LOGICAL TERMNL, OUTPUT
      INTEGER I1, ITASK, INSETS

      CALL RDSETS (..., INSETS)

      DO 5 I1=0,INSETS
         CALL RDFROM (I1, ...)

*        initialization of TIMER and PLANT subroutines
         ITASK  = 1
         TERMNL = .FALSE.
         TIME   = 0.0
         FINTIM = 100.
         DELT   = 1.0
         CALL TIMER (ITASK, DELT, PRDEL, FINTIM, TIME, DAY, TERMNL, OUTPUT)
         CALL PLANT (ITASK, OUTPUT, TERMNL, TIME, DAY, DELT, ...)

*        run loop as long as TERMNL is not .TRUE.
10       IF (.NOT.TERMNL) THEN
         < main loop contents removed >
         GOTO 10
         END IF

         ITASK = 4
         CALL PLANT (ITASK, OUTPUT, TERMNL, TIME, DAY, DELT, ...)
5     CONTINUE

      STOP
      END
```

The call to `RDSETS` detects the possible presence of the RERUNS.DAT file and then analyses the data file if there is one. 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 internal to the RD routines. To the plant or the soil water balance routines it appears as if the values used originate from the standard data files !!! Therefore no changes are necessary in the calls, such as the ones in Listing 5, to make reruns possible.

Results from the reruns are written to the output file after analysis of the rerun file and after each replacement. 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 data files and the simulation is halted.

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 routine with different output variables is used.

All subroutines write their output to the same output file (whose name is defined in the FSE main program). By using a set of special routines (the OUT routines), output can be written 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 variables, `OUTARR` for arrays 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 8. 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 $\pi/20$) 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 8: Example program for the use of the OUT routines.

```
      PROGRAM SINE
      IMPLICIT REAL (A-Z)
      REAL 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 = FLOAT (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 of Listing 8:

```
*----------------------------------------------------------------------------
* 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 OUTDAT to store the incoming names and numerical values in a temporary file (with unit=21). The number of combinations of name and value 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 information 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 MS-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 call (see listing 8) is actually an 'interface' call to OUTDAT. What the routine does internally is that it 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 1). This procedure can be repeated several times. Separate print plots can be made of dry weights, weather data etc.. The final call to OUTDAT (with 99) deletes the temporary file.

Table 1: Task variable options that should be supplied to OUTPLT to generate the different print plot types.

		Width	
		132	80
Scaling	Individual	4	6
	Common	5	7

Weather data

The weather data used in the model are read from external files. The weather data file definition, however, is different from those for the RD routines. The weather data system used, has been developed jointly by the Centre for Agrobiological Research and the Department of Theoretical Production Ecology. It is especially suited for use in crop growth simulation models and has been documented in a separate report (van Kraalingen *et al.*, 1991) that can be obtained from the same sources as this documentation. It is an easy system to understand and we will outline it briefly using some introductory paragraphs from van Kraalingen *et al.* (1991). The latter report also contains a list of weather data from all around the world that are currently available on request (this list is reproduced in Appendix D).

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.

The reading program consists of a set of subroutines and functions, only two of which are intended to be called by the user (Listing 10, `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 (screen and/or log file), and 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 (`A` and `B`) pertaining to the selected station, 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 data file requested does not exist). The location parameters can later be used to calculate daylength (from latitude) or average air pressure (from altitude).

After this initialization procedure, weather data for specific days can be obtained by calls to the second subroutine (`WEATHR`) with day number 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 (`RAD`), minimum and maximum air temperature (`TMMN` and `TMMX`), vapour pressure (`VAPOUR`), wind speed (`WIND`) and rainfall (`RAIN`). In Listing 9, the weather data for 1984 from the meteorological station in Wageningen are extracted from the file `NL1.984`:

Listing 9: Example of use of the weather data system.

```
      PROGRAM EXTR
      IMPLICIT REAL (A-Z)
      INTEGER IFLAG, ISTN,  IYEAR, ISTAT, IDATE
      CHARACTER WTRDIR*80, CNTR*6

      IFLAG  = 1101         <- errors to screen and log file, warnings to log file
      WTRDIR = ' '          <- weather files on current directory
      CNTR    = 'NL'       <- country name
      ISTN    = 1          <- station number
      IYEAR   = 1985       <- year number

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

      DO 10 IDAY=1,365
         CALL WEATHR (IDAY, ISTAT, RAD, TMMN, TMMX, VAPOUR, WIND, RAIN)
         < weather for day= IDAY is known, calculations can be done >
10    CONTINUE

      STOP
      END
```

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

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 (RESULTS.OUT). 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 main program that subsequently terminates the simulation run. 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 FSE-MAIN:  cannot  read  weather  directory  and  country  name
                    from TIMER file

WARNING from OUTDAT: zero length variable name
```

5 How to operate the model and its data files

This Chapter describes how the model operates and the syntax of the corresponding data files. Chapter 6 explains how to modify the source code of the program. We assume that you have successfully compiled and linked the FSE program and that you know how to run the model and are able to use an editor to create and modify the data files (otherwise, see Chapter 9).

The following steps have to be taken to use the model:

1) Modify data files if necessary

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 implemented in the data only.

The following data files are used when SUCROS is used with the FSE program :

Name:	**Used by:**	**Contents:**
TIMER.DAT	TIMER	time variables (year, time step, etc.), weather station, country, control variables,
PLANT.DAT	SUCROS	plant parameters and initial values of state variables,
Weather data files	STINFO, WEATHR	daily weather data.
< optional>		
RERUNS.DAT	RDSETS, RDFROM	TIMER and plant parameters (used in reruns),

Note: When models other than SUCROS are used with the FSE program it is possible for more data files to be used and/or for them to have different names.

The data files TIMER.DAT, 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 identical sets of variables. The following syntax rules apply to TIMER.DAT and PLANT.DAT:

- the file consists of names and numerical values of variables, separated by an `'='` sign. So: `PLMX = 20.`, is a valid specification,
- the name of a variable cannot exceed six characters,
- for array variables, more than one numerical value may follow the equal sign, separated by commas,

- identical numerical array values may be given as n*<numerical value>,
- variables may appear in the file in any order,
- 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 record of the data file are read,
- The decimal point in a real value like `20.` may be omitted,
- No tabs or other control and extended ASCII characters are allowed in the file.

These rules are illustrated in Listing 10.

Listing 10: Example data file:

<start of datafile>
```
* example data file
A = 10.                      ! single value
B = 0., 2., 3., 4.           ! array of four elements
C = 10., 20.,                ! array continued on next line
    30., 40.
D = 100*10.                  ! array of 100 elements
E = 10.; F = 20.; G = 30.    ! more than one parameter on single
                             ! line
```
<end of file>

The TIMER.DAT file

This data file specifies 1) the value of the time variables such as time step of integration, time between different outputs to file, etc., 2) the directory in which the weather data are stored, the country code, station number and year, and 3) some miscellaneous control variables. An example file is given in Appendix A. The meaning of the different variables of this file is:

First line with exclamation mark (!):
Very often, large numbers of weather data files will be used. For this reason it is convenient to store these data in a separate directory. By typing a directory name after this exclamation mark, you can direct the weather system to read weather data from that directory. An example line could be:

```
!DISK$DUA1:[<account>.SYS.WEATHER]   <- example directory for VAX-VMS
!C:\SYS\WEATHER\                     <- example directory for IBM-PC and compatibles
!HD40:WEATHER:                       <- example directory for Apple Macintosh
```

Note the colon (:) and backslash (\) characters following some directory strings.

Second line with exclamation mark (!):

This line contains the abbreviated country code; it too is written immediately after the exclamation mark. For a list of available weather data files, their corresponding country codes and station numbers, see van Kraalingen *et al.* (1990) and Appendix D. Example lines are:

`!NL`	<- country code for The Netherlands
`!UK`	<- country code for United Kingdom
`!ITALY`	<- country code for Italy

You may have noted that according to the rules of the data files, any text immediately following an exclamation mark is regarded as comment by the RD routines. However, it is not yet possible to read character strings from a data file with the RD routines. Separate code has been added to the FSE main program to extract these two character strings from the data file without interfering with the RD routines. The rules that apply to these two character strings differ from those that apply to the rest of the file. The rules are: 1) the weather directory should be given immediately after an exclamation mark on the first line after a comment block with asterisks, 2) the country code should be given immediately after an exclamation mark on a new line following the weather directory (see Appendix A). Unlike other variables, these character strings cannot be used in reruns.

`ISTN` and `IYEAR`

These variables refer to weather data and indicate the station number and year that should be used from the country following the second exclamation mark. For example, when the country code is `NL` (The Netherlands), `ISTN=1` and `IYEAR=1984`, the 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 NL1.984 on the directory given after the first exclamation mark.

`DAYB`, `FINTIM`, `PRDEL` and `DELT`

These variables represent the time parameters of the model. `DAYB` is the start day of the whole program; its value should be between 1 and 365. `FINTIM` is the finish time of the simulation. Note that this is counted from the start of simulation! For example when `DAYB = 93`, and `FINTIM = 10`, the simulation will continue until `DAY = 103`. The variable `PRDEL` indicates the time between consecutive outputs to file. For example, when `PRDEL = 5`, output is given each time that `TIME` is a multiple of `PRDEL` (`TIME=5,10,15` etc.). Irrespective of the value of `PRDEL`, output is always given at the start of the simulation (`TIME=0`) and when the simulation is terminated (either `FINTIM = TIME` or some other finish criterion). 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. `DELT` is the time step of integration. The value of `DELT` cannot be chosen freely, but is normally determined by the model that you are using. For SUCROS a value of one day is normally required.

ITABLE, IDTMP and HARDAY

The variable ITABLE defines whether an output table is required (no output table: ITABLE = 0) and if so, what the format should be. A multiple column table (ITABLE = 4) is sufficient for normal printing and viewing. But the normal table format is not very suitable to be imported in spreadsheet or graphics programs. Using ITABLE = 5, a tab-delimited multiple column table which is easily imported in programs such as Excel is generated. A two-column format, suitable for further processing with TTPLOT, is generated using ITABLE = 6.

The variable IDTMP defines whether the file of temporary output data (RES.TMP) should be deleted at termination of the simulation (IDTMP = 0, do not delete, IDTMP = 1, 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 RESULTS.OUT 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 temporary file after termination of the simulation with the TTSELECT program in combination with the TTPLOT graphing program. For this special purpose the temporary file should not be deleted.

The variable HARDAY can be used to force output at day numbers for which harvest data from the field are available. In many cases these harvest 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). A maximum of twenty day numbers can be defined here. A single value of zero indicates that no forced output is required. Examples are:

```
HARDAY = 11.,27.,52.        <- Output is forced on days 11, 27 and 52
HARDAY = 0.                 <- No forced output
```

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, a file named SOIL.DAT containing soil parameters and initial values, has to be present in the appropriate format. Normally, the general syntax rules as discussed above will apply to these data files.

The RERUNS.DAT file

If the file RERUNS.DAT is absent or empty, the model will execute one single run, using the data from the standard data files. By creating a rerun file, the model will execute additional runs with different parameters and/or initial values for the state variables. 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:

```
<start of file>
* 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.
<end of file>
```

Unlike reruns in CSMP, 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.

We discussed the implementation of reruns in the source code of the program in Chapter 4.

2) Run 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 repeatedly on the screen. 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. Data can be recovered from the temporary file, using the OUTREC program (OUTput RECovery, see the section on Error Recovery in this Chapter).

3) Examine output

The model typically creates two output files: RESULTS.OUT and WEATHER.LOG. The RESULTS.OUT file contains the contents of the input files copied into the output file plus the output of the model with the reruns merged below each other in the file. WEATHER.LOG 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. The format of the output file RESULTS.OUT depends on the value of the variable `ITABLE` from the TIMER.DAT file.

As explained in Chapter 3, some of the warnings and error messages generated in the program (except those generated in the weather system), are written to RESULTS.OUT. If this happens it is important to note that the output table is created during the terminal section of the model and that warnings are written to the output file during the dynamic simulation, so that the output table follows the warnings in the output file.

The output can be graphically analysed using the TTSELECT and/or TTPLOT programs.

Error recovery

If a run is terminated by some error from the model, the output file RESULTS.OUT will not contain the results of that specific run. But the results up till the error occurred are written to the temporary file RES.TMP. 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 is generated by a '4' (the default), '5' generates a tab-delimited table (meant to be imported in Excel), '6' generates an output of only two columns at a time. The output table will be written to the file OUTREC.OUT so that any existing RESULTS.OUT file is not deleted.

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

6 How to implement changes to an existing subroutine

You are strongly advised not to make changes in the subroutine structure 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. 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.

If you want to create a new subprocess description (including data file initialization, integration, output, etc.), follow the structure of the subprocess descriptions as closely as possible. This is further outlined in Chapter 7.

Modification of subprocess calculations

First, define the modification in terms of program statements. If new variables are introduced, determine the type for each of them (parameter, driving variable, rate of change or state). If the new variables are local to the subroutine, determine the exact position in the text 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 3). 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. 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                         |  Correct
   ...                        |     ...
ELSE IF (ITASK.EQ.2) THEN     |  ELSE IF (ITASK.EQ.2) THEN
   ...                        |     ...
   A = B*...                  |     B = ...
   B = ...                    |     A = B*...
ELSE IF (ITASK.EQ.3) THEN     |  ELSE IF (ITASK.EQ.3) THEN
   ...                        |     ...
```

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

Add a variable to the output list

In general the output list should appear at the end of the ITASK=2 section. Output however, should only be generated if either the output flag (OUTPUT), or the terminal flag (TERMNL) 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.

Add a finish condition

Each subprocess can terminate the run by setting 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=3 section, as shown in Listing 4. Any additional finish condition should be added here, similar to the existing ones.

Add titles to the output file

As shown in the listings of the modules, in the initial section a subroutine, OUTCOM, that accepts a text string, is called. 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).

Add print plotted variables

In the terminal sections of the subprocesses, calls to OUTPLT are given. The calls with a '1' as the first argument define a 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 3. 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)).

7 How to create a new subprocess description

It will have become clear from the previous chapters, that if a new subprocess is implemented within the FSE program, the actual subroutine should distinguish the four different tasks. Listing 12 shows an empty subroutine that can be used as a starting point for a new subprocess description. Experience has shown that the best sequence for creating a new subprocess description is:

1) begin by defining the integration section and the finish condition(s),
2) initialize the states in the initial section,
3) define the driving variables and the rates in the rate section,
4) define the parameters in the initial section,
5) 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,
6) 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.

To obtain rates and states at every PRDEL and to have both pertain to the same time, it is essential not to change the locations where output is generated and where the finish condition is tested. Even if the value of a rate or driving variable terminates the simulation, the check should be done in the integral section.

The subroutine call should be inserted at four different locations in the main program. These locations have been indicated in the main program with the line:

```
*----< Insert plant call here >
```

Listing 12: Empty subroutine that can be used with the FSE program.

```
      SUBROUTINE PLANT (ITASK, IUNITP, IUNITO, FILEP, OUTPUT, TERMNL,
     &                  TIME , DAY   , DELT  ,
     &                  LAT  ,
     &                  DTR  , TMMN  , TMMX)
      IMPLICIT REAL (A-Z)

*     Formal parameters

      INTEGER   ITASK, IUNITP, IUNITO
      LOGICAL   OUTPUT, TERMNL
      CHARACTER*(*) FILEP

*     Standard local declarations

      INTEGER ITABLE, ITOLD
      SAVE

      DATA ITOLD /4/

      CALL CHKTSK ('PLANT', IUNITO, ITOLD, ITASK)

      IF (ITASK.EQ.1) THEN

         CALL RDINIT (IUNITP, IUNITO, FILEP)

*        Insert RD calls here

         CLOSE (IUNITP, STATUS='DELETE')


      ELSE IF (ITASK.EQ.2) THEN

*        Rate calculation section
         GLV = ...

*        output of states and rates only if it is required
         IF (OUTPUT .OR. TERMNL) THEN
            CALL OUTDAT (2, 0, 'variable name', variable name)
            ...
         END IF

      ELSE IF (ITASK.EQ.3) THEN

*        Integration
         new_state   = INTGRL (old_state  , rate, DELT)

*        Determine the finish conditions of the simulation

         IF (state.GE.???) TERMNL = .TRUE.

      ELSE IF (ITASK.EQ.4) THEN

*        Terminal section

*        Define graph for output
         CALL OUTPLT (1, 'variable name')
         ...
         CALL OUTPLT (6, 'Printplot')

      END IF

      ITOLD = ITASK

      RETURN
      END
```

8 Installing the FSE program

Requirements for running the FSE program

There are few requirements for running the FSE program. Any standard FORTRAN 77 compiler should be able to compile the program successfully, because it has been developed on VAX-VMS, Atari 520ST+, Apple Macintosh II, and several IBM compatibles, using standard FORTRAN 77 compilers.

The minimum RAM memory requirement depends on the computer and the compiler. It should be at least 512 kb.

If you intend to do serious development work with the FSE program or any other FORTRAN program, we recommend you 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 capacities are much better than in most compilers and this will save much time when debugging any FORTRAN program.

If you are working on an IBM-PC or compatible computer, you are advised to use Microsoft FORTRAN 5.0 or any later version. A batch file and some object files are provided on the floppy disk to be used with this compiler. 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 utility source files. Note that complete source listings of all the relevant programs are provided in the appendices.

Contents of the disk:

The disk you receive is a 5.25" double sided 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 (specify your hardware configuration).

In the following we assume that you have received the FSE program with the 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 are:

A:\ FSE \ WHEAT \	FSEWHT.FOR	FSE main program with wheat SUCROS subroutines,
	PLANT.DAT	Plant parameters and initial state variables,
	TIMER.DAT	Weather, time and control variables,
	RERUNS.DAT	Example rerun file,

A:\ SYS \ WEATHER \	NL1.970 ... NL1.990	Weather data, Netherlands, Wageningen, 1970-1990,
A:\ SYS \ F77 \	TTUTIL.LIB	Object library of TTUTIL routines compiled with Microsoft FORTRAN 5.0,
	WEATHER.LIB	Object library of weather system compiled with Microsoft FORTRAN 5.0,
	EXE.BAT	Batch file for compilation, linking and execution for Microsoft FORTRAN 5.0 compiler,
	OUTREC.FOR	Source code of program to recover output when an error has occurred during dynamic simulation. Has to be compiled with the same compiler you will be using with the FSE program.
A:\ FSE \ SUBROUT \	57 FORTRAN files	Source code of TTUTIL routines,
	WEATHER.FOR	Source code of weather system,

General installation

You are advised to create the same directory structure on the disk of the machine you will be working on and to copy the files manually onto these directories. If you are short of disk space you could omit the FORTRAN source code files from the A:\FSE\SUBROUT\ directory as these files are not used during normal work.

Working with Microsoft FORTRAN 5.0 on IBM PC's and compatibles

If you will be working with the Microsoft FORTRAN compiler, make sure you have 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.

Install the MS-compiler files also in the directory C:\SYS\F77. Make sure that the directory of the compiler files (FL.EXE, etc.) is 'in' the PATH.

You can now compile the FORTRAN files, link with the object libraries, and run the program using the following commands (important: do not add '.FOR' to the file name):

`EXE file`	plain compilation, linking and execution of 'file.for',
`EXE file D`	compilation, linking and debugging of 'file.for' (requires much more memory !).

After successful compilation and linking, repeated execution can also be invoked by typing the name of the program. Typing the name of the program after a successful 'EXE file D' command will execute the program without debugging. Renewed debugging can be invoked by typing:

```
CV   file
```

(CV stands for CodeView, the debugger of the Microsoft languages.)

The executable files, created by the EXE command, will run on both XT, AT and PS/2 type computers. They do not require a coprocessor but will use one if it is present.

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 recompile the source files on the \FSE\SUBROUT\ directory 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.

Working on a VAX computer of CABO or TPE

On VAX machines of both computing centres, simple compilation, linking and execution can be done with (note that the TTUTIL and WEATHER logicals refer to another account so that you don't have to have the object libraries):

```
$ FORTRAN/CHECK=BOUNDS/STANDARD file          compilation of 'file.for',
$ LINK file,TTUTIL/LIB,WEATHER/LIB            linking of 'file.obj',
$ RUN file                                    execution of 'file.exe'.
```

Compilation, linking and execution with debugging can be done with:

```
$ FORTRAN/CHECK=BOUNDS/STANDARD/DEBUG/NOOPTIMIZE file
                                              compilation of 'file.for',
$ LINK/DEBUG file,TTUTIL/LIB,WEATHER/LIB      linking of 'file.obj',
$ RUN file                                    execution of 'file.exe'.
```

Note: 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 WEATHER_DATA: as the directory in the TIMER.DAT file.

Working on another VAX computer

Before you can start work with the FSE program you have to compile the TTUTIL and WEATHER.FOR source files on the [<account>.FSE.SUBROUT] directory. This can be done with the commands:

`$ CREATE TTUTIL.F77`	creates an empty file,
`^Z`	
`$ APPEND *.FOR TTUTIL.F77`	appends all TTUTIL files to the empty file,
`$ RENAME TTUTIL.F77 TTUTIL.FOR`	change the file type,
`$ FORTRAN/CHECK=BOUNDS TTUTIL.FOR`	compilation of TTUTIL file,
`$ FORTRAN/CHECK=BOUNDS WEATHER.FOR`	compilation of ' WEATHER.FOR ',

Now insert the object files into an object library:

```
$ LIBRARY/CREATE TTUTIL.OLB
$ LIBRARY/INSERT TTUTIL.OLB TTUTIL.OBJ
$ LIBRARY/CREATE WEATHER.OLB
$ LIBRARY/INSERT WEATHER.OLB WEATHER.OBJ
```

Assign two logicals:

```
$ DEFINE TTUTIL  <diskname>:[<account>.FSE.SUBROUT]TTUTIL.OLB
$ DEFINE WEATHER <diskname>:[<account>.FSE.SUBROUT]WEATHER.OLB
```

(for permanent use, insert these two lines in your LOGIN.COM file)

The FSE program can subsequently be compiled, linked and executed with:

`$ FORTRAN/CHECK=BOUNDS/STANDARD file`	compilation of ' file.for ',
`$ LINK file,TTUTIL/LIB,WEATHER/LIB`	linking of 'file.obj',
`$ RUN file`	execution of 'file.exe'.

Debugging can be done with:

```
$ FORTRAN/CHECK=BOUNDS/STANDARD/DEBUG/NOOPTIMIZE file
                                                      compilation of ' file.for ',
$ LINK/DEBUG file,TTUTIL/LIB,WEATHER/LIB              linking of 'file.obj',
$ RUN file                                            execution of 'file.exe'.
```

Note: 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
```

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

The easiest solution in general is to obtain the two object libraries and link these to the main program. Another solution is to add `INCLUDE` statements at the end of your program, though this may give problems with debugging. The use of a library, is definitely the best solution here, but it is difficult to create a library yourself because the routines have to be inserted in a specific order because the linker cannot resolve backward references.

The object library can be obtained directly from the Department of Theoretical Production Ecology or by submitting a request to one of the addresses mentioned in the introduction (specify your processor and coprocessor type !).

9 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: A FORTRAN-77 Verifier and Programming aid, E.W. Kruyt, Dept. of Physiology. Leiden University. PO Box 9604. 2300 RC Leiden. The Netherlands.

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

Hahn, B.D., Problem solving with FORTRAN-77. Edward Arnold Ltd. London. 247 pp.

Kraalingen, D.W.G. van, and 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).

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

Penning de Vries, F.W.T., D.M. Jansen, H.F.M. ten Berge, A. Bakema, 1989. Simulation of Ecophysiological Processes of Growth in Several Annual Crops. Simulation Monograph 29. PUDOC Wageningen and IRRI, Los Baños. 271 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).

Wagener, J.L., 1980. FORTRAN 77, principles of programming. John Wiley & Sons. New York. 370 pp.

Appendix A: FSE main program, with SUCROS wheat and data files

This Appendix gives the listings of the main program with the SUCROS wheat subroutines, the timer and plant data files. These programs can be found on the disk, in the directory A:\FSE\WHEAT\.

The file order in this Appendix is as follows:

FSEWHT.FOR	Main program and SUCROS subroutines,
TIMER.DAT	Data file containing time and weather variables,
PLANT.DAT	Data file containing parameters and initial values of the states for a crop,

File: A:\FSE\WHEAT\FSEWHT.FOR

```
*----------------------------------------------------------------------*
*                                                                      *
*                          FSE-SUCROS-WHEAT                            *
*                 Simple and Universal CROp Simulator                  *
*                          Version July 1991                           *
*                                                                      *
*                                                                      *
* FORTRAN version of the crop growth simulator SUCROS. This version is *
* based on earlier versions, written in CSMP. References:              *
*                                                                      *
* F.W.T. Penning de Vries & H.H. van Laar (Eds), 1982:                 *
* Simulation of plant growth and crop production.                      *
* Simulation Monograph Series. Pudoc Wageningen. 308 pp.               *
*                                                                      *
* R. Rabbinge, S.A. Ward & H.H. van Laar (Eds), 1989:                  *
* Simulation and systems management in crop protection.                *
* Simulation Monograph 32. PUDOC Wageningen. 420 pp.                   *
*                                                                      *
* The model is programmed, using the FORTRAN Simulation Environment    *
* developed by D.W.G. van Kraalingen. Simulation Reports CABO-TT No.23 *
*                                                                      *
*                                                                      *
* External datafiles needed:  timer.dat                                *
*                             plant.dat                                *
*                             Weather data files                       *
*                             reruns.dat (only when reruns are needed) *
*----------------------------------------------------------------------*


*----------------------------------------------------------------------*
*                                                                      *
*                            MAIN PROGRAM                              *
*                 FORTRAN Simulation Environment (FSE)                 *
*                                                                      *
*----------------------------------------------------------------------*


      PROGRAM MAIN

*-----Standard declarations

      IMPLICIT REAL (A-Z)
      INTEGER ITASK , INSETS, IRUN  , I1, I2, I3
      INTEGER IUNITR, IUNITT, IUNITO, IUNITP, IUNITS
      INTEGER ISTAT1, ISTAT2, IDAY  , IYEAR , ISTN  , ILEN
      INTEGER ITABLE, IDTMP , IMNHD , INHD

      LOGICAL OUTPUT, TERMNL, WTRMES, EOF

      CHARACTER*80 WTRDIR, FILER, FILET, FILEO, FILEP, FILES
      CHARACTER    CNTR*7, DUMMY*1

      PARAMETER (IMNHD=20, TINY=1.E-4)
      REAL HARDAY(IMNHD)

*-----Insert here declarations for use with compartimentalized
*     waterbalances

*-----Unit numbers for rerun (R), timer (T), output (O), plant data (P)
*     and soil data (S) files. WTRMES flags any messages from
*     the weather system
```

```
      IUNITR = 20
      IUNITT = 30
      IUNITO = 40
      IUNITP = 50
      IUNITS = 60
      WTRMES = .FALSE.

*-----Ditto file names

      FILER  = 'RERUNS.DAT'
      FILET  = 'TIMER.DAT'
      FILEO  = 'RESULTS.OUT'
      FILEP  = 'PLANT.DAT'
      FILES  = 'SOIL.DAT'

*-----Open output file and copy contents of timer file to output file

      CALL FOPEN  (IUNITO, FILEO, 'NEW', 'DEL')
      CALL COPFIL (IUNITT, FILET, IUNITO)

*-----Get directory and country name of weather data from timer file

      CALL FOPEN  (IUNITT, FILET ,'OLD', 'NVT')
      CALL GETREC (IUNITT, WTRDIR, EOF)
      CALL GETREC (IUNITT, CNTR  , EOF)
      IF (EOF) CALL ERROR ('FSE-MAIN',
     &         'unexpected end_of_file in TIMER file')
      I2 = ILEN (WTRDIR)
      I3 = ILEN (CNTR)
      IF (I2.NE.0.AND.I3.NE.0) THEN
         I2 = MAX (2,I2)
         I3 = MAX (2,I3)
         WTRDIR(1:I2) = WTRDIR(2:I2)//' '
         CNTR (1:I3)  = CNTR(2:I3)//' '
      ELSE
         CALL ERROR ('FSE-MAIN',
     &'cannot read weather directory and country name from TIMER file')
      END IF
      CLOSE (IUNITT)

*-----Read number of rerun sets, if rerun sets have been defined
*     copy contents of rerun file to output file

      CALL RDSETS (IUNITR, IUNITO, FILER, INSETS)
      IF (INSETS.GT.0) CALL COPFIL (IUNITR+1, FILER, IUNITO)


*======================================================================*
*======================================================================*
*                                                                      *
*                   Main loop and rerun begins here                    *
*                                                                      *
*======================================================================*
*======================================================================*


      DO 10 I1=0,INSETS

      IRUN = I1+1
      WRITE (*,'(A)') ' '

*-----Select data set
      CALL RDFROM (I1, .TRUE.)


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

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

      ITASK  = 1
      TERMNL = .FALSE.

*-----Read variables from timer file

      CALL RDINIT (IUNITT  , IUNITO, FILET)
      CALL RDSREA ('DAYB'  , DAYB  )
      CALL RDSREA ('FINTIM', FINTIM)
      CALL RDSREA ('PRDEL' , PRDEL )
      CALL RDSREA ('DELT'  , DELT  )
      CALL RDSINT ('IYEAR' , IYEAR )
      CALL RDSINT ('ISTN'  , ISTN  )
      CALL RDSINT ('ITABLE', ITABLE)
      CALL RDSINT ('IDTMP' , IDTMP )
      CALL RDAREA ('HARDAY', HARDAY, IMNHD, INHD)
      CLOSE (IUNITT, STATUS='DELETE')


*-----Initialize TIMER and OUTDAT routines
      CALL TIMER (ITASK, DAYB, DELT, PRDEL, FINTIM,
     &            IYEAR, TIME, DAY , IDAY , TERMNL, OUTPUT)

      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 simulation.

      CALL STINFO (1100   , WTRDIR, ' ', CNTR, ISTN, IYEAR,
     &             ISTAT1, LONG   , LAT, ELEV, A, B)
      CALL WEATHR (IDAY, ISTAT2, RDD, TMMN, TMMX, VP, WN, RAIN)
      IF (ISTAT1.NE.0.OR.ISTAT2.NE.0) WTRMES = .TRUE.

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

*     < Insert water balance call here if required >


*     < Insert plant call here >
      CALL SUCROS (ITASK, IUNITP, IUNITO, FILEP, OUTPUT, TERMNL,
     &             DAY   , DELT   ,
     &             LAT   ,
     &             RDD   , TMMN   , TMMX)


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


20    IF (.NOT.TERMNL) THEN

      WRITE (*,'(A,I3,A,I5,A,F7.2)')
     &  ' Run:', IRUN, ', Year:', IYEAR, ', Day:', DAY

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

```
      IF (ITASK.NE.1) THEN

         ITASK = 3

*--------< Insert plant call here >
         CALL SUCROS (ITASK, IUNITP, IUNITO, FILEP, OUTPUT, TERMNL,
     &                DAY  , DELT  ,
     &                LAT  ,
     &                RDD  , TMMN  , TMMX)

*--------< Insert water balance call here if required >

      END IF

      ITASK = 2

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

*-----Get weather data for new day and flag messages
      CALL STINFO (1100   , WTRDIR, ' ', CNTR, ISTN, IYEAR,
     &             ISTAT1, LONG   , LAT, ELEV, A, B)
      CALL WEATHR (IDAY, ISTAT2, RDD, TMMN, TMMX, VP, WN, RAIN)
      IF (ISTAT1.NE.0.OR.ISTAT2.NE.0) WTRMES = .TRUE.

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

      IF (OUTPUT.OR.TERMNL) THEN
         CALL OUTDAT (ITASK, IUNITO, 'TIME', TIME)
         CALL OUTDAT (ITASK, IUNITO, 'DAY' , DAY)
      END IF

*-----------------------------------------------------------------------*
*                      Calculation of rates section                      *
*-----------------------------------------------------------------------*

*-----< Insert plant call here >
      CALL SUCROS (ITASK, IUNITP, IUNITO, FILEP, OUTPUT, TERMNL,
     &             DAY  , DELT  ,
     &             LAT  ,
     &             RDD  , TMMN  , TMMX)


*-----< Insert potential soil evaporation call here if required >


*-----< Insert water balance call here if required >


*-----Time variables update, check for FINTIM and OUTPUT
      CALL TIMER (ITASK, DAYB, DELT, PRDEL, FINTIM,
     &            IYEAR, TIME, DAY , IDAY , TERMNL, OUTPUT)

*-----Generate output to file if day is equal to a harvest day

      IF (HARDAY(1).NE.0.) THEN
         DO 30 I2=1,INHD
            IF (DAY.GT.(HARDAY(I2)-TINY).AND.DAY.LT.(HARDAY(I2)+TINY))
     &          OUTPUT = .TRUE.
30       CONTINUE
      ELSE IF (INHD.GT.1) THEN
         CALL ERROR ('FSE-MAIN',
     &        'harvest data in timer file not correct')
      END IF
      GOTO 20
      END IF

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

      ITASK = 4

*-----Generate output file dependent on option from timer file
      IF (ITABLE.GE.4) CALL OUTDAT (ITABLE, 20, ' ',0.)

*-----< Insert plant call here >
      CALL SUCROS (ITASK, IUNITP, IUNITO, FILEP, OUTPUT, TERMNL,
     &             DAY   , DELT  ,
     &             LAT   ,
     &             RDD   , TMMN  , TMMX)


*-----< Insert water balance call here if required >


*-----Delete temporary output file dependent on switch from timer file
      IF (IDTMP.EQ.1) CALL OUTDAT (99, 0, ' ', 0.)

10    CONTINUE

*-----Delete temporary rerun file if reruns were carried out
      IF (INSETS.GT.0) CLOSE (IUNITR, STATUS='DELETE')

*-----Write message to screen and log file if warnings and/or errors
*     have occurred, pause and wait for return from user

      IF (WTRMES) THEN

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

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

      END IF

      STOP
      END


*----------------------------------------------------------------------*
*  SUBROUTINE SUCROS                                                   *
*  Authors: various authors                                            *
*  Date   : 14-Nov-1990                                                *
*  Purpose: This subroutine is the spring wheat version of SUCROS, to  *
*           calculate growth in situations of potential production.    *
*  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   *
*  IUNITP  I4  Unit of input file with plant data          -       I   *
*  IUNITO  I4  Unit of output file                         -       I   *
*  FILEP   C*  Name of file with plant 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 *
*  DAY     R4  Day number                                      d       I  *
*  DELT    R4  Time step of integration                        d       I  *
*  LAT     R4  Latitude of site                              degrees   I  *
*  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  *
*                                                                         *
*  FATAL ERROR CHECKS (execution terminated, message)                     *
*  condition: FSO < -0.001, DELT < 1, LAT < -98, RDD < -98, TMMN < -98 *
*             TMMX < -98, checks internal to other subroutines            *
*                                                                         *
*  SUBROUTINES and FUNCTIONS called : TOTASS, ASTRO, GLA,                 *
*        CHKTSK, COPFIL, ERROR, OUTCOM, OUTDAT, OUTPLT,                   *
*        RDINIT, RDSREA, RDAREA                                           *
*  FILE usage : IUNITP, IUNITO                                            *
*-------------------------------------------------------------------------*

      SUBROUTINE SUCROS (ITASK, IUNITP, IUNITO, FILEP, OUTPUT, TERMNL,
     &                   DAY  , DELT  ,
     &                   LAT  ,
     &                   RDD  , TMMN  , TMMX)

      IMPLICIT REAL (A-Z)

*     Formal parameters

      INTEGER  ITASK , IUNITP, IUNITO
      LOGICAL  OUTPUT, TERMNL

      CHARACTER*(*) FILEP

*     Standard local declarations

      INTEGER ITABLE, ITOLD, IEMERG
      PARAMETER (ITABLE=100)
      LOGICAL INIT

      REAL AMDVST(ITABLE), AMTMPT(ITABLE), DVRVT(ITABLE), RDRTB(ITABLE)
      INTEGER      IAMDVN,         IAMTMN,        IDVRVN,         IRDRN

      REAL  DVRRT(ITABLE),  FSHTB(ITABLE), FLVTB(ITABLE), FSTTB(ITABLE)
      INTEGER      IDVRRN,          IFSHN,         IFLVN,         IFSTN

      SAVE

      DATA ITOLD /4/, INIT /.FALSE./

*     Check the new task against the old task, and check weather data
      CALL CHKTSK ('SUCROS', IUNITO, ITOLD, ITASK)

      IF (LAT.LT.-98..OR.RDD.LT.-98..OR.TMMN.LT.-98..OR.TMMX.LT.-98.)
     &    CALL ERROR ('SUCROS',
     &        'missing weather data, check file WEATHER.LOG')

      IF (ITASK.EQ.1) THEN

*        copy input file to output file once !

         IF (.NOT.INIT) THEN
            CALL COPFIL (IUNITP, FILEP, IUNITO)
            INIT = .TRUE.
         END IF

*        Send title to output file

         CALL OUTCOM ('SUCROS, Plant growth at potential production')
```

```
*        Initialization section

         IF (DELT.LT.1.0) CALL ERROR
     &      ('SUCROS','DELT too small for SUCROS')

         CALL RDINIT (IUNITP, IUNITO, FILEP)

*        Initialization of states
         CALL RDSREA ('NPL'   , NPL)
         CALL RDSREA ('LA0'   , LA0)
         CALL RDSREA ('RGRL'  , RGRL)
         CALL RDSREA ('TMBJUV', TMBJUV)
         CALL RDSREA ('SLA'   , SLA)
         CALL RDSREA ('AMX'   , AMX)
         CALL RDSREA ('EFF'   , EFF)
         CALL RDSREA ('KDIF'  , KDIF)
         CALL RDSREA ('SCV'   , SCV)
         CALL RDSREA ('Q10'   , Q10)
         CALL RDSREA ('MAINSO', MAINSO)
         CALL RDSREA ('ASRQSO', ASRQSO)
         CALL RDSREA ('EAR'   , EAR)
         CALL RDSREA ('WLVG'  , WLVG)
         CALL RDSREA ('WST'   , WST)
         CALL RDSREA ('WSO'   , WSO)
         CALL RDSREA ('WRT'   , WRT)
         CALL RDSREA ('DVS'   , DVS)
         CALL RDSREA ('TMSUME', TMSUME)
         CALL RDSREA ('EMERG' , EMERG)
         CALL RDSREA ('DAYSOW', DAYSOW)

*        read in tables
         CALL RDAREA ('AMDVST', AMDVST, ITABLE, IAMDVN)
         CALL RDAREA ('AMTMPT', AMTMPT, ITABLE, IAMTMN)
         CALL RDAREA ('DVRVT' , DVRVT , ITABLE, IDVRVN)
         CALL RDAREA ('DVRRT' , DVRRT , ITABLE, IDVRRN)
         CALL RDAREA ('FSHTB' , FSHTB , ITABLE, IFSHN)
         CALL RDAREA ('FLVTB' , FLVTB , ITABLE, IFLVN)
         CALL RDAREA ('FSTTB' , FSTTB , ITABLE, IFSTN)
         CALL RDAREA ('RDRTB' , RDRTB , ITABLE, IRDRN)

         CLOSE (IUNITP, STATUS='DELETE')

         WLVD   = 0.
         EAI    = 0.
         LAII   = NPL * LA0 * 1.E-4
         LAI    = 0.
         WLV    = WLVG + WLVD
         TADRW  = WLV+WST+WSO
         LAIP   = LAI+0.5*EAI
         IEMERG = 0

      ELSE IF (ITASK.EQ.2) THEN

*        rate calculation section

*        daily temperature ( C): maximum, minimum, average, daytime and
*        effective

         TMAV   = 0.5 * (TMMX+TMMN)
         TMAVD  = TMMX - 0.25 * (TMMX-TMMN)
         TMELV  = MAX (0., TMAV-TMBJUV)
         TMEFF  = Q10**((TMAV-25.)/10.)

*        subroutine ASTRO computes daylength and daily radiation
*        characteristics from Julian day, latitude and measured daily total
*        global radiation
```

```
      CALL ASTRO (DAY, LAT,
     &            SC , DS0, SINLD, COSLD, DAYL, DSINB, DSINBE)

*     emergence process begins after sowing
      DEMERG = MAX (0., TMAV-3.)

*     leaf photosynthesis rate at light saturation (kg CO2/ha leaf/h)
      AMDVS = LINT (AMDVST, IAMDVN, DVS)
      AMTMP = LINT (AMTMPT, IAMTMN, TMAVD)
      AMAX  = AMX * AMDVS * AMTMP


*     subroutine TOTASS computes daily total gross assimilation (DTGA)
      CALL TOTASS (SC , DAYL, SINLD, COSLD, DSINBE, RDD,
     &             SCV, AMAX, EFF  , KDIF , LAIP  , DTGA)

*     conversion from assimilated CO2 to CH2O
      GPHOT = DTGA * 30./44.

*     maintenance respiration (kg CH2O/ha/d)

      IF (WLV.GT.0) THEN
         MNDVS = WLVG / WLV
      ELSE
         MNDVS = 1.
      ENDIF

      MAINTS = 0.03*WLV + 0.015*WST + 0.015*WRT + MAINSO*WSO
      MAINT  = MIN (GPHOT, MAINTS * TMEFF * MNDVS)

*     fraction of dry matter growth occurring in shoots, leaves, stems,
*     storage organs and roots

      FSH = LINT (FSHTB, IFSHN, DVS)
      FLV = LINT (FLVTB, IFLVN, DVS)
      FST = LINT (FSTTB, IFSTN, DVS)
      FSO = 1. - FLV - FST
      FRT = 1. - FSH

      IF (FSO.LT.-0.001) THEN
         CALL ERROR ('SUCROS', 'FSO negative')
      ELSE IF (FSO.GT.-0.001.AND.FSO.LT.0.) THEN
         FSO = 0.
      END IF

*     assimilate requirements for dry matter conversion (kgCH2O/kgDM)
      ASRQ = FSH * (1.46*FLV + 1.51*FST + ASRQSO*FSO) + 1.44*FRT

*     relative death rate of leaves and development rate
      IF (IEMERG.EQ.0) THEN
         RDR = 0.
         DVR = 0.
      ELSE IF (DVS.LT.1.) THEN
         RDR = 0.
         DVR = LINT (DVRVT, IDVRVN, TMAV)
      ELSE
         RDR = LINT (RDRTB, IRDRN , TMAV)
         DVR = LINT (DVRRT, IDVRRN, TMAV)
      ENDIF

      GTW   = (GPHOT - MAINT) / ASRQ
      GSH   = FSH  * GTW
      GLV   = FLV  * GSH
      GST   = FST  * GSH
      GSO   = FSO  * GSH
      GRT   = FRT  * GTW
      DLV   = WLVG * RDR
```

```
         GLAI = IEMERG * GLA (TMELV, DVS, LAII, LAI, RGRL, TMSUME, SLA,
     &                           GLV   , DLV)

         IF (DVS.LT.1.3) THEN
            GEAI = 0.
         ELSE
            GEAI = -RDR * EAI
         ENDIF

*        output of states and rates only if it is required

         IF (OUTPUT .OR. TERMNL) THEN

*           states
            CALL OUTDAT (2, 0, 'DVS'    , DVS)
            CALL OUTDAT (2, 0, 'TADRW'  , TADRW)
            CALL OUTDAT (2, 0, 'LAI'    , LAI)
            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, 'WLVG'   , WLVG)
            CALL OUTDAT (2, 0, 'WLVD'   , WLVD)
            CALL OUTDAT (2, 0, 'EAI'    , EAI)
            CALL OUTDAT (2, 0, 'TMSUME', TMSUME)
            CALL OUTDAT (2, 0, 'EMERG'  , EMERG)

*           driving variables and rates
            CALL OUTDAT (2, 0, 'RDD'    , RDD)
            CALL OUTDAT (2, 0, 'TMAV'   , TMAV)
            CALL OUTDAT (2, 0, 'TMAVD'  , TMAVD)
            CALL OUTDAT (2, 0, 'AMDVS'  , AMDVS)
            CALL OUTDAT (2, 0, 'AMTMP'  , AMTMP)
            CALL OUTDAT (2, 0, 'AMAX'   , AMAX)
            CALL OUTDAT (2, 0, 'GPHOT'  , GPHOT)
            CALL OUTDAT (2, 0, 'MAINTS', MAINTS)
            CALL OUTDAT (2, 0, 'TMEFF'  , TMEFF)
            CALL OUTDAT (2, 0, 'MNDVS'  , MNDVS)
            CALL OUTDAT (2, 0, 'MAINT'  , MAINT)
            CALL OUTDAT (2, 0, 'FSH'    , FSH)
            CALL OUTDAT (2, 0, 'FLV'    , FLV)
            CALL OUTDAT (2, 0, 'FST'    , FST)
            CALL OUTDAT (2, 0, 'FSO'    , FSO)
            CALL OUTDAT (2, 0, 'FRT'    , FRT)
            CALL OUTDAT (2, 0, 'ASRQ'   , ASRQ)
            CALL OUTDAT (2, 0, 'RDR'    , RDR)
            CALL OUTDAT (2, 0, 'DVR'    , DVR)
            CALL OUTDAT (2, 0, 'GTW'    , GTW)
            CALL OUTDAT (2, 0, 'GLV'    , GLV)
            CALL OUTDAT (2, 0, 'GST'    , GST)
            CALL OUTDAT (2, 0, 'GSO'    , GSO)
            CALL OUTDAT (2, 0, 'GRT'    , GRT)
            CALL OUTDAT (2, 0, 'DLV'    , DLV)
            CALL OUTDAT (2, 0, 'GLAI'   , GLAI)
         END IF

      ELSE IF (ITASK.EQ.3) THEN

*        integration
         WLVG   = INTGRL (WLVG  , GLV-DLV, DELT)
         WLVD   = INTGRL (WLVD  , DLV    , DELT)
         WST    = INTGRL (WST   , GST    , DELT)
         WSO    = INTGRL (WSO   , GSO    , DELT)
         WRT    = INTGRL (WRT   , GRT    , DELT)
         EAI    = INTGRL (EAI   , GEAI   , DELT)
         LAI    = INTGRL (LAI   , GLAI   , DELT)
         DVS    = INTGRL (DVS   , DVR    , DELT)
         TMSUME = INTGRL (TMSUME, TMELV*IEMERG , DELT)
         EMERG  = INTGRL (EMERG , DEMERG , DELT)

*        operations on state variables

         IF (EMERG.GE.120..AND.IEMERG.EQ.0) THEN
            IEMERG = 1
            LAI = LAII
         END IF

         WLV   = WLVG + WLVD
         TADRW = WLV + WST + WSO

*        at DVS > 0.8 ears are pushed into the green area:

         IF (DVS.GT.0.8 .AND. EAI.EQ.0.) EAI = EAR * TADRW
         LAIP = LAI + 0.5 * EAI

*        Determine the finish conditions of the simulation

         IF (DVS.GE.2.0) TERMNL = .TRUE.

      ELSE IF (ITASK.EQ.4) THEN

*        Define graph for output
*        use individual scale, small plot width for output

         CALL OUTPLT (1, 'TADRW')
         CALL OUTPLT (1, 'LAI')
         CALL OUTPLT (1, 'WSO')
         CALL OUTPLT (6, 'Printplot of SUCROS')

      END IF

      ITOLD = ITASK

      RETURN
      END

*----------------------------------------------------------------------*
* Function GLA:                                                        *
* Computes daily increase of leaf area index (ha leaf/ ha ground/ d)   *
*----------------------------------------------------------------------*

      REAL FUNCTION GLA (TMELV, DVS, LAII, LAI, RGRL, TMSUME, SLA,
     &                   GLV  , DLV)
      IMPLICIT REAL (A-Z)
      SAVE

      IF (DVS.LT.0.3 .AND. LAI.LT.0.75) THEN
*        during juvenile growth:
         GLA = LAII * RGRL * TMELV * EXP (RGRL * TMSUME)
      ELSE
*        during mature plant growth:
         GLA = SLA * (GLV - DLV)
      ENDIF

      RETURN
      END

*----------------------------------------------------------------------*
*  SUBROUTINE ASTRO                                                    *
*  Authors: Daniel van Kraalingen                                      *
*  Date   : 9-Aug-1987                                                 *
*  Modified by Jan Goudriaan 4 Febr 1988                               *
*  Modified by Jan Goudriaan and Kees Spitters 7 December 1989         *
*  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 *
*  ----   ---- -------                                      -----  ----- *
*  DAY     R4  Day number (Jan 1st = 1)                       -       I  *
*  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  Astronomical 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                                        *
*                                                                        *
*  SUBROUTINES and FUNCTIONS called : ERROR                              *
*  FILE usage : none                                                     *
*-----------------------------------------------------------------------*

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

*-----PI and conversion factor from degrees to radians
      PARAMETER (PI=3.141592654, RAD=0.017453292)

*-----check on input range of parameters
      IF (LAT.GT.67.)  CALL ERROR ('ASTRO','LAT > 67')
      IF (LAT.LT.-67.) CALL ERROR ('ASTRO','LAT < -67')

*-----declination of the sun as function of daynumber (DAY)
      DEC = -ASIN (SIN (23.45*RAD)*COS (2.*PI*(DAY+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 (DS0)
      SC  = 1370.*(1.+0.033*COS (2.*PI*DAY/365.))
      DS0 = SC*DSINB

      RETURN
      END

*-----------------------------------------------------------------------*
*  SUBROUTINE TOTASS                                                     *
*  Authors: Daniel van Kraalingen                                        *
*  Date   : 10-Dec-1987                                                  *
*  Modified by Jan Goudriaan 5-Febr-1988                                 *
*  Modified by Jan Goudriaan and Kees Spitters 7 December 1989           *
*  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 *
*  ----   ---- -------                                   -----  ----- *
*  SC      R4  Solar constant                           J m-2 s-1 I   *
*  DAYL    R4  Astronomical daylength (base = 0 degrees     h      I   *
*  SINLD   R4  Seasonal offset of sine of solar height      -      I   *
*  COSLD   R4  Amplitude of sine of solar height            -      I   *
*  DSINBE  R4  Daily total of effective solar height        s      I   *
*  RDD     R4  Daily total of global radiation           J/m2/d    I   *
*  SCV     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 use efficiency             kg CO2/J/  I   *
*                                                       ha/h m2 s      *
*  KDIF    R4  Extinction coefficient for diffuse light            I   *
*  LAI     R4  Leaf area index                            ha/ha    I   *
*  DTGA    R4  Daily total gross assimilation          kg CO2/ha/d O   *
*                                                                      *
*                                                                      *
*  SUBROUTINES and FUNCTIONS called : ASSIM                            *
*  FILE usage : none                                                   *
*----------------------------------------------------------------------*

      SUBROUTINE TOTASS (SC , DAYL, SINLD, COSLD, DSINBE,
     &                   RDD, SCV , AMAX , EFF  , KDIF  , LAI, DTGA)
      IMPLICIT REAL(A-Z)
      REAL XGAUSS(3), WGAUSS(3)
      INTEGER I1, IGAUSS
      SAVE

      PARAMETER (PI=3.141592654)

      DATA IGAUSS /3/
      DATA XGAUSS /0.1127, 0.5000, 0.8873/
      DATA WGAUSS /0.2778, 0.4444, 0.2778/

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

      DO 10 I1=1,IGAUSS

*--------at the specified HOUR, radiation is computed and used to compute
*        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 (FRDIF) from atmospheric
*        transmission (ATMTR)
         PAR   = 0.5*RDD*SINB*(1.+0.4*SINB)/DSINBE
         ATMTR = PAR/(0.5*SC*SINB)

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

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

*--------diffuse PAR (PARDIF) and direct PAR (PARDIR)
         PARDIF = MIN (PAR, SINB*FRDIF*ATMTR*0.5*SC)
         PARDIR = PAR-PARDIF

         CALL ASSIM (SCV,AMAX,EFF,KDIF,LAI,SINB,PARDIR,PARDIF,FGROS)
```

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

10    CONTINUE

      DTGA = DTGA * DAYL

      RETURN
      END

*----------------------------------------------------------------------*
*  SUBROUTINE ASSIM                                                    *
*  Authors: Daniel van Kraalingen                                      *
*  Date   : 10-Dec-1987                                                *
*  Modified by Jan Goudriaan 5-Febr-1988                               *
*  Purpose: This subroutine performs a Gaussian integration over       *
*           depth of canopy by selecting three 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 *
*  ----   ---- -------                                  -----  ----- *
*  SCV     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 use efficiency            kg CO2/J/ I   *
*                                                      ha/h m2 s     *
*  KDIF    R4  Extinction coefficient for diffuse light          I   *
*  LAI     R4  Leaf area index                           ha/ha   I   *
*  SINB    R4  Sine of solar height                        -     I   *
*  PARDIR  R4  Instantaneous flux of direct radiation (PAR) W/m2 I   *
*  PARDIF  R4  Instantaneous flux of diffuse radiation(PAR) W/m2 I   *
*  FGROS   R4  Instantaneous assimilation rate of       kg CO2/  O   *
*              whole canopy                            ha soil/h     *
*                                                                      *
*  SUBROUTINES and FUNCTIONS called : none                             *
*  FILE usage : none                                                   *
*----------------------------------------------------------------------*

      SUBROUTINE ASSIM (SCV, AMAX, EFF, KDIF, LAI, SINB, PARDIR, PARDIF,
     &                  FGROS)
      IMPLICIT REAL(A-Z)
      REAL XGAUSS(3), WGAUSS(3)
      INTEGER I1, I2, IGAUSS
      SAVE

*-----Gauss weights for three point Gauss
      DATA IGAUSS /3/
      DATA XGAUSS /0.1127, 0.5000, 0.8873/
      DATA WGAUSS /0.2778, 0.4444, 0.2778/

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

*-----extinction coefficient for direct radiation and total direct flux
      CLUSTF = KDIF / (0.8*SQV)
      KDIRBL = (0.5/SINB) * CLUSTF
      KDIRT  = KDIRBL * 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)*PARDIF*KDIF  *EXP (-KDIF  *LAIC)
         VIST  = (1.-REFS)*PARDIR*KDIRT *EXP (-KDIRT *LAIC)
         VISD  = (1.-SCV) *PARDIR*KDIRBL*EXP (-KDIRBL*LAIC)

*--------absorbed flux (J/M2 leaf/s) for shaded leaves and assimilation of
*        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.-SCV) * PARDIR / 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(-KDIRBL*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: A:\FSE\WHEAT\TIMER.DAT

```
* Put the directory where your weather data are stored behind
* the first exclamation mark and the country code behind the second.
* These variables cannot be used in rerun files !
* do not use blank lines between this comment block and the two lines
* with the exclamation mark !!
* Directories are likely to be on
*
*    PC : C:\SYS\WEATHER\
*    VAX: <diskname>:[<account>.SYS.WEATHER]
*    MAC: HD40:WEATHER:
*
!C:\SYS\WEATHER\
!NL

ISTN   = 1         ! Station number of weather data
IYEAR  = 1984      ! Year of weather data

*
* Time variables and output file options
*

DAYB   = 90.       ! Start day of simulation
FINTIM = 1000.     ! Finish time of simulation
PRDEL  = 10.       ! Time between consecutive outputs to file
DELT   = 1.        ! Time step of integration
ITABLE = 4         ! Format of output file:
                   ! (0 = no output table, 4 = normal table,
                   !  5 = Tab-delimited (for Excel), 6 = TTPLOT format)
IDTMP  = 1         ! Switch variable what should be done with the
                   ! temporary output file (0 = do not delete,
                   ! 1 = delete)
HARDAY = 0.
                   ! List of harvest data for which output is required
                   ! 0 = no harvest data
```

File: A:\FSE\WHEAT\PLANT.DAT

```
* Data file with species parameters for spring wheat

* Initial conditions

WLVG   = 0.
WST    = 0.
WSO    = 0.
WRT    = 0.
DVS    = 0.
TMSUME = 0.
EMERG  = 0.
DAYSOW = 90.
NPL    = 210.
LA0    = 0.57
RGRL   = 0.014
TMBJUV  = 0.
SLA    = 0.0022
AMX    = 40.
EFF    = 0.45
KDIF   = 0.6
SCV    = 0.2
Q10    = 2.
MAINSO = 0.01
ASRQSO = 1.41
EAR    = 6.3E-5

* AMDVST, relative effect of development stage on Amax
  ~~AMDVST  = 0.,1.0,   1.,1.0,     2.,0.5,     2.5,0.~~

* AMTMPT, relative effect of temperature on Amax
  AMTMPT = -30.,0.,  0.,0.,  10.,1.0,  25.,1.0,   35.,0.,  50.,0.

* DVRVT, effect of temperature on rate of development in vegetative stage
  DVRVT  = -30.,0.0,      0.,0.,    30.,0.0377

* DVRRT, effect of temperature on rate of development in reproductive stage
  DVRRT  = -30.,0.,      0.,0.,    30.,0.033

* FSHTB, partitioning to shoot as affected by DVS
  FSHTB =  0.0, 0.500,   0.1, 0.500,   0.2, 0.600,   0.350, 0.780,
           0.4, 0.830,   0.5, 0.870,   0.6, 0.900,   0.700, 0.930,
           0.8, 0.950,   0.9, 0.970,   1.0, 0.980,   1.100, 0.990,
           1.2, 1.000,   2.5, 1.000

* FLVTB, partitioning of shoot growth to leaves as affected by DVS
  FLVTB = 0.0, 0.650,  0.10, 0.650,   0.25, 0.7,  0.5,0.5,
          0.7, 0.150,  0.95, 0.000,   2.50, 0.0

* FSTTB, partitioning of shoot growth to stems as affected by DVS
  FSTTB = 0.0, 0.35,   0.10, 0.35,  0.25, 0.3,   0.5,0.5,
          0.7, 0.85,   0.95, 1.00,  1.05, 0.0,   2.5,0.0

* RDRTB, relative death rate of leaves as affected by temperature
  RDRTB = 0.0, 0.03,  10.0,0.03,  15.0,0.04,  30.0,0.09
```

Appendix B: Listing of names of variables in the main program

Name	Meaning	Unit
A	A parameter of Ångström formula (only used with sunshine duration data)	-
B	B parameter of Ångström formula (only used with sunshine duration data)	-
CNTR	Country name	-
COPFIL	Routine to copy one file into another (from library TTUTIL)	-
DAY	Day of simulation (REAL)	d
DAYB	Start day of simulation	d
DELT	Time step of integration	d
DUMMY	Dummy variable used in interactive input	-
ELEV	Elevation of the site	m
ERROR	Subroutine to handle errors that have occurred in the program (errors terminate the program) (from library TTUTIL)	-
EOF	Flag that indicates an end_of_file condition	-
FILEO	Name of output file	-
FILEP	Name of file containing plant data	-
FILER	Name of file containing rerun data	-
FILES	Name of file containing soil data	-
FILET	Name of file containing timer data	-
FINTIM	Finish time of simulation	d
FOPEN	Subroutine to open sequential, formatted files (from library TTUTIL)	-
GETREC	Subroutine to read non-empty, non-comment lines from a file (from library TTUTIL)	-
HARDAY	Array with harvest data at which output is forced	d
I1	Integer counter used for miscellaneous purposes	-
I2	Integer counter used for miscellaneous purposes	-
I3	Integer counter used for miscellaneous purposes	-
IDAY	Day number of simulation (INTEGER)	d
IDTMP	Variable indicates if temporary file should be deleted or not	-
ILEN	Function to determine significant length of string (from library TTUTIL)	-
IMNHD	Maximum number of harvest data in HARDAY	-
INHD	Actual number of harvest data in HARDAY	-
INSETS	Number of rerun sets found on rerun file	-
IRUN	Run number	-
ISTAT1	Status code of STINFO weather routine	-
ISTAT2	Status code of WEATHR routine	-
ISTN	Station number of weather data	-
ITABLE	Format of output table	-
ITASK	Task that should be carried out by subroutines	-
IUNITO	Unit of file used for output	-
IUNITP	Unit of file containing plant data	-

IUNITR	Unit of file containing rerun data	-
IUNITS	Unit of file containing soil data	-
IUNITT	Unit of file containing timer data	-
IYEAR	Year of simulation	-
LAT	Latitude of weather station	[degrees].[decimal minute]
LONG	Longitude of weather station	[degrees].[decimal minute]
OUTDAT	Subroutine to generate output file (from library TTUTIL)	-
OUTPUT	Flag that indicates if output should be generated	-
PRDEL	Time difference between output intervals	d
RAIN	Daily rainfall	mm d^{-1}
RDAREA	Subroutine to read array of real values from file (from library TTUTIL)	-
RDD	Daily shortwave irradiation from weather system	kJ m^{-2} d^{-1}
RDFROM	Subroutine to select a rerun set (from library TTUTIL)	-
RDINIT	Subroutine to initialize RD routines (from library TTUTIL)	-
RDSETS	Subroutine to find number of rerun sets (from library TTUTIL)	-
RDSINT	Subroutine to read single integer variable from file (from library TTUTIL)	-
RDSREA	Subroutine to read single real variable from file (from library TTUTIL)	-
SUCROS	Subroutine that simulates a crop at potential production	-
STINFO	Subroutine to read weather data into internal buffer (from CABO weather system)	-
TERMNL	Flag that indicates if simulation will continue for another time step	-
TIME	Time after start of simulation (always starts at zero !)	d
TIMER	Subroutine that increases time variables (from library TTUTIL)	-
TINY	Small number to avoid problems with rounding	-
TMMN	Daily minimum temperature from weather system	°C
TMMX	Daily maximum temperature from weather system	°C
VP	Daily vapour pressure of air from weather system	kPa
WEATHR	Subroutine to get weather data from internal buffer (from CABO weather system)	-
WN	Daily average wind speed at 2 m height	m s^{-1}
WTRDIR	Directory of the weather data	-
WTRMES	Flag that indicates if warnings and/or errors have occurred from the weather system	-

Appendix C: Listing of names of variables in SUCROS and other subroutines

Name	Meaning	Unit
AMAX	Actual CO_2 assimilation rate at light saturation for individual leaves	kg $(CO_2)ha^{-1}$ (leaf) h^{-1}
AMDVS	Factor accounting for effect of development stage on AMX	-
AMDVST	Table for AMDVS	-
AMTMP	Factor accounting for effect of daytime temperature on AMX	-
AMTMPT	Table for AMTMP	-
AMX	Potential CO_2 assimilation rate at light saturation for individual leaves	kg $(CO_2)ha^{-1}$ (leaf) h^{-1}
AOB	Intermediate variable	-
ASIN	Arcsine function (intrinsic FORTRAN function)	radians
ASRQ	Assimilate (CH_2O) requirement for dry matter production	kg (CH_2O) kg^{-1} (d.m.)
ASRQSO	Assimilate requirement for dry matter production of storage organs	kg (CH_2O) kg^{-1} (d.m.)
ASSIM	Subroutine to calculate FGROS	-
ASTRO	Subroutine to compute astronomic- and photoperiodic day length	-
ATMTR	Atmospheric transmission coefficient	-
CHKTSK	Subroutine to check new task against previous task of simulating subroutines	-
CLUSTF	Cluster factor	-
COS	Cosine function (intrinsic FORTRAN function)	-
COSLD	Intermediate variable in calculating daylength	-
DAY	Day number (1 January =1)	d
DAYL	Daylength	h d^{-1}
DAYSOW	Sowing day, day number since 1 January	d
DEC	Declination of the sun	radians
DELT	Time step of integration (=time step of the model)	d
DEMERG	Daily increase in temperature sum	°C d^{-1}
DLV	Death rate of leaves	kg (leaf) ha^{-1} d^{-1}
DS0	Daily extra-terrestrial irradiation	J m^{-2} d^{-1}
DSINB	Integral of SINB over the day	s d^{-1}
DSINBE	As DSINB, but with a correction for lower atmospheric transmission at lower solar elevations	s d^{-1}
DTGA	Daily total gross assimilation of the crop	kg (CO_2) ha^{-1} d^{-1}
RDD	Daily total solar irradiation	J m^{-2} d^{-1}
DVR	Development rate	d^{-1}
DVRRT	Development rate in pre-anthesis phase as a function of temperature	d^{-1}
DVRVT	Development rate in post-anthesis phase as a function of temperature	d^{-1}
~~DVS~~	~~Development stage of the crop~~	-
EAI	Ear area index	ha (ear) ha^{-1} (soil)
EAR	Ear area ratio	ha (ears 2*one-sided projection) kg^{-1} (shoot)
EFF	Initial light use efficiency for individual leaves	(kg $(CO_2)ha^{-1}$ (leaf) h^{-1})(J m^{-2} (leaf) $s^{-1})^{-1}$

EMERG	Temperature sum for crop to emerge	°C
EXP	Exponent function (intrinsic FORTRAN function)	-
FGL	CO_2 assimilation rate at one depth in the canopy	kg (CO_2) ha^{-1} (leaf) h^{-1}
FGROS	Instantaneous CO_2 assimilation rate of the crop (per unit ground area)	kg (CO_2) ha^{-1} (ground) h^{-1}
FGRS	Intermediate variable for calculation of assimilation of sunlit leaves	-
FGRSH	CO_2 assimilation rate at one depth in the canopy for shaded leaves	kg (CO_2) ha^{-1} (leaf) h^{-1}
FGRSUN	CO_2 assimilation rate at one depth in the canopy for sunlit leaves	kg (CO_2) ha^{-1} (leaf) h^{-1}
FILEP	String variable that contains name of plant-data file	-
FLV	Fraction of shoot d.m. increase allocated to leaves	-
FLVTB	Table for FLV	-
FOPEN	Subroutine to open sequential, formatted files (from library TTUTIL)	-
FRDIF	Diffuse irradiation as a fraction of measured shortwave irradiation	-
FRT	Fraction of total d.m. increase allocated to roots	-
FSH	Fraction of total d.m. increase allocated to shoots	-
FSHTB	Table for FSH	-
FSLLA	Fraction of leaf area that is sunlit	-
FSO	Fraction of shoot d.m. increase allocated to storage organs	-
FST	Fraction of shoot d.m. increase allocated to stems	-
FSTTB	Table for FST	-
GEAI	Net growth rate of ear area index	ha (ear) ha^{-1} d^{-1}
GLA	Subroutine to calculate daily increase of leaf area index	-
GLAI	Net growth rate of leaf area index of the crop	ha (leaf) ha^{-1} (soil) d^{-1}
GLV	D.m. growth rate of leaves	kg (leaf) ha^{-1} d^{-1}
GPHOT	Daily total gross assimilation (CH_2O)	kg (CH_2O) ha^{-1} (soil) d^{-1}
GRT	D.m. growth rate of roots	kg (roots) ha^{-1} (soil) d^{-1}
GSH	D.m. growth rate of shoots	kg (shoots) ha^{-1} (soil) d^{-1}
GSO	D.m. growth rate of storage organs	kg (stor. organ) ha^{-1} (soil) d^{-1}
GST	D.m. growth rate of stems	kg (stem) ha^{-1} (soil) d^{-1}
GTW	Total d.m. growth rate of the crop	kg (d.m.) ha^{-1} (soil) d^{-1}
HOUR	Hour during the day	h
I1	DO-loop counter	-
I2	DO-loop counter	-
IAMDVN	Integer variable for number in table for AMDVS	-
IAMTMN	Integer variable for number of elements in table for AMTMP	-
IDVRRN	Integer variable for number of elements in table for DVRR	-
IDVRVN	Integer variable for number of elements in table for DVRV	-
IEMERG	Variable that indicates if emergence has taken place	-
IFLVN	Integer variable for number of elements in table for FLV	-
IFSHN	Integer variable for number of elements in table for FSH	-
IFSTN	Integer variable for number of elements in table for FST	-
IGAUSS	Number of Gauss points that is used in some numerical integrations	-

ILEN	Function to determine significant length of string (from library TTUTIL)	-
INIT	Flag used to carry out operations only once	-
INTGRL	Function for doing: `state = state+rate*delt` (from library TTUTIL)	-
IRDRN	Integer variable for number of elements in table for RDR	-
ITABLE	Integer variable to set size of local arrays	-
ITASK	Integer variable to indicate the required action of subroutines in main program	-
ITOLD	As ITASK but for a previous call	-
IUNITO	Unit number for output file	-
IUNITP	Unit number for plant data file	-
KDIF	Extinction coefficient for diffuse PAR flux	LAI^{-1}
KDIRBL	Extinction coefficient for direct component of direct PAR flux	LAI^{-1}
KDIRT	Extinction coefficient for total direct PAR flux	LAI^{-1}
LA0	Extrapolated leaf area at field emergence	cm^2 $plant^{-1}$
LAI	Leaf area index	ha (leaf) ha^{-1} (soil)
LAIC	Partial cumulative leaf area index at various canopy depths	ha (leaf) ha^{-1} (soil)
LAII	Initial leaf area index	ha (leaf) ha^{-1} (soil)
LAIP	Total green area to calculate daily assimilation	ha (gr. area) ha^{-1} (soil)
LAT	Latitude of the site	[degrees].[decimal minute]
LINT	Function for linear interpolation (from library TTUTIL)	-
LOG	Function for natural logarithm (intrinsic FORTRAN function)	-
MAINSO	Maintenance respiration coefficient of storage organs	kg (CH_2O) kg^{-1} (d.m.)
MAINT	Maintenance respiration (CH_2O) of the crop	kg (CH_2O) ha^{-1} (soil) d^{-1}
MAINTS	Maintenance respiration (CH_2O) of the crop at reference temperature	kg (CH_2O) ha^{-1} (soil) d^{-1}
MAX	Function for determining highest value (intrinsic FORTRAN function)	-
MIN	Function for determining lowest value (intrinsic FORTRAN function)	-
MNDVS	Factor accounting for effect of development stage on maintenance respiration	-
NPL	Plant density	plants m^{-2}
OUTCOM	Subroutine to write text string to output file (from library TTUTIL)	-
OUTDAT	Subroutine to generate output file (from library TTUTIL)	-
OUTPLT	Subroutine to print plot (from library TTUTIL)	-
OUTPUT	Flag that determines if output has to be generated	-
PAR	Flux of incoming photosynthetically active irradiation	J m^{-2} (soil) s^{-1}
PARDIF	Diffuse flux of incoming PAR	J m^{-2} (soil) s^{-1}
PARDIR	Direct flux of incoming PAR	J m^{-2} (soil) s^{-1}
PI	Ratio of circumference to diameter of circle	-
Q10	Factor accounting for increase in maintenance respiration with a 10 °C rise in temperature	-
RAD	Factor to convert degrees to radians	radians $degree^{-1}$
RDAREA	Subroutine to read array of real values from file (from library TTUTIL)	-
RDD	Daily total solar irradiation	J m^{-2} d^{-1}
RDINIT	Subroutine to initialize RD routines (from library TTUTIL)	-

RDR	Relative death rate of leaves	d^{-1}
RDRTB	Table for RDR	-
RDSINT	Subroutine to read single integer variable from file (from library TTUTIL)	-
RDSREA	Subroutine to read single real variable from file (from library TTUTIL)	-
REFH	Reflection coefficient of canopy with horizontal leaf angle distribution	-
REFS	Reflection coefficient of canopy with spherical leaf angle distribution	-
RGRL	Relative growth rate during exponential leaf area growth	$cm^2\ cm^{-2}\ °C^{-1}\ d^{-1}$
SC	Solar constant, corrected for varying distance sun-earth	$J\ m^{-2}\ s^{-1}$
SCV	Scattering coefficient of leaves for PAR	-
SIN	Function for sine (intrinsic FORTRAN function)	-
SINB	Sine of solar inclination above the horizon	-
SINLD	Intermediate variable in calculating solar declination	-
SLA	Specific area of new leaves	ha (leaf) kg^{-1} (leaf)
SQRT	Function for square root (intrinsic FORTRAN function)	-
SQV	Intermediate variable in calculation of reflection coefficient	-
TADRW	Total above-ground dry weight	kg (d.m.) ha^{-1} (soil)
TERMNL	Logical variable indicating if simulation run has to come to an end	-
TMAV	Daily average temperature	°C
TMAVD	Daily average daytime temperature	°C
TMBJUV	Base temperature for juvenile leaf area growth	°C
TMEFF	Factor accounting for effect of temperature on maintenance respiration	-
TMELV	Daily effective temperature for leaf growth	$°Cd^{-1}$
TMMX	Daily maximum temperature	°C
TMMN	Daily minimum temperature	°C
TMSUME	Temperature sum after emergence	°Cd
TOTASS	Subroutine to calculate gross CO_2 assimilation of canopy	-
VISD	Absorbed direct component of direct flux per unit leaf area (at depth LAIC)	$W\ m^{-2}$ (leaf)
VISDF	Absorbed total direct flux per unit leaf area (at depth LAIC)	$W\ m^{-2}$ (leaf)
VISPP	Absorbed light flux by leaves perpendicular on direct beam	$W\ m^{-2}$ (leaf)
VISSHD	Total absorbed flux for shaded leaves per unit leaf area (at depth LAIC)	$W\ m^{-2}$ (leaf)
VISSUN	Total absorbed flux for sunlit leaves in one of three Gauss point classes	$W\ m^{-2}$ (leaf)
VIST	Absorbed total direct flux per unit leaf area (at depth LAIC)	$W\ m^{-2}$ (leaf)
WGAUSS	Array containing weights to be assigned to Gauss points	-
WLV	Dry weight of leaves (green + dead)	kg (leaves) ha^{-1}
WLVD	Dry weight of dead leaves	kg (dead leaves) ha^{-1}
WLVG	Dry weight of green leaves	kg (green leaves) ha^{-1}
WRT	Dry weight of roots	kg (roots) ha^{-1}
WSO	Dry weight of storage organs	kg (stor. organs) ha^{-1}
WST	Dry weight of stems	kg (stems) ha^{-1}
XGAUSS	Array containing Gauss points	-

Appendix D: List of available weather data (June 1991)

Standardized data files are currently available from the following meteorological stations. Year number 1000 indicates long term average data.

Country name	Station name	Country code	Station code	Years
Austria	Wien	AUSTRI	1	2: 1985-1986
Bangladesh	Joydebpur	BDESH	1	3: 1983-1984, 1986
Belgium	Uccle	BELG	1	4: 1985-1987, 1000
China	Nanjing	CHINA	1	2: 1983-1984
Colombia	Palmira	COLOM	1	4: 1983-1986
Cyprus	Akhelia	CYPRUS	1	2: 1985-1986
Denmark	Thorshavn	DK	1	1: 1000
Denmark	Ålborg	DK	2	1: 1000
Denmark	København / Landbohojskolen	DK	3	1: 1000
Denmark	Roskilde	DK	4	2: 1985-1986
Egypt	Sakha	EGYPT	1	1: 1984
~~France~~	~~Marseille / Marignane~~	~~FRANCE~~	~~1~~	~~1: 1000~~
France	Nice / Côte d' Azur	FRANCE	2	1: 1000
France	Perpignan	FRANCE	3	1: 1000
France	Ajaccio / Campo del	FRANCE	4	1: 1000
France	Tours	FRANCE	5	1: 1000
France	Le Puy en Velay	FRANCE	6	1: 1000
France	Biarritz	FRANCE	7	1: 1000
France	Les Escaldes	FRANCE	8	1: 1000
France	Brest / Guipavas	FRANCE	9	1: 1000
France	Trappes	FRANCE	10	1: 1000
France	Paris / Le Bourget	FRANCE	11	1: 1000
France	Nancy / Essey	FRANCE	12	1: 1000
France	Strasbourg / Entzhe	FRANCE	13	1: 1000
France	Nantes	FRANCE	14	1: 1000
France	Bourges	FRANCE	15	1: 1000
France	Dijon	FRANCE	16	1: 1000
France	Limoges / Bellegard	FRANCE	17	1: 1000
France	Lyon / Bron	FRANCE	18	1: 1000
France	Bordeaux / Mérignac	FRANCE	19	1: 1000
~~France~~	~~Toulouse / Blagnac~~	~~FRANCE~~	~~20~~	~~1: 1000~~
France	Nîmes / Courbessac	FRANCE	21	1: 1000
France	La Minière (Versailles)	FRANCE	22	2: 1983-1984
France	Grignon (Versailles)	FRANCE	23	2: 1985,1987

France	Le Rheu (Rennes)	FRANCE	24	2: 1987-1988
France	Dijon	FRANCE	25	2: 1985-1986
France	La Revanche-Lectoure	FRANCE	26	4: 1986-1989
Germany	Schleswig	GERM	1	1: 1000
Germany	Hamburg / Fuhlsbutt	GERM	2	1: 1000
Germany	Emden-Nesserland	GERM	3	1: 1000
Germany	Hannover	GERM	4	1: 1000
Germany	Berlin / Tempelhof	GERM	5	1: 1000
Germany	Essen	GERM	6	1: 1000
Germany	Kassel	GERM	7	1: 1000
Germany	Geisenheim	GERM	8	1: 1000
Germany	Stuttgart / Cannstadt	GERM	9	1: 1000
Germany	Nürnberg	GERM	10	1: 1000
Germany	München / Riem	GERM	11	1: 1000
Germany	Zugspitze	GERM	12	1: 1000
Germany	Kahler Asten	GERM	13	1: 1000
Germany	Feldberg	GERM	14	1: 1000
Germany	Kiel	GERM	15	5: 1000, 1983-1986
Germany	Göttingen	GERM	16	2: 1985-1986
Germany	Höhenheim	GERM	17	2: 1985-1986
Greece	Thessaloniki / Mikr	GREECE	1	1: 1000
Greece	Kerkyra	GREECE	2	1: 1000
Greece	Athinai / Nat. Obs	GREECE	3	1: 1000
Greece	Athinai / Helleniko	GREECE	4	1: 1000
Greece	Kalamath	GREECE	5	1: 1000
Greece	Hiraklion / Crete	GREECE	6	1: 1000
Greece	Kawala	GREECE	7	1: 1000
Greece	Tricala	GREECE	8	1: 1000
Greece	Larisa	GREECE	9	1: 1000
India	Patancheru, ICRISAT	INDIA	1	10: 1975-1984
India	Bijapur, Karnataka	INDIA	2	10: 1971-1980
India	Coimbatore	INDIA	3	3: 1983-1985
India	Cuttack	INDIA	4	3: 1983-1985
India	Hyderabad	INDIA	5	2: 1983-1984
India	Kapurthala	INDIA	6	3: 1983-1985
India	Pattambi	INDIA	7	3: 1983-1985
Indonesia	Muara	INDON	1	2: 1983-1984
Indonesia	CRIFC/Sukamandi (West Java)	INDON	2	3: 1983-1985
Ireland	Belfast / Aldergrov	IRL	1	1: 1000
Ireland	Valentia Observat	IRL	2	1: 1000
Ireland	Cork (Airport)	IRL	3	1: 1000
Ireland	Shannon (Airport)	IRL	4	1: 1000
Ireland	Dublin (Airport)	IRL	5	1: 1000

Ireland	Belmullet	IRL	6	1: 1000
Ireland	Malin Head	IRL	7	1: 1000
Ireland	Carlow Dublin	IRL	8	2: 1985-1986
Israel	Gilat (Migda)	ISR	1	23: 1962-1984
Israel	Bet Dagan	ISR	2	1: 1988
Italy	Milano / Linate	ITALY	1	1: 1000
Italy	Verona / Villafranc	ITALY	2	1: 1000
Italy	Venezia / Tessera	ITALY	3	1: 1000
Italy	Trieste	ITALY	4	1: 1000
Italy	Pisa / S. giusto	ITALY	5	1: 1000
Italy	Pescara	ITALY	6	1: 1000
Italy	Roma / Fiumicino	ITALY	7	1: 1000
Italy	Napoli / Capodichin	ITALY	8	1: 1000
Italy	Brindisi	ITALY	9	1: 1000
Italy	Messina	ITALY	10	1: 1000
Italy	Trapani / Birgi	ITALY	11	1: 1000
Italy	Catania / Fontanaro	ITALY	12	1: 1000
Italy	Alghero	ITALY	13	1: 1000
Italy	Cagliari / Elmas	ITALY	14	1: 1000
Italy	Udine Rivolto	ITALY	15	1: 1000
Italy	Ancona	ITALY	16	1: 1000
Italy	L'Aquila	ITALY	17	1: 1000
Italy	Foggia	ITALY	18	1: 1000
Italy	Potenza	ITALY	19	1: 1000
Italy	Policoro	ITALY	20	3: 1986-1988
Kenya	Ahero	KENYA	1	1: 1986
Luxembourg	Clerveaux	LUX	1	1: 1000
Luxembourg	Luxembourg / Findel	LUX	2	1: 1000
Mali	Station du Sahel (Niono)	MALI	1	4: 1976-1979
Mali	Mopti-Sevare	MALI	2	32: 1000,1959-1989 [1]
Mali	Bankass	MALI	3	30: 1959-1988 [1]
Mali	Koro	MALI	4	30: 1959-1988 [1]
Mali	Djenne	MALI	5	30: 1959-1988 [1]
Mali	Douentza	MALI	6	30: 1959-1988 [1]
Mali	Hombori	MALI	7	30: 1959-1988 [1]
Mali	Niafunke	MALI	8	30: 1959-1988 [1]
Mali	Sarafere	MALI	9	5: 1965,1973,1986-1988 [1]
Mali	Tonka	MALI	10	2: 1961-1962[1]
Mali	Samanko (Bancoumana)	MALI	11	4: 1983-1986

[1] Rainfall only

Mali	Samanko (Makandiana)	MALI	12	4: 1983-1986
Nepal	Parwanipur	NEPAL	1	2: 1983-1984
Netherlands	Wageningen (Haarweg)	NL	1	38: 1954-1991,1000
Netherlands	Swifterbant	NL	2	16: 1974-1989
Netherlands	De Kooy (Den Helder)	NL	3	10: 1976-1985
Netherlands	De Bilt	NL	4	1: 1000
Netherlands	Zuid Limburg (Beek airport)	NL	5	2: 1987-1988
Netherlands	Eelde Airport	NL	6	1: 1987
Netherlands	Lelystad (Exp. farm)	NL	7	4: 1987-1990
Netherlands	Valthermond	NL	8	1: 1988
Netherlands	Randwijk	NL	9	2: 1980,1988
Netherlands	Vredepeel	NL	10	1: 1986,1987
Netherlands	Vlissingen	NL	11	3: 1986-1988
Netherlands	De Bilt	NL	12	30: 1959-1988
New Zealand	Lincoln	NZ	1	4:1984-1987
Nigeria	Samaru	NIGIA	1	5: 1980-1984
Pakistan	Dokri	PAKIS	1	1: 1986
Peru	San Camilo	PERU	1	2:1981-1982
Philippines	IRRI wet station site	PHIL	1	11: 1979-1989
Philippines	IRRI dry station site	PHIL	2	11: 1979-1989
Philippines	Masapang	PHIL	3	1: 1984
Philippines	Los Baños (UPLB)	PHIL	4	26: 1959-1984
Portugal	Santa Maria / Acores	PORT	1	1: 1000
Portugal	Lisboa / Portela	PORT	2	1: 1000
Portugal	Porto / Pedras Ruba	PORT	3	1: 1000
Portugal	Faro	PORT	4	1: 1000
Portugal	Beja	PORT	5	1: 1000
Portugal	Penhas Douradas	PORT	6	1: 1000
Portugal	Braganca	PORT	7	1: 1000
Senegal	Nioro du Rip	SENEG	1	1: 1988
South Korea	Milyang	SKREA	1	2: 1983-1984
South Korea	Suweon	SKREA	2	2: 1983-1984
Spain	La Coruña	SPAIN	1	1: 1000
Spain	Valladolid	SPAIN	2	1: 1000
Spain	Madrid / Retiro	SPAIN	3	1: 1000
Spain	Mahon Menorca / SA	SPAIN	4	1: 1000
Spain	Badajoz	SPAIN	5	1: 1000
Spain	Barcelona	SPAIN	6	1: 1000
Spain	Palma De Mallorca	SPAIN	7	1: 1000
~~Spain~~	~~Ponta Delgada / AC~~	~~SPAIN~~	~~8~~	~~1: 1000~~
Spain	Santander	SPAIN	9	1: 1000
Spain	Leon	SPAIN	10	1: 1000
Spain	Soria	SPAIN	11	1: 1000

Spain	Zaragoza	SPAIN	12	1: 1000
Spain	Ciudad Real	SPAIN	13	1: 1000
Spain	Murcia	SPAIN	14	1: 1000
Spain	Cordoba	SPAIN	15	1: 1000
Spain	Granada	SPAIN	16	1: 1000
Spain	Malaga	SPAIN	17	1: 1000
Spain	Madrid Ciudad Universitaria	SPAIN	18	2: 1987-1988
Spain	San Esteban De Gomaz	SPAIN	19	1: 1988
Sri Lanka	Paranthan	SRILA	1	3: 1983-1985
Switzerland	Santis	SWIT	1	1: 1000
Syria	Breda	SYRIA	1	8: 1980-1986, 1988
Syria	Ghreriffe	SYRIA	2	2: 1985-1986
Syria	Jindiress	SYRIA	3	7: 1980-1986
Syria	Kafr Antoon	SYRIA	4	4: 1980-1983
Syria	Khanasser	SYRIA	5	7: 1980-1986
Syria	Tel Hadya	SYRIA	6	11: 1978-1986, 1989-1990
Syria	Homs	SYRIA	7	3: 1988-1990
Syria	Izra'a	SYRIA	8	3: 1988-1990
Taiwan	Pingtun	TAIW	1	3: 1983-1985
Thailand	Sanpatong	THAIL	1	4: 1983--1986
United Kingdom	Dalwhinnie	UK	1	1: 1000
United Kingdom	Lerwick	UK	2	1: 1000
United Kingdom	Stornoway	UK	3	1: 1000
United Kingdom	Aberdeen / Dyce	UK	4	1: 1000
United Kingdom	Tiree	UK	5	1: 1000
United Kingdom	Edurgh / Royal	UK	6	1: 1000
United Kingdom	Eskdalemuir	UK	7	1: 1000
United Kingdom	Valley	UK	8	1: 1000
United Kingdom	Manchester Airport	UK	9	1: 1000
United Kingdom	Waddington	UK	10	1: 1000
United Kingdom	Birmingham Airport	UK	11	1: 1000
United Kingdom	Glamorgan / Rhouse	UK	12	1: 1000
United Kingdom	London / Gatwick	UK	13	1: 1000
United Kingdom	Plymouth / Mount	UK	14	1: 1000
United Kingdom	Durnemouth / Hurn	UK	15	1: 1000
United Kingdom	Cambridge	UK	16	2: 1985-1986
United Kingdom	Cambridge	UK	16	4: 1985-1988
United Kingdom	Sutton Bonington	UK	17	6: 1980-1985
United Kingdom	Mlynefield	UK	18	3: 1985-1987
United Kingdom	Invergowree	UK	19	4: 1984-1987
United States	Hancock, Wisconsin	USA	1	3: 1985-1987
United States	Bakersfield, California	USA	2	1: 1988

United States	Davis, California	USA	3	1: 1000
United States	Ithaca, New York	USA	4	1: 1987
United States	Tulelake, California	USA	5	1: 1988

Appendix E: OUTREC, program to recover lost output data

```
      PROGRAM OUTREC
      INTEGER IOPT, ILU1, ILU2
      LOGICAL TERMNL
      COMMON /OUTCUT/ TERMNL, ILU1, ILU2
      DATA IOPT /4/

      ILU2 = 20
      ILU1 = 21

      WRITE (*,'(A,/)') ' Output options:'
      WRITE (*,'(A)')   ' 4 = normal table'
      WRITE (*,'(A)')   ' 5 = tab-delimited table'
      WRITE (*,'(A,/)') ' 6 = two column format table'

      CALL ENTDIN ('Which output option', IOPT, IOPT)
      CALL FOPEN (20, 'OUTREC.OUT', 'NEW', 'UNK')
      CALL OUTDAT (IOPT, 0, 'Recovered file !!!!', 0.)

      WRITE (*,'(A)')
     &   ' Output successfully recovered in file OUTREC.OUT !'

      STOP
      END
```