WORKING PAPER

"FIELD MANAGER" APPLICATION PACKAGE

V. Mazourik

April 1989
WP-89-009

IIASA
International Institute
for Applied Systems Analysis

"FIELD MANAGER" APPLICATION PACKAGE

V. Mazourik

April 1989
WP-89-009

Computer Center of the USSR Academy of Sciences Moscow USSR

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

Foreword

One of the important problems in the development of Decision Support Systems regards the issue of designing and implementation of man-machine interface. Importance of this component of the DSS follows from the fact that the end--user is usually not a computer specialist and, therefore, even the most useful decision–theoretic framework will be rejected if communication with the computer is too difficult for him. From the other side, design and implementation of user interface requires a lot of experience from the system designer, big resources for programming and a long time for for debugging and coding. Therefore, every attempt to simplify this aspect of DSS design and development is important.

In this paper the software package *Field Manager* is presented. This package allows easy and quick development of user interfaces. The design is based on two novel ideas in the field of software management - the *abstract data type approach* and *object-oriented software specification*. The package has been applied in several practical applications and the collected experience has shown its effectiveness and simplicity.

Alexander B. Kurzhanski
Chairman
System and Decision Sciences Program

Table of Contents

Page

"FIELD MANAGER" APPLICATION PACKAGE

V. Mazourik

1. Introduction.

One of the most advanced scientific areas nowadays is the area of software development for personal computers. Due to the wide distribution of personal computers this area obviously contributes a lot to the new information technology which is a leading trend today.

There exist very many classes of software packages for the PC. The most popular and sucessfull from the commercial point of view are packages which are oriented to inexperienced user who is not familiar with computers at all.

Programmers who create those end-user packages use instrumental software which includes programming languages and compilers, graphic packages, numeric methods libraries, window managers etc. Software development tools provide programmers with powerful means to create more effective application-oriented packages.

This paper contributes to the instrumental software, which deals with interactive systems development. The Field Manager (FM) Package is described. Unlike the traditional window managers this package provides the programmer with the means to organize structured windows. Window can include any number of overlapped fields which are linked to the objects of the application program. Layout of the screen (fields number, their shape, position, linkage to the objects etc) is fully under user's control in dialogue session.

The abstract data types approach was used as a semantic and implementation basis of the FM package. It proved to be very effective to simplify procedural interface between the FM package and application programs.

2. The package goals.

The FM application package provides the user with flexible interactive capabilities for numerical analysis algorithms.

The numerical analysis programs under consideration can be described in the following way. There exists some mathematical model which consists of different objects: vectors, matrices, functions etc. There also exists a set of actions on these objects which are implemented in the form of numerical procedures. In other words we have the abstract interface to the model which permits us to manipulate the objects of the model without knowing its implementation in detail. The model analysis can be fulfilled by activating some numerical procedures on the objects and investigating the results thus obtained. We can treat each computational process on the model as a sequence of actions which have their own goals and need some control on the part of the user. The process may choose the proper moment for interaction with the user and select the control mode using information about the current situation. There exists another possibility when the user asynchronously controls the solution process and interferes if necessary.

Besides the objects of the model the numerical process can choose the list of its inner variables to be controlled during the solution. For example, the optimization method can introduce its parameters for the control. Permitting the control and stopping it are fully dependent on the inner logic of the numerical process. Let's now consider the control problems from the user interface point of view. The capabilities which are listed below are provided for the user by the FM package. At any given moment the computer terminal screen is occupied with some window which has a list of fields in it. The fields are homogeneous in the sense that they are not typified. The only two attributes which are connected with any field are the number of the numerical items in the field and the size of the item (the number of digits in it). The user can access the values of scalars, vectors and matrices using these fields. The package chooses the proper output form comparing the dimension of the object which is currently linked to the field with the attributes of this field. For example, if a scalar object is linked to a field which has several vertical items then the iterative output of this scalar value will be produced with automatic scrolling in the field.

The fields may or may not be linked to the objects of the numerical process. The user can make this connection for any object which is controllable at the given moment. The package makes some checks comparing the object and field attributes to be linked together. The window shows the name of the object currently linked to a given field (if any). Changing this name, the user causes another object to be linked to the same field. We can treat this in the following way. For a given linkage scheme each field "looks" at the memory cells which contain the current value of the object. It means that any changes of these values are automatically visible on the screen, and vise versa, the objects are accessible through the fields to change their values. This scheme improves the controllabili-

ty of the numerical algorithms. Let's consider some examples. The optimization method, being loaded, permits control of its parameters. It does not mean that these parameters will immediately appear on the screen. The user can create a couple of fields in the window. By changing the linkage of the parameters to these fields he can give all the parameters their initial values. If it is important to have permanent control of some parameter during the computational process the user can link it constantly to one of these fields using another one for some other purposes.

The problem of control variation can be easily solved using this technique, in particular, the problem of numerical methods debugging. The method can introduce its debugging parameters thus giving the possibility not only to control it by the end-user, but to debug it by the mathematician.

It is often convenient to double the control of the objects, especially in the hierarchical numerical algorithms, by linking the same object to different fields in different windows, or by linking different slices of a very long vector to several fields. The package will automatically produce the proper output in all these cases due to the object-oriented approach of the interface. The numerical algorithm calls the abstract output procedure of the current object value. This procedure will provide the proper output to all the fields which are linked to the given object.

Mathematic models consist of the sets of interrelated objects. The user chooses the proper positions of the fields at the screen in accordance with these relations. For example, in linear programming model the column Ax will be situated to the right to the A matrix, and vector of independent variables x will stand below matrix A. The shapes of the corresponding fields are also related in a proper way. If the cursor moves from left to right through the matrix field and meets the boundary of the field then automatic scrolling will take place in order to demonstrate the new matrix positions which were not previously visible. This movement violates the position correspondence between A matrix and x vector. The FM package makes it possible to improve this situation. The user can define horizontal and vertical groups of the objects. When the scrolling appears due to the cursor movement through any object of the group then all other members of the group will scroll automatically. The user can define groups and eliminate them by special procedures invocation in the program.

Flexibility of the control is also provided by the following feature. The user can enter the window editing mode which makes it possible to create new fields, modify the existent fields attributes or eliminate the field from the window. It is important to note, that all these modifications do not influence the numerical processes in operation. If, for example, all the fields which are linked to the object are eliminated from the window the

output procedure for this object will just display nothing.

Interactive algorithms of different types are often in use in the numerical packages. Many of the optimization algorithms are based on the iterative schemes. Some set of interrelated values is calculated at every iteration: the current value of independent variables vector, the correspondent value of the goal function etc. The characteristics of the solution process can be investigated by analyzing these values through the solution trajectory. The analogous sets of interrelated values appear in the multicriteria optimization, but they represent the solution itself in the form of Pareto points.

The FM package provides the user with the capability to define ordered sets of object values, to fill them in the iterations and to analyze them afterward using both numeric and graphic representation. It is possible to switch the field mode between the numeric and graphic one. Special attribute of the field controls the mode of field visualization. It has the values "numeric", "horizontal histogram", "vertical histogram", "deviation level", which will be described later. There also exist several forms of graphical representation of the function values. The package provides the menu capability. The menu is not a special object. The procedure written by the user may be linked to the object. If the user presses the "enter" button when the cursor is positioned in the field linked to the object the package will automatically call this procedure with the horizontal and vertical coordinates of the cursor in the field as the parameters. Thus the menu capability can be connected with any object by the description of the choice procedure for this object. The object types which are known to the package include the string type. This makes it possible to create ordinary menus with the symbolic items structured in vector or matrix form. In any sense the menus are ordinary objects and obey the general rules of the package. The user can for example link the menu to several different fields in different windows.

The menus provide the means to activate the numerical processes making it possible not only to control the numerical parameters but also to change the dialogue structure itself, accommodating it to the needs of the user. The menu capability is only one example of the more general and powerful mechanism of filtering procedures. This mechanism will be discussed later. The user can create his own structured windows adjusting them for different control schemes. He can save these standard interface patterns in files and then modify them adding some new fields, changing the linkage to the objects etc.

The package provides the asynchronous control of the numerical algorithms. It is possible to move the cursor through the window, to create new fields or modify the old ones, to change the linkage of the objects to the fields etc, without stopping the active numerical algorithm.

3. Interactive means of the package.

The main idea of the package is the division of the control and visualization capabilities between the application program and the dialogue monitor of the package itself. This approach permits the further improvement of the both components minimizing their interconnection. From the user point of view the package consists of the set of abstract procedures of access and visualization of the program variables. Using these procedures the user describes the main structure of the interaction with the program: in the proper points of the program he permits the control of some objects by introducing them or stops it, calls the demo procedures etc. These are the means of the static control of the numerical algorithms.

Let's describe the interactive control capabilities which the package provides for the user during the dialogue session.

To activate the package the user must initiate the window calling the procedure "wnd_init", then make some objects of the program to be known to the system calling the procedure "introduce" and then initiate dialogue session with the procedure "obj_interface":

```
main () {
char key;
wnd_init ();
introduce ...
do {
obj_interface ();
} while (key != 'q');
}
```

The "key" variable contains the code of the symbol which is currently pressed on the keyboard. The "q" symbol quits the dialogue session. The "obj_interface" procedure analyses every input symbol from the keyboard and creates the proper response to it. The symbol input in the "obj_interface" procedure is implemented using no-wait mode, which makes it possible to imitate the asynchronous mode of interaction. The codes of the numerical methods include the "obj_interface" procedure calls thus providing this imitation.

There exist several window modes:

- "active cursor" mode;
- "inside the field" mode;

- "editing" mode.

Let's consider the situation when the window has several fields and the mode of the window is "active cursor". In this mode the user can move the cursor at any position in the window. The "plus" button will change the mode to the "inside the field". The field is the rectangle in the window, which consists of several horizontal and vertical items. When entering the field, the cursor changes its shape to cover one item of the field. Pushing the arrows will then move the cursor to the next item of the field. If the object size is more than the field size this movement will result in automatic scrolling if necessary. This is a well known situation for the users of the spreadsheets.

If the "plus" button is pressed when the cursor is positioned out of any field, a new field will be created. Its size is initially equal to the size of one elementary symbol.

If the user presses the "plus" button in the "inside the field" mode then the mode will be changed to "editing". The "editing" mode consists of three submodes: moving the field through the window, changing the size of the item in the field and changing the number of items in the field. To return to the "inside the field" mode the user must press the "minus" button.

To explain the FM interface capabilities let's consider some typical situations:

1) Creation of the new field; current mode: "active cursor"; actions:
   - move cursor to the chosen position;
   - press "plus" (resulting in the minimum size field creation);
   - press "plus" (change the mode to "editing");
   - press "." (enter the editing submode to change the size of the field item);
   - press the arrows buttons to install the proper item size;
   - press "." (enter the editing submode to change the number of the items in the field);
   - press the arrows buttons to install the proper number of the items;
   - press "minus" (change the mode to "inside the field").
2) Editing the current field; current mode: "active cursor"; actions:
   - move cursor to the field;
   - press "plus" (change the mode to "inside the field");
   - press "plus" (change the mode to "editing"; in this mode the arrows will move the field through the window);

- then - as in (1).

3) Linking the object to the field; current mode: "inside the field"; actions:
   - press ">" (the comment "enter the object name" appears in the window);
   - enter the object name; when the user finishes entering the name ("enter" button) the field items will contain the current value of the object; the linkage is possible only for the objects which were previously defined with the help of the "introduce" procedure.

4) Changing the object value in the position of the field; current mode: "inside the field"; actions:
   - press " " (i.e. space; the comment "enter new value appears in the window);
   - enter new value;

5) Storing the window structure in the file; current mode: "active cursor"; actions:
   - press "F1" functional button (the comment "enter the file name to write the window" appears in the window);
   - enter the file name; when the user finishes entering the name ("enter" button) the window structure will be stored in the file; this window can be later retrieved from the file.

6) Retrieving the window structure from the file; current mode "active cursor"; actions:
   - press "F2" functional button (the comment "enter the file name" appears in the window);
   - enter the file name; when the user finishes entering the name ("enter" button) the window structure will be retrieved from the file.

7) Eliminating field from the window; current mode: "inside the field"; actions:
   - press <BACKSPACE>;

4. Procedural interface description.

The advantage of the FM package is the significant simplification of the procedural interface which is the consequence of the strict division of the main functions between the application program and the user who controls the computational process in interactive mode. The main idea of this division is the following one.

The application program takes into account only two aspects of the user interface.

First, it initiates the control over the given objects by the "introduce" procedure call or stops the control by the "forget" procedure call. This means that the program itself is responsible for the set of the objects which are available for the user to control their values at any given moment.

Second, the application program initiates the demo of the object values thus turning the user's attention to some computational situations. All these actions are fully under control of the application program which is quite natural because only the program has the adequate and full picture of the computational process. The actions of introducing objects and demonstrating their current values have the goal of providing the user with as much information as possible for decision making.

On the other hand it is unnatural to make the application program responsible for the implementation features of the interface such as amount and position of the fields in the window, their attributes (color, shape) etc. The application program only provides the user with control resources and the user may decide by himself how to use them more effectively. He selects the objects to be controlled depending on the current situation and the goals of the control at the given moment.

Let's describe the FM package procedure interface. This interface is oriented for the use in the C language environment.

Window initialization procedure:

wnd_init ()

This procedure without parameters must be called before introducing any objects, because it is responsible for the tables initialization.

Object introduce procedure:

```
introduce (obj_type, elm_type, obj_name, comment,
value_ref, x_size, y_size)
  int  obj_type; /*  object type  */
  int  elm_type; /*  element type  */
  char  * obj_name;  /* object name  */
  char  * comment;  /* comments  */
  char  * value_ref;  /* value reference  */
  int  hor_size,  ver_size; /* object dimensions */
```

The object types known to the package are SCALAR, VECTOR, MATRIX. The element types are CHAR, STRING, INT, LONG_INT, FLOAT, DOUBLE. All these literals are defined in the fm.def file. The user must include it to the application program.

The filters may be linked to the object. The filters are user defined procedures, which are invoked when some events occur. These events are listed below (the procedure specification is given):

BOUND_FILTER - cursor crosses the field boundary;
no parameters;

VAL_FILTER - the value in the field position is changed;
val_filter (old_value, new_value, hor_index, ver_index)
int * old_value;
int * new_value;
int hor_index;
int ver_index;

ENTER_FILTER - position choice (menu);
enter_filter (hor_index, ver_index)
int hor_index;
int ver_index;

PRED_LINK_FILTER - linking the object to the field;
no parameters;

PRED_ENTER_FILTER - entering the field;
no parameters;

POST_ENTER_FILTER - leaving the field;
no parameters;

POST_LINK_FILTER - unlinking the object from the field;
no parameters;

POSITION_FILTER - changing the cursor position within
the field;
position_filter (hor_index, ver_index)
int hor_index;
int ver_index;

DEMO_FILTER - calling the demo procedure;
the filter is invoked before object demonstration;
no parameters;

SYMBOL_FILTER - pressing the keyboard button which
doesn't have the predefined system reaction;
symbol_filter (symbol, hor_index, ver_index)
char symbol;
int hor_index;

```
int ver_index;
```

In order to install the filter for the object one must call the procedure:

```
filter_ready (obj_name, filter_type, proc)
char * obj_name;
int   filter_type;
int (* proc) ();
```

For example, the procedure call

```
extern int s_filter ();
filter_ready ("s_vector", PRED_ENTER_FILTER, s_filter);
```

yields in the following: s_vector object is linked to the user defined procedure s_filter.

This procedure will be automatically invoked when the cursor is inside any field which is currently connected with s_vector and the user presses the <ENTER> button. The actual parameters of the s_filter are the horizontal and vertical coordinates of the cursor position inside the field. More strictly, taking into account possible scrolling they are the coordinates of the current position inside the structured object. As was mentioned above this is a way to create menu capability.

Obviously, any object (for example, numerical matrix) may play the role of menu.

The system will invoke the filter if proper situation occurs and if the filter for this situation is active at the moment.

The procedure to eliminate control over the object:

```
forget (obj_name)
  char * obj_name;
```

The demo procedure:

```
demo (obj_name, attention_mode)
  char * obj_name;
  int attention_mode;
```

The "attention_mode" parameter may have two values:

NORMAL, ALARM. The object value will blink in ALARM mode. The procedure to send the message to the screen:

```
message (text, attention_mode, wait_factor)
char * text;
int attention_mode;
int wait_factor;
```

The last parameter defines the time delay after the message:

number - delay in seconds;

NO_WAIT- no delay (corresponds to number = 0);

YES_NO - delay till pressing any button.

The procedure to create horizontal group of the objects:

create_hor_group (group_name, group_size, name_1, ...)

int group_name; int group_size; char * name_1; ...

This procedure with variable number of parameters creates the named group which consists of group_size objects. The names of the objects are name_1, etc. The attribute "horizontal" means, that the simultaneous horizontal scrolling is switched on for all the objects in the group.

Analogous procedure to create vertical group of objects:

create_ver_group (group_name, group_size, name_1, ...)

The object can be the member of horizontal and vertical group simultaneously (for example, the matrix in linear programming task). Several groups can coexist with the only restriction that their intersection is empty in both horizontal and vertical classes. Any subset of the group members can be linked to the fields of the window.

Separate demonstration of subsets of some large group is thus possible. Procedures to eliminate previously defined group linkage:

```
elim_hor_group (group_name)
  char * group_name;
elim_ver_group (group_name)
  char * group_name;
```

Procedure to create ordered list of objects:

list_introduce (name, comment, obj_number, name1, ...)

```
char * name;
char * comment;
int obj_number;
char * name1;
```

Each chain of the list consists of obj_number elements which have the names name1 etc.

Procedure to add new chain to the list:

```
add_to (list_name)
  char * list_name;
```

New chain in the list is created. The current values of the objects are saved in this chain.

The list object can be linked to any field in the window. The hierarchy of link capabilities exists for the list object which is due to the inner hierarchy of the list structure. First the list itself is linked to the field.

Some operations are available in this state. One of them - list reordering by the object attributes in the chain. Pressing <PgDwn> descends to the lower level. The chains of the list are obtainable on this level. Pressing <PgUp> lifts again to the list level.

The lists are automatically linked to system defined filter procedures which are invoked when pressing <CTRL-arrow> buttons. Vertical arrows change the linkage to the field from one object in the chain to another in the order which was specified in list_introduce procedure call. Horizontal arrows make the movement through the list. It means, that the sequence of the previously saved object values will appear in the field. The field itself may have any attributes, including histogram output, which makes it possible to analyze the iterative solution process in graphic form.

When leaving the field the system saves the current link: name of the object in chain and index of value in list. Entering the field will restore this state.

The object in the link is not restricted in its type. It may have list type, which is the way to organize complicated structures. For example, the ordered list of optimization trajectories may be easily created to be analyzed thoroughly afterwards.

Procedure to save the window structure in the file:

```
wnd_write (file_name)
  char * file_name;
```

Procedure to restore the window structure from the file:

```
wnd_read (file_name)
  char * file_name;
```

Using this procedure the application program can invoke previously defined layout of the window. The typical situation is that the user takes some layout, which is recommended by the system designers and then he accommodates it to himself creating his own layout modes.

5. Graphic capabilities.

The goal of the FM package is to provide the user with the instrument of interactive control of the numerical algorithms. The efficiency of these capabilities to simplify the decision making for the user is the main criterion for their design. The visualization capabilities are very important for the decision making. It is well known that the graphic form of output is accepted by the user at least two times as fast as the numerical information. The windows with dynamically created fields proved to be effective instrument for the user to choose the proper control level over the objects of numerical models. In addition to it the FM package has two more capabilities which are oriented on the graphic form of the interface.

The first is pictogram mode of interface. The second gives several ways to create graphic images of functions which are defined in n-dimensional space.

The pictogram interface is becoming very popular recently. It is almost a necessary attribute of the new operating environments and operating systems for the personal computers. We can interpret the pictogram interface as the use of the nominal scale, when we are interested not in the numerical characteristics of the object but only in the fact that the object belongs to some class. We can for example use the pictograms to mark the different files in the directory. It is simple to understand without any explanation that the file which is represented by the sheet of paper contains the textual information.

Let's consider the numerical analysis programs. The numerical form of output is not always the best one. Very often it is more convenient to output only the current status of the object using some picture which is obviously connected with this status. The status list and the corresponding pictograms may be linked to the object when it is introduced to the package. The application program calculates the current status of the object and the demo procedure chooses the proper pictogram to show it on the screen.

The graphic form of output for the numerical values of the objects is implemented in FM in the following way. Among the field attributes to control its shape, position, color etc. there are some others which define the graphic alternatives of data output. One is the histogram mode of output, another - deviation mode.

There exists three modes of output to the positions of the field. First is traditional numerical formated output. Second is histogram mode. The FM package provides the user with two submodes: horizontal and vertical histograms. The histogram output is a convenient way to compare visually the values in a given set.

If a matrix object is linked to the field and the field has the output mode "horizontal histogram", then rows of the matrix will be shown in the histogram form with automatic

scaling through all the values in every row. Histogram bars will be drawn in the same positions in the field, which are usually occupied by the numbers. The most suitable visual shape of the positions can be easily obtained when entering the editing mode of the field.

Vertical histogram makes it possible to visually compare the values in every column of the matrix.

Let's give an example of a histogram output. Create a new field which has one vertical and several horizontal positions. Let the positions be like narrow long vertical bars. Let the output attribute of the field be "horizontal histogram". Link the matrix object to this field and immediately the histogram which corresponds to the first row of the matrix will appear on the screen. Cursor movement in horizontal direction will (if necessary) activate scrolling and previously hidden histogram positions will become visible. Vertical cursor movement will cause the next row of the matrix in histogram form to appear in the same field positions. If rows represent some values in fixed moments, then vertical scrolling through the matrix demonstrates visually some time dependent tendencies of those values.

Before discussing the deviation level attribute let's describe the sample value comparison mechanism. The implementation of this feature in the FM package is due to the fact that often we are interested not in the object value itself, but in the deviation between this value and some given value.

Any field can be linked to so called sample object. To create this link you must press "=" button and the system will ask the name of etalon object. Pressing the "!" button will eliminate this link. The deviations between the current object values and sample values are automatically represented in the field while sample object reference is nonzero. The sample object may have Scalar, Vector or Matrix type. The obvious rule of dimension adjustment is based on the etalon values reduction or copying. If, for example, the linked object is matrix A and etalon object is vector b, then the values A__dev which we shall see are:

A__dev [i, j] = A [i, j] - b [j]

The deviation level attribute of the field provides the user with the means to demonstrate visually the level of deviation between the etalon value val__e and the current value val__c. This attribute is a positive number L. If |val__c - val__e| > L then the field 5 position has one color, if not - then another.

Besides the comparison with etalon values this attribute is useful for the sparse matrix analysis. To analyze the number of zeros you can create the field with large amount of tiny positions, make the L attribute to be small number which slightly differs from zero

and link the matrix to this field. The color picture of zeros distribution will immediately appear at the screen.

The functions defined on the n-dimensional space and their properties are the main objects of investigation for the numerical analysis algorithms. The FM package provides the user with some means to analyze and demonstrate function values. One of them is new type of the graphic object, which is FUNCTION BY DIRECTION. The procedure to introduce this type of object to the system is the following:

fd_introduce (name, image_type, comment, fun, dimens, base, direction, step, number_of_steps, fun_values, fun_low, fun_up, fun_current)

```
char      * name;
int       image_type;
char      1* comment;
double    (* fun) ();
int       dimens;
double    * base;
double    * direction;
double    * step;
int       number_of_steps;
double    * fun_values;
double    * fun_low;
double    * fun_up;
double    * fun_current;
```

The object of this type can be linked to any field. The field will demonstrate the curve of the function values through the direction defined by the "direction" argument. "base" is initial point, "step" is step value in this direction. "number_of_steps" function values are automatically calculated and saved in "fun_values" vector. Min and max values "fun_low" and "fun_up" are also calculated in "fun_values" domain. "fun_current" contains current function value which corresponds to the cursor position in the field. Each position in the field corresponds to one step in the given direction. If the field has less than "number_of_steps" horizontal positions, then scrolling will be automatically invoked if necessary. Any change of the field shape results in recalculation of the picture.

The "name" parameter defines as usual the name of the graphic object. Let it be, for example, sin. Then the following objects are generated by the system (the equivalent introduce call is given to describe the semantics of the objects):

- sin.base -
introduce (VECTOR, DOUBLE, "sin.base", " ",
base, dimens);

This introduction means that the user can link the object with name "sin.base" to the field thus obtaining full control of its value. Note, that another object which has the same value reference "base" can be created by the user. In this case "sin.base" can be treated as a synonym name to it. The same remarks are valid for other objects - attributes of sin.

- sin.direction - introduce (VECTOR, DOUBLE. "sin.direction", " ", direction, dimens);

- sin.step - introduce (SCALAR, DOUBLE, "sin.step", " ", step);

- sin.fun_low - introduce (SCALAR, DOUBLE, "sin.fun_low", " ", fun_low);

- sin.fun_up - introduce (SCALAR, DOUBLE, "sin.fun_up", " ", fun_up);

- sin.fun_values - introduce (VECTOR, DOUBLE. "sin.fun_values", " ", fun_values, number_of_steps);

- sin.fun_current - introduce (SCALAR, DOUBLE, "sin.fun_current", " ", fun_current);

If the user links "sin.fun_current" object to some field, then the current numerical value of the function which corresponds to the cursor position in "sin" field will be visible. The user can mark the graphic field with two fields, linked to "sin.fun_low" and "sin.fun_up", to have information about the current function values scaling.

As it was mentioned above, any object can be linked with the filtering procedures. Graphical objects are not the exception. It is possible, for example, to implement digitizing of graphic object using the proper <ENTER> filter. The discreet digitizing grid is defined by the amount and shape of the positions inside the field, and the use can change these attributes of the field without disturbing the link of the field with the graphic object.

Another graphic object makes it possible to demonstrate the function values in the vicinity of the given point:

fp_introduce (name, comment, fun, dimens,
mode, level, base,
hor_step, hor_index, ver_step, ver_index,
fun_low, fun_up,
point_current, fun_current)

```
char        * name;
cha.        * comment;
double      (* fun) ();
int         dimens;
int         * mode;
double      * level;
double      * base;
double      * hor__step;
int         * hor__index;
double      * ver__step;
int         * ver__index;
double      * fun__low;
double      * fun__up;
double      * point__current;
double      * fun__current;
```

The function values are calculated at the discreet points of the plane which is defined in "dimens"-dimensional space by "base" point and two coordinate axis with "hor__index" and "ver__index". "hor__step" and "ver__step" are steps of discretization in two given directions. The number of points in the grid is defined by the amount of positions in the field linked with this graphical object. "fun__low" and "fun__up" have the obvious values in the domain of the grid.

Visualization mode is controlled by "mode" attribute. The value mode = 0 makes the mapping of the values to the maximum color palette for the given adaptor. The user can see color pattern of function values in the field, the central position being occupied by the base point value. The value mode = 1 makes binary branching of the values using "level" attribute.

All the attributes of this graphic object are defined as objects in the same way as it was described above. For example, if "my__fun" is the name of a graphic object, then the user can link "my__fun.fun__current" object to some field to have the numerical function values at the screen while cursor moves through "my__fun". "my__fun.point__current" object will simultaneously show the numerical value of the current point. Changing "my__fun. hor__step" and "my__fun.ver__step" the user can find the most suitable scale of the function representation.

Using this type of graphic object one can show equilevel lines of the function in discrete approximation. These lines are implemented as sequences of field positions with the same color. If, for example, the function describes the pollution of some territory,

then visual analysis of the pollution levels can be easily done by changing "level" attribute of the graphic function representation.

All the attributes of graphic objects can be controlled both manually and automatically by the application program. The last possibility is due to the fact that attributes are ordinary variables in the program as one can easily see in the procedure parameters definition given above.

Let's describe now the FM capabilities to produce the graphical image of iterative processes. This class includes optimization algorithms, differential equations integration etc. Numerical results of iterative processes are saved as ordered sets of object values, those values being linked together in chains. The procedures for list creation, adding chains to the lists etc. were described above.

The chain contents is defined by the user. It can contain objects of different types: scalars, vectors, matrices. The list consists of the current values of these objects through the iterations. We can treat these discrete sets of values as a relation. The domain of it is cartesian product of vector spaces for the chain objects.

Let's give some examples of these relations. For the multicriterion optimization task the chain may represent Pareto point, which is described by parameters vector and corresponding criteria vector. In this case linear order of the chains in a list is not important.

For nonlinear programming task the chain consists of the independent variables vector, equality and inequality restrictions vectors, scalar of goal function value, dual variables vector etc. If the problem is to investigate the sensibility of the results for the method parameters they can be also included into the chain. The linear order in the list gives the sequence of the solution trajectory, which is important information for the user.

The FM package provides the user with two graphic objects to visualize multidimensional discrete relations. The first one creates projection of a given relation to the pair of axis:

```
r__introduce (name, comment, list__name,
hor__object__name, hor__object__index,
ver__object__name, ver__object__index,
hor__low, hor__up, ver__low, ver__up,
hor__current, ver__current)

char        * name;
char        * comment;
char        * list__name;
```

```
char        * hor__object__name;
int         * hor__object__index;
char        * ver__object__name;
int         * ver__object__index;
double      * hor__low;
double      * hor__up;
double      * ver__low;
double      * ver__up;
double      * hor__current;
double      * ver__current;
```

"list__name" is the name of the list which contains information for the graphic object. Two-dimensional projection of the discrete relation is defined by the names of two objects: "hor__object__name", "ver__object__name" (which must be the members of the chain for a given list) and their component indices "hor__object__index", "ver__object__index". They are obviously related to the horizontal and vertical directions in the field. The scaling of coordinates is automatically fulfilled to prepare the picture of the projection.

All the attributes of the graphic object can be controlled by the user as usual. For example, "hor__current" and "ver__current" objects make it possible to get current cursor coordinates in the chosen object values domain. The menu filter can be linked to this object to analyze cursor coordinates when it is visually positioned to some place within the field and to invoke proper reaction.

This type of graphic object may help to analyze the solution of multicriteria optimization problem which has the form of Pareto set approximation. In this case both "hor__object__name" and "ver__object__name" are the names of criteria vector, and indices define the particular pair of scalar criteria values to be represented in the field.

Another graphic object provides the user with the means to make the projection of a given set of points to two-dimensional subspace:

```
rp__introduce (name, comment, mode, list__name, object__name,
low, up, hor__axis, ver__axis,
hor__low, hor__up,
ver__low, ver__up)
char        * name;
char        * comment;
int         * mode;
```

```
char        * list__name;
char        * object__name;
double      * low;
double      * up;
double      * hor__axis;
double      * ver__axis;
double      * hor__low;
double      * hor__up;
double      * ver__low;
double      * ver__up;
```

The plane is defined in vector space of the "object__name" object values. Its orientation is given by the vectors "hor__axis", "ver__axis". All the points which correspond to the "object__name" object values in the chains of "list__name" list and which are inside the box "low", "up" will be shown in the field as projections to the given plane. "hor__low" , ... , "ver__up" define the least possible rectangle which contains all the projections. "mode" parameter defines the mode of points representation. Simultaneous creation of several planes with different orientation provides the user with the convenient visual patterns to analyze the trajectory of the iteration process in multidimensional space.

6. Future extensions.

New graphic capabilities are going to be included to the FM package. One of them is parameterized picture:

```
pp__introduce (name, comment, painter, par__vector)
  char          * name;
  char          * comment;
  void          (* painter) ();
  double  * par__vector;
```

This object makes it possible to create any graphic pattern in the field. User__defined "painter" procedure is the creator of the picture in the field. The body of "painter" can be implemented using any graphic means which are available. The picture is parameterized in a sense that "painter" may take the values from "par__vector" to create the picture. The FM package provides the "painter" with the procedural access to all the necessary information of the field shape and position.

New object is the base for the application-oriented graphic interface in FM. For example, the parameterized picture can become the graphic output in the iterations of the optimization problem, giving the direct visual interpretation of the optimization process.

Another extension of the FM package will improve the means to exchange data with other popular systems like dBase3, Framework etc.

Recent version has the limitation, that only the objects which were previously introduced in the body of application program are known to the system. The new release will provide the user with dynamically created objects which he defines in dialogue session.

Another limitation is that the FM package now provides very rough means to embed the asynchronous control to application program. What you need to do to make the system react to keyboard when application program runs is to recompile it, embedding the call "obj_interface ();" after every few lines of code. The new release with much more convenient asynchronous control mechanism based on time interrupts is being tested now.

The new release will have menu-oriented layout and on-line help capabilities.

7. Technical characteristics.

The FM package is available for the IBM PC XT and AT compatible personal computers with CGA. EGA adaptor is also valid, but at the moment the FM package doesn't use its high resolution mode.

In fact, the FM package can be treated as an extension of C language. The package itself is implemented in C language and has about 2000 lines of source code. The Microsoft C compiler 4.0 with a big memory model was used. The FM object library has less than 40 kbytes size. The package consists of three files:

- fm.def - definition file to be included in application program #include "fm.def" before the compilation;
- fm.lib - library to be linked with application program;
- demo.exe - demo file.

8. Appendix. FM finite automata.

Definition of screen modes:

free cursor	cursor has small shape and moves through the screen using arrows
inside the field	cursor is inside the field and has the shape of one field position
free cursor in field area	free cursor which is positioned within the field area
edit_move	first mode of field editing: moving the field in the window
edit_size	second mode of field editing: changing the size of field position
edit_number	last mode of field editing: changing the number of positions inside the field

mode	key	new mode	remark
free	<F1>	free	write layout to the file
free	<F2>	free	read layout from the file
free+field	+	inside	enter the field
free	+	edit_move	create new field with one elementary position and enter first edit mode
inside	+	edit_move	move the field
edit_move	.	edit_size	change position size
edit_size	.	edit_number	change positions number
edit_move	-	inside	finish editing
edit_size	-	inside	finish editing
edit_number	-	inside	finish editing
inside	-	free	free cursor state
inside	>	inside	link the field to the obj
inside	<	inside	eliminate linkage
inside	-	inside	enter new value
inside	<BACKSPACE>	free	delete the field
inside	<CTRL-right>	inside	background field color
inside	<CTRL-left>	inside	foreground field color
free	<CTRL-right>	inside	background window color
free	<CTRL-left>	inside	foreground window color
free	<CTRL-F9>	free	enter DOS level
free	<SHIFT-F9>	free	graphic mode (temporary)

Note: this description does not include the obvious response to arrows. For example, being in "edit_move" mode one must press <right-arrow> button to move the field one position to the right.