SIMUBASE
Using Data Base Capabilities to Implement Interface Free Modules for Building a Simulation Management Environment

By

Abdenacer Moussaoui

A RESEARCH PAPER
submitted to
Oregon State University

in partial fullfillment of
the requirements for the
degree of

Master of Science

1989

ACKNOWLEDGEMENTS

I wish to give thanks to Dr. Ted G. Lewis for his guidance during the writing of this research.

I also wish to thank Toshi Minoura and Tim Budd for their interest.

Special regards to my brother Abdennour who helped me in numerous occasions, and to my brother Mohamed for his support.

To all the others which I have not included here I would like to say: THANK YOU!

I dedicate this thesis to my mother and father.

Abdennour Boulouis
5/16/89

Table of Contents

1 Abstract

AN ABSTRACT OF THE RESEARCH PAPER OF

Abdenacer Moussaoui for the degree of Master of Science in Computer Science presented in April 1989

Title: SIMUBASE, Using Data Base Capabilities to Implement Interface Free Modules for Building, a Simulation Management Environment

Abstract approved:

Ted G. Lewis

This is an attempt to deal with interface problems of interacting modules in a large simulation system.

In simulation, scientists require routine substitution of modules for comparison of alternative models. These changes have propagating side affects and from a programmer's point of view are essentially system redesigns.

This paper presents comparisons of several approaches attempting to resolve the above problem. Proposed here is an accessible design. The features of a Data Base Management System (DBMS) are used to implement a prototype (SIMUBASE) showing the feasibility of this approach. SIMUBASE is a simulation environment for managing simulation modules.

Index terms

- Modular Programming, Plugable Module, Reusable, Data-Item, Automatic Making, Standard Module Description, Version Selection Procedure, Module Master Name, Module Version Name, Simulation Scripts.

2 Introduction

Simulation generally deals with experimentation, which often is done through repetitive trials of different variations of a model. The process is to test various approaches to obtain a simulation model that is close to the real world. The experimenter's steps usually can be summarized as follows:

- 1- Create Simulation Program Code
- 2- Run simulation
- 3- Save Output Data
- 4- Analyze Data, Comparison of the outputs of two or more models
- 5- Modify Program or Approach
- 6- Repeat again at step 2

When programming these various approaches of a simulation it becomes a tedious process of generating multiple programs. These programs are generally the same, except for parts of the programs which accommodate the variations in the approaches. It is very often the case that a program code is repetitively modified and re-compiled to meet the needs of the experimenter(s). Many of these variations have unpredictable or predictably unpleasant side-effects [Beladi-1979]

My goal in this research is to create an environment that is most suitable for creating computer simulation models and to proficiently experiment with variations of these models without having to worry about re-coding or re-compiling, and to keep the experimentation organized and well documented.

3 Background/History

Unfortunately, part of what initiated this research is a pool of ill-defined modules written mostly in the FORTRAN and BASIC programming languages dealing with agricultural simulation models. (See side bar)

These modules have been developed over the years by various researchers with basic knowledge about programming, and very limited concepts about good programming techniques. It is often the case that these modules are nothing but an amalgamated program code with unclear parts and very limited **reusability**. In computer science terms, we call such programming practices un-structured, non-modular and non-reusable programming methodologies.

Because of the availability of this huge library of modules (despite their current state) the researchers question whether the following goals can be achieved:

- Can a researcher with limited programming knowledge, be able to pick a number of modules from this pool, and miraculously combine and run them, to experiment with the specific case he is working on?
- Can the researcher substitute any given version of a module with a different one in order to compare different test runs?

Even if this pool of modules were to be written in a single unified programming language, the goals above are far from being accomplished effectively even with todays state-of-the-art programming techniques as is shown in further sections of this paper.

About PLANTEMP Simulations...

This research started around PLANTEMP. The PLANTEMP simulation dealt with following the development of selected crops, approximation of rain usage, effects of varying numerous parameters such as precipitation, row spacing, solar radiation, etc...

The PLANTEMP project was intended to be a quick transfer of a simple BASIC program to modular form. The project led to a much greater appreciation of the problems of software design and more importantly a realization of the magnitude of change imposed on a program by the change of one or more of the models upon which a program was based.

Changes that scientists require as routine substitution for comparison of alternative models are essentially system redesigns from a programmer's point of view. [Rickman-1987]

4 Objective

It would be advantageous to design a large library capable of holding well defined modules that are easily manageable such that a non-programmer can efficiently use it without having to code, compile, or do other tedious chores that even professional programmers have to struggle with.

The reader should keep in mind that a suitable simulation development system will accommodate two kinds of users. On one hand there will be users who develop, debug, and test simulation modules. On the other hand the system is specifically aimed at the non-programmer scientist who wishes to use the system purely for experimentation. Such users will be using subsets of modules developed by other more experienced users and/or system managers.

In attempting to narrow down the technical challenges from what has been expressed above by the researchers we re-phrase the goals as:

- Provide the ability to easily plug in and plug out different implementations of a simulation sub-part; being able to work with multiple versions of module(s).
- Eliminate or minimize the impact of routine changes on the whole system.
- Combine a set of modules to create simulation systems. Form variations by altering the set and its configuration much like a child does with LEGO[1] pieces.
- The development environment system in mind should be as friendly as possible for novices and experts alike.
- Use a simple yet powerful programming language which can even be used by a novice computer user.
- No need to re-compile simulation models.
- More complex simulation models can be created by combining simpler existing models.
- Simulation model development should be rapid and straight forward.
- This type of environment will encourage researchers to experiment and share their creativity with others due to the elimination of rigid specifications (language, data-storage, modules communication, etc...).

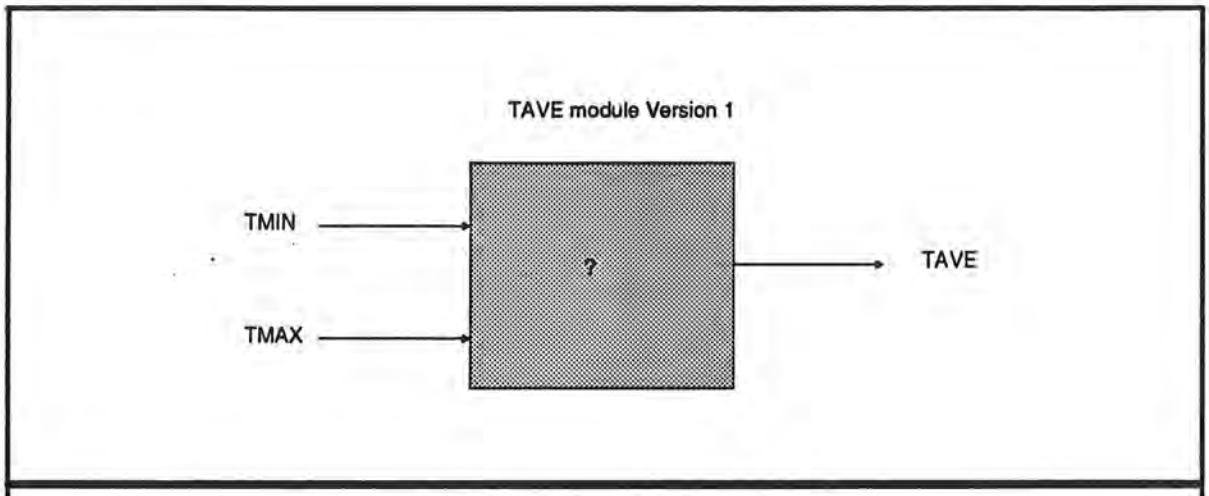
(1) LEGO pieces have fixed standard interfaces.

4-1 Illustration

In showing how the concept of **plugable** modules is useful, let's take the example of a module that computes the daily average temperature. The output of this module is the average temperature which is to drive a simulation model, such as a plant growth.

Let's imagine a scientist wanting to experiment with this simulation model. The experimenter would like to test two different simulation runs. Each run uses a different variation/version of the daily average temperature module which I refer to in this paper as the TAVE module.

In simulation run one, the implementation of average is defined as follows: the module receives the daily minimum and maximum temperatures (referred to as TMIN and TMAX respectively) and computes the daily average temperature. The interface to

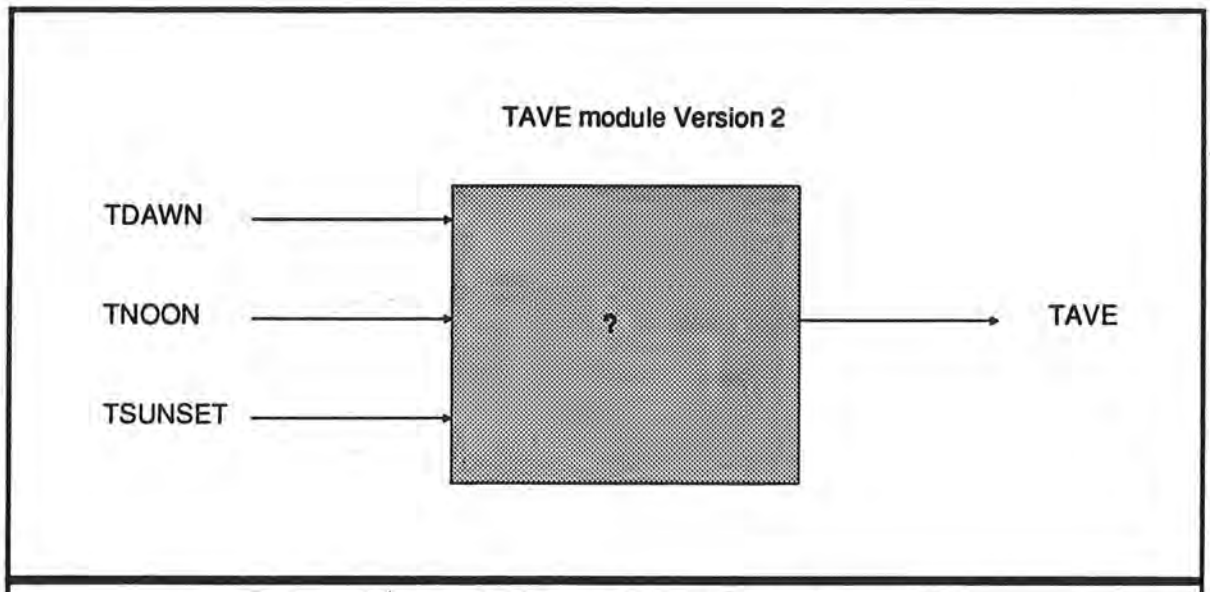


Interface to TAVE module version 1 --- Figure 1

such a module may be represented as in figure 1, or in Pascal as the function

```
FUNCTION  TAVE(
                    TMIN,
                    TMAX : REAL ) : REAL ;
BEGIN
        {implementation code}
END ;
```


In the second simulation run the experimenter would like to substitute a different version of the TAVE module. The implementation of this second version of TAVE receives



Interface to TAVE module version 2 --- Figure 2

three collected daily temperatures to compute an equivalent result, but using the interface depicted in figure 2, or in Pascal as

```
FUNCTION  TAVE(
                    TDAWN,
                    TNOON,
                    TSUNSET : REAL ) : REAL ;
BEGIN
        {implementation code}
END ;
```

The change of arguments to this module alters its interface. A change in the interface of a certain module, implies the modification of any part of the system that makes reference to this process. Imagine a large program where this function may be called repeatedly from numerous places. This would imply modifying every single call to TAVE when we substitute one version for another that has a different interface. Most of the time changing the calls to TAVE is not enough, because the code manager needs to undertake other tasks such as adding new declarations as well as the removal of old

ones. This is indeed a drawback that even the modular[2] programming concept and practice fails to resolve.

A tedious and error prone process of changing code can be easily imagined from the simple Pascal example shown comparatively below. In this instance, the modeler sub-

Version 1 of the Pascal simulation code of DEMO1	Version 2 of the Pascal simulation code of DEMO1

PROGRAM **DEMO1-1**(INPUT, OUTPUT) ;
CONST
LAST_DAY = 20 ;
 	
TYPE
ARRAY_T = ARRAY[1..LAST_DAY] OF REAL ;
 	
VAR
NEW_STOCK,
INTEREST,
STOCK : ARRAY_T ;
DAY : INTEGER ;
 	
FUNCTION INTEREST1(
DAY : INTEGER) : REAL ;
BEGIN
INTEREST1 := SQRT(DAY) ;
END ;
 	
BEGIN
STOCK[1] := 1000 ;
 	
FOR DAY := 1 TO LAST_DAY-1 DO
BEGIN
INTEREST[DAY] := INTEREST1(DAY) ;
NEW_STOCK[DAY] := STOCK[DAY] + INTEREST[DAY] ;
STOCK[DAY+1] := NEW_STOCK[DAY] ;
END ;
 	
WRITELN('DAY':10, 'STOCK':10, 'INTEREST':10, 'NEW_STOCK':10) ;
FOR DAY := 1 TO LAST_DAY-1 DO
BEGIN
WRITELN(DAY:10, STOCK[DAY]:10:2, INTEREST[DAY]:10:2, NEW_STOCK[DAY]:10:2) ;
END ;
END .
	PROGRAM **DEMO1-2**(INPUT, OUTPUT) ;
	CONST
	LAST_DAY = 20 ;
	
	TYPE
	ARRAY_T = ARRAY[1..LAST_DAY] OF REAL ;
	
	VAR
	NEW_STOCK,
	INTEREST,
	STOCK : ARRAY_T ;
	DAY : INTEGER ;
	
	FUNCTION INTEREST2(
	DAY : INTEGER ;
	STOCK : REAL) : REAL ;
	BEGIN
	INTEREST2 := SIN(DAY) * SQRT(STOCK) ;
	END ;
	
	BEGIN
	STOCK[1] := 1000 ;
	
	FOR DAY := 1 TO LAST_DAY-1 DO
	BEGIN
	INTEREST[DAY] := INTEREST2(DAY, STOCK[DAY]) ;
	NEW_STOCK[DAY] := STOCK[DAY] + INTEREST[DAY] ;
	STOCK[DAY+1] := NEW_STOCK[DAY] ;
	END ;
	
	WRITELN('DAY':10, 'STOCK':10, 'INTEREST':10, 'NEW_STOCK':10) ;
	FOR DAY := 1 TO LAST_DAY-1 DO
	BEGIN
	WRITELN(DAY:10, STOCK[DAY]:10:2, INTEREST[DAY]:10:2, NEW_STOCK[DAY]:10:2) ;
	END ;
	END .

(2) A programming design strategy in which one of the goals is to minimize global impact on a system due to a small change in a part of the system.

stitutes one interest function with another to create a new simulation run. The highlighted code shows what is being modified from one run to another. Not only will the modeler have to write two extensive versions of very similar programs, but considerable effort in coding will be required if the modeler desires to save data to a file (both in declaring the needed data types and variables for storage and coding the I/O logic).

"A remarkable amount of effort is spent hand coding the logic for moving structured data between memory and I/O devices. Many applications involve parsing input data, processing it, and then reconstituting the data on the output stream. This I/O code is often more time consuming to develop and test than the processing logic, and it must be continually adjusted as data structures evolve during development." [Cox-1984]

In an average simulation composed of several thousand lines of code, the magnitude of change is enormous. The existence of different interfaces is not caused by erroneous design nor insufficient definition of requirements, but is intended as a feature of the system. I am not concerned with minimizing the effects of introducing a substitute module due to a normal software evolution, as discussed in most of the literature, [Beladi-1981], [Lehman-1978], [Cox-1984], [Korson&Vaishnavi-1986], [Pressman-1982]. More precisely, the intention is to provide the ability to create **deliberately** new configurations by selecting some desired sets of modules. Change in simulation systems is inevitable, so we do not wish to limit it or bound it, but manage it efficiently.

"The unending sequence of modification that software systems undergo is thus not necessarily or primarily due to the short sightlessness or lack of planning. It is intrinsic to the very being of the system. Therefore, the need for change must be taken into account at all stages of specification, design and development. Changeability is, in the long run, as crucial a factor in the total cost-effectiveness of a program as are attributes such as its resource usage, speed or freedom from faults." [Lehman-1978].

Summary

The function/module TAVE can be easily conceived as a low level building block in a large system. If this system is to allow substitution of different versions of this process for testing and evaluation it means it must resolve connectivity and complexity issues in the system.

The mechanism of intermodule communication -global data and parameter passing- have far-reaching effects on software module configuration and hence system integration, [Pizzarello-1984].

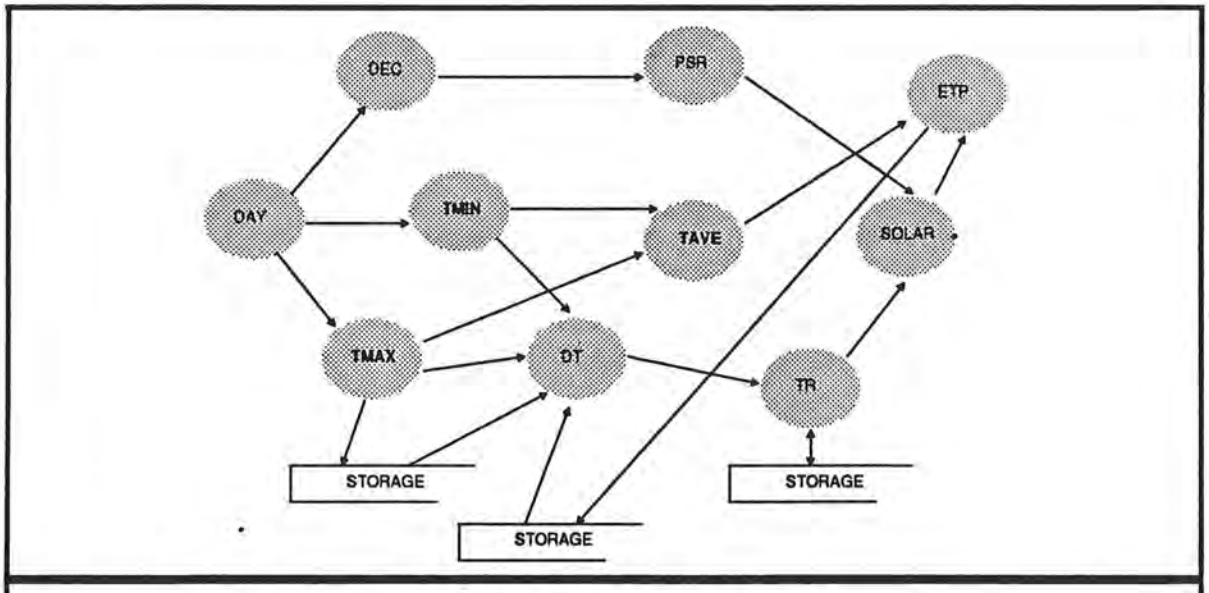
My primary focus is attempting to provide a mechanism for **changeability** regarding specifically interface specification. If this is indeed possible, **plugable modules** will be achieved.

5 Approaches

In this section I will present some different programming techniques and strategies for a modular design. Discussed are the trade-offs vis-a-vis the impact caused by the change in the interface of the two implementations of the TAVE module described in the previous section.

5-1 Approach 1

In programming languages like PASCAL, C, or FORTRAN, data and control is passed from one module to another through interface parameters. Modules are structured and no global data is used except through parameter lists. See figure 3



DFD for Approach 1 --- Figure 3

Notice in this strategy the high level of module intercommunication. Hence the problem with this approach is that whenever a change is made to the system a chain reaction takes place which triggers the need for other parts of the system to be changed as well. This rippling effect is caused by the variables passing between modules and the strong typing of the structured programming languages.

The maintainable-object method (M-Object) [Lewis-1986] attempts to limit the propagation of changes by restricting the interfaces among modules. But a simple change in interface can cause widespread changes in other modules. For example, suppose an additional parameter is added to a method that is called from a thousand places in an application program. Not only must we change the interface definition, but we must locate and modify all references to this interface. The location of such places may not be readily accessible/available nor do we want to constantly modify such references. Writing well structured code does not necessarily guarantee modular code nor does modular code imply reusability of code.

Consequences

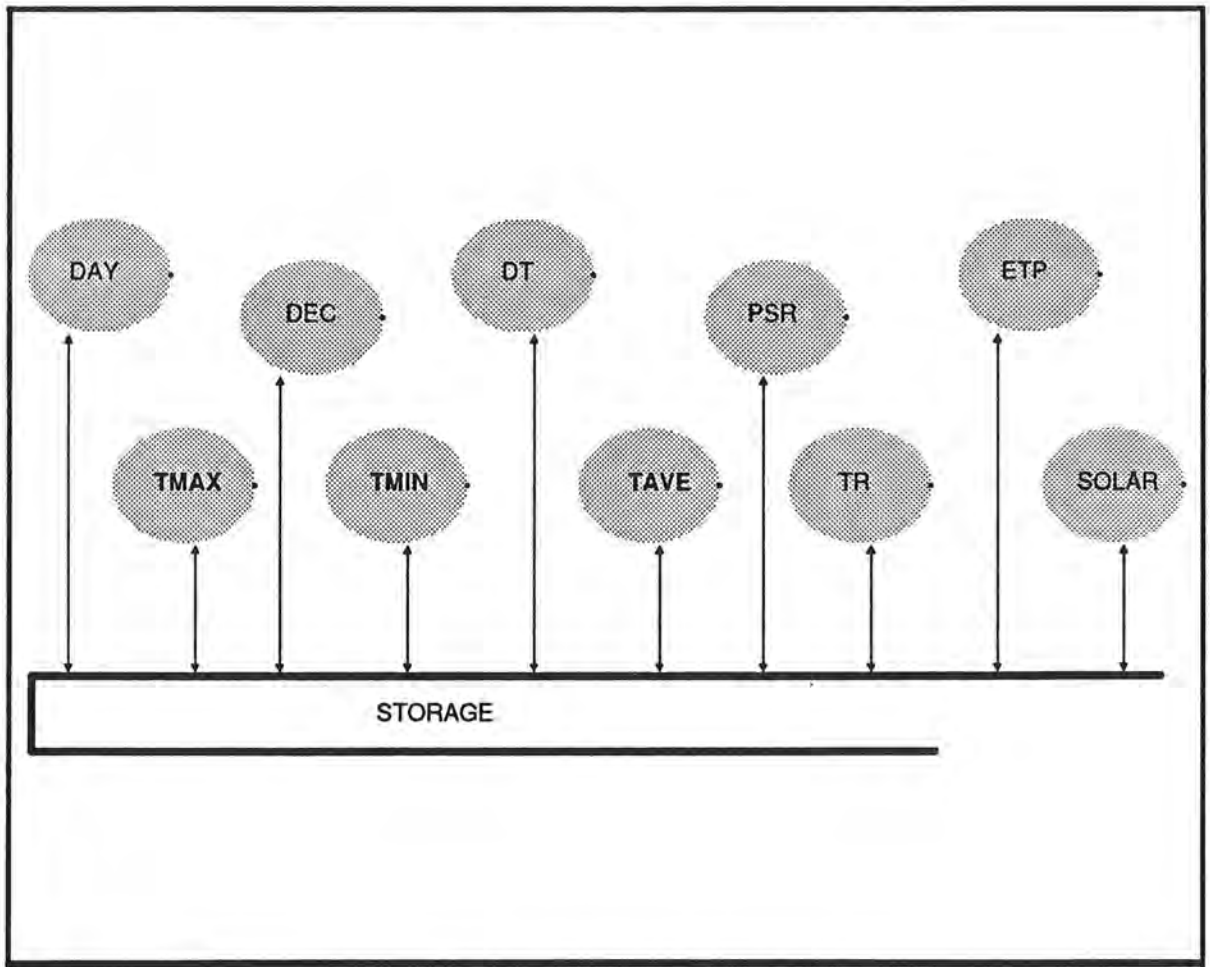
+ Well defined interfaces
- All other modules referencing the modified module will have to be modified.
- Re-compilation of all or part of the system due to data structures and interfaces changes.
- A high level of coupling[3].

(3) Coupling is a measure of interconnection among modules in a software structure. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and which data pass across the interface.[Pressman-1982]

5-2 Approach 2

The second Approach is similar to the first approach, but with the following restriction: All module interfaces are parameterless. Thus, module communication can be achieved through files, and more precisely, structured-files[4]. See figure 4

In this approach modules access files to acquire input. After they process the data, output is generated by modifying the accessed files or creating new ones.



DFD for Approach 2 --- Figure 4

(4) Structured files offer a great deal more accessibility and performance while ASCII files don't because of the usual sequential access imposed on their structure.

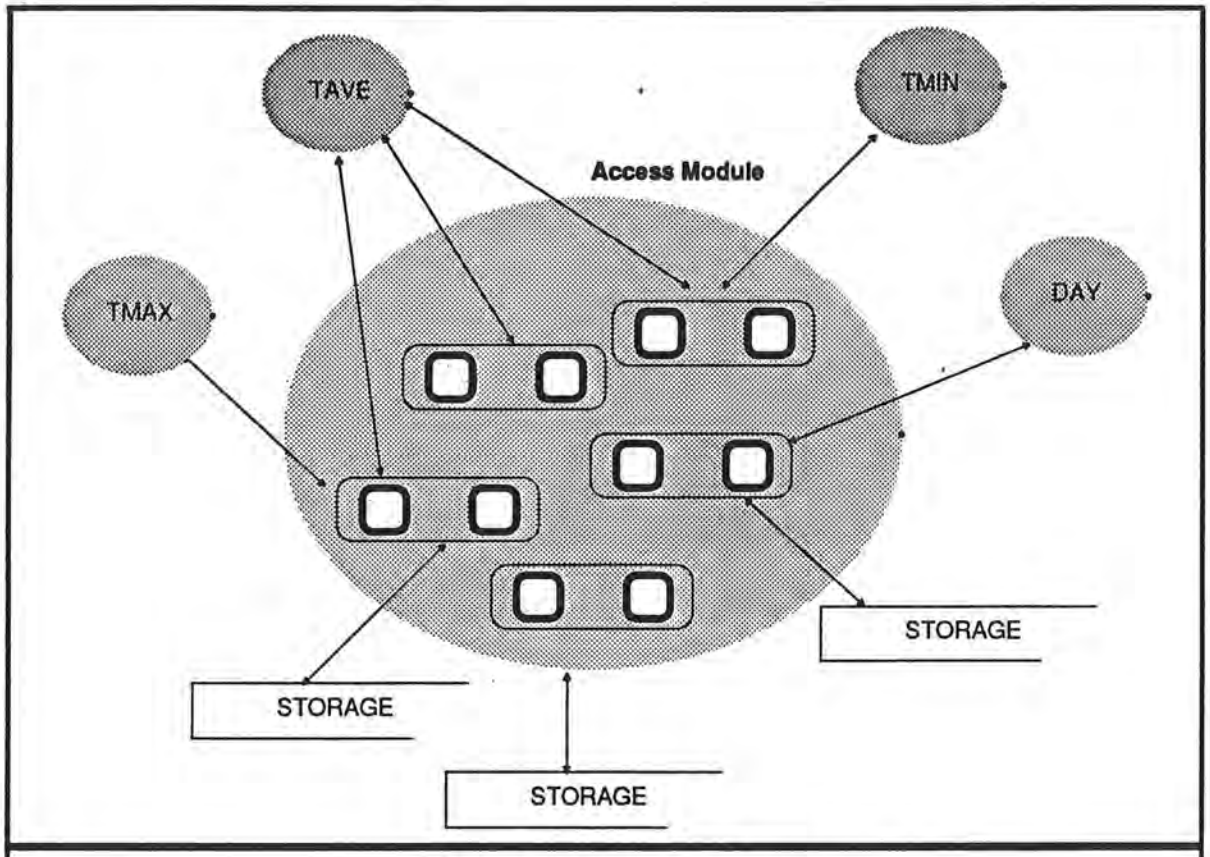
Consequences

+ Minimize the ripple-effect, because references to modules do not have to be modified. Since the module interfaces are parameterless there is no need to modify the declarations and references to these modules.

- All modules have to know at all times of any alteration to structures
- Re-compile all or part of the system due to the change in the data structures and file formats.
- Slow execution of the model due to file manipulations.

5-3 Approach 3

The third approach is to use an access module to handle all access to data that is public (global). Develop a library of routines (functions/procedures) in the access module to provide access and manipulation of each data held by the access module. For each global data, routines in the access module are used by other modules to either retrieve, store, or update data. See figure 5



DFD for Approach 3 --- Figure 5

Each module introducing new (public) data to the system must provide the access routines which are added to the access module, so that other modules can manipulate the data.

"If several programs share the same data, then it is even more important that data be associated with its own access code, otherwise each program using these data must contain the access code individually. Thus, if the implementation of the data structure changes, this must be reflected in each program - a certainly error prone process." [Beladi-1985]

In summary the simulation system is composed of:

- A structured shared data store consisting of one or more data structures.
- A large access module composed of access routines to the shared data. These routines are the principal accesses to interface with a given module.
- A set of modules that contain the code of the access routines.

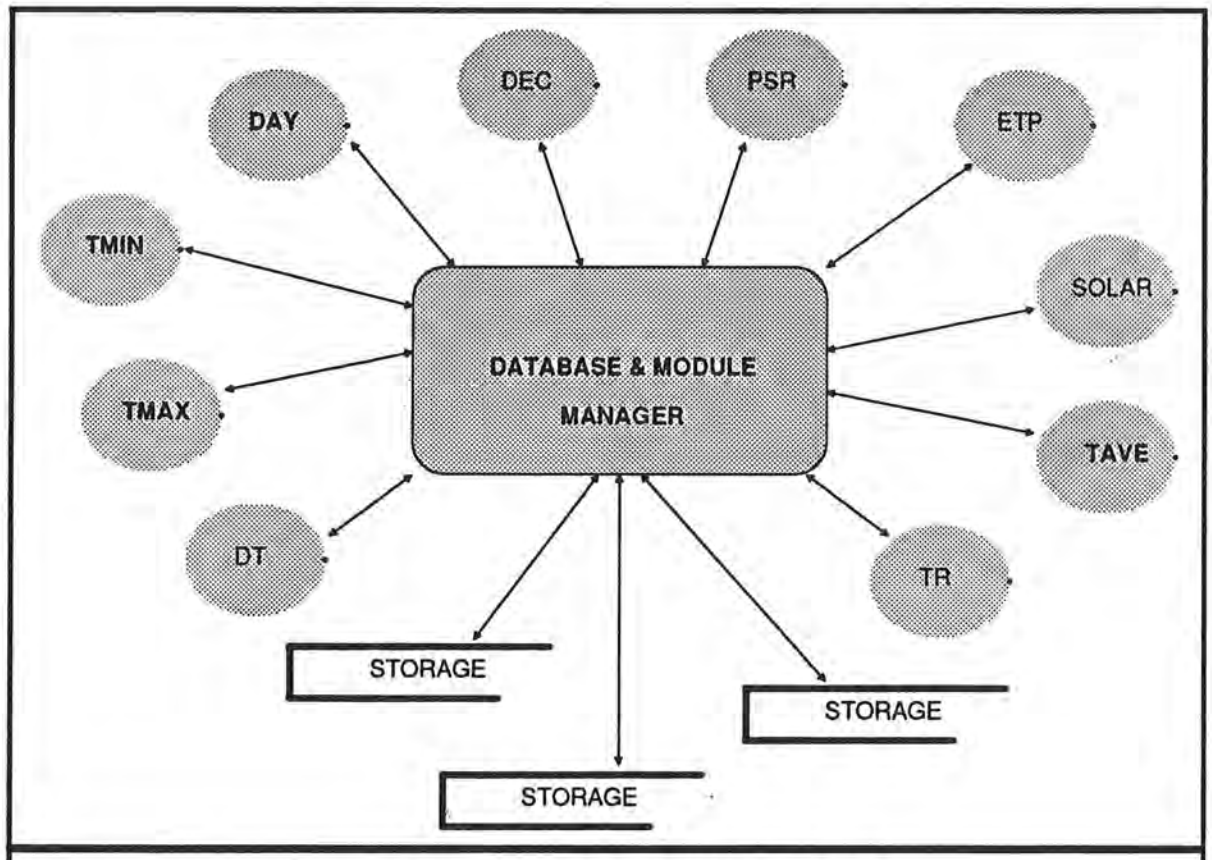
Consequences

- The access module needs to be re-compiled almost every time a new module is added or replaces a previous one.
- Access procedures should use a consistent naming convention which tends to lead to errors. This affects *all exported* items (access routines, shared data types, etc...)
- When a module is removed the corresponding data structures and access routines now unused, need to be removed.
- Although reduced to a great extent there still exist a high level of intercommunication (i.e. module coupling). Notice for example, the TAVE module is still explicitly calling the TMIN and TMAX modules.

+ The rippling effect of modification propagation is minimized because knowledge is now restricted to access routines.
+ In this approach the module serves to localize the access routines so modification and re-compilation is reduced.

6 Prospective Approach

Approach 3 suggests centralizing the global data and then providing access routines around the data. I opted to extend this idea by further abstracting the shared data. This can be achieved by storing the shared data in a common area and restricting the knowledge of the underlying structure and data management to a single manager. Modules create, retrieve and store their needed data to and from the data storage through an intelligent data manager. The data creation, passing, and update, happens through a simple data retrieval language. See figure 6 .



DFD for Prospective Approach --- Figure 6

We further add the restriction that a module never calls another one in the system (i.e. no module direct inter-communication). In other words modules do not call one another directly so that no interface dependency may result, in fact <u>an explicit module interface need not exist at all.</u>

"A good methodology should provide an ***implicit*** record of the internal connections in a program [module inter-connection] and avoid explicit interface rigid specification which causes a <u>high level of coupling</u>". [Lehman&Beladi-1985].

If a module needs data which is provided by another module, it needs only to request it from the data manager which in turn resolves which module is responsible for the creation and/or update of the requested data. The manager will call the needed module(s) including all dependent modules and then upon return passes the data to the module making the request. In case the data has been already computed by some previous activation, the data manager should detect this situation and possibly shortcut a chain of module activations simply by furnishing the stored data.

Consequences

+ Modules have only knowledge of how to reference global data. That is a module developer need only to be aware of the existing module logical names to reference them.
+ A process does not need not to know where and how the data it needs is stored.
+ Coding a process becomes fairly easy and reflects mostly the process algorithm. Almost all language intricacies such as parameter matching, and file handling are removed.
+ A scientist in practice would not have to modify someone else's implementation module in order to use it.
+ A scientist need only to run the process desired and all the needed sub-processes will be invoked on a need basis.

This last point is very significant; it relieves the user from keeping track of the dependency relationship between modules. Later discussion will show how the system can automatically derive dependency relations among modules without having them being explicitly specified.

7 Design and Specification

In this section I present how a suitable simulation management environment might be conceived to support the ideas described in the prospective approach.

Every shared data (i.e. global) in the system is treated as a data-item. A data-item is any item referred to by more than one module. Once a data-item is created any process can request it.

In our previous example the modules names TAVE, TMIN and TMAX also become data-items in this new strategy. Hence, since every module in the current scope of this paper will introduce one and only one data-item, the words data-item and module will be used inter-changeably in the following sections.

7-1 Standard Module Description

Since in this new strategy all structural knowledge of modules is restricted to the module manager we need a **Standard Module Description**. A module description is the means by which any given module gets defined in the system. Hence, modules can be introduced in the system by providing the system manager with a set of module attributes that describe most of the characteristics of any given module. Of course this set of attributes can be augmented when more information about a given module is needed. Therefore, every data-item in the system may have the following attributes attached to it as the standard module description:

- a **Module Master Name**
  is the name referred to throughout the system and which should be **unique**. In other words this is the module **logical** name.
- a **Module Version Name**
  is a module version-specific name. This can be thought of as the name of a specific version of the module responsible for the data-item.
- **description**
  Textual description of what the data item represents.
- **type specification**
  this should include things such as type, size, decimal places, units, etc...
- **implementation reference**
  This may include things such as: Model/Algorithm reference, Who is the author? How he can be reached? etc...

- **implementation description**
  This description can provide information on the method used in the implementation of the algorithm, source references, etc...
- **creation date**
  when was this data-item last introduced in this system.
- **modification date & time**
  when was this data-item last computed. This particular information is critical to the correct making/sequencing of the modules.
- **creator process code**
  is the actual source algorithm that updates the data-item. The modules use a simple yet powerful language to interface with a database manager in order to add, retrieve, update, or delete data.

Note

Coding a module description can be achieved through an entry form such as the one seen in figure 7.

Module Master Name	Module Version Name	Creation Date
TAVE	TAVE2	12/03/86

Description	Modified Date/Time
Daily Average Temperature	05/23/88 -- 2:30p

Implementation Description/Reference
Computes the daily average temperature, Campbell Model 1986

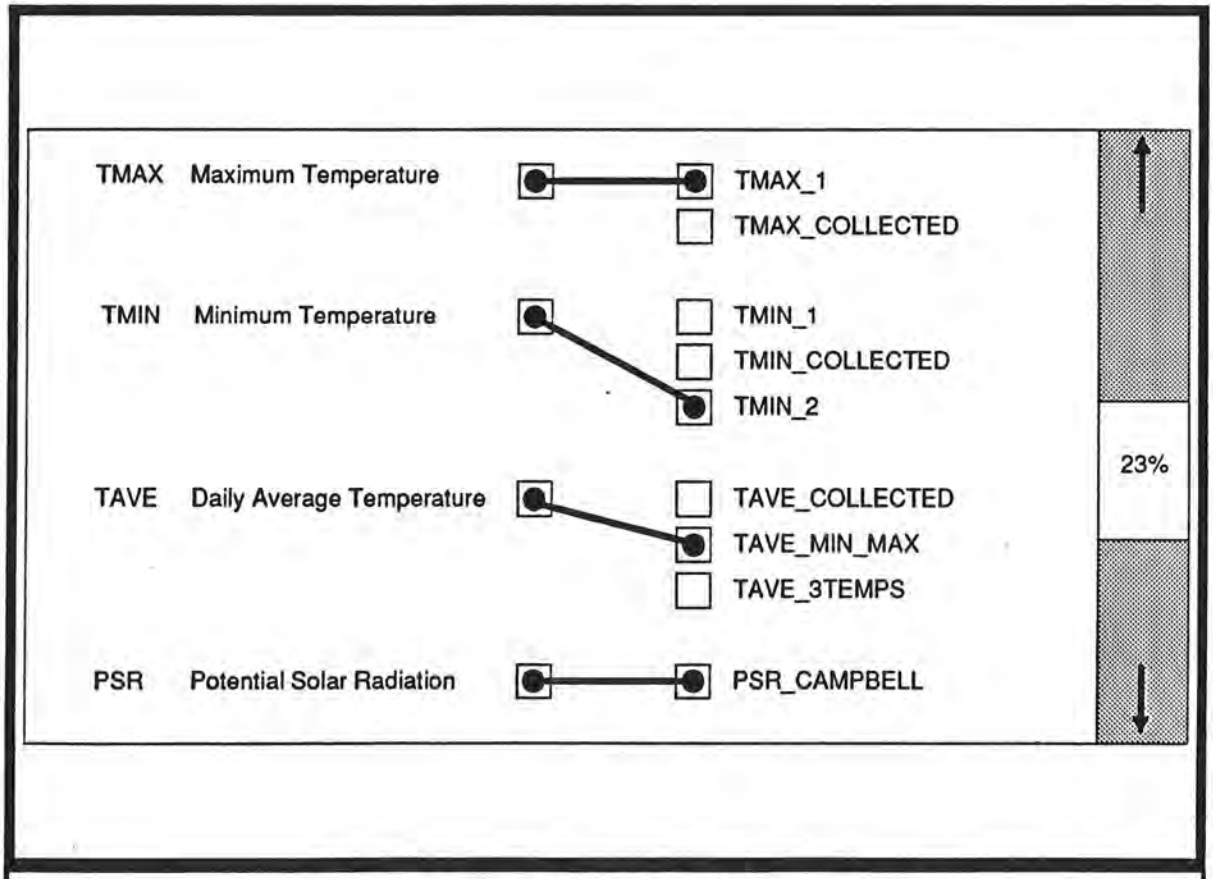
Specifications	
Type	Numeric
Field Width	10
Decimals	3
Units	Celcius

Sample Module Description Screen Form --- Figure 7

7-2 Version Selection Procedure

Since one of my objectives is to allow the existence of more than one version of any given module, I must specify for every data-item (i.e. module master name) which version is to be used. Hence the system includes a **Version Selection Procedure** in which every data-item is assigned an existing version of the module responsible for it. In a typical session the user is presented with a list of all data-items present in the library of modules and the versions available for each module master name. He selects by toggling flags to indicate which master name(s) is to be bound to which version (of course the system insures a unique binding). Any given configuration can be saved in **Version Configuration Files** which can then be re-loaded.

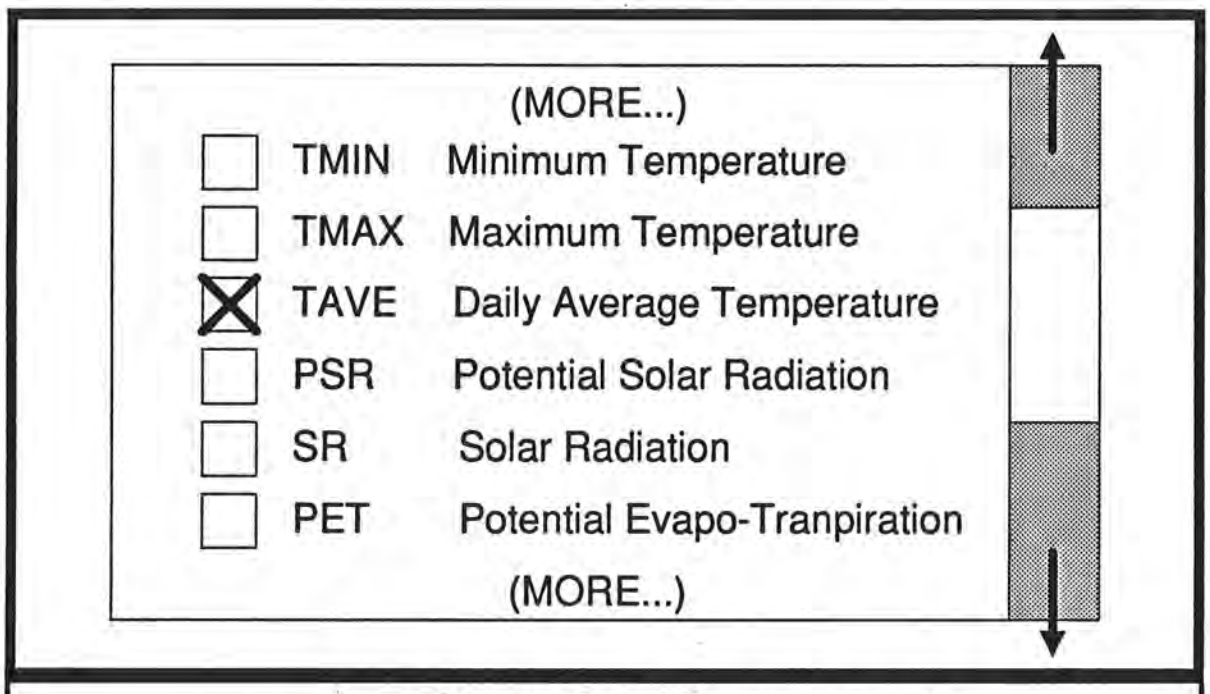
This procedure can be either textual or implemented as a graphical user interface such as the one seen in figure 8



Sample Version Selection Procedure --- Figure 8

7-3 Simulation Selection Procedure

So far the module description template provides ample information on any given module and using the version selection procedure, the system is now aware of which specific versions are to be used. However the reader may question how these modules tie together and what describes the running order and interaction, and indeed the system definitely needs some knowledge about which set of modules are to be run. This is why an activation mechanism is needed. Therefore the system also provides a **Simulation Selection Procedure** in which the user selects which modules he wishes to include in a given simulation, see figure 9. The user can generate different test runs by toggling different simulation configurations and saving each one of them in separate **Simulation Script Files** using this procedure. A script is the mechanism by which some desired modules are invoked in some order to activate a simulation run.



Sample Simulation Selection Procedure --- Figure 9

During the simulation selection procedure, the user is required to select only the top module or set of top modules that are disjoint. The system will figure out all of the dependent modules. This can be achieved by using the knowledge of the last update date and looking into the algorithm source code for identifiers that reference module master names. We will see how this derivation is achieved when we apply this design strategy to our example.

7-4 Reporting Procedure

Finally when a simulation configuration is activated the system should provide some different means of viewing and reporting the results of the simulation runs. This can be done in tabular form, graphical data plots, or visual animated graphical icons, [Stella-1987]. This reporting feature can be similar to the simulation selection procedure in that the user picks the module master names to report on. However since the user has already selected a simulation configuration, the system restricts the list of modules presented to the user, to the ones he has picked during the simulation selection procedure.

Besides the built-in reporting capabilities the system should allow the simulation data to be exported to other systems for more sophisticated analysis.

7-5 Summary

The user introduces new modules to the system by defining their module descriptions through a standard entry form. Using the version selection procedure the users selects which specific version of certain modules he wishes the system to specifically use. The user builds a simulation run using the simulation selection procedure to select a set of top modules. Finally the user requests reports on desired data in the simulation.

7-6 Applying this Design Strategy to Our Example

Now let us consider applying this targeted design to the experiment mentioned in the introduction.

As was mentioned earlier, the names TMIN, TMAX and TAVE also become data-items in this new approach and will have respective description forms defined in the system.

Let's assume that TMIN and TMAX data-items are already defined in the system. In order for our scientist to try the first simulation run of his experiment he would have to provide a process to retrieve the TMIN and TMAX data-items and compute a new data-item TAVE. Hence the first version of TAVE will need to have the following simplified form description associated with it in the system:

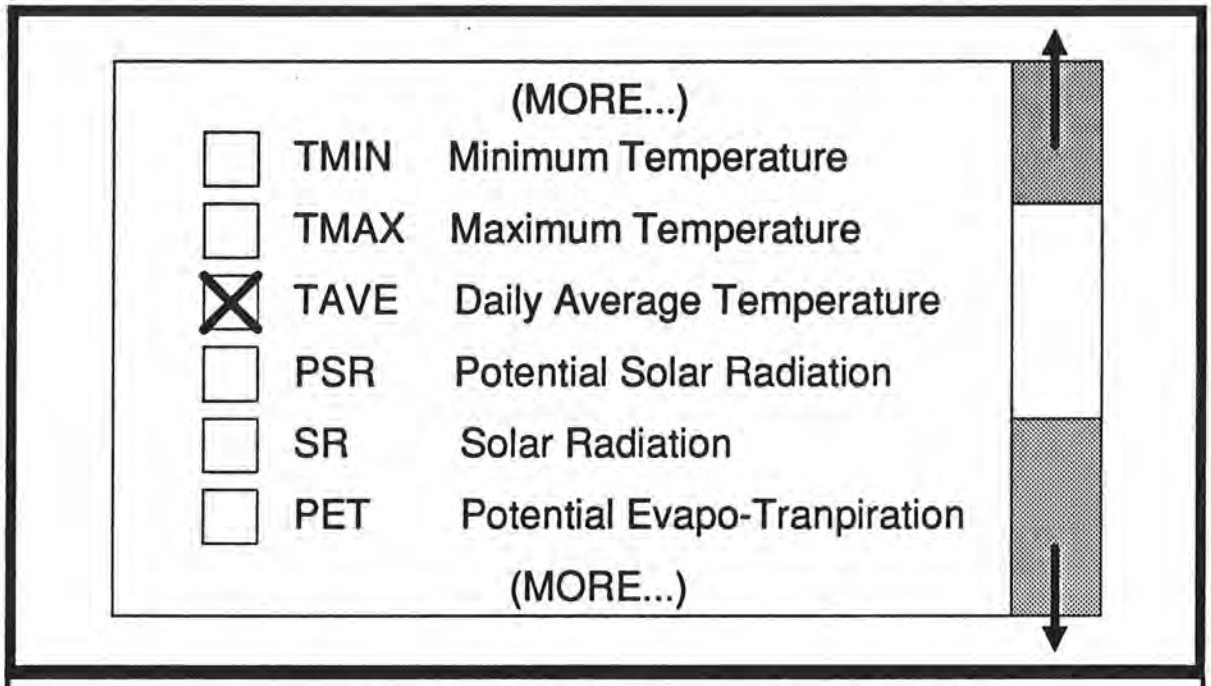
- module-master-name: . . . . . TAVE
- module-version-name: . . . . TAVE_min_max
- description: . . . . . daily average temperature
- field-type: . . . . . numeric
- field-length: 10
- field-decimals: 3
- implementation-description: . . min max method
- implementation-reference: . . . Campbell model
- creation: 11/15/87
- modification: . . . . . 3/20/88 - 12:03
- implementation-code: . . . . . TAVE = (TMIN + TMAX) / 2

If this module description has not already been defined the user would have to enter it into the system otherwise the user need only to pick this module during the simulation selection procedure.

Respectively the form description for version 2 of the module TAVE used in the second run would be:

- module-master-name: . . . . . TAVE
- module-version-name . . . . . TAVE_3TEMPS
- description: . . . . . daily average temperature
- field-type: . . . . . numeric
- field-length: 10
- field-decimals: 3
- implementation-description: . . 3 collected temperatures
- implementation-reference: . . . unknown
- creation: 12/15/87
- modification: . . . . . 3/20/88 - 12:34
- implementation-code: . . . . . TAVE = (TDAWN + TNOON + TSUNSET) / 3

Now that we have a convenient way of defining our modules we need to use the mechanism described as the simulation selection procedure to link them in some desired manner. So in order to run the simulation we need to select some desired modules. In our case the top module is the module TAVE. Thus selecting the TAVE module using the selection procedure, may look like figure 10



Selecting TAVE module for a simulation run --- Figure 10

This simulation selection procedure is an equivalent form of writing a script such as:

```
DO TAVE
REPORT ON TAVE
```

which also happens to include a simple reporting command. The first line of the script tells the system to invoke the module that computes the daily average temperature.

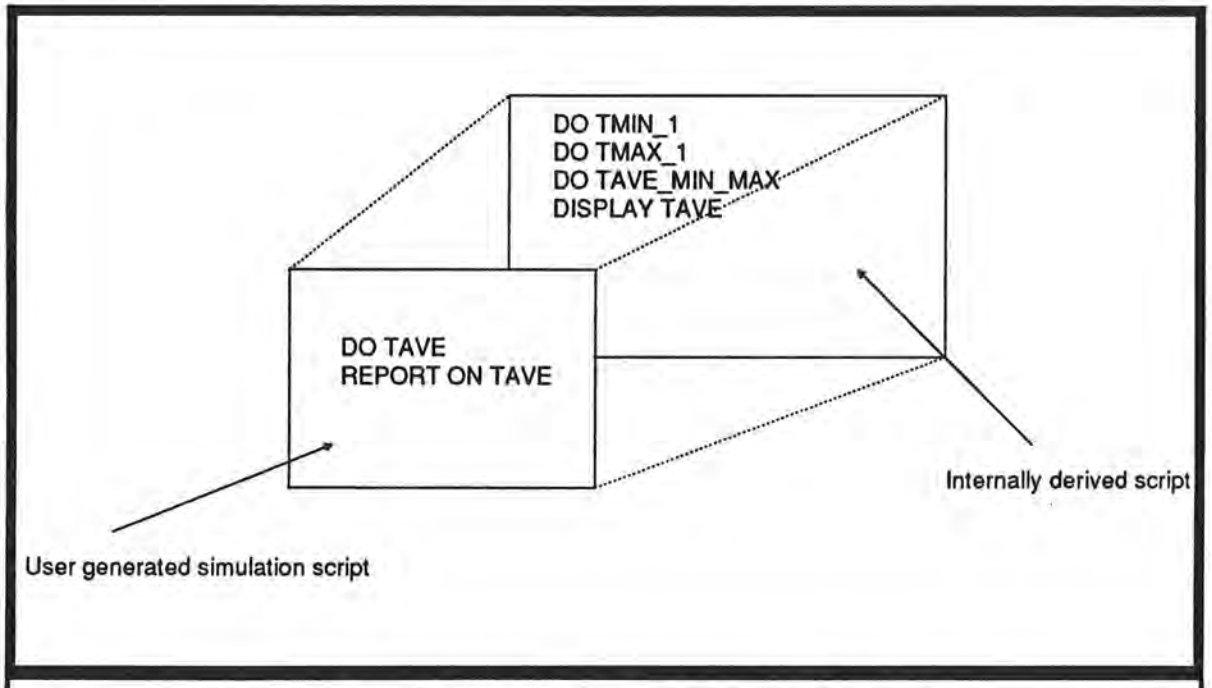
Assuming the user has already used the version selection procedure and the relevant current binding of data-items to specific versions here expressed in textual form is:

```
TAVE is-assigned-to TAVE_MIN_MAX
TMIN is-assigned-to TMIN_1
TMAX is-assigned-to TMAX_1
```

Then as we have seen from the module description of TAVE version TAVE_MIN_MAX (and more specifically from the algorithm) that it refers to TMIN and TMAX data-items. Thus, this particular TAVE module depends not only on the existence of these data-items, but also on the correct values of the data associated with them. The system can derive this dependency relation which is **not explicitly** given.

This translates into: the module TAVE_MIN_MAX depends on the TMIN_1 and TMAX_1. The system uses this knowledge along with the modification dates associated with every data-item to decide if the data-items values are up to date or need to be re-computed. Hence, these data-items (TMIN and TMAX) need to be made prior to the TAVE data-item.

Therefore, the actual internal executed script is like the one seen in figure 11: Where the single line DO TAVE causes the activation of the physical modules TMIN_1, TMAX_1, and then TAVE_MIN_MAX in this order.



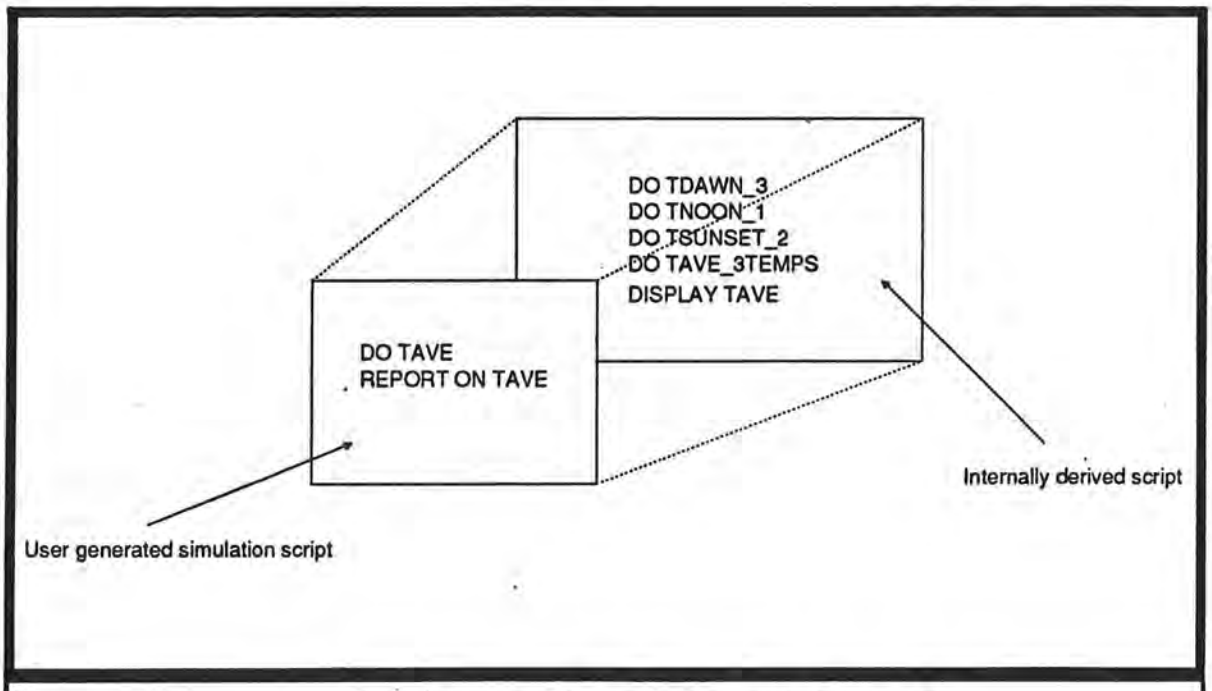
Internal execution for simulation run 1 --- Figure 11

Note that this process by which the system derives a dependence relation between modules is very useful. This automatic-making causes the simulation configuration file, or script defined in the simulation selection procedure to be **reusable**. This is true, since it only refers to the module master names and not to specific versions of the modules.

Now to execute the second simulation run, the user does not need to re-select a different simulation configuration since he is interested only in using a different version of the TAVE module. However, he must ensure the proper **binding** of data-items to specific modules. This can be done by selecting different versions using the version selection procedure, or by loading a previously stored version binding (i.e. version configuration file). Hence, the second simulation run will require the following necessary bindings:

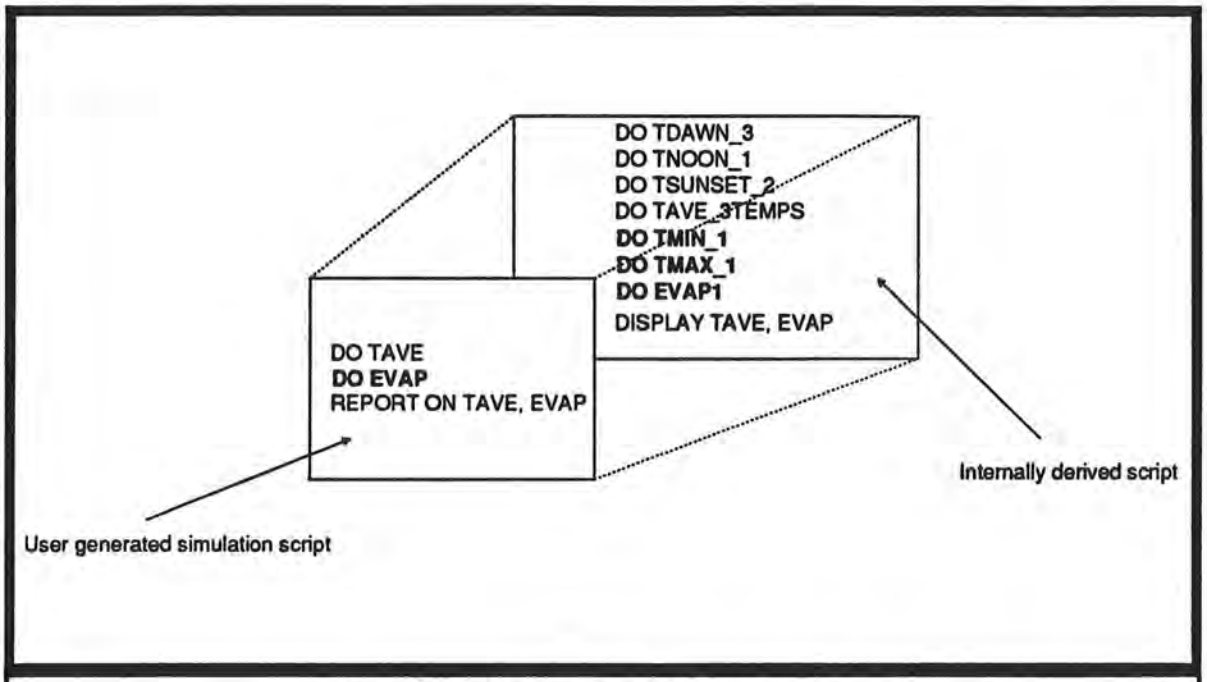
```
TAVE is-assigned-to TAVE_3TEMPS
TDAWN is-assigned-to TDAWN_3
TNOON is-assigned-to TNOON_1
TSUNSET is-assigned-to TSUNSET_2
```

Then according to the dependency relation derived by the system the actual internally executed script is seen in figure 12. Notice that this time the module TAVE causes the activation of four completely different modules.



Internal execution for simulation run 2--- Figure 12

Finally, consider the simulation script in figure 13. Although the TAVE module is still assumed assigned to the TAVE_3TEMPS module, both of TMIN_1 and TMAX_1 are still activated. This is due to the fact that the EVAP module in this case assigned to EVAP1 depends on these two modules, and hence they are also included in part of the derived internal simulation script.



Internal script when EVAP is added --- Figure 13

7-7 Module Dependence Relation Resolution

Before the system can execute a simulation run, the user must define to the system which versions are to be used, by setting the binding between the module's master-name and the desired corresponding module version-name, using the version selection procedure. This can be done also by loading a version configuration file. As the system sequentially executes, the simulation script, it encounters a reference to a module master name, and by using the currently defined bindings, it accesses the specific version source algorithm. If the source code makes reference to other modules' master names this process is repeated. These references eventually end, and the system activates the module's algorithms recursively.

Summary

Using the user generated script, the system generates the internal scripts, using a simplified algorithm such as:

```
1- For each Master name encountered in the script
        a- replace it by its current module version name
2- For each module version name
        a- access its implementation code
        b- insert all references to master names prior to the module version
           name into the script
Repeat the process until all references to master names are resolved.
```

If efficiency is an issue, some of the dependency resolution can be derived incrementaly as modules are introduced into the system, instead of being exclusively derived at run-time.

7-8 Iterative and Branching Examples

The examples discussed in the previous section invoke each module once in a sequential fashion. This implies that each module will compute all of its output and then return control to the script to activate the successive module.

Our design also supports simulation where each module can only compute the current instance of its output with respect to the simulation execution. In other words module depend on each other and have to wait for each other computations. The script facility adapts to this situation by providing a looping mechanism by which each module is invoked in a particular order. Each module will compute its current output and must return control to the script.

For example consider the simulation script for the daily activities of a banking account shown below:

```
DO WHILE NOT END-OF-SIMULATION
        DO DAY
        DO STARTING-BALANCE
        DO COMPUTE-WITHDRAWALS
        DO COMPUTE-DEPOSITS
        DO COMPUTE-CHARGES
        DO COMPUTE-INTEREST
        DO ENDING-BALANCE
END-WHILE
```

Clearly, to compute the daily starting balance we need to compute the ending balance of the previous day. To compute the ending balance we need to compute the current

day activities such as interest which in turn depends on the current balance. In this situation we say there is a mutual dependency between modules, and hence modules cannot be assumed to compute all of their values through a single activation. Instead the simulation must iterate through the modules.

Another module which cannot run independently of the other modules because it relies on the current state of the account is the COMPUTE-CHARGES module. If our model for simulating service charges is:

- charge five dollars monthly,
- charge ten dollars if balance drops below a minimum of fifty dollars
- charge nothing otherwise

its implementation which requires conditional computation may look like the following:

```
REPLACE CHARGES WITH 0
IF BALANCE  50
        REPLACE CHARGES WITH CHARGES + 10
IF DAY-OF-MONTH = 1
        REPLACE CHARGES WITH CHARGES + 5
```

Note: service charges are absolute values.

Summary

Looping and conditional execution are both powerful constructs in computer programming and our design permits their usage either at the script level, or the module level or both.

8 SIMUBASE: Current Implementation

Our prospective approach has pointed out the use of a common area for storing the shared data and providing a data manager around it. This suggests the use of databases and a DataBase Management System (DBMS). Hence, we have attempted to implement part of the above design strategy using an available database management system and tailoring it to our needs. We have considered two popular relational DBMS systems on the PC, namely DBASE-III and FoxBase+. These DBMS are very compatible. In fact FoxBase is a super set of the Dbase-III language and thus provides more built-in functions and features.

8-1 Previous Use of Data Base Concepts in Simulation

The idea of using database in simulation is not a new one. [Standridge&Pritsker-1982] A paper which discusses SDL, a Simulation Data Language, which provides modelers with data management techniques needed to handle simulation-related data, See also side bar. Nonetheless, most of the literature talks about simulation languages, and no one seems to mention the need of a simulation management environment; an environment with modular-programming that allows plugable modules. This is what SIMUBASE is all about.

DataBase Capabilities & Simulation

In several studies, existing general purpose data base systems have been incorporated into simulation languages for a specific purpose. One example is the development of INDECS, a general conveyorized facilities simulation package, by [Nof and Wilson]. A computer program written in the GASP II simulation language was used to simulate the system. A CODASYL data base was used to hold model data, such as the exact configuraion to be simulated. Another example by [Joseph and Roberts] involved the integration of the GPLAN data base system into the INS simulation language. The INS analys program causes the detailed output of a simulation run to be stored in a GPLAN data base. Thus, instead of being limited to summary statistics, the user is given access to the step-by-step information generated by the simulation of his system. A somewhat similar approach was taken by [Duket&Wortman], who stored the output of a SAINT simulation in a computer file. Then the Statistical Package of the Social Sciences was used to produce the desired analysis. [Markowitz] describes a data base system which could be incorporated into SIMSCRIPT. By designing the data base to be compatible with the representation of model entities, these entities, their attributes, and their relations can be stored directly. [Markowitz] proposes the entity-attribute-set representation as a flexible, general framework for organizing data.

8-2 Physical Storage

8-2-1 Simulation Data

In earlier reference to the prospective approach, the need for a common data storage was cited. In order to store the data generated by SIMUBASE, every data-item has been implemented as a field in a database (runtime database). This will permit the computation and storage of data generated during a simulation run.

8-2-2 Standard Module Description

As was pointed out in earlier sections, the textual standard module descriptions may take one of many layouts when implemented as screen forms. Hence, to allow a fast and a convenient module definition in this implementation of SIMUBASE, I have chosen the layout seen in figure 14. Since the form does not accommodate enough space for the module algorithm source code, there will be an accessible editor window associated with every module definition. See figure 15

Besides defining a module to the system, introducing a new module in the system translates usually into introducing a new data-item and subsequently in this implementation, a new record into the module database. In SIMUBASE the user may add a new

Module/Script Name	Code File	File Type	Creation Date
TAVE	TAVE2	FUN	12/03/86

Short Description	Last Modification Date
Daily Average Temperature	05/23/88

Author	Last Modification Time
Abdenacer	12:35pm

Long Description

Computes the daily average temperature, Campbell Model 1986

Specifications	
Type	Numeric
Field Width	10
Decimals	3

SIMUBASE Standard Description Form for TAVE --- Figure 14

module using the module manager options. This is done by filing an empty form description for the desired module.

```
Edit: H:\DB\CAMPBELL\TAVE1   .PRG       Ins       Caps
/=================#====================#==============#====================\
# CURSOR  <-- -->  #         UP   DOWN   #    DELETE    # Insert Mode:  Ins  #
#  Char:   <   >   # Field:   v     ^    # Char:  Del   # Insert line:  ^N   #
#  Word:  Home End # Page:  PgUp  PgDn   # Word:   ^T   # Save: ^W  Abort:Esc#
#  Line:   ^<  ^>  # Find:   ^KF         # Line:   ^Y   # Readfile:    ^KR   #
#  Reformat: ^KB   # Refind: ^KL         #              # Writefile:   ^KW   #
\=================#====================#==============#====================/

*******************************
** Code file: TAVE1
** Description: Daily Average Temperature  Min-Max
** Author: Abdenacer
** Created on: 12/18/88 -- 11:10:32
*******************************


REPLACE ALL TAVE WITH ( TMIN + TMAX ) / 2

RETURN 0
```

Header created automatically from the Standard Module Description

SIMUBASE Editor Window --- Figure-15

Summary

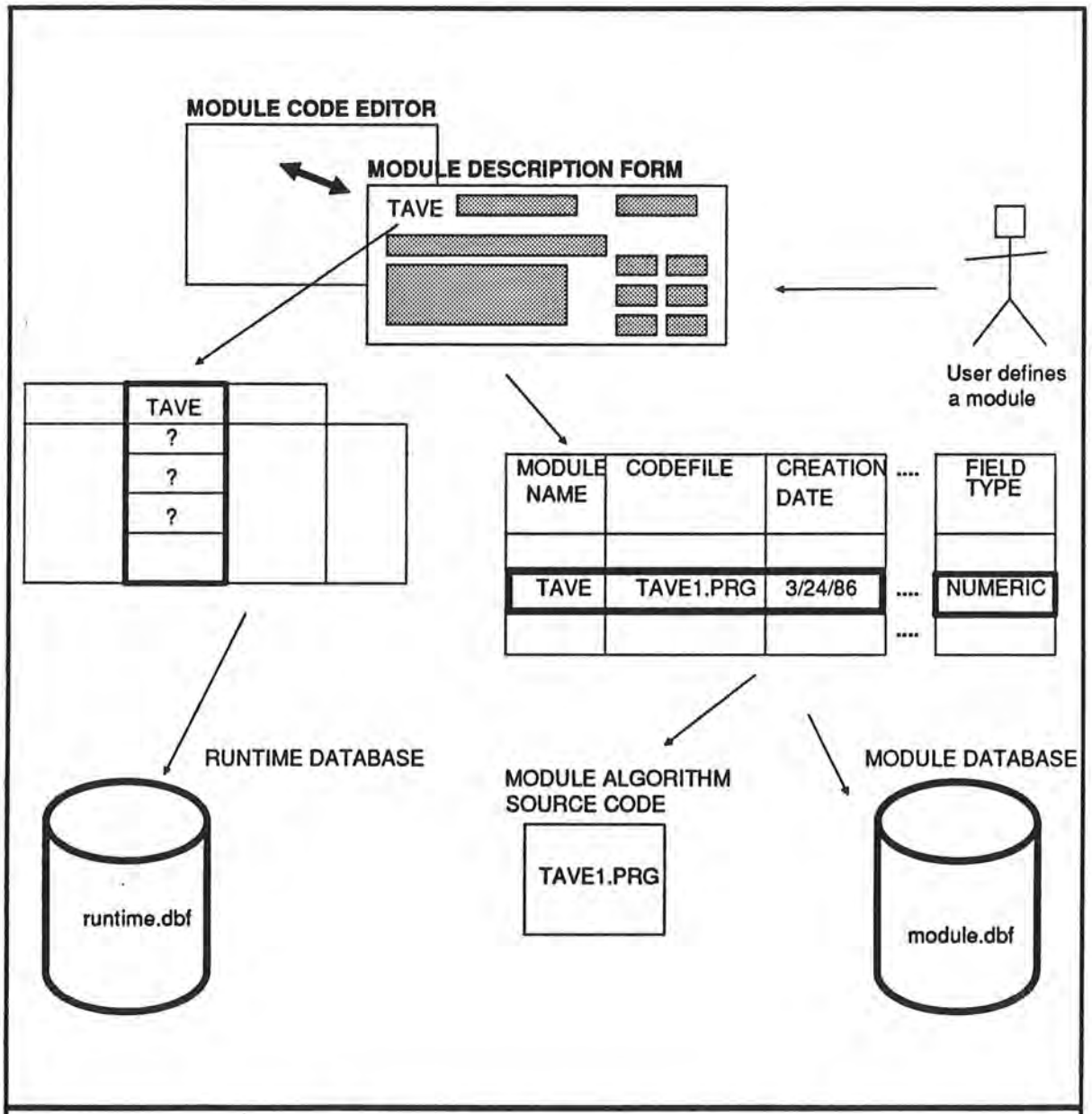
The actual steps taken by the current implementation of SIMUBASE, are shown in figure 16, and clearly reveals the current database design. After a standard module description form has been filled, SIMUBASE stores this form in the module database (modules.dbf), which stores all of the module descriptive attributes except the algorithm source codes. The algorithm source codes are stored in separate files named after the module versions' specific name; for example (tave1.prg). From then on, SIMUBASE adds a new field to the run-time database (runtime.dbf). This field's name is nothing but the module master name or logical name. This implies that for every set of module versions there will be a single field namely the common master name associated with these modules in the runtime database.

8-2-3 Simulation Scripts and Reporting

Once enough modules are defined in the system, one may wish to write a script module to combine and run the desired modules. Script modules will usually end by using reporting commands on the desired data-items.

For simplicity, in SIMUBASE we have chosen to implement scripts in the same fashion as modules. Their information is defined using the same description format, and the scripts contents are generated with the same editor and stored in similar procedure files; for example (script1.prg).

Although in SIMUBASE, scripts share a lot of the characteristics of modules such as a master name, version specific name, and others; SIMUBASE handles them quite differently in that unlike modules they do not reference data directly, but activate other

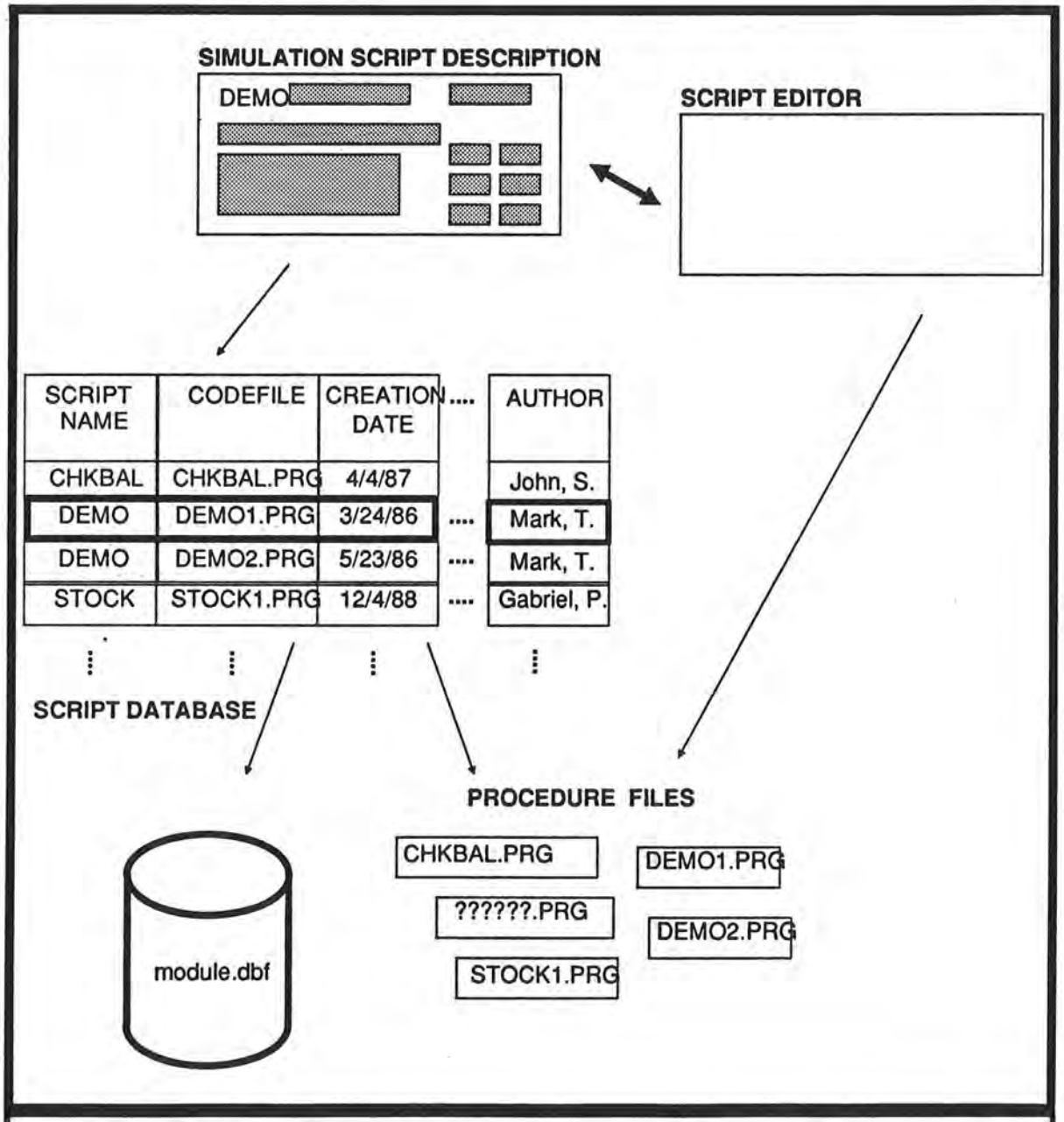


SIMUBASE internal mechanism --- Figure-16

modules or scripts. Moreover, they do not have data directly associated with them in the runtime database. The actual process of defining a script is shown in figure 17

8-2-4 Version Selection

Because of the host language inadequacy as we will see in a later section, this current implementation of SIMUBASE does not derive any module dependence relations.



Simulation Script Storage Management --- Figure 17

Since there is no provision for the automatic-making described earlier in the design strategy, the simulation scripts have to reference the module versions' specific names. For this reason also, there is no need for a version selection procedure. Although this feature of the system has not been implemented, it does not diminish the power of the current environment.

Because there is no provision for automatic-making of simulation scripts the user has to hand craft them. Hence, simulation scripts are similar to the automatic, internally derived scripts, discussed earlier in the design.

Nevertheless, even as it stands now, the current system establishes a possible solution to the problems described in this paper.

8-3 How Does Our Example Work Under the Current Implementation ?

The module definition in SIMUBASE is similar to the one described in the design strategy, except for the algorithm source code construct. In this implementation, for instance, the algorithm/formula:

TAVE = (TMIN + TMAX) / 2

is coded using DBase language constructs such as:

REPLACE TAVE WITH (TMIN + TMAX) / 2

Since this language is very English-like, the user should not encounter any difficulties using it.

Thus, the data-items needed for running our experiment, using the daily average temperature TAVE_MIN_MAX method, will be represented in SIMUBASE as shown in figure 18

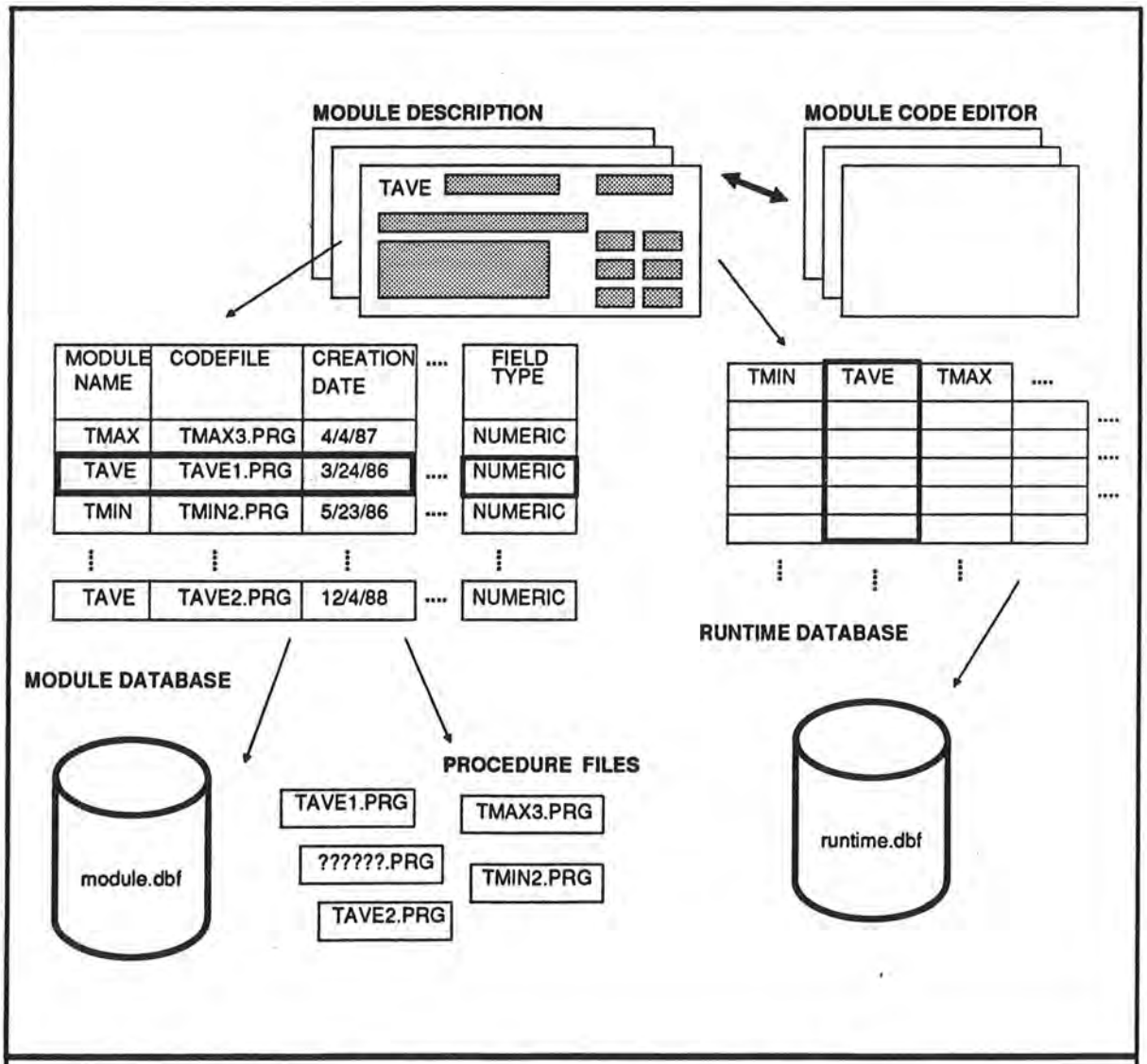
The script for TAVE_MIN_MAX will appear as:

```
DO TMIN1
DO TMAX1
DO TAVE1
DISPLAY ALL TMIN, TMAX, TAVE
```


The output from the simulation run will have the following tabular structure of fields in the system.

```
........... TMIN ..... TMAX .... TAVE
..............40 ..... .....50 .......45
..............50 .........60 .......55
..............55 .........65 .......60
```

Although the fields shown are ordered in a particular fashion; it is not essential. In fact their relative order as well as their existence is entirely governed by the database



Modules Storage Management --- Figure 18

manager. The field names shown here are the master names for the data-items involved in this particular simulation run.

8-4 Features of the Current System

- integrated environement with built-in editor, extensive on-line help, debugging facilities, and automatic system documentation.
- save output to file for later analysis with statistical packages. This also allows the comparison of the outputs of two or more models.
- running a single module without a script, allows debugging of the module before including it in a actual simulation script.
- generation of reports concerning information about the models defined in the system, such as module and script attributes, etc...
- automatic header generation for the code file, using the information recorded from the standard module description.

8-5 Limitations of the Current System

By no mean is the current prototype near the targeted design. Although it clearly demonstrates module interface independence, there is still a large gap between it and the intended system. One of the important aspects that is currently absent, is the intelligent behavior in the system such as:

- Ability to derive dependency relations among the different modules
- Automatic running of subordinate modules. Currently the scripts have to be explicit and manually crafted.
- Detection of attempts to use nonexistent modules, and provision of various alternatives.

Other limitations are:

- slow execution especially as the number of modules increase due to file manipulation which can be improved by better management of the runtime database.
- ability to handle complex data.

8-6 About the Language of Implementation and its Effectiveness

In the current implementation of SIMUBASE, module source codes as well as the simulation scripts are the standard clauses available in the Dbase language. In fact, it is interesting to note that not only module algorithms and simulation scripts are written in Dbase, but this simulation environment prototype (SIMUBASE) itself, has been developed, using the same language.

The fact that data items have been stored as fields in the database, and that DBase clauses can reference these fields independently of their physical storage, has helped implement this version of SIMUBASE. This ability has been exploited to modify data-item with the REPLACE clause and report on data-items with the LIST and DISPLAY clauses. Refer to the dBase manual or the on-line help in SIMUBASE for a thorough description of these clauses.

8-7 Host Language Inadequacy

The following sections will discuss the obstacles encountered during the development of the targeted design.

I would like to emphasize that neither the current simulation prototype system nor the development environment used to implement it, are flawless. The researcher would like to point out that he has undergone non-orthodox programming practices to achieve the current prototype. For example the intrinsic command/procedure to add a field to an existing database structure is not available. Dbase programmers usually create databases manually and then develop Dbase code to manage the data stored in them. Consequently, I had to implement complicated code to add and delete fields used in conjunction with managing data-items.

Activating a procedure file at run-time was another battle. The Dbase DO <procedure-file> command expects a literal, and not a variable. However, the trick around this was to force the project modules to be written as parameterless User Defined Function which has caused the appearance of the terminating clause RETURN 0 in their implementation code. By using a dummy variable named EXECUTE and assigning to it the a user defined function through the use of Dbase macro substitution, I was able to activate the project modules and simulation scripts stored as Dbase procedure files. (refer to the source code run_mod.prg and run_bat.prg for more detail).

Foxbase does not provide any mean to process a file containing a module implementation code. This lack of text file processing in the Dbase language made it difficult to

attempt implementing the module dependency resolution and hence the implementation of the algorithm to derive internal scripts from generic reusable scripts has been left for future consideration.

Nevertheless, even as it stands now, the current system establishes a viable solution to all the problems described in this paper.

For more detail about common problems in simulation languages refer to [Miller&Morgan-1976].

8-8 Consideration when Selecting a Host Language Suited for Such a Prototype Development

In the following sections I will attempt to highlight the key features that make a suitable development environment for such a system, as well as the ones that contribute to a friendly simulation development environment.

8-8-1 Library routines, Interpreted .vs. Compiled code environment

Many DBMS systems are now popular on Personal Computers. However, because of their availability, our options have been narrowed to Dbase-III and FoxBase+. We are currently using FoxBase because some of our simulation processes require trigonometric functions. Although none of the DBMS provide an immediate support for them, FoxBase however allows User-Defined-Function. As the name dictates, the user can implement functions to complement/extend the intrinsic ones. Thus, in order to support the simulation sample we happen to work with, I implemented in grief, an approximation of some trigonometric functions [Rektorys-1969]. Of course there exist third party vendors who supply libraries written in conventional languages such as C that can be linked to the DBMS language. However, this would have implied compiling code, a penalty that the developer opted not to pay during the development course.

8-8-2 Self-interpretation and Partial Compilation

In many instances, programmers face the problem of having to build a small interpreter part of an application. This implies re-inventing the wheel each time our application includes some language interpretation. It is useful to have a programming language which can interpret itself or a subset of its constructs. Through the tricks described above, I was able to activate user created procedure file and allow their modification at run-time.

It's understood that eventually we would like to compile certain portions of the code while maintaining others as source. In other words, we would like to compile the application code and keep the application user code in source form to be modified and interpreted at run-time.

9 SIMUBASE Overview

SIMUBASE is both a simulation code manager and a sophisticated programming environment. SIMUBASE was designed to work under FoxBase+ to create a complete program development system; one which allows you to harness the tremendous power of FoxBase+.

9-1 Fundamental Concepts

9-1-1 What is a SIMUBASE module?

A module in SIMUBASE is a collection of descriptive attributes bound to a master data-item name. Each module is responsible for a single piece of data and describes how it is to be computed.

9-1-2 What is a script?

A script is the mechanism by which some desired modules are invoked in some order to activate a simulation run. The script facility also allows reporting on selected data-items to be viewed on the screen, printed to paper, or committed to a file.

Although in SIMUBASE, scripts share a lot of the characteristics of modules such as a master name, version specific name, and others; SIMUBASE handles them quite differently in that unlike modules they do not reference data directly but activate other modules or scripts. Moreover, they do not have data directly associated with them in the runtime database.

9-1-3 What is a project?

A project is a set of related modules and scripts that belong to a certain experiment. The grouping of these experimentation modules and scripts into projects will be controlled by the modeler. The project feature is provided currently for organizational purpose, to keep the experimentation environment relatively small and accessible.

9-2 SIMUBASE Module and Script Manager

The module and script manager allows the introduction of new modules and simulation scripts into the system or modify existing ones.

Creating modules is achieved by choosing the module management option and filling module descriptive forms. Defining simulation scripts is done in a similar way.

9-3 SIMUBASE Documenter

SIMUBASE also allows the user to generate extensive reports on the project desired. Some of these reports can be included in the final system documentation such as data-dictionary, module and script reports.

9-4 Multiple Projects

SIMUBASE environment can handle more than one project. This will allow the module developer to organize module databases by related subjects or experiments. This will also allow the modules user to easily locate the modules they are interested in for a particular experimentation.

When a new project is created SIMUBASE creates a separate sub-directory for it. This directory will be used to store all of the projects specific files. These files include the run-time database, the module descriptions database, modules algorithm code files, scripts files and the project constants database.

9-5 Who Can Use SIMUBASE?

Fundamental knowledge of personal computer operation including DOS and file structures are the only prerequisites.

This type of environment will encourage researchers to experiment and share their creativity with others due to the elimination of rigid specifications (language, data-storage, modules communication, etc...)

10 SIMUBASE Menus

The menu system provides a handy way to invoke frequently used command with minimum keystrokes. If you want to exit the menus without invoking any option, press the ESC key.

```
00000000=------------------------------------------------------------=00000000
0000                 M A I N   M E N U                                 0000
00                                                                      00
|      M- Module/Script Manager...              O-open a project        |
|                                                                       |
|      E- Quick-Edit Modules or Scripts code    H-system help...        |
|                                                                       |
|      S- Run a Simulation Script (batch of modules)                    |
|      D- Run a module (for debugging)                                  |
|                                                                       |
|      P- Peek at the data-items runtime database                       |
|      C- Modify user/system constants                                  |
|      7- Sort modules database                 Z-zap deleted modules   |
|      R- Report generator...                                           |
|      0- to quit                               X- shell to DOS         |
|                                                                       |
00                                                                      00
0000                                                                  0000
00000000=------------------------------------------------------------=00000000
Enter your option
```

SIMUBASE MAIN MENU --- Figure 1

Some menu options have three dots (...) beside them. These options contain other options. If you select a menu option with tree dots, a whole new menu (a sub-menu) joins the one you're working with. You can select various options from the sub-menu.

10-1 Main Menu

The main menu includes many options, see figure 19. Most of these options will be described in the following sections.

10-2 Selecting a project

SIMUBASE allows the user to organize modules and scripts intro projects, this option is usually used to select and open a project. After a project has been opened it becomes the current project and all other options in the menu relate to this project until a different project is selected.

10-3 Module/Script Manager

This is were you define a data-item to the system, by giving the data-item master name, the data-item code file name, and the data item type specification. This may include type, field width, and decimal places. This option will also allow the user to enter a short description of the data-item, as well a full description regarding the implementation, reference, etc... (Refer to the Standard Module Description for more detail)

This option features full screen editing with searching and browsing abilities. It also allows the user to mark modules for later deletion.

10-4 Running a Simulation Script

This option allows the user to select among the script available in the current opened project and activate it.

10-5 Project Constants Manager

This system also incorporates the ability to introduce global constants that may be referenced throughout the system by any module. The project constants manager is also able to include a description of the constant being added to system. It enables the user to edit its value, or its description, or delete the constant at any given time.

10-6 Viewing the Run-time database

SIMUBASE allows the user at any time to peek at the run-time database. In fact, this database is were SIMUBASE keeps the values generated by a module, when it was last run. Viewing such a database is somewhat similar to examining a core dump or register values in another programming language environment. Being able to peek at this data allows the module developer to quickly trace the values a particular module is producing, as well as those produced by other modules he may have activated.

```
*** Alphabetic Index **

<ALIAS>   |<HISTORY>|<OPERATOR|<PATH>    |<SCOPE>   |?         |@         |ABS()
ACCEPT    |ALIAS()  |APPEND   |ASC()     |AT()      |AVERAGE   |BOF()     |BROWSE
CALL      |CANCEL   |CHANGE   |CHR()     |CLEAR     |CLOSE     |CMONTH()  |COL()
CONTINUE  |COPY     |COUNT    |CREATE    |DATE()    |DAY()     |DBF()     |DELETE
DELETED()|DIMENSION|DIR      |DISKSPACE|DISPLAY   |DO        |DOW()     |DTOC()
EDIT      |EJECT    |EOF()    |ERASE     |ERROR()   |EXIT      |EXP()     |FCOUNT()
FIELD()   |FILE()   |FIND     |FKLABEL()|FKMAX()   |FLOCK()   |FLUSH     |FOUND()
GATHER    |GETENV()|GO       |HELP      |IF        |IIF()     |INDEX     |INKEY()
INPUT     |INSERT   |INT()    |ISALPHA()|ISCOLOR()|ISLOWER()|ISUPPER()|JOIN
KEYBOARD  |LABEL    |LEFT()   |LEN()     |LIST      |LOAD      |LOCATE    |LOG()
LOOP      |LOWER()  |LTRIM()  |LUPDATE()|MACROS-&  |MAX()     |MENU      |MESSAGE()
MIN()     |MOD()    |MODIFY   |MONTH()  |MULTIUSER|NDX()     |NOTE      |ON
OS()      |PACK     |PARAMETER|PCOL()   |PRIVATE   |PROCEDURE|PROW()    |PUBLIC
QUIT      |READ     |READKEY()|RECALL   |RECCOUNT |RECNO()   |RECSIZE()|REINDEX
RELEASE   |RENAME   |REPLACE  |REPLICATE|REPORT    |RESTORE   |RESUME    |RETRY
RETURN    |RIGHT()  |ROUND()  |ROW()    |RTRIM()   |RUN/!     |SAVE      |SCATTER
SCROLL    |SEEK     |SELECT   |SELECT()|SET       |SKIP      |SOUNDEX   |SORT
SPACE()   |SQRT()   |STORE    |STR()    |STUFF()   |SUBSTR()|SUM       |SUSPEND
SYS()     |TEXT     |TIME()   |TOTAL    |TRANSFORM|TRIM()    |TYPE      |TYPE()
```

Aphabetic Index for the dBase commands --- Figure 2

10-7 Help Menu

SIMUBASE provides an extensive assistance help on all the commands available in the language. The help menu is subdivided into logical categories to facilitate the task of locating the desired help topic more accurately. An alphabetized index for all FoxBASE+ commands is also accessible. See figure 20

PAGE NO. 1 CAMPBELL MODULE(s)
03/22/89
LISTING

Master Name	Version	Short Description	Author	Created	Modified
DA	DA	DAY	Abdenacer	12/28/88	03/19/89
DAY	DAY	ALL YEAR	Abdenacer	12/27/88	03/21/89
DAY	DAY1	1 TO 50 STEP 1	Abdenacer	12/25/88	03/22/89
DAY	DAY10	EVERY 10 DAYS	Abdenacer	12/26/88	03/21/89
DAY	DAY30	EVERY 30 DAYS	Abdenacer	12/24/88	03/22/89
DEC	DEC1	DECLINATION	Abdenacer	12/10/88	03/22/89
EP	EP	PARTITION PET INTO PE	Abdenacer	01/08/89	03/22/89
ETP	ETP	EVAPO TRANSPIRATION	Abdenacer	12/29/88	03/21/89
FI	FI	Fractional	Abdenacer	12/11/88	03/22/89
Interception					
LAI	LAI	Leaf Area Index	Abdenacer	12/12/88	03/22/89
NETSRAD	NETSRAD	Net Solar Radiation	Abdenacer	12/13/88	03/22/89
PE	PE	PE?	Abdenacer	01/07/89	03/22/89
PEVAP	PEVAP	Potential evaporation	Abdenacer	12/14/88	03/22/89
PRAD	PRAD1	Potential radiation	Abdenacer	12/15/88	03/22/89
PSR	PSR1	POTENTIAL SOLAR	Abdenacer	01/03/89	03/21/89
RADIATION					
PT	PT	PARTITION PET INTO PT	Abdenacer	01/09/89	03/22/89
PTRANS	PTRANS	Potential	Abdenacer	12/16/88	03/22/89
transpiration					

Project Module Report --- Figure 3

PAGE NO. 1 CAMPBELL SIMULATION SCRIPT(s)
03/22/89
LISTING

Master Name	Version	Short Description	Author	Created	Modified
CAMBBELL	CBELL2	script for campbell	Abdenacer	01/01/89	03/19/89
version2					
CAMPBELL	CAMPBELL	script for campbell	Abdenacer	12/31/88	03/19/89
version1					
PAGE37	PAGE37	PLANTEMP PROGRESS	Abdenacer	12/05/88	03/22/89
REPORT					
PAGE38	PAGE38	PLANTEMP PROGRESS	Abdenacer	12/06/88	03/22/89
REPORT					
PAGE40	PAGE40	PLANTEMP PROGRESS	Abdenacer	12/07/88	03/22/89
REPORT					
PAGE40_1	PAGE40_1	1 TO 50 STEP 1	Abdenacer	12/08/88	03/22/89
TAVE	STAVE1	DEMO TAVE WITH 2	Abdenacer	01/06/89	03/22/89
TEMPERATURES					
TAVE	STAVE2	demo tavg with 3	Abdenacer	12/09/88	03/22/89
temperatures					

Project Simulation Scripts Report --- Figure 4

10-8 Sorting The Modules Database

At any time the user may choose to organize the listing of modules in the order to satisfy his needs. Since the number of modules may grow quite large at any time, this option facilitates locating a module within any given project.

10-9 Reporting and System Documentation

SIMUBASE also allows the user to generate extensive reports on the system and the project desired. Some of these reports can be included in the final system documentation, such as data-dictionary, and module reports.

10-9-1 Project Modules Report

This report includes the modules currently defined in the project. The report can provide a great deal of help to a new user so that he becomes familiar with what is currently defined in the system. See figure 21

10-9-2 Project Scripts Report

This report includes the scripts currently defined in the project, which will include the script names, description, date of creation, and date of modification. See figure 22

```
PAGE NO.     1                    CAMPBELL
03/22/89
PROJECT CONSTANTS REPORT

Constant Name Constant Value                Full Description
_____________ _____________                 ________________


PI                3.14159256400
LAT           0.84                          Latitude
K1                1.35000000000
K2               -0.35000000000
K3                0.67000000000
K4                0.44000000000
ALBEDO            0.22000000000             Surface Albedo
PLDA          119                           Planting date
EMDA          130                           Emergence date
MTDA          180                           Maturity date
FC            0.25                          Field Capacity
RDMAX         2                             Maximum rooting depth - meters
KC            0.4                           ave daily canopy transm coeff.
```

Project Constants Report --- Figure-5

10-9-3 Project Constants Report

This report includes the constants declared in the current project. Included are such things as constant name, value, description, etc... A sample report is shown in figure 23

10-10 Exit/Shell to DOS

This option permits you to use all DOS commands without interrupting your work in SIMUBASE. Currently SIMUBASE does not support DOS commands. No problem - just choose this menu option and you are in DOS, while everything in SIMUBASE stays in memory. When you're done with DOS, just type EXIT and you're back in SIMUBASE again, with everything as you left it.

11 A Guided Tour

SIMUBASE lets you control the creation, the managing and the execution of your modules. SIMUBASE is a highly-interactive environment that encourages rearranging and experimentation.

When using SIMUBASE you don't have to page through heavy reference manuals each time you decide to lookup a command in the language; on-line help is provided for your convenience. Simubase also includes a built-in editor to provide a complete integrated environment.

11-1 SIMUBASE Demo Project

The attached disk contains some demo SIMUBASE projects which include complete sets of modules that you can experiment with.

- DEMO1 . . . . . Stock Simulation
- DEMO2 . . . . . Check-Book Simulation
- CAMPBELL . . . . . Winter Wheat Dry Matter Simulation

Using a simulation demo project

To access a demo project, you start by opening the demo project and selecting the [open project] option from the menu. You may then select one of the several scripts to activate a simulation run.

11-2 Running The Simulation Demo

Project DEMO1 is a small set of modules and simulation scripts that simulates the behavior of a financial stock. The two scripts available in the project will demonstrate the ability to substitute one model/computation of the INTEREST module for another, with little effort from the user.

In project DEMO1 there are already two simulation scripts, and you will be able to create more. For now let us experiment with the one already provided.

To activate a script use the [run simulation script] option from the main menu. The system will then present a list with the available scripts. Using the cursor-keys, position the highlight-bar over the script named SIMU1, and press the CTRL-END to choose it. The system will now open this script and start activating the modules. As each module executes, the system will monitor this activity on the screen. Eventually the activation of modules terminates, and the script executes some reporting command on selected data-items. If these reports exceed the screen size the display will pause and allow the user to step through the report, viewing the data at his/her own pace.

Activating the second script is done in a similar way, except that this time you highlight the script named SIMU2 and press the CTRL-END to activate it.

You have now completed running these scripts. You may not yet appreciate the power of SIMUBASE, but to provide a similar demo in Pascal for example, you would have to write a fair amount of code (refer to the two examples given in the objective).

Sending the report in SIMUBASE to the printer is just as simple as adding the TO PRINT words at the end of the Dbase reporting clause. A more sophisticated option for re-direction of output to selectable devices (screen, printer, and files) can be achieved by inserted the commands DO PRINTBEG and DO PRINTEND respectively before and after the reporting command.

11-3 Creating a Module

Now if you would like to experiment with your own version of the interest module, you can define it as follows:

First choose the [module/script manager] option from the main menu. Next, use the append command by pressing the letter A. You will then be presented with a blank module description form. Proceed by entering the master name, INTEREST, and the version name INT3. For the file code field enter FUN (if you were defining a script type BAT). Enter a short description for your version and modify the type specification if necessary; or accept the defaults, provided other fields can be left blank and updated at a later time. Exit from the append option by pressing the letter F to exit from the append mode. To enter your own formula for INTEREST, use the M option, and enter something like:

REPLACE INTEREST WITH MOD(DAY, 30)

or use your own formula. Exit from the editor with CTRL-END.

To try out your implementation modify the second script SIMU2 by replacing the line DO INT2 with DO INT3 and then active the script as described earlier from the main menu. You could have created a whole new script, however, modifying an existing one is a much quicker process.

11-4 Summary

This guided tour ends here, however your experimentation does not. You have now acquired necessary knowledge on how to activate simulation scripts, create your own module and test it. You must now review your steps and experiment with the other features of this system using the other projects available.

12 Where to Go From Here

In order to render SIMUBASE environment thoroughly suitable for the vast array of simulation applications, the following features need to be considered in future work.

- Import and export modules between project and different sites.
- Module dependency relation resolution and automatic making of scripts.
- Currently SIMUBASE does not provide direct support and knowledge of units of measurements, therefore it would be nice if automatic support of units existed.
- Detection of reference to undefined modules.
- Investigate whether template driven code generator can be used in conjunction with the current implementation for automatic code generation.
- If the implementation language allows it, the programming language on-line help, as well as other help screens will be available from within the module editor window.
- Graphical representation of simulation generated data.
- Symbolic representation of dataflow during a simulation modeling such as [Stella-1987].

13 Conclusion

The prototype that was implemented in part of this research, creates a simple and yet powerful language and simulation development environment, which can be used by even novice computer users. SIMUBASE clearly establishes the ability to easily interchange different versions of module(s) without having to modify other modules or re-compile them. Simulation model development can be rapid and straight forward.

SIMUBASE allows the creation of more complex simulation models by incrementally combining simpler existing models. SIMUBASE is a dynamic environment that adequately suits simulation research which continuously evolves.

Existing simulators can be easily ported to SIMUBASE, because only the pure computational process of those simulators need to be transferred. This reduces transition time and cost factors that often hinders the migration to newer and more suitable technologies.

END

14 Bibliography

Software Engineering, Reusable Software, Object-Oriented programming, Prototyping.

[Beladi-1981] "Modifiability of Large Software Systems", Proc 14th IBM Comp Soc Symp Tokyo, Jan 8-13, Oct 1981, or (Program Evolution: Processes of Software Change, Ch 17), Academic Press.

[Lehman-1978] "Laws of Program Evolution - Rules and Tools for Programming Management", Proc Infotech State of the Art Conf, 'Why Software Projects Fail', Apr 1978, pp 11/1-11/25. (Ch 1, 14, 17, 19) or (Program Evolution: Processes of Software Change, Ch 12), Academic Press.

[Pizzarello-1984], "Development and Maintenance of Large Software Systems", Lifetime Learning Publications, Belmont, California.

[Lewis-1986], "Software Design For Ease of Maintenance and Reuse", Computer Science, Oregon State University.

[Rickman-1987], "Progress Report for CO2Wheat: Plantemp", Columbia Plateau Conservation Research Center, Pendleton, OR 97801.

[Cox-1984], "Message/Object Programming: An Evolutionary Change in Programming Technology", IEEE Software, Vol. 1 No. 1, Jan. 1984, pp. 50-61.

[Snodgrass-1983], "An Object-Oriented Command Language", IEEE Trans. Soft. Eng. Vol. SE-9, No. 1, Jan. 1983

[Cox-1983], "The Object-Oriented Precompiler - Programming Smalltalk-80 Methods in C languages", ACM Sigplan Notices, Vol. 18, No. 1, Jan. 1983, pp. 15-22.

[Korson&Vaishnavi-1986], "An Empirical Study of the Effects of Modularity on Program Modifiability", (Empirical Studies of Programmers, First Workshop on Empirical Studies of Programmers, June 5-6, 1986), Ablex Publishing Corporation.

[Pressman-1982] "Software Engineering, A practioner's Approach", McGraw-Hill Book Company.

[McCabe-1976], "A Software Complexity Measure," IEEE Transactions on Software Engineering, vol. 2, Dec 1976, pp. 308-320.

[Halstead-1977], "Elements of Software Science", North Holland, 1977.

Simulation Languages and Data Base Capabilities

[Standridge&Pritsker-1982], "Using Data Base Capabilities In Simulation", (Progress In Modelling and Simulation 1982, The Swiss federal Institute of Technology, Zurich), Academic Press.

[Babad&Schrae-1978], "A New Look at Process Oriented Simulation Languages", Graduate School of Business, University of Chicago.

[Duket&Worman-1976], "An Example to Illustrate the Use of SPSS for the Analysis of a SAINT Model", Report to the Aerospace Medical Division, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio, by Pritsker & Associates, Inc., West Lafayette, Indiana, July 1976.

[Joseph&Robert-1977], "Utility of Data-Base Management to Analyze the Output from Complex Simulations", Proceedings, 1977 Winter Simulation Conference, December , 1977.

[Markowitz-1977], "An Entity, Attribute, Set and Event View of Data Base Systems. IBM Research Report RC6811 (#29158), Yortown Heights, New York, October 1977.

[Miller&Morgan-1976], "Simulation Language Features in 1976: Existing and Needed", Proceedings, 1976 Winter Simulation Conference, December 1976.

[Nof&Wilson-1976], "INDECS: General Conveyorized Facilities Description and Simulation", Proceedings, 1976 Winter Simulation Conference, December 1976.

[Pritsker&Kiviat-1969], "Simulation with GASP II", Englewood Cliffs, J.J. Prentice-Hall, Inc., 1969.

[Pritsker-1974], "The GASP IV Simulation Language", New York, John Wiley and Sons, Inc., 1974.

Inter-process communication mechanisms, Modular Programming

[Stankovic-1982], "Software communication mechanisms: Procedure calls versus Messages", Computer Vol 14,4,p19-25

[Staunstrup-1982], "Message Passing Communication Versus Procedure Call Communication", SP&E Vol 12,3,p222-234.

[Welsh&Bustard-1979], "Pascal-Plus - Another Language for modular multiprogramming", SP&E Vol 9,11,p947,957. (The paper introduces Pascal-plus, a language building on Pascal that uses monitors for communication between processes).

[Wirth-1977], "Modula", SP&E Vol 7,1,p3-84. (The papers introducing the language Modula, describing its implementation and illustrating its usage in practical programming)

Simulation Environments, Miscellaneous

[Stella-1987], "Stella for Business", High Performance Systems, Inc. 1987

SIMUBASE Source Code Implementations

[Rektorys-1969], "Survey of Applicable Mathematics", M.I.T. Press Cambridge, Massachussetts.

[Miriam&Liskin-1987],"Advanced Dbase III PLUS Programming and Techniques", Osborne McGraw-Hill.

[FoxView], "FoxBASE+ Screen Painter & Application Generator - User Guide", Fox Software, Inc., May 1988.

[FoxDoc], "FoxBASE+ Documentation System-User Guide", Fox Software, Inc., May 1988

Index Listing

Appendix A

STATS.DOC

```
System: SIMUBASE: Data Base Based Simulation
Author: Mr. Abdenacer Moussaoui
03/19/89   19:15:36
System Summary

-------------------------------------------------------------------------------


This system has:
  1628 lines of code
    31 program files
     1 procedure files
    17 procedures and functions
     5 databases
     1 index files
     2 report forms
     0 format files
     0 label forms
     0 memory variable files
   300 cross-referenced tokens


See the tree diagram for programs, procedures, functions and format files

                    Index          Report         Label          Memory
  Databases         Files          Forms          Forms          Files
-------------- -------------- -------------- -------------- --------------

&PROJECTD.PCON &PROJECTD.MODULF1.FRM
PROJECTS.DBF                  CONST.FRM
&PROJECTD.MODU
&PROJECTD.RUNT
DBM.DBF

-------------------------------------------------------------------------------
FoxDoc created the following documentation files:
         H:\DB\OUT\STATS.DOC
         H:\DB\OUT\TREE.DOC
         H:\DB\OUT\FILELIST.DOC
         H:\DB\OUT\DATADICT.DOC
         H:\DB\OUT\FRMSUMRY.DOC
         H:\DB\OUT\ERROR.DOC
         Action diagram files
         UPDATE.BAT   to update program source files in H:\DB
         BACKDBF.BAT  to backup databases, indexes and memory files
         BACKPRG.BAT  to backup program files, report forms and format files
         PRINTDOC.BAT to print documentation files
```

TREE.DOC

```
System: SIMUBASE: Data Base Based Simulation
Author: Mr. Abdenacer Moussaoui
03/19/89   19:15:33
Tree Diagram

-------------------------------------------------------------------------------

TOP.PRG
+----SETE.PRG
+----ADDSPATH.PRG
+----SETPCONST.PRG
+----GETPROJ.PRG
|    +----SELECTWA.PRG
|    +----INFORM.PRG
+----OPENPROJ.PRG
+----MENU1.PRG
+----MM.PRG
|    +----MM_OPEN.PRG
|    |    +----MM_AREA (procedure in MM_PROC.PRG)
|    +----MM_EDIT.PRG
|         +----MM_FORM (procedure in MM_PROC.PRG)
|         +----SAYREC (procedure in MM_PROC.PRG)
|         |    +----STATLINE (procedure in MM_PROC.PRG)
|         |    +----MM_SAYS (procedure in MM_PROC.PRG)
|         +----GETKEY (procedure in MM_PROC.PRG)
|         +----MM_APPE.PRG
|         |    +----MM_FORM (procedure in MM_PROC.PRG)
|         |    +----MM_SAYS (procedure in MM_PROC.PRG)
|         |    +----MM_STOR (procedure in MM_PROC.PRG)
|         |    +----STATLINE (procedure in MM_PROC.PRG)
|         |    +----SAYLINE (procedure in MM_PROC.PRG)
|         |    +----MM_KEYS (procedure in MM_PROC.PRG)
|         |    +----YESNO.PRG
|         |    +----MM_GETS (procedure in MM_PROC.PRG)
|         |    +----GETKEY (procedure in MM_PROC.PRG)
|         |    +----MM_REPL (procedure in MM_PROC.PRG)
|         |    +----CODEFILE.PRG
|         |         +----YESNO.PRG
|         +----MEDITOR.PRG
|         |    +----MODIDATE.PRG
|         +----MODIDATE.PRG
|         +----MM_SEEK (procedure in MM_PROC.PRG)
|         +----SAYLINE (procedure in MM_PROC.PRG)
|         +----SAYEOF (procedure in MM_PROC.PRG)
|         +----MM_STOR (procedure in MM_PROC.PRG)
|         +----MM_KEYS (procedure in MM_PROC.PRG)
|         +----MM_GETS (procedure in MM_PROC.PRG)
|         +----MM_REPL (procedure in MM_PROC.PRG)
|         +----ZAPITEM.PRG
|         |    +----ZAPFIELD.PRG
|         +----ADDITEM.PRG
|         |    +----ADDFIELD.PRG
|         +----DOGOTO (procedure in MM_PROC.PRG)
|         |    +----GETKEY (procedure in MM_PROC.PRG)
|         |    +----GOTOREC (procedure in MM_PROC.PRG)
|         +----STATLINE (procedure in MM_PROC.PRG)
+----MEDITOR.PRG
|    +----MODIDATE.PRG
+----ADDITEM.PRG
|    +----ADDFIELD.PRG
+----RUN_MOD.PRG
+----RUN_BAT.PRG
+----SELECTWA.PRG
+----SORTMOD.PRG
+----PACK.PRG
|    +----YESNO.PRG
|    +----ZAPITEM.PRG
|         +----ZAPFIELD.PRG
```

```
+----DOREPORT.PRG
|    |    F1.FRM (report form)
|    |    CONST.FRM (report form)
|    +----MENUREPO.PRG
|    +----PRINTBEG.PRG
|    +----PRINTEND.PRG
|    +----SELECTWA.PRG
+----DOHELP.PRG
     +----MENUHELP.PRG
```

DATADICT.DOC

```
System: SIMUBASE: Data Base Based Simulation
Author: Mr. Abdenacer Moussaoui
03/19/89   19:15:29
Database Structure Summary

--------------------------------------------------------------------------------

5 databases in the system
   &PROJECTD.PCONST
   PROJECTS.DBF
   &PROJECTD.MODULES
   &PROJECTD.RUNTIME
   DBM.DBF
--------------------------------------------------------------------------------

&PROJECTD.PCONST is a macro unknown to FoxDoc


FoxDoc did not find any associated index files

This database appears to be associated with report form(s):
       : CONST.FRM

Used by: TOP.PRG
       : DOREPORT.PRG

--------------------------------------------------------------------------------


Structure for database : PROJECTS.DBF
Number of data records :       3
          Last updated :  03/19/89 at 18:38
Field  Field name  Type        Width    Dec    Start      End
    1  PROJNAME    Character      15              1       15
    2  DIRECTORY   Character      20             16       35
    3  MEMO        Character      30             36       65
** Total **                      66


FoxDoc did not find any associated index files

FoxDoc did not find any associated report forms

Used by: GETPROJ.PRG

--------------------------------------------------------------------------------


&PROJECTD.MODULES is a macro unknown to FoxDoc
       Alias: DBM

This database appears to be associated with index file(s):
       : &PROJECTD.MODULE1.IDX  (index key not found)

This database appears to be associated with report form(s):
       : F1.FRM

Used by: OPENPROJ.PRG
```

&PROJECTD.RUNTIME is a macro unknown to FoxDoc

FoxDoc did not find any associated index files

FoxDoc did not find any associated report forms

Used by: OPENPROJ.PRG

Structure for database : DBM.DBF
Number of data records : 39
Last updated : 03/19/89 at 12:57

Field	Field name	Type	Width	Dec	Start	End
1	FLDNAME	Character	10		1	10
2	FLDTYPE	Character	1		11	11
3	FLDLEN	Numeric	2		12	13
4	FLDDEC	Numeric	2		14	15
5	FUNCNAME	Character	8		16	23
6	CODE	Character	3		24	26
7	METHOD	Character	32		27	58
8	D1	Character	40		59	98
9	D2	Character	40		99	138
10	D3	Character	40		139	178
11	AUTHOR	Character	25		179	203
** Total **			204			

FoxDoc did not find any associated index files

FoxDoc did not find any associated report forms

Used by: SORTMOD.PRG

System: SIMUBASE: Data Base Based Simulation
Author: Mr. Abdenacer Moussaoui
03/19/89 19:15:30
Data Dictionary

Field Name	Type	Len	Dec	Database
AUTHOR	C	25	0	DBM.DBF
CODE	C	3	0	DBM.DBF
D1	C	40	0	DBM.DBF
D2	C	40	0	DBM.DBF
D3	C	40	0	DBM.DBF
DIRECTORY	C	20	0	PROJECTS.DBF
FLDDEC	N	2	0	DBM.DBF
FLDLEN	N	2	0	DBM.DBF
FLDNAME	C	10	0	DBM.DBF
FLDTYPE	C	1	0	DBM.DBF
FUNCNAME	C	8	0	DBM.DBF
MEMO	C	30	0	PROJECTS.DBF
METHOD	C	32	0	DBM.DBF
PROJNAME	C	15	0	PROJECTS.DBF

FILELIST.DOC

```
System: SIMUBASE: Data Base Based Simulation
Author: Mr. Abdenacer Moussaoui
03/19/89   19:15:36
File List

--------------------------------------------------------------------------------

Programs and procedures:
  ADDFIELD.PRG
  ADDITEM.PRG
  ADDSPATH.PRG
  CODEFILE.PRG
  DOCONT              (procedure in MM__PROC.PRG)
  DOGOTO              (procedure in MM__PROC.PRG)
  DOHELP.PRG
  DOLOCATE            (procedure in MM__PROC.PRG)
  DOREPORT.PRG
  GETKEY              (procedure in MM__PROC.PRG)
  GETPROJ.PRG
  GOTOREC             (procedure in MM__PROC.PRG)
  INFORM.PRG
  MEDITOR.PRG
  MENU1.PRG
  MENUHELP.PRG
  MENUREPO.PRG
  MM.PRG
  MM__APPE.PRG
  MM__AREA            (procedure in MM__PROC.PRG)
  MM__EDIT.PRG
  MM__FORM            (procedure in MM__PROC.PRG)
  MM__GETS            (procedure in MM__PROC.PRG)
  MM__KEYS            (procedure in MM__PROC.PRG)
  MM__OPEN.PRG
  MM__REPL            (procedure in MM__PROC.PRG)
  MM__SAYS            (procedure in MM__PROC.PRG)
  MM__SEEK            (procedure in MM__PROC.PRG)
  MM__STOR            (procedure in MM__PROC.PRG)
  MODIDATE.PRG
  OPENPROJ.PRG
  PACK.PRG
  PRINTBEG.PRG
  PRINTEND.PRG
  RUN_BAT.PRG
  RUN_MOD.PRG
  SAYEOF              (procedure in MM__PROC.PRG)
  SAYLINE             (procedure in MM__PROC.PRG)
  SAYREC              (procedure in MM__PROC.PRG)
  SELECTWA.PRG
  SETE.PRG
  SETPCONST.PRG
  SORTMOD.PRG
  STATLINE            (procedure in MM__PROC.PRG)
  TOP.PRG
  YESNO.PRG
  ZAPFIELD.PRG
  ZAPITEM.PRG

Procedure files:
  MM__PROC.PRG

Databases:
  &PROJECTD.MODULES
  &PROJECTD.PCONST
  &PROJECTD.RUNTIME
  DBM.DBF
  PROJECTS.DBF

Index files:
```

```
  &PROJECTD.MODULE1.IDX

Report forms:
  CONST.FRM
  F1.FRM
```

PRCSUMRY.DOC

```
System: SIMUBASE: Data Base Based Simulation
Author: Mr. Abdenacer Moussaoui
03/19/89   19:15:32
Procedure and Function Summary

--------------------------------------------------------------------------------

1 files containing procedure in the system
   MM__PROC.PRG
--------------------------------------------------------------------------------

MM__PROC.PRG -- Last updated:  03/19/89 at 18:27

   Contains: SAYREC
      Called by: DOCONT            (procedure in MM__PROC.PRG)
      Called by: MM__EDIT.PRG
          Calls: STATLINE          (procedure in MM__PROC.PRG)
          Calls: MM__SAYS          (procedure in MM__PROC.PRG)
   Contains: GETKEY
      Called by: DOGOTO            (procedure in MM__PROC.PRG)
      Called by: DOCONT            (procedure in MM__PROC.PRG)
      Called by: MM__EDIT.PRG
      Called by: MM__APPE.PRG
   Contains: STATLINE
      Called by: SAYREC            (procedure in MM__PROC.PRG)
      Called by: MM__EDIT.PRG
      Called by: MM__APPE.PRG
   Contains: SAYEOF
      Called by: DOLOCATE          (procedure in MM__PROC.PRG)
      Called by: DOCONT            (procedure in MM__PROC.PRG)
      Called by: MM__EDIT.PRG
   Contains: SAYLINE
      Called by: DOLOCATE          (procedure in MM__PROC.PRG)
      Called by: DOCONT            (procedure in MM__PROC.PRG)
      Called by: MM__EDIT.PRG
      Called by: MM__APPE.PRG
   Contains: GOTOREC
      Called by: DOGOTO            (procedure in MM__PROC.PRG)
   Contains: DOGOTO
      Called by: MM__EDIT.PRG
          Calls: GETKEY            (procedure in MM__PROC.PRG)
          Calls: GOTOREC           (procedure in MM__PROC.PRG)
   Contains: DOLOCATE
          Calls: SAYLINE           (procedure in MM__PROC.PRG)
          Calls: SAYEOF            (procedure in MM__PROC.PRG)
          Calls: DOCONT            (procedure in MM__PROC.PRG)
   Contains: DOCONT
      Called by: DOLOCATE          (procedure in MM__PROC.PRG)
          Calls: SAYREC            (procedure in MM__PROC.PRG)
          Calls: SAYLINE           (procedure in MM__PROC.PRG)
          Calls: GETKEY            (procedure in MM__PROC.PRG)
          Calls: SAYEOF            (procedure in MM__PROC.PRG)
   Contains: MM__AREA
      Called by: MM__OPEN.PRG
   Contains: MM__SEEK
      Called by: MM__EDIT.PRG
   Contains: MM__KEYS
      Called by: MM__EDIT.PRG
      Called by: MM__APPE.PRG
   Contains: MM__FORM
      Called by: MM__EDIT.PRG
```

```
      Called by: MM__APPE.PRG
   Contains: MM__SAYS
      Called by: SAYREC            (procedure in MM__PROC.PRG)
      Called by: MM__APPE.PRG
   Contains: MM__GETS
      Called by: MM__EDIT.PRG
      Called by: MM__APPE.PRG
   Contains: MM__STOR
      Called by: MM__EDIT.PRG
      Called by: MM__APPE.PRG
   Contains: MM__REPL
      Called by: MM__EDIT.PRG
      Called by: MM__APPE.PRG
```

APPENDIX B

TOP.PRG

```
*:*********************************************************************
*:
*:        Program: TOP.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     17:09
*:
*:          Calls: SETE.PRG
*:               : ADDSPATH.PRG
*:               : SETPCONST.PRG
*:               : GETPROJ.PRG
*:               : OPENPROJ.PRG
*:               : MENU1.PRG
*:               : MM.PRG
*:               : MEDITOR.PRG
*:               : ADDITEM.PRG
*:               : RUN_MOD.PRG
*:               : RUN_BAT.PRG
*:               : SELECTWA.PRG
*:               : SORTMOD.PRG
*:               : PACK.PRG
*:               : DOREPORT.PRG
*:               : DOHELP.PRG
*:
*:           Uses: &PROJECTD.PCONST
*:
*:     Documented: 03/19/89 at 19:14              FoxDoc version 1.0
*:*********************************************************************

   *_____________________________________________________
   * Created on 23:16:11  1/30/1988 by Abdenacer
   * Updated on 20:39:28  1/31/1988
   * Updated on
   * Updated on
   * Updated on
   *_____________________________________________________

   DO SETE

   INITSPATH = CRTSPATH()

   CRTPATH = CRTDRIVE() + CRTDIR()
   DO ADDSPATH WITH CRTPATH

   DO SETPCONST WITH 'SYSCONST'        && init system constants

   PROJECTD = ''
   DO GETPROJ
   DO SETPCONST WITH PROJECTD+'PCONST'  && init PROJECT constants
   DO OPENPROJ                         && open project databases
   * DO OPENDB                          && open system databases


 +=DO WHILE .T.
 |    OPTION='?'
```

```
|    DO MENU1 WITH OPTION
|
|  +=DO CASE
|  +=CASE OPTION='1'                          && quick module manager
|  |    SELECT DBM
|  |    BROWSE LOCK 1
|  |    SELECT RUNTIME
|  |
|  +=CASE OPTION='M'                         && module manager
|  |    DO MM
|  |
|  +=CASE OPTION='E'                          && edit a module
|  |    && DO LOCATE_E
|  |  +=DO WHILE .T.
|  |  |    SELECT DBM                                && ALLOW PICKING
|  |  |    BROWSE FIELDS FLDNAME, METHOD, FUNCNAME, CODE
|  |  |  +-IF ESCAPED()
|  |  v=======EXIT
|  |  |  +-ENDIF
|  |  |    DO MEDITOR WITH PROJECTD + FUNCNAME
|  |  +=ENDDO
|  |    CLEAR PROGRAM                          && have to be explicit at run-time
|  |
|  |
|  +=CASE OPTION='D'                           && run a module
|  |    SELECT DBM                               && ALLOW PICKING
|  |    SET FILTER TO CODE  'BAT'
|  |    KEYBOARD CHR(5) + CHR(24)
|  |    BROWSE FIELDS FLDNAME, METHOD, FUNCNAME, CODE
|  |    SET FILTER TO
|  |  +-IF ESCAPED()
^==========LOOP
|  |  +-ENDIF
|  |  +=DO CASE
|  |  +=CASE CODE = 'FUN' .OR. CODE = 'KEY'
|  |  |    DO ADDITEM
|  |  |    OLDSPATH = CRTSPATH()
|  |  |    DO ADDSPATH WITH PROJECTD
|  |  |    DO RUN_MOD
|  |  |    SET PATH TO &OLDSPATH
|  |  |
|  |  +=OTHERWISE
|  |  |    ? CHR(7) + 'UNKNOW CODE:' + CODE
|  |  |    WAIT
|  |  +=ENDCASE
|  |
|  +=CASE OPTION='S'                         && run a batch of modules
|  |    SELECT DBM
|  |    GO TOP
|  |    SET FILTER TO CODE = 'BAT'
|  |    BROWSE FIELDS FLDNAME, FUNCNAME, METHOD, CODE
|  |    SET FILTER TO
|  |  +-IF ESCAPED()
^==========LOOP
|  |  +-ENDIF
|  |    OLDSPATH = CRTSPATH()
|  |    DO ADDSPATH WITH PROJECTD
|  |    DO RUN_BAT
|  |    SET PATH TO &OLDSPATH
|  |
|  +=CASE  OPTION='C'                        && modify constants
|  |    DO SELECTWA
|  |    USE &PROJECTD.PCONST
|  |    GO TOP
|  |    BROWSE
|  |    USE
|  |    DO SETPCONST WITH PROJECTD+'PCONST'
|  |
|  +=CASE  OPTION='7'                        && sort modules
|  |    DO SORTMOD
|  |
|  +=CASE  OPTION='P'                        && peek at runtime database
```

```
 |  |    SELECT RUNTIME
 |  |    GO TOP
 |  |    BROWSE
 |  |    && zap deleted modules
 |  +=CASE  OPTION='Z'
 |  |    DO PACK
 |  |
 |  +=CASE  OPTION='R'
 |  |    DO DOREPORT
 |  |    ** open project
 |  +=CASE OPTION='O'
 |  |    DO GETPROJ
 |  |  +-IF ESCAPED()
^==========LOOP
 |  |  +-ENDIF
 |  |    DO SETPCONST WITH PROJECTD+'PCONST'   && re-init project constant
 |  |    DO OPENPROJ
 |  |
 |  +=CASE  OPTION='H'
 |  |    DO DOHELP
 |  |
 |  +=CASE  OPTION='0'
 |  |    SET PATH TO &INITSPATH
v=======EXIT
 |  |
 |  +=OTHERWISE
^=======LOOP
 |  +=ENDCASE
 |
+=ENDDO
  *: EOF: TOP.ACT
```

DOHELP.PRG

```
*:*****************************************************************
*:
*:        Program: DOHELP.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     17:54
*:
*:      Called by: TOP.PRG
*:
*:          Calls: MENUHELP.PRG
*:
*:     Documented: 03/19/89 at 19:15               FoxDoc version 1.0
*:*****************************************************************

   ** DOHELP.PRG
   ** Created on 03/19/89 Time:10:44:10
   ** Updated on
   **
 +=DO WHILE .T.
 |    OPTION = '?'
 |    DO MENUHELP WITH OPTION
 |
 |  +=DO CASE
 |  +=CASE OPTION = 'H'
 |  |    HELP
 |  |
 |  +=CASE OPTION = 'E'
 |  |    HELP LIST
 |  +=CASE OPTION = 'F'
 |  |    HELP DISP
 |  +=CASE OPTION = 'M'
 |  |    HELP COPY
 |  +=CASE OPTION = 'N'
 |  |    HELP APPEND
```

```
 |  |
 |  |
 |  +=CASE OPTION = '0'
 v=======EXIT
 |  +=ENDCASE
 |
 +=ENDDO
   *: EOF: DOHELP.ACT
```

DOREPORT.PRG

```
*:*****************************************************************
*:
*:        Program: DOREPORT.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     17:33
*:
*:      Called by: TOP.PRG
*:
*:          Calls: MENUREPO.PRG
*:               : PRINTBEG.PRG
*:               : PRINTEND.PRG
*:               : SELECTWA.PRG
*:
*:           Uses: &PROJECTD.PCONST
*:
*:   Report Forms: F1.FRM
*:               : CONST.FRM
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************

   ** doreport.PRG
   ** Created on 03/19/89 Time:17:00:15
   ** Updated on
   **


   OPTION = '?'

 +=DO WHILE .T.
 |    DO MENUREPO  WITH OPTION
 |  +=DO CASE
 |  +=CASE OPTION = 'M'
 |  |    SELECT DBM
 |  |    DO PRINTBEG
 |  |    REPORT FORM F1 FOR CODE  'BAT' HEADING 'MODULE(s)'
 |  |    DO PRINTEND
 |  |
 |  +=CASE OPTION = 'S'
 |  |    SELECT DBM
 |  |    DO PRINTBEG
 |  |    REPORT FORM F1 FOR CODE = 'BAT' HEADING 'SIMULATION SCRIPT(s)'
 |  |    DO PRINTEND
 |  |
 |  +=CASE OPTION = 'C'
 |  |    DO SELECTWA
 |  |    USE &PROJECTD.PCONST
 |  |    DO PRINTBEG
 |  |    REPORT FORM CONST
 |  |    DO PRINTEND
 |  |    USE
 |  |
 |  +=CASE OPTION = '0'
 v=======EXIT
 |  +=ENDCASE
 |
```

```
  +=ENDDO

    *: EOF: DOREPORT.ACT
```

MENU1.PRG

```
*:*****************************************************************
*:
*:        Program: MENU1.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     18:46
*:
*:      Called by: TOP.PRG
*:
*:     Documented: 03/19/89 at 19:14              FoxDoc version 1.0
*:*****************************************************************
    * Created on 23:55:30  1/30/1988 by Abdenacer
    * Updated on
    * Updated on
    * Updated on
    * MAIN MENU
    PARAMETER OPTION

    CLEAR
Text
==
                M A I N   M E N U

      M- Module Manager                        O-open a project

      E- Edit Modules or Scripts code          H-system help

      S- Run a Simulation Script (batch of modules)
      D- Run a module (for debugging)

      P- Peek at the data-items runtime database
      C- Modify user/system constants
      7- Sort modules database                 Z-zap deleted modules
      R- Report generator
      0- to quit


==
                          Enter your option
ENDTEXT

    @@ ROW(), COL()+2 GET OPTION PICTURE '! '          && RANGE 0,9
    READ

    *: EOF: MENU1.ACT
```

MENUHELP.PRG

```
*:*****************************************************************
*:
*:        Program: MENUHELP.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     17:53
*:
*:      Called by: DOHELP.PRG
```

```
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************

    ** MENUHELP.PRG
    ** Created on 03/19/89 Time:10:45:22
    ** Updated on
    **
    PARAMETER OPTION

    CLEAR
Text
==
                    H E L P   M E N U
  Date  function                    Database update
     1- DTOC                          A- REPLACE
      2- CTOD                          B-
      3- DOW                           C-
      4- MONTH
      5- DATE()                      Reporting commands
      6- TIME()                        E- LIST
                                       F- DISPLAY
     Export/Import
      M- COPY TO
      N- APPEND FROM

      H- ALPHABETIC INDEX
      0- EXIT


==
                            Enter your option
ENDTEXT

    @@ ROW(), COL()+2 GET OPTION PICTURE '! '
    READ

    *: EOF: MENUHELP.ACT
```

MENUREPO.PRG

```
*:*****************************************************************
*:
*:        Program: MENUREPO.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     17:19
*:
*:      Called by: DOREPORT.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************

    ** MENUREPO.PRG
    ** Created on 03/19/89 Time:17:16:21
    ** Updated on
    **

    PARAMETER OPTION

    CLEAR
Text
==
                    R E P O R T   M E N U


      M- Modules listing
```

```
  STORE METHOD      TO MMETHOD
  STORE AUTHOR      TO MAUTHOR
  STORE D1          TO MD1
  STORE D2          TO MD2
  STORE D3          TO MD3
  STORE FLDTYPE     TO MFLDTYPE
  STORE FLDLEN      TO MFLDLEN
  STORE FLDDEC      TO MFLDDEC
  RETURN


  PROCEDURE MM__REPL
  * ---Using MODULES.DBF
+-IF .NOT. EOF()
|    * ---Replace only if there is an available record.
|    REPLACE;
|    FLDNAME    WITH MFLDNAME,;
|    FUNCNAME   WITH MFUNCNAME,;
|    CODE       WITH MCODE,;
|    METHOD     WITH MMETHOD,;
|    AUTHOR     WITH MAUTHOR,;
|    D1         WITH MD1,;
|    D2         WITH MD2
|    REPLACE;
|    D3         WITH MD3,;
|    FLDTYPE    WITH MFLDTYPE,;
|    FLDLEN     WITH MFLDLEN,;
|    FLDDEC     WITH MFLDDEC
+-ENDIF
  RETURN

  * EOF: MM__PROC.PRG
  *: EOF: MM__PROC.ACT
```

```
@@  0, 0 SAY "Record:"
@@  0,72 SAY DATE()
SET COLOR TO &PROMPTATR
@@ PROMPTROW-1,0 SAY PROMPTBAR
*
SET COLOR TO R/N
@@ 1,0,21,79 BOX ""
@@  2, 3 SAY "Module/Script Master Name "
@@  2,30 SAY "Code File "
@@  2,45 SAY "Code Type "
@@  2,57 SAY "Creation Date    "
@@  5,57 SAY "Modification Date    "
@@  8,57 SAY "Modification Time"
@@  6, 3 SAY "Short Description "
@@  9, 3 SAY "Author   "
@@ 12,48 SAY "Type Specifications"
@@ 13,47,19,77 BOX ""
@@ 14, 5 SAY "Full description"
@@ 15, 2 SAY "D1 "
@@ 16, 2 SAY "D2 "
@@ 17, 2 SAY "D3 "
@@ 14,52 SAY "Field Type "
@@ 16,51 SAY "Field Width "
@@ 18,49 SAY "Field Decimal "
RETURN


PROCEDURE MM__SAYS
* ---Using MODULES.DBF
SET COLOR TO ,N/W
@@  3, 3 GET FLDNAME    PICTURE "@!"
@@  3,30 GET FUNCNAME   PICTURE "@!"
@@  3,45 GET CODE       PICTURE "@!"
SET COLOR TO N/W
@@  3,57 SAY CREA_DATE
@@  6,57 SAY MODI_DATE
@@  9,57 SAY MODI_TIME
@@  7, 3 GET METHOD
@@ 10, 3 GET AUTHOR
@@ 15, 5 GET D1
@@ 16, 5 GET D2
@@ 17, 5 GET D3
@@ 14,63 GET FLDTYPE
@@ 16,63 GET FLDLEN     PICTURE "99"
@@ 18,63 GET FLDDEC     PICTURE "99"
CLEAR GETS
RETURN


PROCEDURE MM__GETS
* ---Using MODULES.DBF
SET COLOR TO ,N/W
@@  3,45 GET MCODE      PICTURE "@@!" VALID(MCODE = 'FUN' .OR. CODE = 'BAT' .OR. CODE =
'KEY')
@@  7, 3 GET MMETHOD    VALID(LEN(TRIM(MMETHOD))0)
@@ 10, 3 GET MAUTHOR
@@ 15, 5 GET MD1
@@ 16, 5 GET MD2
@@ 17, 5 GET MD3
@@ 14,63 GET MFLDTYPE
@@ 16,63 GET MFLDLEN    PICTURE "99" RANGE 1,15
@@ 18,63 GET MFLDDEC    PICTURE "99" RANGE 0,10 VALID(MFLDDEC  MFLDLEN)
READ
RETURN


PROCEDURE MM__STOR
* ---Using MODULES.DBF
* ---Initialize memvars with field contents.
STORE FLDNAME    TO MFLDNAME
STORE FUNCNAME   TO MFUNCNAME
STORE CODE       TO MCODE
```

```
+=DO WHILE CHOICE = "Y" .AND. .NOT. EOF()
|     OLDRECNUM = RECNO()
|     DO SAYREC
|     DO SAYLINE WITH ROW+1,"Continue? (y/n)"
|     DO GETKEY WITH CHOICE,"YN"+RETURNKEY
|     @ ROW+1,0 CLEAR
|  +-IF CHOICE = "Y"
|  |     CONTINUE
|  +-ENDIF
+=ENDDO
+-IF EOF()
|     DO SAYEOF WITH ROW,OLDRECNUM
+-ENDIF
  RETURN

  PROCEDURE MM__AREA
  SELECT &DBFAREA
  DBFNAME = "MODULES.DBF"
  DBFTEMP = "MODULES$.DBF"
  NDXNAM1 = "MODULES1.IDX"
  NDXKEY1 = "FLDNAME+FUNCNAME"
  NDXORDER = "1"
  LASTREC = RECCOUNT()
  RETURN

  PROCEDURE MM__SEEK
  PARAMETER ROW
  PRIVATE EXPR
+-IF NDXORDER = "0"

+-ENDIF
  SET COLOR TO &PROMPTATR
  @@ ROW,0 CLEAR
+=DO CASE
+=CASE NDXORDER = "1"
|     MFLDNAME = SPACE(10)
|     MFUNCNAME = SPACE(8)
|     @ ROW,  0 SAY "Enter Fldname" GET MFLDNAME PICTURE "@!"
|     @ ROW+1,0 SAY "      Funcname" GET MFUNCNAME PICTURE "@!"
|     READ
|     EXPR = TRIM( MFLDNAME+MFUNCNAME )
|  +-IF ""  EXPR
|  |     SEEK EXPR
|  +-ENDIF
+=ENDCASE
  RETURN

  PROCEDURE MM__KEYS
  PARAMETER EXPR,ISBLANK,ISUNIQUE
  EXPR = ""
  ISBLANK = .F.
  ISUNIQUE = .F.
  SET COLOR TO ,N/W
  @@  3, 3 GET MFLDNAME   PICTURE "@!" VALID(LEN(TRIM(MFLDNAME))0)
  @@  3,30 GET MFUNCNAME  PICTURE "@!" VALID(LEN(TRIM(MFUNCNAME))0)
  READ
  ISBLANK = ("" = TRIM( MFLDNAME ))
  ISBLANK = ISBLANK .AND. ("" = TRIM( MFUNCNAME ))
  EXPR = MFLDNAME+MFUNCNAME
  ISUNIQUE = .T.
  RETURN

  PROCEDURE MM__FORM
  SET COLOR TO &SCREENATR
  CLEAR
  SET COLOR TO &STATUSATR
  @@  0, 0 SAY SPACE(80)
```

```
  PROCEDURE SAYLINE
  PARAMETER ROW,STRG
  SET COLOR TO &PROMPTATR
  @@ ROW,0 CLEAR
  @@ ROW,0 SAY STRG
  RETURN


  PROCEDURE GOTOREC
  PARAMETER ROW,RECNUM,LASTRECNUM
  RECNUM = 0
  SET COLOR TO &PROMPTATR
  @@ ROW,0 CLEAR
  @@ ROW+1,17 SAY "{ 1 to "
  @@ ROW+1,24 SAY SUBSTR( STR( LASTRECNUM + 1000000,7 ),2 ) + " } + {Return}"
  @@ ROW,0 SAY "Enter RECORD number" GET RECNUM;
  PICTURE "@Z 9999999" RANGE 0,LASTRECNUM
  READ
  @@ ROW,0 CLEAR
+-IF RECNUM  0
|    GOTO RECNUM
+-ENDIF
  RETURN


  PROCEDURE DOGOTO
  PARAMETER ROW,RECNUM,LASTRECNUM
  RECNUM = 0
  SET COLOR TO &PROMPTATR
  @@ ROW,0 CLEAR
  @@ ROW,0 SAY "GOTO:  {T}op  {B}ottom  {R}ecord#  {Return} "
  DO GETKEY WITH CHOICE,"TBR"+RETURNKEY
  @@ ROW,0 CLEAR
+=DO CASE
+=CASE CHOICE = RETURNKEY


+=CASE CHOICE = "T"
|    GOTO TOP
|    RECNUM = RECNO()
+=CASE CHOICE = "B"
|    GOTO BOTTOM
|    RECNUM = RECNO()
+=CASE CHOICE = "R"
|    DO GOTOREC WITH ROW,RECNUM,LASTRECNUM
+=ENDCASE
  RETURN


  PROCEDURE DOLOCATE
  PARAMETER ROW,EXPR
  PRIVATE OLDRECNUM
  OLDRECNUM = RECNO()
  DO SAYLINE WITH ROW,"Locating..."
  LOCATE FOR &EXPR
+-IF EOF()
|    DO SAYEOF WITH ROW,OLDRECNUM
+-ELSE
|    @ ROW,0 CLEAR
|    @ ROW,0 SAY "LOCATE FOR" GET EXPR
|    CLEAR GETS
|    DO DOCONT WITH ROW
+-ENDIF
  RETURN


  PROCEDURE DOCONT
  PARAMETER ROW
  PRIVATE OLDRECNUM
  CHOICE = "Y"
```

```
  * Version.: FoxBASE+, revision 2.10
  * Notes...: PROCEDURE file for MODULES.DBF
  *

  PROCEDURE SAYREC
  * ---"SayRec" is used by the EDIT program and PROCEDURE DoCONT.
  *
  DO STATLINE WITH RECNO(),DELETED()
  DO MM__SAYS
  *
  * ---If you are calling "SayRec" from more than one
  * ---application, you may wish to replace the above
  * ---line with a DO CASE structure, as follows:
  *
  *    * ---"appnum" is the application ID number.
  *    DO CASE
  *    CASE appnum = 1
  *       DO AP1_SAYS
  *    CASE appnum = 2
  *       DO AP2_SAYS
  *    ENDCASE
  *
  RETURN


  PROCEDURE GETKEY
  PARAMETER CHOICE,KEYCHARS
  PRIVATE KEYCODE
  CHOICE = "*"
+=DO WHILE .NOT. (CHOICE $ KEYCHARS)
|    KEYCODE = INKEY()
|  +-IF KEYCODE  0
|  |    CHOICE = UPPER(CHR(KEYCODE))
|  +-ENDIF
|    * ---A keyfilter can be implemented here, as follows:
|    *
|    *    * ---FROM:  {F1}  ^leftarrow  ^rightarrow
|    *    * ---INTO:  "H"    leftarrow   rightarrow
|    *    fromkeys = CHR(28) + CHR(26) + CHR(2)
|    *    intokeys = "H" + CHR(19) + CHR(4)
|    *    choice = SUBSTR( "*"+intokeys,AT(choice,fromkeys) + 1,1 )
+=ENDDO
  RETURN


  PROCEDURE STATLINE
  PARAMETER RECNUM,ISDELETED
  SET COLOR TO &STATUSATR
  @@ 0, 8 SAY SUBSTR( STR( RECNUM + 1000000,7 ),2 )
+-IF ISDELETED
|    @ 0,50 SAY "*DELETED*"
+-ELSE
|    @ 0,50 SAY "         "
+-ENDIF
  RETURN


  PROCEDURE SAYEOF
  PARAMETER ROW,OLDRECNUM
  SET COLOR TO &PROMPTATR
  @@ ROW,0 CLEAR
+-IF EOF()
|    @ ROW,0 SAY "END-OF-FILE encountered"
+-ELSE
|    @ ROW,0 SAY "BEGINNING-OF-FILE encountered"
+-ENDIF
  WAIT
  @@ ROW,0 CLEAR
+-IF OLDRECNUM  0
|    GOTO OLDRECNUM
+-ENDIF
  RETURN
```

```
*:*********************************************************************
    * Program.: MM__OPEN.PRG
    * Author..: Mr. Abdenacer Moussaoui
    * Date....: 11/28/88
    * Notice..: Copyright (c) 1988, MTS
    * Version.: FoxBASE+, revision 2.10
    * Notes...: OPEN program for MODULES.DBF
    *
    * ---INKEY() constant values.
    PGDN = CHR(3)
    PGUP = CHR(18)
    RETURNKEY = CHR(13)
    DELRECORD = CHR(7)
    *
    * ---SET COLOR TO values.
    SCREENATR = "R+/N,N/W"
    STATUSATR = "GU/N,N/W"
    WINDOWATR = "R+/N,N/W"
    PROMPTATR = "GR+/N,N/W"
    HILITEATR = "N/W"
    *
    * ---Initialize global variables.
    STORE 0 TO LASTREC,OLDRECNUM,RECNUM,MENUCHOICE
    STORE " " TO CHOICE,EXPR
    PROMPTROW = 23
    PROMPTBAR = REPLICATE( CHR( 196 ),80 )
    *

    * ---Initialize database variables for current workarea.
    DO MM__AREA
    *
    RETURN
    * EOF: MM__OPEN.PRG
    *: EOF: MM__OPEN.ACT
```

MM__PROC.PRG

```
*:*********************************************************************
*:
*: Procedure file: MM__PROC.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     18:27
*:
*:  Procs & Fncts: SAYREC
*:               : GETKEY
*:               : STATLINE
*:               : SAYEOF
*:               : SAYLINE
*:               : GOTOREC
*:               : DOGOTO
*:               : DOLOCATE
*:               : DOCONT
*:               : MM__AREA
*:               : MM__SEEK
*:               : MM__KEYS
*:               : MM__FORM
*:               : MM__SAYS
*:               : MM__GETS
*:               : MM__STOR
*:               : MM__REPL
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*********************************************************************
    * Program.: MM__PROC.PRG
    * Author..: Mr. Abdenacer Moussaoui
    * Date....: 03/19/89
    * Notice..: Copyright (c) 1989, MTS
```

```
 |  |     OLDRECNUM = RECNO()
 |  |     SKIP
 |  |  +-IF EOF()
 |  |  |     DO SAYEOF WITH ROW,OLDRECNUM
 |  |  +-ELSE
 |  |  |     DO SAYREC
 |  |  +-ENDIF
 |  |
 |  +=CASE EDITCHOICE = "P"
 |  |     * ---Previous record.
 |  |     OLDRECNUM = RECNO()
 |  |     SKIP -1
 |  |  +-IF BOF()
 |  |  |     DO SAYEOF WITH ROW,OLDRECNUM
 |  |  +-ELSE
 |  |  |     DO SAYREC
 |  |  +-ENDIF
 |  |
 |  +=CASE EDITCHOICE = "E"
 |  |     * ---Edit the record.
 |  |     ISEDITED = .T.
 |  |     DO MM__STOR
 |  |     DO SAYLINE WITH ROW,"Press {Ctrl-W} to Exit"
 |  |     * ---If you don't want the user to edit the
 |  |     * ---key fields, then delete the following line.
 |  |     DO MM__KEYS WITH EXPR,ISBLANK,ISUNIQUE
 |  |     DO MM__GETS
 |  |     DO MM__REPL
 |  |
 |  |     DO ZAPITEM
 |  |     DO ADDITEM
 |  |     SELECT DBM
 |  |
 |  +=CASE EDITCHOICE = "G"
 |  |     * ---Goto a record.
 |  |     DO DOGOTO WITH ROW,RECNUM,LASTREC
 |  |  +-IF RECNUM  0
 |  |  |     DO SAYREC
 |  |  +-ENDIF
 |  +=CASE EDITCHOICE = DELRECORD
 |  |     * ---Delete the record.
 |  |     ISEDITED = .T.
 |  |  +-IF DELETED()
 |  |  |     RECALL
 |  |  +-ELSE
 |  |  |     DELETE
 |  |  +-ENDIF
 |  |     DO STATLINE WITH RECNO(),DELETED()
 |  +=ENDCASE
 +=ENDDO
  RETURN
  * EOF: MM__EDIT.PRG
  *: EOF: MM__EDIT.ACT
```

MM__OPEN.PRG

```
*:*********************************************************************
*:
*:        Program: MM__OPEN.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     15:07
*:
*:      Called by: MM.PRG
*:
*:          Calls: MM__AREA         (procedure in MM__PROC.PRG)
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
```

```
*:                : MM__GETS          (procedure in MM__PROC.PRG)
*:                : MM__REPL          (procedure in MM__PROC.PRG)
*:                : ZAPITEM.PRG
*:                : ADDITEM.PRG
*:                : DOGOTO            (procedure in MM__PROC.PRG)
*:                : STATLINE          (procedure in MM__PROC.PRG)
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*********************************************************************
    * Program.: MM__EDIT.PRG
    * Author..: Mr. Abdenacer Moussaoui
    * Date....: 11/28/88
    * Notice..: Copyright (c) 1988, MTS
    * Version.: FoxBASE+, revision 2.10
    * Notes...: EDIT program for MODULES.DBF
    *
    PARAMETER ISEDITED
    PRIVATE ROW,LASTPAGE,EDITCHOICE,NDXCHOICE
    PRIVATE ISBLANK,ISUNIQUE
    ROW = PROMPTROW
    EXPR = ""
    STORE .F. TO ISEDITED,ISBLANK,ISUNIQUE
    DO MM__FORM
    DO SAYREC
    EDITCHOICE = "*"
    NDXCHOICE = "*"
    * ---Loop until {Return} is pressed.
    * ---The following loop is really a "REPEAT/UNTIL  ".
  +=DO WHILE .T.
  |    SET COLOR TO &PROMPTATR
  | +-IF .NOT. (EDITCHOICE $ "NP"+DELRECORD)
  | |    @ ROW,0 CLEAR
  | +-ENDIF
  |    @ ROW+1,12 SAY "{N}ext-record  {P}rev-record  {M}odule code   0,.
=Exit"
  |    @ ROW,0 SAY "EDIT/VIEW:  {E}dit  {F}ind  {G}oto  {A}ppend    {M}odify source"
  |    DO GETKEY WITH EDITCHOICE,"EFGLNPMA0"+DELRECORD+RETURNKEY
  | +=DO CASE
  | |
  | +=CASE .NOT. (EDITCHOICE $ "AH") .AND. (LASTREC = 0)
  | |    @ 17, 0 SAY "EMPTY DATABASE:  Only Append and Help are available."
  | |    WAIT
  | |
  | +=CASE EDITCHOICE = 'A'
  | |    DO MM__APPE
  | |    DO MM__FORM
  | |    DO SAYREC
  | |
  | +=CASE EDITCHOICE = 'M'
  | |    DO MEDITOR WITH PROJECTD+FUNCNAME
  | |    DO MODIDATE
  | |    DO MM__FORM
  | |    DO SAYREC
  | |
  | |
  | +=CASE EDITCHOICE = RETURNKEY .OR. EDITCHOICE = '0'
  v=======EXIT
  | |
  | +=CASE EDITCHOICE = "F"
  | |    * ---Find a record.
  | |    OLDRECNUM = RECNO()
  | |    DO MM__SEEK WITH ROW
  | | +-IF EOF()
  | | |    DO SAYLINE WITH ROW,"No find."
  | | |    WAIT
  | | |    GOTO OLDRECNUM
  | | +-ELSE
  | | |    DO SAYREC
  | | +-ENDIF
  | |
  | +=CASE EDITCHOICE = "N"
  | |    * ---Next record.
```

```
   | |    "APPEND:
:add-another  {C}arry-add  {E}dit  {F}inished  "
   | |     DO GETKEY WITH CHOICE,"CEFM"+DELRECORD+RETURNKEY
   | |  +=DO CASE
   | |  +=CASE CHOICE = DELRECORD
   | | |    * ---Toggle IsDeleted flag.
   | | |    ISDELETED = .NOT. ISDELETED
   | | |    DO STATLINE WITH LASTREC+RECNUMOFS,ISDELETED
   | | |
   | |  +=CASE CHOICE = "E"
   | | |    * ---Re-edit the record.
   | | |    ISDELETED = .F.
   | | |
   | |  +=CASE CHOICE $ "CF"+RETURNKEY
   | | |    * ---Finished, Add-another, or Carry-add.
   | | |    ISCARRY = (CHOICE = "C")
   | | |  +-IF ISDELETED
   | | | |    * ---Reset offset so as not to increment.
   | | | |    RECNUMOFS = RECNUMOFS - 1
   | | |  +-ELSE
   | | | |    * ---Save the memvar values.
   | | | |    APPEND BLANK
   | | | |    DO MM__REPL
   | | | |    REPLACE CREA_DATE WITH DATE()
   | | | |    DO CODEFILE
   | | |  +-ENDIF
   | |  +=ENDCASE
   | |
   | |    * ---Condition to exit inner loop.
   | |  +-IF CHOICE $ "CEF"+RETURNKEY
   |  v=======EXIT
   | |  +-ENDIF
   |  +=ENDDO
   |
   |    * ---Condition to exit outer loop.
   |  +-IF CHOICE = "F"
   v=======EXIT
   |  +-ENDIF
   +=ENDDO

    LASTREC = LASTREC + RECNUMOFS
    * GOTO TOP
    RETURN
    * EOF: MM__APPE.PRG
    *: EOF: MM__APPE.ACT
```

MM__EDIT.PRG

```
*:*****************************************************************
*:
*:        Program: MM__EDIT.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     19:13
*:
*:      Called by: MM.PRG
*:
*:          Calls: MM__FORM          (procedure in MM__PROC.PRG)
*:               : SAYREC            (procedure in MM__PROC.PRG)
*:               : GETKEY            (procedure in MM__PROC.PRG)
*:               : MM__APPE.PRG
*:               : MEDITOR.PRG
*:               : MODIDATE.PRG
*:               : MM__SEEK          (procedure in MM__PROC.PRG)
*:               : SAYLINE           (procedure in MM__PROC.PRG)
*:               : SAYEOF            (procedure in MM__PROC.PRG)
*:               : MM__STOR          (procedure in MM__PROC.PRG)
*:               : MM__KEYS          (procedure in MM__PROC.PRG)
```

```
| |     * ---Add another record.
| |     RECNUMOFS = RECNUMOFS + 1
| |  +-IF .NOT. ISCARRY
| |  |     * ---Initialize memory variables with blanks.
| |  |     GOTO BOTTOM
| |  |  +-IF .NOT. EOF()
| |  |  |    SKIP
| |  |  +-ENDIF
| |  |     DO MM__SAYS
| |  |     DO MM__STOR
| |  |     GOTO BOTTOM
| |  |     * ---Initialize fields/memvars.
| |  |     STORE "N" TO MFLDTYPE
| |  |     STORE 8 TO MFLDLEN
| |  |     STORE 2 TO MFLDDEC
| |  +-ENDIF
| |     ISCARRY = .F.
| +-ENDIF
|
|   DO STATLINE WITH LASTREC+RECNUMOFS,ISDELETED
|   @ 0,50 SAY "*BLANK*  "
|   * ---Check for duplicate record.
| +=DO WHILE .T.
| |     DO SAYLINE WITH ROW,"Press {Ctrl-W} to Exit"
| |     * ---Enter key field values.
| |     DO MM__KEYS WITH EXPR,ISBLANK,ISUNIQUE
| |  +-IF ISBLANK .OR. .NOT. ISUNIQUE
| v=======EXIT
| |  +-ENDIF
| |     * ---Check for duplicate key in master file.
| |     SEEK EXPR
| |  +-IF EOF()
| |  |     * ---No duplicate key found, so leave.
| v=======EXIT
| |  +-ELSE
| |  |     * ---Found a duplicate record in the file.
| |  |     SET COLOR TO &STATUSATR
| |  |     @ 0,50 SAY "*DELETED*"
| |  |     DO SAYLINE WITH ROW,;
| |  |     "DUPLICATE KEY encountered.  Record cannot be appended."
| |  |     ANS = 'Y'
| |  |     DO YESNO WITH ANS, 'Blank it?'
| |  |  +-IF ANS = 'Y'
| |  |  |     MFLDNAME = SPACE(8)
| |  |  |     MFUNCNAME = SPACE(8)
| |  |  +-ENDIF
| |  +-ENDIF
| +=ENDDO
|
| +-IF ISBLANK
| |     ISDELETED = .T.
| +-ELSE
| |     DO MM__GETS
| +-ENDIF
|   DO STATLINE WITH LASTREC+RECNUMOFS,ISDELETED
|
|   * ---Loop until Add, Carry, Edit, or Finished is selected.
|   * ---The following loop is really a "REPEAT/UNTIL  ".
| +=DO WHILE .T.
| |     * ---You can add other prompts and options in this inner loop.
| |     * ---For example, to add an invoicing routine:
| |     *
| |     *    (1) Insert "{I}nvoice" in the prompt line below,
| |     *    (2) Include "I" in the values for GetKey, and
| |     *    (3) Add a CASE to the DO CASE structure, such as:
| |     *
| |     *        CASE choice = "I"
|  |    *           DO
gram_name
| |     *
| |     DO SAYLINE WITH ROW,;
```

```
PRIVATE LASTREC,RECNUM,OLDRECNUM,EXPR,ISVALID
STORE .F. TO LASTREC,RECNUM,OLDRECNUM,EXPR,ISVALID
* ---Declare field memory variables.
PRIVATE;
MFLDNAME,MFUNCNAME,MCODE,MMETHOD,MAUTHOR,;
MFLDTYPE,MD1,MD2,MFLDLEN,MD3,;
MFLDDEC
STORE " " TO;
MFLDNAME,MFUNCNAME,MCODE,MMETHOD,MAUTHOR,;
MFLDTYPE,MD1,MD2,MD3
STORE 0.00 TO;
MFLDLEN,MFLDDEC
*
* ---Initialize Global memory variables and OPEN file(s).
DBFAREA = "1"
DO MM__OPEN
*

* ---DO EDIT/VIEW.
ISEDITED = .F.
DO MM__EDIT WITH ISEDITED
*: EOF: MM.ACT
```

MM__APPE.PRG

```
*:*****************************************************************
*:
*:        Program: MM__APPE.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     18:38
*:
*:      Called by: MM__EDIT.PRG
*:
*:          Calls: MM__FORM         (procedure in MM__PROC.PRG)
*:               : MM__SAYS         (procedure in MM__PROC.PRG)
*:               : MM__STOR         (procedure in MM__PROC.PRG)
*:               : STATLINE         (procedure in MM__PROC.PRG)
*:               : SAYLINE          (procedure in MM__PROC.PRG)
*:               : MM__KEYS         (procedure in MM__PROC.PRG)
*:               : YESNO.PRG
*:               : MM__GETS         (procedure in MM__PROC.PRG)
*:               : GETKEY           (procedure in MM__PROC.PRG)
*:               : MM__REPL         (procedure in MM__PROC.PRG)
*:               : CODEFILE.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************
    * Program.: MM__APPE.PRG
    * Author..: Mr. Abdenacer Moussaoui
    * Date....: 11/28/88
    * Notice..: Copyright (c) 1988, MTS
    * Version.: FoxBASE+, revision 2.10
    * Notes...: APPEND program for MODULES.DBF
    *
    PRIVATE ROW,RECNUM,RECNUMOFS
    PRIVATE ISBLANK,ISUNIQUE,ISCARRY,ISDELETED
    * ---Initialize local memory variables.
    ROW = PROMPTROW
    RECNUMOFS = 0
    STORE .F. TO ISBLANK,ISUNIQUE,ISCARRY,ISDELETED
    EXPR = ""
    DO MM__FORM
    * ---Start by adding one record.
    CHOICE = RETURNKEY
    * ---The following loop is really a "REPEAT/UNTIL  ".
  +=DO WHILE .T.
  |  +-IF (CHOICE = RETURNKEY) .OR. ISCARRY
```

```
** Updated on
** delete module master name from runtime database

SELECT DBM

FLD_NAME = TRIM( FLDNAME )
FLD_TYPE = FLDTYPE
FLD_LEN = FLDLEN
FLD_DEC = FLDDEC

FUNC_NAME = TRIM(FUNCNAME)              && get physical module name

SELECT RUNTIME
DB_NAME = PROJECTD+'RUNTIME'
DO ZAPFIELD WITH DB_NAME, FLD_NAME


*: EOF: ZAPITEM.ACT
```

MM.PRG

```
*:*****************************************************************
*:
*:        Program: MM.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     18:59
*:
*:      Called by: TOP.PRG
*:
*:          Calls: MM__OPEN.PRG
*:               : MM__EDIT.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************
* Program.: MM.PRG
* Author..: Mr. Abdenacer Moussaoui
* Date....: 03/19/89
* Notice..: Copyright (c) 1989, MTS
* Version.: FoxBASE+, revision 2.10
* Notes...: MAIN program for MODULES.DBF
*
* ---SET environment.
SET TALK OFF
SET STATUS OFF
SET HELP OFF
SET BELL OFF
SET MENUS OFF
SET SAFETY OFF
SET ESCAPE OFF
SET SCOREBOARD OFF

* ---Open PROCEDURE file.
SET PROCEDURE TO MM__PROC

* ---Declare Global memory variables.
PRIVATE;
PGDN,PGUP,RETURNKEY,DELRECORD,;
SCREENATR,STATUSATR,WINDOWATR,PROMPTATR,HILITEATR,;
DBFNAME,DBFTEMP,DBFAREA,DBFPAGEMAX,NDXORDER,ISEDITED,;
PROMPTBAR,PROMPTROW,MAINCHOICE,MENUCHOICE,CHOICE
STORE .F. TO;
PGDN,PGUP,RETURNKEY,DELRECORD,;
SCREENATR,STATUSATR,WINDOWATR,PROMPTATR,HILITEATR,;
DBFNAME,DBFTEMP,DBFAREA,DBFPAGEMAX,NDXORDER,ISEDITED,;
PROMPTBAR,PROMPTROW,MAINCHOICE,MENUCHOICE,CHOICE
PRIVATE NDXNAM1,NDXKEY1
STORE .F. TO NDXNAM1,NDXKEY1
```

```
USE DBM
*: EOF: SORTMOD.ACT
```

ADDITEM.PRG

```
*:*****************************************************************
*:
*:        Program: ADDITEM.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     18:45
*:
*:      Called by: TOP.PRG
*:               : MM__EDIT.PRG
*:
*:          Calls: ADDFIELD.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************

*_____________________________________________________
* Created on 11/15/1988 by Abdenacer
* Updated on
* Updated on
* Updated on
* Updated on
* add module master name to runtime database
*_____________________________________________________
** try to add item

SELECT DBM

FLD_NAME = TRIM( FLDNAME )          && FNAME   DOES NOT LIKE IT ??
&&   BECAUSE META-DB
FLD_TYPE = FLDTYPE
FLD_LEN = FLDLEN
FLD_DEC = FLDDEC

FUNC_NAME = TRIM(FUNCNAME)              && get physical module name

SELECT RUNTIME
DB_NAME = PROJECTD+'RUNTIME'
DO ADDFIELD WITH DB_NAME, FLD_NAME, FLD_TYPE, FLD_LEN, FLD_DEC
*: EOF: ADDITEM.ACT
```

ZAPITEM.PRG

```
*:*****************************************************************
*:
*:        Program: ZAPITEM.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     18:44
*:
*:      Called by: PACK.PRG
*:               : MM__EDIT.PRG
*:
*:          Calls: ZAPFIELD.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************

** ZAPITEM.PRG
** Created on 03/19/89 Time:16:10:50
```

```
+-IF DELETEIT
|    PACK
+-ENDIF
  *: EOF: PACK.ACT
```

SETE.PRG

```
*:*****************************************************************
*:
*:        Program: SETE.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     14:39
*:
*:      Called by: TOP.PRG
*:
*:     Documented: 03/19/89 at 19:14              FoxDoc version 1.0
*:*****************************************************************

    ** SETE.PRG
    ** Created on 03/19/89 Time:14:39:04
    ** Updated on
    **

    SET TALK OFF
    SET ECHO OFF
    SET SAFETY OFF
    SET STATUS ON
    *: EOF: SETE.ACT
```

SORTMOD.PRG

```
*:*****************************************************************
*:
*:        Program: SORTMOD.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/18/88     12:22
*:
*:      Called by: TOP.PRG
*:
*:           Uses: DBM.DBF
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************

    *_____________________________________________________
    *_____________________________________________________
    * algo:
    *  -
    *_____________________________________________________
    * Created on 0:14:17  1/12/1988 by Abdenacer
    * Updated on
    * Updated on 01/20/88
    * Updated on
    * Updated on
    *_____________________________________________________

    SELECT DBM
    ? 'Sorting on code then field name...'
    SORT TO DBM.TMP ON CODE, FLDNAME
    USE
    ERASE DBM.DBF
    RENAME DBM.TMP TO DBM.DBF
```

```
    * Updated on
    *_____________________________________________________

    && module database
    * ---Open database file.
    SELECT A
  +-IF .NOT. FILE( PROJECTD+"MODULES.DBF" )
  |     ? [PROJECD+"MODULES.DBF" not found]
  |     WAIT
QUIT
  +-ENDIF
    USE &PROJECTD.MODULES ALIAS DBM

    * ---Open INDEX file(s).
  +-IF .NOT. FILE( PROJECTD+"MODULE1.IDX" )
  |     ? [Creating index "MODULE1.IDX"...]
  |     INDEX ON FLDNAME+FUNCNAME TO &PROJECTD.MODULE1.IDX
  +-ENDIF
    SET INDEX TO &PROJECTD.MODULE1.IDX


    && actual data-items db
    SELECT 2
    USE &PROJECTD.RUNTIME
    *: EOF: OPENPROJ.ACT
```

PACK.PRG

```
*:*****************************************************************
*:
*:        Program: PACK.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     18:44
*:
*:      Called by: TOP.PRG
*:
*:          Calls: YESNO.PRG
*:               : ZAPITEM.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************

    ** PACK.PRG
    ** Created on 03/19/89 Time:16:47:17
    ** Updated on
    **

    DELETEIT = .F.

    SELECT DBM
    GO TOP
  +=DO WHILE ! EOF()
  |  +-IF DELETED()
  |  |     ANS = 'N'
  |  |     DO YESNO WITH ANS, 'Delete -' + FLDNAME+ '--' +FUNCNAME
  |  |     ?
  |  |  +-IF ANS = 'Y'
  |  |  |     DO ZAPITEM
  |  |  |     DELETEIT = .T.                && at least one needs to be deleted
  |  |  +-ELSE
  |  |  |     RECALL
  |  |  +-ENDIF
  |  +-ENDI
  |     SELECT DBM
  |     SKIP
  +=ENDDO
```

```
*:*****************************************************************
*:
*:        Program: GETPROJ.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 01/16/89     20:59
*:
*:      Called by: TOP.PRG
*:
*:          Calls: SELECTWA.PRG
*:               : INFORM.PRG
*:
*:           Uses: PROJECTS.DBF
*:
*:     Documented: 03/19/89 at 19:14              FoxDoc version 1.0
*:*****************************************************************

    ** GETPROJ.PRG
    ** Created on 01/16/89 Time:20:12:01
    ** Updated on
    **
    DO SELECTWA
    USE PROJECTS
    GO TOP
    BROWSE
    PROJECTD = TRIM( DIRECTORY )
  +-IF ! FILE( PROJECTD +'RUNTIME.DBF' ) ;
  |    .AND. ! FILE( PROJECTD + 'MODULES.DBF' ) ;
  |    .AND. ! FILE( PROJECTD + 'PCONST.DBF' )
  |
  |    DO INFORM WITH 'Creating project'
  |    DD = LEFT( PROJECTD, LEN( PROJECTD )-1 )
  |    ! MD &DD
  |    COPY FILE RUNTIME.DBF TO &PROJECTD.RUNTIME.DBF
  |    COPY FILE MODULES.DBF TO &PROJECTD.MODULES.DBF
  |    COPY FILE PCONST.DBF TO &PROJECTD.PCONST.DBF
  +-ENDIF
    USE

    *: EOF: GETPROJ.ACT
```

OPENPROJ.PRG

```
*:*****************************************************************
*:
*:        Program: OPENPROJ.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     15:12
*:
*:      Called by: TOP.PRG
*:
*:           Uses: &PROJECTD.MODULES    Alias: DBM
*:               : &PROJECTD.RUNTIME
*:
*:        Indexes: &PROJECTD.MODULE1.IDX
*:
*:     Documented: 03/19/89 at 19:14              FoxDoc version 1.0
*:*****************************************************************

    * Open project databases.
    *
    * Created on 9:27:11  11/3/1988 by Abdenacer
    * Updated on
    * Updated on
    * Updated on
```

```
*:        Program: MODIDATE.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     15:20
*:
*:      Called by: MEDITOR.PRG
*:               : MM__EDIT.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*******************************************************************

    *    IF READKEY() = 270
    REPLACE MODI_DATE WITH DATE()
    REPLACE MODI_TIME WITH TIME()
    MODIFIED = .T.
    *    ENDIF
    *: EOF: MODIDATE.ACT
```

CODEFILE.PRG

```
*:*******************************************************************
*:
*:        Program: CODEFILE.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     15:44
*:
*:      Called by: MM__APPE.PRG
*:
*:          Calls: YESNO.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*******************************************************************
    *
    * create header for module code file
    *
    MFILE = PROJECTD+TRIM(FUNCNAME)+'.PRG'

  +-IF FILE( MFILE )
  |    ANS = 'N'
  |    DO YESNO WITH ANS, 'file already exists, overwrite?'
  |  +-IF .NOT. ANS = 'Y'

  |  +-ENDIF
  +-ENDIF

    SAVE SCREEN
    SET ALTE TO &MFILE
    SET ALTE ON
    ? [?'] + FUNCNAME + '...' + METHOD + ['']
    ? [** Author: ] + AUTHOR
    ? [** Created on: ]
    ?? DATE()
    ?? ' ' + TIME()
    ? [**]
    SET ALTE OFF
    SET ALTE TO
    RESTORE SCREEN
    *: EOF: CODEFILE.ACT
```

GETPROJ.PRG

```
*:
*:          System: SIMUBASE: Data Base Based Simulation
*:          Author: Mr. Abdenacer Moussaoui
*:       Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     11:47
*:
*:       Called by: TOP.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************

    *_____________________________________________________
    * Created on 23:16:11  1/30/1988 by Abdenacer
    * Updated on
    * Updated on
    * Updated on
    * Updated on
    *_____________________________________________________

    SELECT DBM

    FUNC_NAME = TRIM(FUNCNAME)              && get physical module name
    SHORT_DESC = TRIM(METHOD)
    ** run module
    SELECT RUNTIME
    GO TOP                                  && extra??
    ? 'executing - ' + FUNC_NAME + '--' + SHORT_DESC
    EXECUTE = &FUNC_NAME()                  && execute module


    *: EOF: RUN_MOD.ACT
```

MEDITOR.PRG

```
*:*****************************************************************
*:
*:        Program: MEDITOR.PRG
*:
*:          System: SIMUBASE: Data Base Based Simulation
*:          Author: Mr. Abdenacer Moussaoui
*:       Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     16:44
*:
*:       Called by: TOP.PRG
*:                : MM__EDIT.PRG
*:
*:           Calls: MODIDATE.PRG
*:
*:     Documented: 03/19/89 at 19:15              FoxDoc version 1.0
*:*****************************************************************
    * MODULE EDITOR

    PARA FILENAME

    SAVE SCREEN

    KEYBOARD FILENAME + CHR(13)
    MODI COMM

    DO MODIDATE

    RESTORE SCREEN
    *: EOF: MEDITOR.ACT
```

MODIDATE.PRG

```
*:*****************************************************************
*:
```

```
      S- Simulation Scripts listing

      C- Project Constant repor




      0- EXIT


==
                              Enter your option
ENDTEXT

    @@ ROW(), COL()+2 GET OPTION PICTURE '! '
    READ

    *: EOF: MENUREPO.ACT
```

RUN_BAT.PRG

```
*:*********************************************************************
*:
*:        Program: RUN_BAT.PRG
*:
*:         System: SIMUBASE: Data Base Based Simulation
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 03/19/89     11:48
*:
*:      Called by: TOP.PRG
*:
*:     Documented: 03/19/89 at 19:15               FoxDoc version 1.0
*:*********************************************************************

    *_____________________________________________________
    * Created on 14:29:11  1/31/1988 by Abdenacer
    * Updated on
    * Updated on
    * Updated on
    * Updated on
    *_____________________________________________________

    SELECT DBM
    FUNC_NAME = TRIM(FUNCNAME)
    SHORT_DESC = TRIM(METHOD)

    SELECT RUNTIME
    * GO TOP
    * USE RUNTIME                    && go to top

    ? 'Batch - ' + FUNC_NAME + ' ' + SHORT_DESC + ' is currently running..  Wait...'
    ?

    EXECUTE = &FUNC_NAME()


    *: EOF: RUN_BAT.ACT
```

RUN_MOD.PRG

```
*:*********************************************************************
*:
*:        Program: RUN_MOD.PRG
```

APPENDIX C

COS.PRG

```
*:*****************************************************************
*:
*:        Program: COS.PRG
*:
*:         System: ** Personal Library **  --Not For Sale--
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 01/31/88     17:54
*:
*:      Called by: DUMMYTOP.PRG
*:
*:     Documented: 03/19/89 at 19:23              FoxDoc version 1.0
*:*****************************************************************
    *
    *_____________________________________________________
    *  An approximation of the cosinus function.
    *_____________________________________________________
    * algo:
    *  -
    *_____________________________________________________
    * Created on 0:39:31  1/12/1988 by Abdenacer
    * Updated on
    * Updated on 01/20/88
    * Updated on
    * Updated on
    *_____________________________________________________

   PARAMETER  X

   PRIVATE FCT
   PRIVATE PWR
   PRIVATE SIGN
   PRIVATE I

   PI2 = 2 * 3.141592564
  +=DO WHILE  X  PI2                && TO AVOID THE FACT GETTING BIG
  |    X = X - PI2
  +=ENDDO


   FCT = 2
   PWR = X * X
   TMP = 1
   SIGN = -1
   I = 3

  +=DO WHILE   I  30
  |    TMP = TMP + SIGN * ( PWR / FCT )
  |    FCT = FCT * I * (I+1)
  |    PWR = PWR * X * X                          && CAN SET X*X TO CONSTANT??
  |    SIGN = - SIGN
  |    I = I + 2
  +=ENDDO

   RETURN TMP
   *: EOF: COS.ACT
```

SIN.PRG

```
*:*****************************************************************
*:
*:        Program: SIN.PRG
*:
*:         System: ** Personal Library **  --Not For Sale--
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 02/01/88      1:18
*:
*:      Called by: DUMMYTOP.PRG
*:
*:     Documented: 03/19/89 at 19:23              FoxDoc version 1.0
*:*****************************************************************

    *
    *  An approximation of the sinus function.
    *
    * ATTEMP TO  SIN  IMPLEMENTATION USING   MAC-LAURIN SERIES
    *                  3     5     7
    *                 X     X     X
    *   SIN(X) = X - --- + --- - --- + ....
    *                 3!    5!    7!
    *
    * algo:
    *  -
    *
    * Created on 0:14:17  1/12/1988 by Abdenacer
    * Updated on
    * Updated on 01/20/88
    * Updated on
    * Updated on
    *

   PARAMETER  X

   PRIVATE FCT
   PRIVATE PWR
   PRIVATE SIGN
   PRIVATE I

   PI2 = 2 * 3.141592654
  +=DO WHILE  X  PI2                && TO AVOID THE FACT GETTING BIG
  |    X = X - PI2
  +=ENDDO

   FCT = 6
   PWR = X * X * X
   TMP = 0.0 + X
   SIGN = -1.0
   I = 4

  +=DO WHILE   I  30
  |    TMP = TMP + SIGN * ( PWR / FCT )
  |    FCT = FCT * I * (I+1)
  |    PWR = PWR * X * X
  |    SIGN = - SIGN
  |    I = I + 2
  +=ENDDO

   RETURN TMP
   *: EOF: SIN.ACT
```

ATAN.PRG

```
*:*****************************************************************
*:
*:        Program: ATAN.PRG
```

```
*:
*:          System: ** Personal Library **  --Not For Sale--
*:          Author: Mr. Abdenacer Moussaoui
*:       Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 01/31/88     17:55
*:
*:       Called by: DUMMYTOP.PRG
*:
*:      Documented: 03/19/89 at 19:23              FoxDoc version 1.0
*:*****************************************************************
    *
    *
    *  An approximation of the ARC-TANGENT function.
    *
    * algo:
    *  -
    *
    * Created on 0:14:17  1/12/1988 by Abdenacer
    * Updated on
    * Updated on
    * Updated on
    * Updated on
    *

    PARAMETER  X

    PRIVATE FCT
    PRIVATE PWR
    PRIVATE SIGN

    PI2 = 2 * 3.141592564
  +=DO WHILE X  PI2                && TO AVOID THE FACT GETTING BIG
  |    X = X - PI2
  +=ENDDO


    FCT = 3
    PWR = X * X * X
    TMP = X
    SIGN = -1

  +=DO WHILE   FCT  30
  |    TMP = TMP + SIGN * ( PWR / FCT )
  |    FCT = FCT + 2
  |    PWR = PWR * X * X
  |    SIGN = - SIGN
  +=ENDDO

    RETURN TMP
    *: EOF: ATAN.ACT
```

ADDFIELD.PRG

```
*:*****************************************************************
*:
*:        Program: ADDFIELD.PRG
*:
*:          System: ** Personal Library **  --Not For Sale--
*:          Author: Mr. Abdenacer Moussaoui
*:       Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 11/09/88     23:06
*:
*:       Called by: DUMMYTOP.PRG
*:
*:            Uses: &DB_NAME
*:                : &TMP_FNAME.DBF
*:                : &TMP2_FNAME
*:                : &ORG_FNAME
*:
*:      Documented: 03/19/89 at 19:23              FoxDoc version 1.0
```

```
*:*****************************************************************
   *
   *
   *  Given the name of a database and a field *description*
   *  will *add* the field and its data.
   *
   * algo:
   *  - open database in current area
   *  - check if field already exist and do nothing if it does
   *  - copy the structure as data to a tmp database   ie. meta-db
   *  - add the field description to the meta-db
   *  - create the structure of the original database from this data
   *  - append all matching data from original db
   *  - rename tmp file to the original name
   *
   * Created on 1/7/1988 by Abdenacer
   * Updated on 2:04:37  1/8/1988
   * Updated on
   * Updated on
   * Updated on
   *

   PARAMETER  DB_NAME, FLD_NAME, FLD_TYPE, FLD_LEN, FLD_DEC

   PRIVATE OTHER_FLDS
   PRIVATE CRT_FLD
   PRIVATE FLD_IDX
   PRIVATE ORG_FNAME
   PRIVATE TMP_FNAME

   USE &DB_NAME                          && open db in question

   FLD_IDX = 1
   CRT_FLD = FIELD( 1 )
   FLD_EXISTS = .F.

   SET EXACT ON                          && for string comparison

   && search for field
  +=DO WHILE   LEN( CRT_FLD )  0
  |  +-IF  CRT_FLD = FLD_NAME
  |  |    FLD_EXISTS = .T.
  v=======EXIT  && SHOULD WE EXIT LOOP??
  |  +-ENDIF
  |    FLD_IDX = FLD_IDX + 1
  |    CRT_FLD = FIELD( FLD_IDX )
  +=ENDDO

   * CLEAR
   * IF FLD_EXISTS
   *  ? "ALREADY THERE " + FLD_NAME
   * ELSE
   *  ? "ABOUT TO ADD.." + FLD_NAME
   * ENDIF

  +-IF  FLD_EXISTS

                          && avoid add it
   +-ENDIF                           && MAY BE SHOULD DISPLAY MESSAGE??



   TMP2_FNAME = DB_NAME + ".STR"          && database with new field
   TMP_FNAME = DB_NAME + ".EXT"           && meta_database
   ORG_FNAME = DB_NAME + ".DBF"           &&  HOW ABOUT USING DBF() ??

   COPY TO &TMP_FNAME STRUCTURE EXTENDED
```

```
     USE &TMP_FNAME
     APPEND BLANK
      REPLACE  FIELD_NAME WITH FLD_NAME, FIELD_TYPE WITH FLD_TYPE, FIELD_LEN WITH FLD_LEN,
FIELD_DEC WITH FLD_DEC

     CREATE &TMP2_FNAME FROM &TMP_FNAME       && same structure as orginal db
     APPEND FROM &ORG_FNAME

     DELETE FILE  &TMP_FNAME
     DELETE FILE  &ORG_FNAME
     USE
     RENAME  &TMP2_FNAME  TO  &ORG_FNAME
     Use &db_name

     *: EOF: ADDFIELD.ACT
```

ZAPFIELD.PRG

```
 *:*****************************************************************
 *:
 *:        Program: ZAPFIELD.PRG
 *:
 *:         System: ** Personal Library **  --Not For Sale--
 *:         Author: Mr. Abdenacer Moussaoui
 *:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
 *:  Last modified: 01/11/88     22:53
 *:
 *:      Called by: DUMMYTOP.PRG
 *:
 *:           Uses: &DB_NAME
 *:               : &TMP_FNAME.DBF
 *:
 *:     Documented: 03/19/89 at 19:23              FoxDoc version 1.0
 *:*****************************************************************
     *
     *_____________________________________________________
     *  Given the name of a database and a field name
     *  will remove the field and its data.
     *_____________________________________________________
     * algo:
     *  - open database in current area
     *  - build a list of field name except the one to be deleted
     *  - copy all field and data to a tmp file
     *  - delete old file
     *  - rename tmp file to the original name
     *_____________________________________________________
     * Created on 1/6/1988 by Abdenacer
     * Updated on 0:13:24  1/7/1988
     *_____________________________________________________

     PARAMETER  DB_NAME, FLD_NAME

     PRIVATE OTHER_FLDS
     PRIVATE CRT_FLD
     PRIVATE FLD_IDX
     PRIVATE ORG_FNAME
     PRIVATE TMP_FNAME

     USE &DB_NAME

     TMP_FNAME = DB_NAME + ".TTT"
     ORG_FNAME = DB_NAME + ".DBF"            &&  HOW ABOUT DBF() ??

     FLD_IDX = 1
     CRT_FLD = FIELD( 1 )
     OTHER_FLDS = ""
     && build field names not to be deleted
   +=DO WHILE   LEN( CRT_FLD )  0
   |  +-IF  CRT_FLD  FLD_NAME
   |  |    OTHER_FLDS = OTHER_FLDS + ", " + CRT_FLD
```

```
|  +-ENDIF
|     FLD_IDX = FLD_IDX + 1
|     CRT_FLD = FIELD( FLD_IDX )
+=ENDDO

   OTHER_FLDS = SUBSTR( OTHER_FLDS, 2 )      && trim first comma

   COPY TO &TMP_FNAME  ALL  FIELDS &OTHER_FLDS

   USE                                       && to delete it should not be in use
   DELETE FILE  &ORG_FNAME
   RENAME  &TMP_FNAME  TO  &ORG_FNAME
   USE  &DB_NAME
   *: EOF: ZAPFIELD.ACT
```

ADDSPATH.PRG

```
*:*****************************************************************
*:
*:        Program: ADDSPATH.PRG
*:
*:         System: ** Personal Library **  --Not For Sale--
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 11/08/88     23:53
*:
*:      Called by: DUMMYTOP.PRG
*:
*:     Documented: 03/19/89 at 19:23              FoxDoc version 1.0
*:*****************************************************************
   * push a search path onto the current one

   PARA NEWSPATH

   OLDSPATH = CRTSPATH()
   SET PATH TO &NEWSPATH;&OLDSPATH
   *: EOF: ADDSPATH.ACT
```

SETPCONS.PRG

```
*:*****************************************************************
*:
*:        Program: SETPCONS.PRG
*:
*:         System: ** Personal Library **  --Not For Sale--
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 11/24/88     15:04
*:
*:      Called by: DUMMYTOP.PRG
*:
*:           Uses: &P_DB
*:
*:     Documented: 03/19/89 at 19:23              FoxDoc version 1.0
*:*****************************************************************


   * set public constants.
   * Start initially on 7/3/88
   * Created on 21:55:33  11/2/1988
   * Updated on
   *
   * string have to be stored quoted I guess??

   PARA P_DB

   PRIVATE;
   M;
```

```
    V;

    SELECT 10
    USE &P_DB

   +=DO WHILE .NOT. EOF()
   |    M = NAME                && move field to mem variable so we can use macro
   |    PUBLIC &M               && make it global
   |  +-IF TYPE( SVALUE ) = 'U'
   |  |    WAIT  NAME + '=' + SVALUE + ' - Value is UNDEFINED and skiped!'
   |  +-ELSE
   |  |    V = SVALUE              && move field to mem variable
   |  |    &M = &V                 && evaluate and set mem variable
   |  +-ENDIF
   |    SKIP
   +=ENDDO

    USE
    *: EOF: SETPCONS.ACT
```

SELECTWA.PRG

```
*:*****************************************************************
*:
*:        Program: SELECTWA.PRG
*:
*:         System: ** Personal Library **  --Not For Sale--
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 11/03/88     10:11
*:
*:      Called by: DUMMYTOP.PRG
*:
*:     Documented: 03/19/89 at 19:23              FoxDoc version 1.0
*:*****************************************************************
    PRIVATE I,WA
    I = 0
   +=DO WHILE I _10
   |    I = I + 1
   |    WA = STR(I)
   |    SELECT &WA
   |  +-IF NULLSTR( ALIAS() )

   |  +-ENDIF
   +=ENDDO

    WAIT 'ERROR, NO EMPTY WORKAREA DEFAULT TO 10'
    SELECT 10
    *: EOF: SELECTWA.ACT
```

INFORM.PRG

```
*:*****************************************************************
*:
*:        Program: INFORM.PRG
*:
*:         System: ** Personal Library **  --Not For Sale--
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 01/16/89     22:25
*:
*:      Called by: DUMMYTOP.PRG
*:
*:          Calls: CLEARLIN.PRG
*:
*:     Documented: 03/19/89 at 19:23              FoxDoc version 1.0
*:*****************************************************************
```

```
* INFORM.PRG
* echo message to the user
*
PARA  MSG
LIN = ROW()
@@ LIN, COL()+1 SAY MSG + ' Hit return to continue! '
DUMMY = ' '
@@ LIN, COL() GET DUMMY
READ
DO CLEARLIN WITH LIN, LIN

*: EOF: INFORM.ACT
```

YESNO.PRG

```
*:*********************************************************************
*:
*:        Program: YESNO.PRG
*:
*:         System: ** Personal Library **  --Not For Sale--
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 01/25/89     21:37
*:
*:      Called by: DUMMYTOP.PRG
*:               : EDITOR.PRG
*:
*:     Documented: 03/19/89 at 19:23              FoxDoc version 1.0
*:*********************************************************************

   * Created on 0:47:03  7/1/1988 by Abdenacer
   * Updated on
   PARAMETER ANS, MSG

   SAVE SCREEN
   @@ ROW(), COL() SAY MSG + '? (Yes/No) '
   @@ ROW(), COL() SAY ANS PICTURE '!'
   * READ
   ** to be used in err.hand. while a read is intr.
   K = INKEY(0)
 +-IF K = 89 .OR. K = 121 .OR. K = 13
 |    ANS = 'Y'
 +-ELSE
 |    ANS = 'N'
 +-ENDIF

   RESTORE SCREEN
   *: EOF: YESNO.ACT
```

PRINTBEG.PRG

```
*:*********************************************************************
*:
*:        Program: PRINTBEG.PRG
*:
*:         System: ** Personal Library **  --Not For Sale--
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 07/19/88     11:23
*:
*:      Called by: DUMMYTOP.PRG
*:
*:          Calls: PRINTSET.PRG
*:
*:     Documented: 03/19/89 at 19:23              FoxDoc version 1.0
*:*********************************************************************
   *
   * Redirected output to desired device
```

```
    * Created on 23:25:57  6/9/1988
    * Updated on 7/13/88
    * Updated on
    * Updated on
    * Updated on
    *

    DO PRINTSET

  +=DO CASE
  +=CASE DEVICE = 'PRINTER'
  |    SAVE SCREEN
  |    CLEAR
Text
 Output is going to be directed to the printer
   - Turn printer on
   - set top of form
 when ready...
ENDTEXT
  |    WAIT
  |    RESTORE SCREEN
  |
  |    *  SET DEVICE TO PRINTER
  |    SET PRINT ON
  |
  +=CASE DEVICE = 'FILE'
  |    SET ALTERNATE TO REPORT.TXT
  |    SET ALTERNATE ON
  +=ENDCASE
    *: EOF: PRINTBEG.ACT
```

PRINTEND.PRG

```
*:*********************************************************************
*:
*:        Program: PRINTEND.PRG
*:
*:         System: ** Personal Library **  --Not For Sale--
*:         Author: Mr. Abdenacer Moussaoui
*:      Copyright (c) 1988, Mr. Abdenacer Moussaoui
*:  Last modified: 07/19/88     11:26
*:
*:      Called by: DUMMYTOP.PRG
*:
*:     Documented: 03/19/89 at 19:23               FoxDoc version 1.0
*:*********************************************************************
    * Reset output redirection to screen
    * Created on 23:23:18  6/9/1988
    * Updated on
    * Updated on
    * Updated on
    * Updated on


  +=DO CASE
  +=CASE DEVICE = 'PRINTER'
  |    *  SET DEVICE TO SCREEN
  |    SET PRINT OFF
  |
  +=CASE DEVICE = 'FILE'
  |    SET ALTERNATE OFF
  |    CLOSE ALTERNATE
  |    ** ! PRINT
  +=ENDCASE
    *: EOF: PRINTEND.ACT
```

1 Appendix D

1-1 System Requirements

Here is the hardware required to run SIMUBASE under Foxbase+.

- IBM PC, XT, AT, or 100% compatible. IBM PC compatible are available from a variety of manufacturers.
- At least 512K of internal memory (RAM). It is advisable to have more memory, ideally 640K (if not more).
- A monitor. Foxbase will run with just about any monitor, but if you have a color monitor you will be able to take advantage of the screen color settings that can be enhanced by careful use of color.
- A printer, correctly attached to the computer. Although a printer is optional, you won't get far in SIMUBASE without the ability to put something on paper.

We will assume you have a computer with 640K of memory, one floppy disk drive (DRIVE A), and a hard-disk.

1-2 Installation

To install the SIMUBASE demo on your hard-disk you must perform the following steps. Assuming you hard-disk drive name is C: and the SIMUBASE demo diskette is residing on drive A: you must type the following command at the DOS prompt:

```
C:
CD \
MD SIMUBASE
CD SIMUBASE
XCOPY A:\*.* C:\SIMUBASE /S
```

An alternative to executing these commands is to activate the provided batch file install.bat by typing INSTALL at the DOS prompt.

At this point you should have completed the installation, to activate SIMUBASE you type SB <return>

Table of Contents