On the Dialog Machine[1] [2]

Andreas Fischlin
& Frank Thommen

ETH Zurich, February 2002

Contents

1 INTRODUCTION 2
2 WHAT IS THE “DIALOG MACHINE”? 3
2.1 A Little Bit of History 3
2.2 For Whom Is the “Dialog Machine” Useful? 4
2.3 How Does the “Dialog Machine” Work? 4
2.4 How Is the “Dialog Machine” Structured? 5
2.4.1 Overview 5
2.4.2 Characteristics of Most Frequently Used Modules 6
3 “DIALOG MACHINE” BASICS 11
3.1 How to Use the “Dialog Machine” 11
3.2. Basic Concepts and Frequently Used Objects 11
3.2.1 Menus 11
3.2.2 Windows 13
3.2.3 Output to Windows 17
3.2.4 Dialogs 19
3.2.5 File I/O and Other Non-Dialog Aspects 21
4 LEARNING BY EXAMPLE 22
4.1 Sample Programs 22
4.1.1 Simple 22
4.1.2 AlmostSimple 22
4.1.3 LessSimple 23
4.1.4 Random 25
5 REFERENCES 32
6 APPENDIX 33
6.1 How to Get the “Dialog Machine” 33
6.2 Hard- and Software Requirements 33
6.3 Disclaimer 34
7 QUICK REFERENCE 35
INDEX 44

[1] Refers to Version 2.2 16/Jan/98 and Version 3.0 9/Jun.2001

[2] The Dialog Machine is one of the fundamental aids belonging to RAMSES (Research Aids for Modeling and Simulation of Environmental Systems).

1 Introduction

Programing on todays interactive computers is too complex. It requires from users who have some programing skills, but who are not computer experts so tremendous efforts, that such users often fall back in programing styles and habits way back and certainly inadequate given todays computers potential. Here the “Dialog Machine” tries to help. It is a simple, yet fully fledged programming environment for non-specialist programmers who like to work in a reliable, robust, and clearly defined environment without having to hack. It provides an easy to learn access to modern working station's user interfaces, which is even highly portable (same programs run on several computer platforms unaltered), and efficiently.

As a consequence, “Dialog Machine” programs offer not only advantages to the programmer, but also to the end-user. They are not only easier to program, but behave usually also more robust. The "philosophy" on which the “Dialog Machine” has been built is that functionality matters to the users most, not appearance (In contrast to most commercial standard software, available e.g. on the Apple® Macintosh®, or IBM PC which do only look similar, but do NOT BEHAVE in the same and consistent way).

I hereby wish to thank all those numerous users and developers of the “Dialog Machine” who made this project possible, in particular Dr. Klara Vancso, Alex Itten, and the other members of the first team from the Pilotproject CELTIA, Project-Centre IDA and all the members like Dr. Olivier Roth, Dimitrios Gyalistras, Thomas Nemecek, Frank Thommen, Juerg Thoeny, Harald Bugmann, of the Systems Ecology Group from the Department of Environmental Sciences at the Swiss Federal Institute of Technology Zürich (ETHZ), who have contributed in one or other form to the success of the “Dialog Machine” project (all the ones I should have forgotten, it's only my fault). I'm especially indebted to Prof. Walter Schaufelberger from the Institute of Automatic Control at ETHZ who encouraged and made the inital embarkement onto the “Dialog Machine” project possible. I'm also greateful to Prof. Niklaus Wirth from the Institute of Computer Sciences at ETHZ; not only has he provided with his excellent work the necessary spirit and fundaments on which we could base our efforts (otherwise we would have built on quicksand), but he has also generously offered several times advice and support. Finally I wish also to thank the Swiss Federal Institute of Technology Zürich (ETHZ) Switzerland, who provided the engineering environment for a natural scientist in which such crazy tasks as designing and implementing a software like the “Dialog Machine” could be completed. Thanks for the opportunity to build a tool, about which we could so far just dream, hope you have fun too and enjoy the “Dialog Machine”. At least what concerns me, I'm glad I can use it!

Zurich, June 1996

Dr. Andreas Fischlin
Systems Ecology ETHZ

2 What Is the “Dialog Machine”?

2.1 A Little Bit of History

The “Dialog Machine” is a software package which has been produced in its first implementation as part of an authoring system under development at the Swiss Federal Institute of Technology Zürich (ETHZ) by the project team CELTIA[3] at the Project-Centre IDA[4] . Project IDA (1986-1991) was headed by the university headquarters (Rektorat) of the Swiss Federal Institute of Technology Zürich (ETHZ) and was in charge of a large five year impulse program to introduce working stations in university teaching financed by the Swiss government. The “Dialog Machine” has been implemented on the Apple® Macintosh® computer (512K RAM or more). In its current version it consists of 26 modules supporting pull-down menus, windows, window related input and output, modal and modeless dialogs, alerts, files, printing and clipboard access.

The “Dialog Machine” was first created, i.e. in the years 85/86, for an easier programming of the *Apple® Macintosh®* in Modula-2. The goal was to have a tool which renders programming of a graphical user interface as simple as possible. Therefore, restricting somewhat its functionality was not much of a concern. The motto was "small and beautiful". In the meantime the “Dialog Machine” has been used for many more purposes and needed to grow a bit (we hope not too much), and it offers now many features which support also quite advanced uses. Essential are just a few, i.e. 30 procedures, which are needed to write the majority of “Dialog Machine” programs. As a benefit the code of programs using the “Dialog Machine” can be short and are written quickly - considering their appearance, robustness, and ease of use.

At various stages, several subprojects were undertaken by different groups and people. For instance, a remote “Dialog Machine” (RDM) was implemented at the institute of Kommunikationstechnik of ETHZ. It made it possible by means of procedure calls to execute parts of “Dialog Machine”-programs on several computers, e.g. a VAX connected via a LAN was computing the content of a window in response to an update request. This demonstrated the flexibility of the “Dialog Machine” design for general programing purposes.

In 1988/89 the “Dialog Machine”was ported to the *MS-DOS/PC* (running under *GEM Desktop*). Apart from a few omissions and limitations, a Modula-2 program using the “Dialog Machine”could easily be ported from the Macintosh to the PC. One only needed to re-compile and link with the DM/PC library. The resulting application could then be executed under GEM and made full use of the mouse, drop-down menus and windows, thereby giving the same user-friendly look and feel as on the Macintosh. On the PC the GEM-version is no longer supported. Instead in 1992/1993 a version for *Windows 3.1* has been developed.

Recently the “Dialog Machine” was ported back into a batch environment (for the particular purposes of implementing a simulation server THOENY *et al.*, 1994). It was possible to implement a so-called batch “Dialog Machine”, which allows to run any interactive program in a batch environment (all based on the original design).

Currently it is possible to port easily “Dialog Machine”-programs from Macintosh computers to other platforms and vice versa. We have done that several times, porting

---

[3] Computer-aided Explorative Learning and Teaching With Interactive Animated Simulation

[4] Informatik Dient Allen; Computer Science for Everybody

software packages consisting of several hundred thousand lines of source code among Sun workstations, IBM-PCs and Macintosh computers without having to change a line of source code and the results obtained with this technique, such as program behavior[5] or simulation results, were identical.

2.2 For Whom Is the “Dialog Machine” Useful?

The “Dialog Machine” has been designed for all those people who wish they could be able to program today's personal computers like the Macintosh or IBM-PCs, but who don't find the time to study phone-book-thick programer's reference manuals such as the “Inside Macintosh”. It is also for programmers with little time devotable to a small programing task; or for people who simply can't figure out how to do it, i.e. for non-computer science faculty more concerned with their own subject than computer science; for application programmers who are fed up with reprogramming for the hundredth time the same, but still slightly different event-loop all over again; and for all those programmers who have a thorough, basic programming skill, but don't understand how to access the assembly, ROM-based toolbox routines inside their Macintosh computer or BIOS etc. The only requirements to use the “Dialog Machine” are the desire to program and some basic programming knowledge in a high-level programming language which supports structured programming, e.g. Pascal.

2.3 How Does the “Dialog Machine” Work?

The “Dialog Machine” separates those program parts which are common to any interactive application program from those program sections which are application-specific. It thus simplifies the programming task of a typical Macintosh application substantially. Only the application-specific program parts need to be written by the programmer; all other program sections are provided by the “Dialog Machine”. Furthermore, the “Dialog Machine” standardizes not only the appearance, but also the behavior of an application program as far as the user dialog is concerned (for instance behavior conforms to the “Macintosh User Interface Guidelines” as published by Apple), thus helping students to orient themselves within complex courseware and reducing the time needed for learning to use new software.

The “Dialog Machine”, once started, attempts to keep control over all run-time activities of an application program. It intercepts all events due to a user action, the so-called user events, such as pressing the mouse button, choosing a menu item, activating a window, clicking an object, or dragging an object on the screen, and reacts to them in a predefined, standard way (primary reaction). Only events, which can not be treated automatically by the “Dialog Machine” are proliferated through the system, i.e. they are transformed into so-called program events, and dispatched to the application-specific program sections, where they cause the application program to perform a certain action (secondary reaction). All user events are rigorously defined and the programmer can interface his application-specific code to the user events in a structured way. For instance, the clicking in the front window, or the closing of a window, which might require some action on related objects, or the pressing of a key, which does not correspond to the choosing of a menu item, are

[5]For Unix workstations only true with restrictions. On Unix machines we have currently no interactive “Dialog Machine” implementation available, only a Batch “Dialog Machine”. If Unix workstations are involved the statement applies only inasmuch as the results the Batch “Dialog Machine” can produce, i.e. file outputs, are identical.

such events. Program control remains with the “Dialog Machine” and is only temporarily passed to application-specific program sections. Consequently, the application-specific software consists of a set of procedures which can be called in an arbitrary sequence, rather than of a conventional program block of statements (straight-line code) to be executed one after the other (Fig. 2.1).

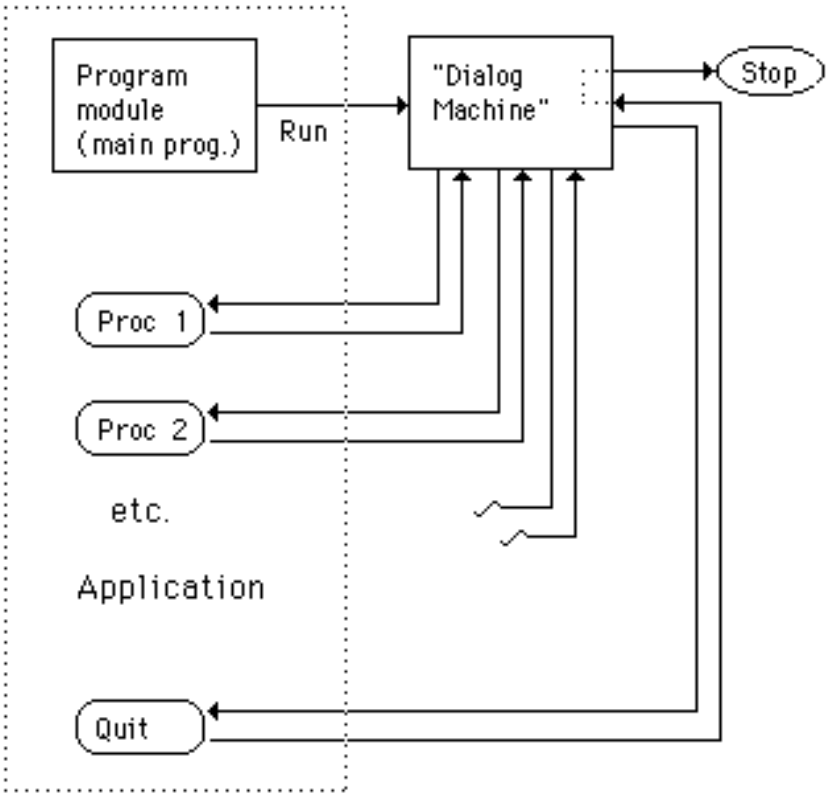


Fig. 2.1. Program control of a “Dialog Machine” program switches back and forth between the “Dialog Machine” layer and the application specific program sections. The resulting program structure is called inverted.

The resulting program structure is called inverted. The “Dialog Machine” can be considered as a versatile software development environment featuring a simplified programming of modern working stations.

2.4 How Is the “Dialog Machine” Structured?

2.4.1 Overview

The core of the “Dialog Machine” consists of nine Modula-2 library modules (DM stands for”Dialog Machine”): *DMMaster*, *DMMenus*, *DMWindows*, *DMWindIO*, *DMMessages*,

DMStorage, *DMLanguage*, *DMConversions*, and *DMSystem*. Each of these modules provides support for instantiating, inspecting, and discarding objects of the class given in

the module name. Of course these modules provide also the basic methods of operation on these objects (Fig. 2.2).

Most program are likely to need other functions too. They are provided by the so-called optional modules, which are only used as needed. They are: *DMAlerts, DMClipboard, DMEditFields, DMEntryForms, DMFiles, DMLanguage, DMMaster, DMMathLib, DMMathLF, DMLongMathLib, DMMenus, DMPrinting, DMPTFiles, DMTextFields, DMWPictIO, DMWTextIO, DMResources, DMClock, DMOpSys, DMFloatEnv, DM2DGraphs, DMPortab, and DMKeyChars*.

The core modules depend mutually on each other and must always be present in order to run the “Dialog Machine”. For instance *DMMessages* is always required at run-time for the display of error or warning messages from any of the other modules. Module *DMConversions* is a module which performs number conversions and it is required by modules like *DMWindIO*, *DMEntryForms*, *DMEditFields*, and *DMFiles*. This core requires about 165 K Bytes of disk respectively memory space.

The optional modules *DMEntryForms*, *DMEditFields*, *DMFiles*, *DMClipboard*, and *DMPrinting* can be used in addition to the core modules depending whether you wish to program dialogs (*DMEntryForms*, *DMEditFields*), read or write to files (*DMFiles*), put data into or get data from the clipboard (*DMClipboard*) or to print texts or graphics (*DMPrinting*). *DMFiles* supports sequential text files similar to the file management of Pascal. The system specific characteristics can be imported from the auxiliary module *DMSystem*.

2.4.2 Characteristics of Most Frequently Used Modules

The following contains short descriptions on the more frequently used “Dialog Machine” modules.

DMMaster. This is the master module maintaining overall control of all actions, in particular user events. User events that can be handled automatically by the “Dialog Machine” are either passed to the other modules or module *DMMaster* responds to them directly.

Some user events can't be handled by *DMMaster* alone. User events of this class require some particular function, possibly operating on objects found only within some other module than the *DMMaster*. E.g. the current mouse position is needed during dragging or the user chooses a menu command. Since the “Dialog Machine” supports only dragging within a window, the data needed by a dragging method is available only from the module dealing with window inputs and output, i.e. *DMWindIO* or currently used menus are only known to module *DMMenus*. In this case *DMMaster* does not handle the event itself, but dispatches it to other “Dialog Machine” modules, the ones responsible for the involved object class.

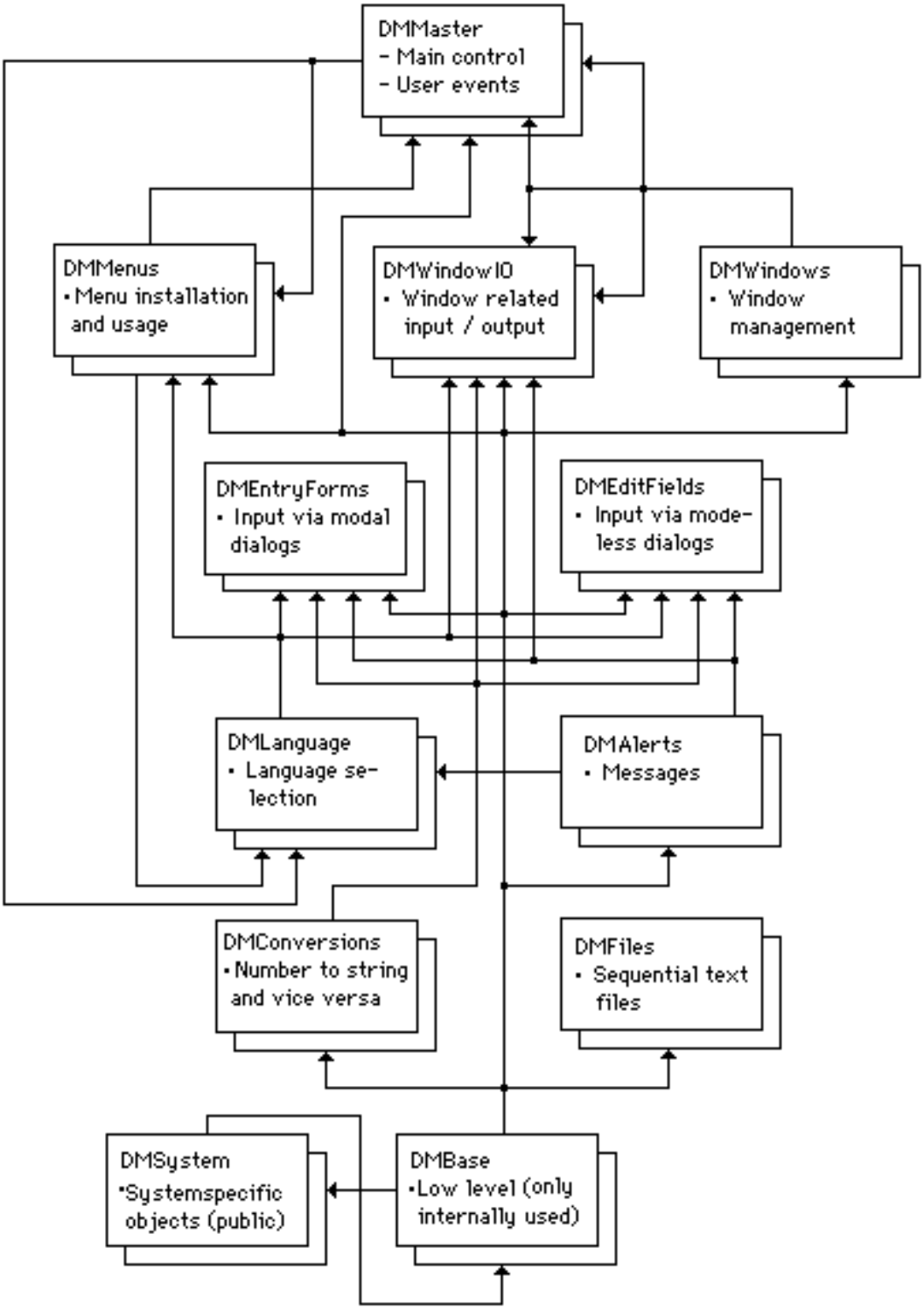


Fig. 2.2. Module structure of the first “Dialog Machine” implementation (note module DMWindowIO has in the meantime be renamed to DMWindIO for better IBM-PC compatibility).

However note, *DMMaster* functions not only as an event dispatcher, since some events like pressing a key on the keyboard is a user event, which is not related to any particular dialog object, such as e.g. a window or menu. Hence, information on user events of this type, e.g. which key has been pressed, have to be exported by module *DMMaster* directly.

The “Dialog Machine” tries to respond to as many user events as possible in an automatic manner. Of course there are also events, to which the “Dialog Machine” can't respond automatically (they are needed, because otherwise you could never program an application specific program). In these cases *DMMaster* dispatches the event to the application specific module. E.g. the module which has installed a particular menu command will receive a message to execute the function associated with a menu command as soon as the end-user of the “Dialog Machine” program chooses that menu command.

All these mechanisms are user-transparent, i.e. not only for the end-user, but also for the programmer; which means, you don't need to understand the implementational details on the event handling of a graphical user interface (GUI) in order to program it. Consequently note, this approach has also a not so obvious implication: It means, as a programer, you can't alter the dialog behavior of the program. At first sight, this may sound restricitve, but the overall result is beneficial to the end-user, since program behavior will be substantially mor robust, because all dialogs function always the same way (it's the same code).

DMMenus. This module supports the installation and management of menus. Typically, application-specific procedures are installed within the “Dialog Machine” and provide the desired action when the end-user chooses the corresponding menu item. All procedures for the activation or deactivation of menus, for changing menu texts or checking a menu item plus all other similar menu management tasks are supported in a simple and straightforward way.

DMWindows. Provides the window management. Several types of windows, which may or may not overlap, are supported: It is possible to create windows of all sorts, i.e. windows with or without scroll bars (scrolling and updating is performed automatically), with a fixed or adjustable size, with or without a close box, at a fixed screen location or movable etc. Typically, a window is created by simply calling the procedure *CreateWindow*, and all other, subsequent window related tasks, such as resizing, activating, or closing of a window, are left to the “Dialog Machine”.
DMWindIO. This module provides graphical input via the pointing device and textual or graphical output based on a small set of simple coordinate systems. The programer can use this module to detect user events, such as the clicking within the content of a window, and to relate them to graphical objects of a round or rectangular shape. Furthermore, routines to drag graphical objects once their seizing has been detected, are offered. Procedures to scroll the window content via the pointing device are also provided.
Output is always related to a coordinate system. The first output coordinate system of module *DMWindIO* supports textual output by addressing character cells in rows and columns (indexed with numbers of type CARDINAL). A second, graphical coordinate system is based on a two-dimensional cartesian coordinate system in pixel units with its origin typically at the lower left corner of the output window (referenced with numbers of type INTEGER). Thirdly turtle graphic routines are offered (pixel based coordinate system). The fourth coordinate system is the so-called user coordinate system, which is used to draw graphs (referenced by coordinates of type REAL). It maps any two-dimensional cartesian coordinate system defined in real numbers to a rectangular portion of the current output window. It is possible to freely switch from one coordinate system to the other, since they coexist all the time and are simultaneously accessible. However, for efficiency reasons, all output is actually done with one common pen only. Hence the various coordinate systems are just convenient means to access the basic underlying output mechanism. Moreover conversion routines are available and it is even possible to display a predefined picture stored as a bitmap.

DMEntryForms *and* ***DMEditFields***. These modules provide means to enter data. Any elementary data type, i.e. character (CHAR), integer (INTEGER, CARDINAL), and real (REAL), may be entered (note, booleans are supported through check boxes, sets to some extent via radio buttons). Data will automatically be checked for syntactic correctness and for whether its value lies within a range defined by the application. Of course, strings can

be entered; however, without any checking. Other dialog elements such as pushbuttons, sets of radio buttons, check boxes, and scroll bars, are supported and may be used in a simple and straightforward manner.

DMEntryForms. Features modal dialogs. They are characterized by the fact that once a modal dialog has started, the user is forced into the correct termination of this dialog, exactly as it has been foreseen by the programmer, before being able to do something else. A separate window, the so-called entry form, is opened up to handle the data entry dialogs.

DMEditFields. Allows to define editing fields for any of the elementary data types and other dialog elements within an ordinary window as managed by module *DMWindows*. Consequently, the user may interrupt or resume the modeless dialog at any moment, the same way as she/he may switch her/his attention from one window to another.

DMMessages. May be used to display warning or error messages in form of a modal dialog. The “Dialog Machine” uses this module also to display error messages.

DMFiles. Provides simple means to store or retrieve data sequentially on a disk file. It is a module which enables the accessing of files via dialogs, i.e. the selection of existing files or the creation of new files. Files can be searched or created by means of the dialog boxes familiar to any Macintosh user. All elementary data types, characters, strings, integers, cardinals, or reals, may be written or read. Several files may be accessed simultaneously.

The following modules do not perfom any dialog, yet they belong to the “Dialog Machine”, since they provide indispensable functionality or are likley to be in high demand by “Dialog Machine” programers.

DMStrings. This module exports string manipulation routines. Note, it is a module which does not perfom any dialog, still it is a core module of the “Dialog Machine”. String manipulation is idispensable, yet error-prone to program. The module is provided for the programer's convenience only (of course it is also heavily used by all “Dialog Machine” core modules themselves).

DMConversions. This module provides conversion routines for converting numbers of type CARDINAL, INTEGER, LONGCARD, LONGINTEFER, REAL, and LONGREAL to a string or vice versa. It belongs again to the same category of modules like *DMStrings*, since it does no dialog whatsoever.

DMStorage. This module provides functions to dynamically allocate (or deallocate) memory, for instance while instantiating an object, and contains also a garbage collector. Again, this module does no dialog, yet is indispensable for any object oriented programing.

DMMathLib, ***DMMathLF*** and ***DMLongMathLib*** These modules provide elementary mathematical functions. They do again no dialog, but they are likely to be in high demand by modelers and simulationists, an important clientele of the “Dialog Machine”. These modules are implemented either for highest accuracy or highest efficiency (*DMMathLF* -f for fast) of computations and has specific hardware requirements in order to function at maximum speed. *DMMathLib* and *DMMathLF* operate on reals (REAL, single precision, 32 bit) and *DMLongMathLib* operates on long reals (LONGREAL, double precision, 64 bit).

Module *DMMathLib* comes in several implementation versions, either using or ignoring SANE (Standard Apple Numerical Environment); use Finder's menu command *Get Info* to learn about the actual version of *DMMathLib.OBM* you are currently using. Module *DMMathLF* requires the presence of a FPU or it will fail entirely. Module *DMLongMathLib* provides always optimal performance, regardless of the presence of an FPU. The use of an FPU is provided by the system software since this module uses SANE throughout, which conforms to the IEEE 754 standard for floating point arithmetic (see e.g. Hough, 1981) and uses the FPU as soon as one is present in the executing computer system.

The following modules are not used often, but provide particular neat functions or are otherwise worth-mentioning, since they play a particular role for the “Dialog Machine”.

DMLanguage. This module allows for the selection of a particular language as the current language. Remember, the first “Dialog Machine” was built to support the authoring of simulation courseware for students at the Swiss Federal Institute of Technology (ETH Zurich). Switzerland knows four official languages. To account also for the needs of English speaking users, the “Dialog Machine” knows also about the existence of English (even as the default language). Use *DMLanguage* to select a particular language and all modules of the dialog machine will adjust automatically to the language chosen and the programmer may also implement another language by writing her/his own implementation for this module.

DMWPictIO and ***DMWTextIO*** These modules can be used to collect all output made by routines from module *DMWindIO* to a particular window in a so-called picture and/or text object. For instance a picture object owned by a particular window consists of all drawing output saved since a partciular start time when the saving mechanism was activated. This technique works regardless of a pictures shape, color, or size. Once output has been collected in this manner, the resulting graphical (or textual) object may be transferred somewhere, e.g. into another window, the clipboard using module *DMClipboard*, written to a file using module *DMPTFiles*, or printed using module *DMPrinting.*

DMSystem. Exports system specific objects, such as hardware dependencies. Some programers absolutely wanna know on which machine the program is running. E.g. this module makes it possible to learn about the screen resolution of the current machine type on which the “Dialog Machine” is running, how many screens are connected (Macintosh machines can have up to eight, coexisting screens connected to the same computer). For instance, a “Dialog Machine” program can distribute outputs according to screen properties, e.g. windows with color graphics to the color screen, windows with texts to the writer's black and white screen.

DMBase. Separates the “Dialog Machine” from the underlying hardware, firmware (ROM) and other system software such as the operating system. In fact this module consists of more than just one module. However, these modules are not of general interest for the “Dialog Machine” client, they rather serve the internal structure of the “Dialog Machine” itself and its portability to other machines. Hence this group of internal modules is depicted as just one pseudo module.

Detailed descriptions of all these modules and many more are provided in form of listings of their DEFINITION modules. You find all these definition modules in the folder *DM* within folder *Docu* of the RAMSES package.

3 “Dialog Machine” Basics

There is an excellent tutorial on the programing with the “Dialog Machine” available (KELLER, 1989). The following text serves only as a short reference for “Dialog Machine” programers, assuming you have read (and hopefully enjoyed) the recommendable 1.

3.1 How to Use the “Dialog Machine”

The “Dialog Machine” is used by importing any of the objects from any of the library modules into another, application specific module. The latter becomes a so-called “Dialog Machine” program you have to write. The importing compilation unit, e.g. a program module, forms the base of the application to be developed. At minimum it must contain the statement which activates the “Dialog Machine”. Hence, the simplest, still executable “Dialog Machine” program consists of five lines only:

```
MODULE Simple;
  FROM DMMaster IMPORT RunDialogMachine;
BEGIN
  RunDialogMachine;
END Simple
```

Typically, a “Dialog Machine” program consists of a set of procedures which are called, when the associated menu items or push buttons are chosen by the end-user (Fig. 2.1). Note, in this respect it is irrelevant wether the “Dialog Machine” program is a single program module or represents an ensemble of several modules. Moreover, every “Dialog Machine” program contains statements "installing" these application specific procedures into the “Dialog Machine”, so that the “Dialog Machine” actually knows about these routines and can properly dispatch user events to them. These statements are usually executed before the last statement of the program, the statement which starts the “Dialog Machine” (see below, e.g. sample program *AlmostSimple*). Such a “Dialog Machine” program requires then compilation (plus linking, depending on the Modula-2 implementation respectively computer platform) and is then ready for execution.

3.2. Basic Concepts and Frequently Used Objects

In the following section you'll find these explanations on the basic concepts of the “Dialog Machine”: how to build a menu bar with its menus, how to manage windows, what types of dialogs are available, and a summary of some of the most frequently used commands of the “Dialog Machine”.

3.2.1 MENUS

Menus are at the heart of the “Dialog Machine”, since they provide the user's main means to issue commands to the computer (for other means to accomplish the same see below, section *Dialogs*). Consequently, every “Dialog Machine” program must have an ensemble of menus, forming the so-called menu bar, which is available to the user everywhere at all times during execution of the “Dialog Machine” program.

Note, it does not matter how the actual menu technique functions, i.e. whether the menus are pop-up menus, pull-down menus etc. The “Dialog Machine” functions the same regardless of implementational details, as long as above condition is always satisfied. This enhances the portability of “Dialog Machine” programs which can be easily adapted to the so-called standard technique of the current computer platform.

This is to the advantage of the user, who is usually used to work in a particular way according to the main user interface of the machine on which he/she works. For instance, if the menu technique is a pop-up menu above requirements are met as soon as a particular command, e.g. clicking a specific mouse button, let's the user access at least one menu command, which lets him/her to quit directly or indirectly the “Dialog Machine” program. Consequently, the “Dialog Machine” doesn't let a programer write a program which has no quit command (see further explanations below).

The menu bar. The menu bar consists of menus which contain themselves so-called menu commands or submenus arranged in a hierarchical fashion. The menu bar always contains at least one menu and at least one menu command. Each menu command is associated with a particular function programmed by the “Dialog Machine” programmer. Its purpose is to perform an application specific operation (cf. Fig. 2.1).

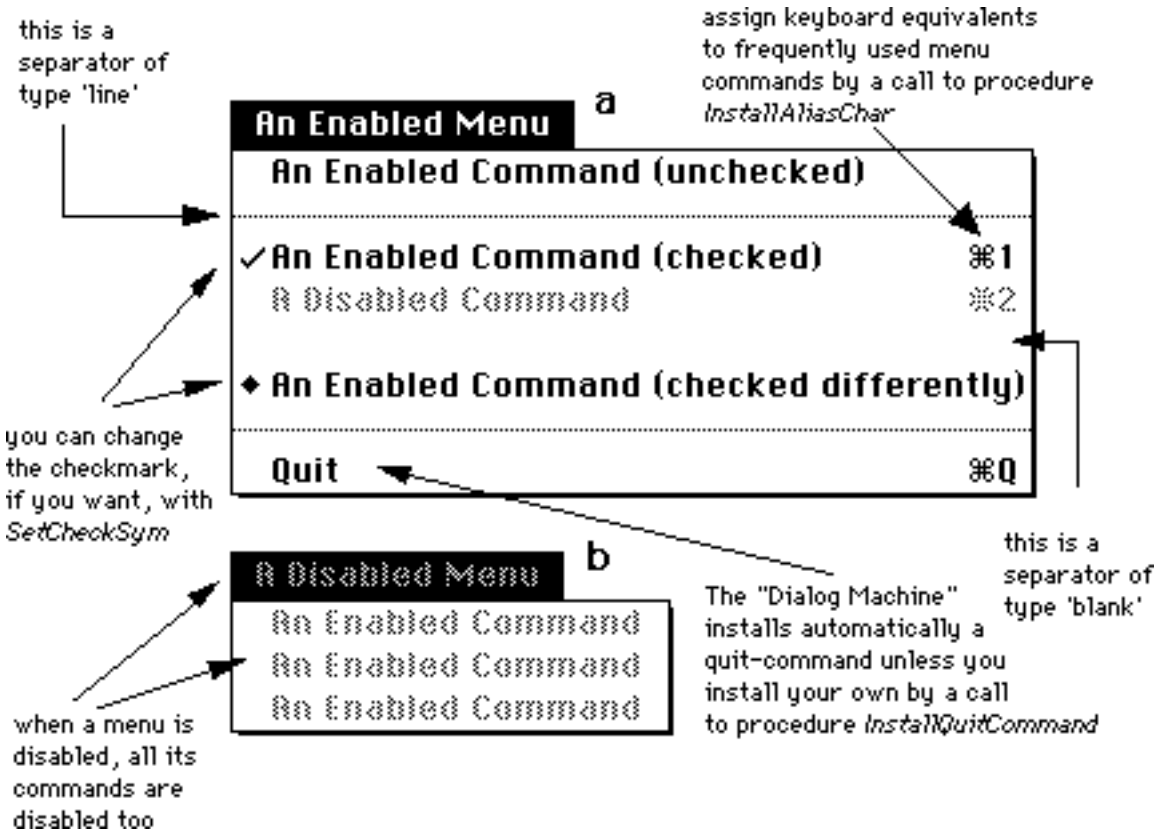


Fig. 3.1: An enabled menu (a) with two different separators and checkmarks, and a disabled menu (b) shown here in the implementation of the “Dialog Machine” for the Macintosh. Note the actual appearance of exactly the same functionality of a “Dialog Machine” program may be quite different from the one shown here. It typically adjusts to the underlying standard user interface of the computer platform on which the “Dialog Machine” program is executed.

A menu bar is built up from left to right by calls to procedure *InstallMenu* from *DMMenus*. The first call to *InstallMenu* creates the first menu, on the Macintosh to the right of the desk accessory menu, the so-called -menu. Calling procedure *InstallCommand* adds a single menu command to an existing menu. The new command is added at the very bottom of the commands already installed in the menu.

Note: With the “Dialog Machine” it is not possible to insert either a menu between two menus, or a menu command between two commands. Menus and commands are always added to the right of the menu bar, or at the bottom of a menu. Hence, to insert a menu, you have first to remove all menus to the right

of the place where you wish to insert it, and reinstall the menus removed before. Similarily, to insert a menu command, you have first to remove all the commands below and reinstall them afterwards.

A call to procedure *InstallSeparator* adds a separator (a dotted line or a blank line) at the bottom of the menu. This is useful for separating groups of related commands from each other (Fig. 3.1).

***The menu command* Quit**. Each “Dialog Machine” has to have a so-called quit command in order to give the user main control over the situation. A call to the procedure *InstallQuitCommand* adds a separator and a menu command to the current bottom of the leftmost menu (so that the user always finds this important command at the same location within the menu bar). The “Dialog Machine” retains control of program execution (inverted control structure), i.e. tries to keep the program alive forever, until the user chooses the menu command quit. If a “Dialog Machine” program never calls *InstallQuitCommand* before it calls *UseMenuBar* (or alternatively *RunDialogMachine* which implicitely calls *UseMenuBar*), the “Dialog Machine” automatically adds a quit command (again at the bottom of the leftmost menu), in order to warrant that every “Dialog Machine” program has a proper exit all the time. In case the “Dialog Machine” programmer has not even installed a menu, no quit command could be installed. In this situation the “Dialog Machine” does even install a menu (English title *Control*), since it is a prerequisite for the quit command installation (see also sample program *Simple*).

Enabling and disabling menu commands. Menu commands can be disabled (their text appears in a light grey as opposed to full black) so that they cannot be selected (Fig. 3.1). This is a very useful feature to tell the user he/she cannot choose certain commands at certain times, e.g. the command “Read data file” should be disabled when no data file is open (see Fig. 3.1); after having opened the file, the command “Read data file” should then become enabled, telling the user that this operation is now ready for execution. Turning a command on and off is done with calls to *DisableCommand* and *EnableCommand.*

Altering the availability of menu commands (or any other type of commands) actually changes the so-called state of the “Dialog Machine” program. What matters are not internal, data dependent states, but those states which matter for the dialog. The “Dialog Machine” programmer should conceptually make a difference between dialog states and internal states, which may or may not be related to each other. It is recommended to analyze in the design phase thoroughly all possible dialog states of the application, list them as a finite set (which should e kept as small as possible), and specify their properties clearly, before embarking on the implementation of the “Dialog Machine” program. What generally results are much more robust and conistently behaving progams, the users will be greatful!

3.2.2 WINDOWS

Creating a window. Windows are the main means of a “Dialog Machine” program to commuicate with the user. They are considered to be something like compupter screens, overcoming somehow the constraints given by the finite size of the physical screen. Every “Dialog Machine” window is completely independent from all other possibly existing windows and displays its content also independently from all other windows.

The actual appearance of the windows does not matter to the “Dialog Machine”, i.e. whether they overlap or are only available as tiled windows (see e.g. Oberon's viewers) etc. Advantages and disadvantages of these window managing techniques matter neither for the “Dialog Machine” nor the should they be of great concern to the programmer (albeit they may of course matter to the user).

	‘normal’ windows				modal windows	
	GrowOrShrinkOrDrag	FixedSize	FixedLocTitleBar	FixedLocation	SingleFrameShadowed	DoubleFrame
Title bar:	yes	yes	yes	no	no	no
zoom box:	yes	no	no	no	no	no
grow box:	yes	no	no	no	no	no
close box:	yes	yes	yes	no	no	no
scroll bars:	yes	yes	yes	yes	(yes)	(yes)

Tab. 3.1. Standard dialog relevant elements conatained in the frame part of a “Dialog Machine” window. Tabulated is their availability in the four basic and two modal window types of the DialogMachine. ‘(yes)’ means, that the elements can be installed, but don’t make very much sense. Dialog elements here shown in their appearance produced by the Macintosh implementation of the “Dialog Machine”.

Windows cosist of two parts: First the so-called frame part, which is generated and maintained exclusively by the “Dialog Machine”. The frame part includes typically standard dialog elements, like titles, scroll bars, grow icons etc., which let the user control window's sizes and positions etc. The second part is the one of prime interest for the “Dialog Machine” programer, it is the window's content. It is basically unlimited (see below *coordinate systems*), but of course, at a time the user can see only a section of it. This is called the working area, which is defined by a rectangle according to this data structure:

```
WindowFrame = RECORD x,y,w,h: INTEGER; END
```

x,y are the coordinates of the lower left corner of the working area given in the screen's pixel resolution, i.e. they determine the placement of the window relative to the screen area. The global coordinate system of the screen has its origin at the lower left corner of the main screen (since any number of screens with contiguous global pixel coordinates are available do a “Dialog Machine” program (see e.g. module *DMSystem*)Multiple screens). This technique to determine the location of a window is adopted by the two procedures of the “Dialog Machine” which allow to place a window on the screen, i.e. *CreateWindow* and *RedefineWindow* from module *DMWindows*.

w and h denote width and height of the working area, i.e. they determine the size of the window, again given in the screen's pixel resolution (see also section *Output to Windows*). The outer frame of the window may vary depending on the type of window, since there are

windows available with a title bar or without etc., facts which are usually of little concern to the programmer. In case the programmer wishes to place windows at particular locations relative to a screen, e.g. while tiling windows on the main screen, there are procedures available to compute from a given working area the so-called outer frame of a window and vice versa (*OuterWindowFrame*, *InnerWindowFrame* from *DMWindows*).

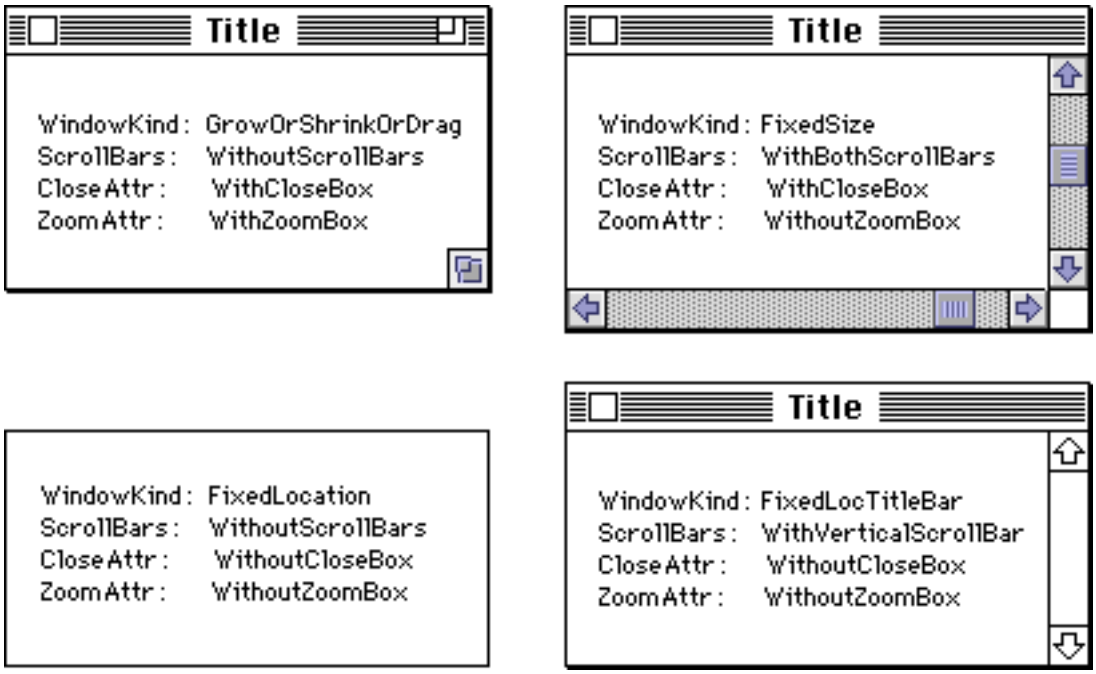


Fig. 3.2: The four basic window types provided by the “Dialog Machine”. Here shown in their appearance produced by the Macintosh implementation of the “Dialog Machine”. Note, the working area of all these windows is of exactly the same size and position (cf. Fig. 3.3), only the frame part varies depending on the number of scroll bars or other window attributes, which can be freely added or not. Note also the special case of the left window at the top, which has to sacrifice some of its working area to the grow icon.

All the types of windows which are supported by the “Dialog Machine” are listed in Tab. 3.1): Windows can have a title bar or not, they can be moveable or sizeable or both, and they can have several other functional dialog elements, like scrollbars. All this is controlled with the parameters of the *CreateWindow* command at time of creation.

Most frequently used window types are the following (see also Fig. 3.2,Tab. 3.1):

1) a very simple window which cannot be resized nor moved (dragged) to another location: **FixedLocation**
2) one that cannot be resized, but moved around: **FixedSize**
3) the same as 2) but with a title bar: **FixedLocTitleBar**
4) a window which can be dragged around and re-sized: **GrowOrShrinkOrDrag**

In addition to these four basic types, the “Dialog Machine” provides two other window types especially suited for the use as windows in modal dialogs (see Tab. 3.1 and Fig. 3.3). They are both of fixed size and position. It’s possible to use scrollbars with them, but it is not recommended and especially in the ‘DoubleFrame’ window they don’t come very well. For details about modal dialogs see also section *Dialogs*.

To make the life of programmers as easy as possible, the “Dialog Machine” provides automatic mechanisms for the restoring and scrolling of windows (*AutoRestoreProc*, from *DMWindows* and *AutoScrollProc* from *DMWindIO*). To ensure the essential restoration of windows during the entire life-time of a window, an update mechanism needs to be provided at the time of creation of the window (parameter of type *RestoreProc* of procedure *CreateWindow*). It can later be changed easily with procedure *SetRestoreProc*.

If the programmer does not like these default methods, he/she can gain full control over the updating of a particular window, however only by programming an update event handler. The handlers duty is simply to restore the window's content by redrawing it,from the application specific data (useful while a graph or diagram ought to be adjusted according to the window's current size). Once installed for a particular window, each time the “Dialog Machine” detects that the window requires restoration of its content, it will send a message to the current update handler asking it to deal with it, i.e. calls the routine. Given the programmer has properly programmed the method (update event handler) all will be fine and the window's content will be restored (see also sample program *LessSimple* which exemplifies these concepts).

For any particular window the programmer can install any number of handlers for all supported window event classes (see type *WindowHandlers* from module *DMWindows*) by calling procedure *AddWindowHandler*. Once a handler has been installed, the “Dialog Machine” will send it a message, i.e. call it, as soon as an event of the given class has been encountered. The most important event class supported is the event class *redefine*; an event of this class is triggered each time the user resizes a window.

Besides, this facility is provided for any other events which may occurr during the execution of a “Dialog Machine” program. This technique is of course at the heart of any object oriented programming technique, to which the “Dialog Machine” adheres as much as possible. In this sense, the procedure associated with a menu command is also nothing else than a handler which responds to the event triggered when the user chooses a menu command (see section *Menus*).

Cleaning and closing windows. The contents of the current output window can be blanked (cleaned) by using the command *EraseContent* (note, *EraseContent* does also clean any drawing possibly hidden from the user, since it took place outside of the current visible area of the window content; this behavior matters as soon as a window's content can be scrolled).

Closing and removing a window from the screen can be programmed by a call to procedure *RemoveWindow*. A removed window has ceased to exist (instantiation nil). Of course, the same effect has the closing of a window by the user, e.g. on the Macintosh by clicking into the so-called close box. Since a window may be associated with particular data or objects, all application specific data might also have to be discarded while a window is closed. Again, the installation of a window handler for the event class *closing* is the recommended technique to program a proper handling of this situation.

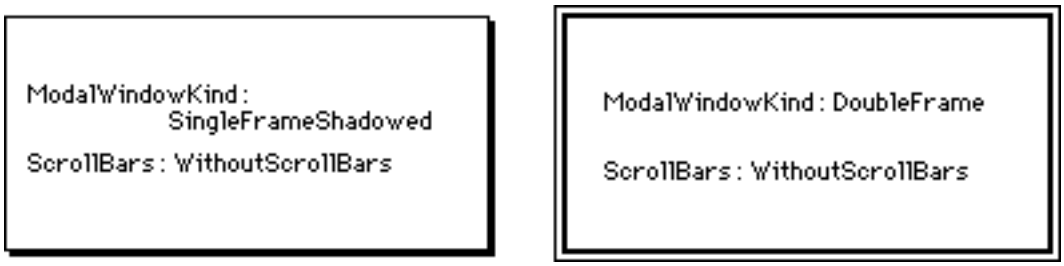


Fig. 3.3: The two modal window types provided by the “Dialog Machine”. Here shown in their appearance produced by the Macintosh implementation of the “Dialog Machine”. Note, the working area of both windows is of exactly the same size and position (is also true for the windows depicted in Fig. 3.2).

3.2.3 OUTPUT TO WINDOWS

The concept of the current output window. The “Dialog Machine” always knows a current output window. All output takes place in this window only. A newly opened window automatically becomes the current output window. With multiple windows it is possible to direct the output to the desired window by using the procedure *SelectForOutput*. This defines the new current output window. Note that the current output window needs not be the top (front) window, allowing to make output to any window regardless of its position relative to the other windows (see also the definition modules of *DMWindow* and *DMWindIO*)

The coordinate system. All drawing and writing into a window is done by referencing one of the window’s coordinate systems.

The basic coordinate system is a pixel based coordinate system (coordinates are of type INTEGER). The origin of this coordinate system lies in the lower left (for ‘bottomLeft’-windows) or in the upper left corner (for ‘topLeft’-windows) of the window. Note, these coordinates apply to the **points between the actual pixels** (Fig. 3.4). The pixel drawn on screen which is associated to such a coordinate is the pixel to the right below the defined point. This principle makes it generally easier to do calculations with coordinates, but in a few cases makes it more difficult e.g. to frame a rectangular area correctly (try to figure out why).

Drawing can be done at every point defined by the coordinate system. Therefore drawing can also take place outside of the window. It will actually be done (which might be important if you scroll the windows content), i.e. execution of such drawing will use up computing time, but the user sees nothing, since the “Dialog Machine” always correctly clips all this drawing exactly at the boundaries of the working area (note, it is not possible to draw into a scroll bar or the grow icon (especially important if you don't have any scroll bars in a window) in contrast to other graphical user interfaces like the Macintosh's toolbox).

An other coordinate system (coordinates are of type CARDINAL) available in the same window refers to character cell coordinates (see next paragraph). For this system you may think in terms of rows and columns, like in a matrix (numbering also follows the conventions of matrix row and column numbering; e.g. leftmost, topmost character cell has the coordinates [1,1]). You can get the height and width of these character cells by calling *DMWindIO.CellHeight()* and *DMWindIO.CellWidth()*. These values are dependent of the current window font.

Actually both coordinate systems apply at the same time. But depending on the type of output you want to make on a window (textual or graphic) the one or the other orientation system is more convenient.

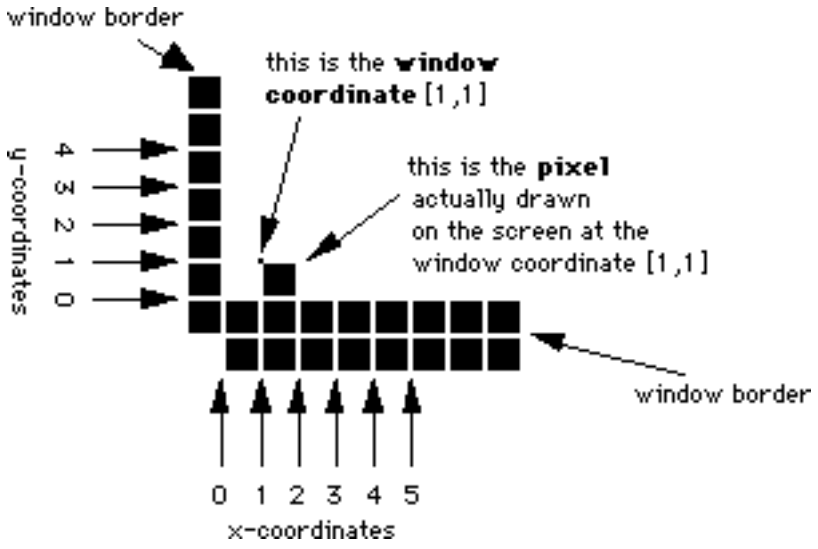


Fig. 3.4: Illustration of the relationship between the window coordinate system of a ‘bottomLeft’-window (never scrolled) and the pixels drawn on the screen (which is what you actually see). Note, all drawing done via the “Dialog Machine” occurs in form of pixels, i.e. squares of a defined size, which extends to the right and below the point specified by the coordinate. Points defined by coordinates are real points in the mathematical sense and have no extension or size. In most cases, this technique allows for easier computations of coordinates and only rarely matters. One exception is the situation shown here, where a call to *DMWindIO.Dot(0,0)* results in no output, since the associated pixel coincides with the window frame and is therefore clipped, despite the fact that the point [0,0] is considered to belong to the window's working area.

A third coordinate system, the so-called user coordinate system, is based on a rectangular graph panel. Although the panel is placed within the window by using the INTEGER based pixel coordinate system, points within the panel are defined by real numbers, which are interpreted as cartesian coordinates (coordinates are of type REAL). The panel is specified by a scale for the bottom horizontal (x-axis) and the left vertical (y-axis) borders of the rectangular graph panel, an interpretation which has to be specified by the user (hence the name).

Finally *DMWindIO* provides also means for so-called turtle graphics, which support drawing in a manner which resembles the working with polar coordinates.

Output to a window. Actual drawing is always done with a single pen . The pen can be placed anywhere in the window content (i.e. inside or outside the window's frame) by using *SetPen* (pixel coordinates X, Y) or *SetPos* (character cell coordinatesLine/Column). All subsequent drawing, writing or painting starts then from the point to which the pen has been moved. In general, **all output routines move the pen position** (except for routine *Dot*), and they do so irrespective of the coordinate system by which the pen's position is given.

Each window has a single pen, which means, this pen is shared among all coordinate systems. Thus, the coordinate systems are nothing else than abstract methods of referencing the pen from different views of actually the very same thing, i.e. the window's content. As a “Dialog Machine” programer you can easily switch from one coordinate system to the other, depending what's currently the most convenient view. There are also routines available, which allow for translating a particular point from every coordinate system to any other (of course, if the destination system is the character cell based

coordinate system, you loose some information due to the discretization of this coordinate system).

Textual output. The “Dialog Machine”provides the usual routines for ASCII output, like: *WriteString, WriteInt, WriteReal, Write, WriteLn* etc., starting from the current pen position.

The current font with its caracteristics (default: plain Geneva 12 on a Mac, the System font on a PC) can be changed with *SetWindowFont.*

Drawing primitives. Drawing of simple graphics can be achieved with the routines *Dot, LineTo* (from the current to the new location), *Circle* and *Area* (paints a rectangular area). The pattern with which all drawing takes place can be changed with *SetPattern.* Procedure *MapArea* is a bit block transfer routine.

Displaying pictures prepared with a graphics tool such as MacPaint . To show more elaborate pictures it is recommended to paint them first with a painting or drawing application such as MacPaint or any other graphics program. Once completed, the picture can be copied to the clipboard (or scrapbook) and from there into a resource file (use the programs *DMResMover* or *ResEdit* to actualy transfer the pictures from the drawing program into a ressource file accessible by the “Dialog Machine”). Simply call routine *DisplayPredefinedPicture* to display the picture in a “Dialog Machine” window.

3.2.4 DIALOGS

Modal dialogs. In a modal dialog the user is forced to terminate the dialog by using one of the exit buttons (usually labeled “CANCEL” and “OK”). No other action outside the dialog box can be taken, i.e. the user is forced into “a mode”.

Modeless dialogs. The modeless dialog on the other hand offers the user the choice to terminate the dialog or to do something else, e.g. selecting another window or choosing a menu command. The modeless dialog requires more effort from the side of the programmer than the modal dialog, although it is friendlier to the user because it leaves more choices open rather than forcing the user to do something particular. However, modal dialogs may increase the reliability of program behavior, since the programing is much more straightforward and therefore less errror-prone.

An example for a modeless dialog is the “find string” box in a word processing program: You can enter a search string, search for the occurrences of the string, or just click into the text window and continue editing without having to terminate the search dialog by pressing “OK”. In contrast, a modal dialog gives the user only a binary choice: Either the user abandons completely the dialog (aborts it - “CANCEL”) or fully completes it (“OK”). There is no inbetween like in a modeless dialog, where editing can temporarily be interrupted, for instance to check some other data, before resuming it where left.

	icons	action	remarks
Check boxes	☒ Checkbox	toggles between checked and unchecked	Toggle
Radio buttons	○ Radio Button 1 ◉ Radio Button 2	selects on of a series of radio buttons	One of a series
Text/String entry fields		enter and edit text and/or numbers	
Default button	OK	closes the dialog window and saves the new values	Is also called by pressing the Return or Enter key.
Cancel button	Cancel	closes the dialog window and discards the new values but keeps the old ones	Is also called by presssing Cmd-. or escape.
Push buttons	Push Button	specific action associated to the button	

Tab. 3.2. Dialog Elements provided by the “Dialog Machine”(modules *DMEntryForms* or *DMEditFields*) and their characteristics

Entry forms (Modal dialog boxes). The module *DMEntryForms* exports procedures to program easily modal dialogs: *StringField, RealField, RadioButton, CheckBox,* etc. (see also the sample programs *FontStyleTest* and *PaintModes* in Keller, 1989). Basically you program a form of a given format, which will contain some fields into which you can fill in information. Typically you label these editable fields and define the type of data you expect the user to enter into those fields, e.g. a positive real number or an integer within a particular range. Once the form is fully defined, the “Dialog Machine” program passes then this form to the “Dialog Machine” by calling procedure *UseEntryForm*. The “Dialog Machine” presents then the entry form to the user as a so-called dialog box by adding at the bottom of the form two push buttons labeled “CANCEL” and “OK”. The “Dialog Machine” enters a "mode", because it requests the user to fill in all fields before the user is "allowed" to do anything else (unless the user pulls the power plug). In this mode the “Dialog Machine” handles all dialog automatically, which includes the editing of numbers or strings, clicking of check boxes, skipping to the next editable field with the TAB key (or backwards with Shift^TAB), transferring data into or from the clipboard, scrolling etc.

The modal dialog can only be exited by pushing one of the two push buttons (“CANCEL” or “OK”) at the bottom of the form. Syntax or range errors are all taken care of by the “Dialog Machine”, but only when the user attempts to exit the dialog by pushing button “OK”. Pushing the latter is interpreted by the “Dialog Machine” as if the user says: All is fine now, the entire form is "OK". The “Dialog Machine” checks now the information the user has provided and any field containing data which violates the specifications, will have to be fixed by the user, or the modal dialog can't be exited. If the user doesn't want to do that, he/she has an alternative (instead of pulling the plug): button “CANCEL”. However, exiting an entry form with “CANCEL” has exactly the same effect as if *UseEntryForm* would never have been called: Any editing, whether it's partly

valid or not, is discarded and the “Dialog Machine” restores all data involved in the form to exactly the same state they had before the dialog.

Consequently, when returning from procedure*UseEntryForm* the programer can be certain that all values passed to the entry form for editing have now valid values, given they had also valid values before (this post condition of a modal dialog is usually quite different from the one resulting from modeless dialogs).

Window based dialogs (Modeless dialog boxes). Modeless dialogs can be constructed by using a window and placing dit fields into it (see module *DMEditFields*). Edit fields are designated areas within the window, which allow the user for editing data like numbers or strings etc. Some of these edit fields operate on elementary data types, like an on-off switch or check box operating on a boolean variable, or selecting a single element from an entire set (radio buttons) etc. (Tab. 3.2).

For instance the procedure *MakePushButton* (from *DMEditFields*) places a push button into a window, a type of dialog element often used in modeless dialogs. Push buttons are special in the sense, that they do not operate on data as most of the other dialog elements, but have the same purpose as menu commands, i.e. are a means for the user to issue a command. The push button is also associated with a procedure, which the “Dialog Machine” calls as soon as the user presses the button, i.e. releases the mouse button while inside the push button's frame.

3.2.5 File I/O and Other Non-Dialog Aspects

The “Dialog Machine” supports besides above summarized dialog elements also more traditional programing techniques. The latter are necessary, because almost every program needs them and the “Dialog Machine” is also a platform independent, general purpose programing environment, which does abstract from any particular hardware or system software properties.

File input/output is at the margin of dialog. *DMFiles* exports two dialog based routines provide access to files, *GetExistingFile* and *CreateNewFile*. The first lets the user open a file which must already exist on the secondary mass storage, typically a hard disk, and lets the user locate the file somewhere in the hierarchical file system. The second routine lets you create a new file by allowing the user to name the new file and to place it somewhere in the file system. Possibly needed dialogs which let the user decide on the overwriting of existing files etc. are of course all also left to the “Dialog Machine”.

All other routines from module *DMFiles* are without any dialogs. They provide means to read from a file or write to a file (in a fashion hidden from the user). A file is considered a sequential data structure similar to the file concepts contained in Pascal (however, random access is also supported).

Finally modules like *DMConversions* or *DMStrings* provide means to convert numbers into strings and vice versa and general purpose string handling routines. Consult the definition modules for details.

For an overview over all other modules see also the quick reference listings of all definition modules which form part of the “Dialog Machine” at the end of this document. To learn about the behaviour of particular routines read the comments in the definition modules contained in folder *Docu* of the RAMSES release package.

4 Learning By Example

As mentioned earlier, there is an excellent tutorial on the programing with the “Dialog Machine” available (KELLER, 1989), which is highly recommendable if you wish to learn how to actually write “Dialog Machine” programs. However, in order to get a first glimpse at how “Dialog Machine” programs are structured and look like, the following sample programs and the given explanations may be more useful.

4.1 Sample Programs

The following sample programs introduce and explain step by step the principles behind the “Dialog Machine”. In particular watch for the inverted control structure.

4.1.1 *SIMPLE*

This is the simplest possible program that uses the “Dialog Machine”. It only installs a new menu with a quit-command. This is always done by the “Dialog Machine” if you don't install any other menus with quit-commands to ensure, that you at least can leave the program.

```
MODULE Simple;
  FROM DMMaster IMPORT RunDialogMachine;
BEGIN
  RunDialogMachine;
END Simple
```

4.1.2 *ALMOSTSIMPLE*

In addition to the program “Simple” the menu is used to install a command that invokes an action. The use of alerts is also showed.

```
MODULE AlmostSimple;  (*A.Fischlin, 26/Jun/86*)

  (*************************************************************

     This program module demonstrates how to install a menu
     and a command before starting the "Dialog Machine" and
     the mechanism to provide an application specific action,
     here the display of the waiting symbol during a delay.
     Furthermore the program demonstrates how to produce
     by means of a so-called alert a message telling the
     user that the application specific action has been
     terminated.

  *************************************************************)

  FROM DMAlerts IMPORT ShowAlert, WriteMessage;

  FROM DMMenus IMPORT Menu, InstallMenu, Command, InstallCommand,
    AccessStatus, Marking;

  FROM DMMaster IMPORT RunDialogMachine,
    ShowWaitSymbol, Wait, HideWaitSymbol;


  VAR
    myMenu: Menu;
      aCommand: Command;


  PROCEDURE HiImFinished;
  BEGIN
    WriteMessage(2,4,"Hello!  I'm finished.");
```

```
  END HiImFinished;

  PROCEDURE TheBigAction;
    VAR i: INTEGER;
  BEGIN
    FOR i:= 1 TO 10 DO
      ShowWaitSymbol;
      Wait(60) (* ª 1 second *);
    END;
    HideWaitSymbol; ShowAlert(3,35,HiImFinished);
  END TheBigAction;


  PROCEDURE InitDM;
  BEGIN
    InstallMenu(myMenu,"Do something",enabled);
    InstallCommand(myMenu,aCommand,"boring...",TheBigAction,
                   enabled,unchecked);
  END InitDM;


BEGIN
  InitDM;
  RunDialogMachine;
END AlmostSimple
```

4.1.3 *LESS SIMPLE*

This program demonstrates window management, use of window update procedures an drawin within windows.

```
MODULE LessSimple; (*A.Fischlin, Mai 86*)

  (*************************************************************)
  (*   Sample program module demonstrating the installation    *)
  (*   of menus, the window management, inclusive content      *)
  (*   restoration and some drawing within the window as       *)
  (*   supported by the "Dialog Machine"                       *)
  (*************************************************************)


  FROM DMMenus IMPORT  Menu, Command, AccessStatus, Marking,
    InstallMenu, InstallCommand, InstallAliasChar, Separator,
    InstallSeparator, DisableCommand, EnableCommand,
    ChangeCommandText;

  FROM DMWindows IMPORT Window, WindowsDone, notExistingWindow,
    WindowKind, ScrollBars, CloseAttr, ZoomAttr, WFFixPoint,
    WindowFrame, CreateWindow, SetRestoreProc, DummyRestoreProc,
    AutoRestoreProc, GetWindowFrame;

  FROM DMWindIO IMPORT SelectForOutput, Circle, Pattern;

  FROM DMMaster IMPORT MouseHandlers, AddMouseHandler,
    AddSetupProc, RunDialogMachine;



  TYPE
    MachineStates = (myWindowDoesNotExist,
                     myWindowExistsButNoAutomaticUpdating,
                     myWindowExistsWithRestoreUpdating,
                     myWindowExistsWithAutoRestoreUpdating);


  VAR
    myMenu: Menu;
    makeWindow, drawCircle, ordUpdating, autoUpdating: Command;
    myWindow: Window; wf: WindowFrame;
    curDMState: MachineStates;


  PROCEDURE CircleRestoreProc(u: Window);
    VAR radius: INTEGER; filled: BOOLEAN; dummyPat: Pattern;
    PROCEDURE Minimum(x,y: CARDINAL): CARDINAL;
    BEGIN (*Minimum*)
      IF x<y THEN RETURN x ELSE RETURN y END
    END Minimum;
```

On the “Dialog Machine”

```
BEGIN (*CircleRestoreProc*)
  GetWindowFrame(u,wf);
  radius:= Minimum(wf.h DIV 3,wf.w DIV 3);
  filled:= FALSE;
  Circle(wf.w DIV 2,wf.h DIV 2,radius,filled,dummyPat)
END CircleRestoreProc;

PROCEDURE DrawCircle;
BEGIN
  SelectForOutput(myWindow);
  CircleRestoreProc(myWindow);
END DrawCircle;



CONST
  clRPStr = "Install your own restore procedure";
  auRPStr = "Install DM's automatic restoring mechanism (AutoRestoreProc)";
  rmClRPStr = "Remove your own restore procedure";
  rmAuRPStr = "Remove automatic restoring";

PROCEDURE SetDMState(s: MachineStates);
BEGIN
  CASE s OF
    myWindowDoesNotExist:
            myWindow:= notExistingWindow;
            EnableCommand(myMenu, makeWindow);
            DisableCommand(myMenu, drawCircle);
            DisableCommand(myMenu, ordUpdating);
            DisableCommand(myMenu, autoUpdating);
  | myWindowExistsButNoAutomaticUpdating:
            DisableCommand(myMenu, makeWindow);
            EnableCommand(myMenu, drawCircle);
            EnableCommand(myMenu, ordUpdating);
            EnableCommand(myMenu, autoUpdating);
            SetRestoreProc(myWindow,DummyRestoreProc);
            ChangeCommandText(myMenu,ordUpdating,clRPStr);
            ChangeCommandText(myMenu,autoUpdating,auRPStr);
  | myWindowExistsWithRestoreUpdating:
            DisableCommand(myMenu, makeWindow);
            DisableCommand(myMenu, drawCircle);
            EnableCommand(myMenu, ordUpdating);
            DisableCommand(myMenu, autoUpdating);
            SetRestoreProc(myWindow,CircleRestoreProc);
            ChangeCommandText(myMenu,ordUpdating,rmClRPStr);
            ChangeCommandText(myMenu,autoUpdating,rmAuRPStr);
  | myWindowExistsWithAutoRestoreUpdating:
            DisableCommand(myMenu, makeWindow);
            EnableCommand(myMenu, drawCircle);
            DisableCommand(myMenu, ordUpdating);
            EnableCommand(myMenu, autoUpdating);
            SetRestoreProc(myWindow,AutoRestoreProc);
            ChangeCommandText(myMenu,ordUpdating,rmClRPStr);
            ChangeCommandText(myMenu,autoUpdating,rmAuRPStr);
  END(*CASE*);
  curDMState:= s;
END SetDMState;


PROCEDURE MakeWindow;
BEGIN
  wf.x:= 50; wf.y:= 50; wf.w:= 200; wf.h:= 200;
  CreateWindow(myWindow,
               GrowOrShrinkOrDrag, WithoutScrollBars,
               WithCloseBox, WithoutZoomBox, bottomLeft,
               wf, "My Window", DummyRestoreProc);
  IF WindowsDone THEN
    SetDMState(myWindowExistsButNoAutomaticUpdating)
  END(*IF*);
END MakeWindow;

PROCEDURE EnableMenuIfWindowCloses(u: Window);
BEGIN
  SetDMState(myWindowDoesNotExist);
END EnableMenuIfWindowCloses;



PROCEDURE ToggleUpdtInstallation;
BEGIN
  IF curDMState = myWindowExistsButNoAutomaticUpdating THEN
    SetDMState(myWindowExistsWithRestoreUpdating);
  ELSIF curDMState = myWindowExistsWithRestoreUpdating THEN
    SetDMState(myWindowExistsButNoAutomaticUpdating);
  END(*IF*);
END ToggleUpdtInstallation;
```

```
  PROCEDURE ToggleAutoUpdtInstallation;
  BEGIN
    IF curDMState = myWindowExistsButNoAutomaticUpdating THEN
      SetDMState(myWindowExistsWithAutoRestoreUpdating);
    ELSIF curDMState = myWindowExistsWithAutoRestoreUpdating THEN
      SetDMState(myWindowExistsButNoAutomaticUpdating);
    END(*IF*);
  END ToggleAutoUpdtInstallation;



  PROCEDURE SettingUp;
    (* Menus are now installed in the "Dialog Machine", since this
    procedure will be called automatically after you have activated
    it by calling DMMaster.RunDialogMachine.  Any menu command texts
    etc. may now be changed the same way as during the ordinary
    running of the "Dialog Machine" *)
  BEGIN
    SetDMState(myWindowDoesNotExist);
  END SettingUp;

  PROCEDURE DMInitialization;
    (* This procedure is called in order to install menus etc. into the
    "Dialog Machine" before it is actually activated by calling procedure
    DMMaster.RunDialogMachine *)
  BEGIN
    InstallMenu(myMenu,"Control",enabled);
    InstallCommand(myMenu, makeWindow,"Open Window", MakeWindow,
                   enabled, unchecked);
    InstallCommand(myMenu,drawCircle,"Draw Circle", DrawCircle,
                   disabled,unchecked);
    InstallAliasChar(myMenu, drawCircle,"D");
    InstallSeparator(myMenu,line);
    InstallCommand(myMenu,ordUpdating,clRPStr,
                   ToggleUpdtInstallation,disabled,unchecked);
    InstallCommand(myMenu,autoUpdating, auRPStr,
                   ToggleAutoUpdtInstallation,disabled,unchecked);
    AddSetupProc(SettingUp, 0);
    AddMouseHandler(CloseWindow,EnableMenuIfWindowCloses, 0);
  END DMInitialization;


BEGIN
  DMInitialization;
  RunDialogMachine
END LessSimple.
```

4.1.4 *RANDOM*

The following sample program is a program as it is typically written using the “Dialog Machine”. The structure of the program essentially reflects the structure of the main menu. Comments have been inserted for better readibility.

The program serves the demonstration of uniformly distributed variates (pseudo random number generation). It has been designed to be used during lectures while the teacher explains the basic principles of linear congruential random number generators. It is possible to define an ad-hoc generator, so that students can provide the values for multiplicator and modulus and immediately observe the resulting behaviour.

In addition to the “Dialog Machine” the module “Randoms” is required. It contains a predefined random number generator specifically tailored to the Macintosh (full period linear congruential generator with prime modulus 2**16 -1 and a multiplicator tested for optimal statistical properties). Module “Randoms” also exports routines to seed the generator and to inspect the current value of the integer series on which the uniformly distributed variates are based.

```
MODULE Random;   (* A.Fischlin  30/6/86 *)
```

```
FROM DMMenus IMPORT Menu, Command, AccessStatus, Marking,
                    InstallAbout,
                    InstallMenu, InstallCommand, InstallAliasChar,
                    Separator, InstallSeparator,
                    InstallQuitCommand,
                    DisableCommand, EnableCommand,
                    ChangeCommandText;

FROM DMWindows IMPORT Window, notExistingWindow,
                      WindowKind, ScrollBars,
                      CloseAttr, ZoomAttr, WFFixPoint,
                      WindowFrame,
                      CreateWindow,
                      AutoRestoreProc, DummyRestoreProc,
                      GetWindowFrame, WindowExists,
                      RemoveWindow;

FROM DMWindIO IMPORT SelectForOutput,
                     ScaleUC, UCDot, UCFrame,
                     SetPen, CellHeight, CellWidth,
                     EraseUCFrameContent,
                     BackgroundWidth, BackgroundHeight,
                     SetPos, WriteReal, Write, WriteString, WriteLn,
                     EraseContent;

FROM DMMaster IMPORT MouseHandlers, InstallMouseHandler,
                     InstallSetUpProc, RunDialogMachine,
                     DialogMachineTask;

FROM DMEntryForms IMPORT FormFrame, WriteLabel, DefltUse,
                         CardField,
                         RadioButtonID, DefineRadioButtonSet, RadioButton,
                         UseEntryForm;

FROM DMAlerts IMPORT WriteMessage, ShowAlert;

FROM SYSTEM IMPORT LONG, VAL;

FROM Randoms IMPORT Seed, GetZ, U;



(**********************)
(* About this program *)
(**********************)

PROCEDURE AboutProc;
BEGIN
  SetPos(2,1);
  WriteString("                                RANDOM"); WriteLn;
  WriteString("     Die Erzeugung von Pseudozufallszahlen"); WriteLn;
  WriteString("                  © Andreas Fischlin, ETHZ"); WriteLn;
  WriteString("                          07/April/1987"); WriteLn; WriteLn;
  WriteString("   This program may be freely copied as long"); WriteLn;
  WriteString("   as it is not used for commercial purposes"); WriteLn;
END AboutProc;



(****************************)
(* DM referencing variables *)
(****************************)

VAR
  myMenu: Menu;
  makeWindows, randGens, oneDot, setPars, seed, clear, quit: Command;
  graphW: Window; wf: WindowFrame; dataW: Window;



(************************************************)
(* program states and state transition procedure *)
(************************************************)

TYPE
  MachineStates = (noWind, withWindsNoRandGen, withWindsAndRandGen);

VAR
  curDMState: MachineStates;

PROCEDURE SetDMState(s: MachineStates);
BEGIN
  CASE s OF
    noWind: IF WindowExists(graphW) THEN RemoveWindow(graphW) END;
```

```
            IF WindowExists(dataW) THEN RemoveWindow(dataW) END;
            EnableCommand(myMenu, makeWindows);
            DisableCommand(myMenu, randGens);
            DisableCommand(myMenu, oneDot);
            EnableCommand(myMenu, setPars);
            EnableCommand(myMenu, seed);
            DisableCommand(myMenu, clear);
  | withWindsNoRandGen:
            DisableCommand(myMenu, makeWindows);
            EnableCommand(myMenu, randGens);
            EnableCommand(myMenu, oneDot);
            ChangeCommandText(myMenu,randGens,
                              "Starte kont. Zufallszahlengeneration");
            EnableCommand(myMenu, setPars);
            EnableCommand(myMenu, seed);
            EnableCommand(myMenu, clear);
  | withWindsAndRandGen:
            DisableCommand(myMenu, makeWindows);
            EnableCommand(myMenu, randGens);
            ChangeCommandText(myMenu,randGens,
                              "Stoppe kont. Zufallszahlengeneration");
            DisableCommand(myMenu, oneDot);
            DisableCommand(myMenu, setPars);
            DisableCommand(myMenu, seed);
            EnableCommand(myMenu, clear);
            (* SelectForOutput(graphW);
            EraseUCFrameContent; *)
  END(*CASE*);
  curDMState:= s;
END SetDMState;



(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)


(*********************************************************************)
(* Global objects: constants, variables, a random number generator *)
(* and some ouput procedures                                         *)
(*********************************************************************)

CONST
  seed0 = 1D;

VAR
  x,y: REAL; z1,z2: LONGINT;
  curU: PROCEDURE (): REAL;
  curGetZ: PROCEDURE (VAR LONGINT);


MODULE AdHocGenerator; (************************************************)

  EXPORT AdHocU, SetParams, GetParams, AdHocSeed, adHocSeed0, AdHocGetZ;

  CONST adHocSeed0 = 30000;

  VAR z: CARDINAL;  A,M: CARDINAL;

  PROCEDURE SetParams(multiplier,modulus: CARDINAL);
  BEGIN
    A:= multiplier;  M:= modulus;
  END SetParams;

  PROCEDURE GetParams(VAR multiplier,modulus: CARDINAL);
  BEGIN
    multiplier:= A;  modulus:= M;
  END GetParams;

  PROCEDURE AdHocU(): REAL;
  BEGIN
    z:= A*z MOD M;
    RETURN FLOAT(z)/FLOAT(M)
  END AdHocU;

  PROCEDURE AdHocSeed(z0: CARDINAL);
  BEGIN
    z:= z0;
  END AdHocSeed;

  PROCEDURE AdHocGetZ(VAR zz: LONGINT);
  BEGIN
    zz:= z;
  END AdHocGetZ;

BEGIN
  AdHocSeed(adHocSeed0); SetParams(7,31);
```

```
END AdHocGenerator; (*************************************************)

PROCEDURE ResetGlobVars;
BEGIN
  x:= 0.0; y:= 0.0; curGetZ(z1); z2:= 0D;
END ResetGlobVars;

PROCEDURE ResetRandGen;
BEGIN
  Seed(seed0); AdHocSeed(adHocSeed0);
  ResetGlobVars;
END ResetRandGen;


PROCEDURE Clear(u: Window);
BEGIN
  SelectForOutput(u);
  EraseContent;
END Clear;


PROCEDURE ScaleGraph;
  CONST m = 35;
  VAR wf: WindowFrame; lm,bm: CARDINAL; lmlab,bmlab: INTEGER;
BEGIN
  GetWindowFrame(graphW,wf);
  wf.x:= m; wf.y:= m;
  wf.w:= wf.w - 7*m DIV 4; wf.h:= wf.h - 7*m DIV 4;
  SelectForOutput(graphW);
  ScaleUC(wf,0.0,1.0,0.0,1.0);
  UCFrame;
  GetWindowFrame(graphW,wf);
  bm:= m - CellHeight(); bmlab:= bm; bmlab:= bmlab-CellHeight() DIV 3;
  SetPen(m,bm); Write("0");
  SetPen(wf.w-3*m DIV 4-CellWidth()*2 DIV 3,bm); Write("1");
  SetPen((wf.w) DIV 2,bmlab); Write("X");
  lm:= m-(3*CellWidth() DIV 2); lmlab:= lm; lmlab:= lmlab-CellWidth() DIV 2;
  SetPen(lm,m); Write("0");
  SetPen(lm,wf.h-3*m DIV 4-CellHeight()*2 DIV 3); Write("1");
  SetPen(lmlab,(wf.h) DIV 2); Write("Y");
END ScaleGraph;


PROCEDURE DocDotData(u: Window);
  CONST
    le = 8;  dig = 5;
  PROCEDURE WriteLongInt(x: LONGINT; n: CARDINAL);
    VAR i: CARDINAL; x0: LONGCARD;
      a: ARRAY [0..11] OF CHAR;
  BEGIN (*WriteLongInt*)
    i := 0; x0 := ABS(x);
    REPEAT a[i] := VAL(CHAR,x0 MOD 10D + LONG(60B));
      x0 := x0 DIV 10D; INC(i)
    UNTIL x0 = 0D;
    IF x < 0D THEN a[i] := "-"; INC(i) END ;
    WHILE n > i DO
      DEC(n); Write(" ")
    END ;
    REPEAT DEC(i); Write(a[i]) UNTIL i = 0
  END WriteLongInt;

BEGIN (*DocDotData*)
  SelectForOutput(u);
  EraseContent;
  SetPos(1,6); WriteString("Z(k)");
  SetPos(2,1);
  WriteLongInt(z1,14);
  SetPos(3,1);
  WriteLongInt(z2,14);
  SetPos(1,19); WriteString("U(k)");
  SetPos(2,15);
  WriteString("X:  "); WriteReal(x,le,dig);
  SetPos(3,15);
  WriteString("Y:  "); WriteReal(y,le,dig);
END DocDotData;


(*********************************)
(* Menu command: "Öffne Fenster" *)
(*********************************)

PROCEDURE MakeWindows;
BEGIN
  ResetGlobVars;
  wf.x:= 25; wf.y:= 25; wf.w:= 250; wf.h:= 250;
  CreateWindow(graphW,
               GrowOrShrinkOrDrag,WithoutScrollBars,
```

```
                WithCloseBox,WithoutZoomBox,bottomLeft,wf,
                'Pseudozufallszahlen',
                AutoRestoreProc);
  ScaleGraph;
  wf.x:= wf.x + wf.w + 25;
  wf.y:= wf.y + wf.h DIV 2;
  wf.w:= 190; wf.h:= 3*CellHeight();
  CreateWindow(dataW,
                FixedSize,WithoutScrollBars,
                WithCloseBox,WithoutZoomBox,bottomLeft,wf,
                'Letzter Punkt',
                DocDotData);
  SetDMState(withWindsNoRandGen);
END MakeWindows;


(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)


(*****************************************************)
(* Menu command: "Starte kont. Zufallszahlengeneration" *)
(*****************************************************)

PROCEDURE GenADot; FORWARD;

PROCEDURE ToggleRandGen;
  PROCEDURE ProdRandGens;
  BEGIN (*ProdRandGens*)
    REPEAT
      GenADot;
      DialogMachineTask;
    UNTIL curDMState <> withWindsAndRandGen
  END ProdRandGens;
BEGIN (*ToggleRandGen*)
  IF curDMState = withWindsNoRandGen THEN
    SetDMState(withWindsAndRandGen);
    ProdRandGens;
  ELSIF curDMState = withWindsAndRandGen THEN
    SetDMState(withWindsNoRandGen);
  END(*IF*);
END ToggleRandGen;


(*******************************************)
(* Menu command: "Erzeuge zwei Zufallszahlen" *)
(*******************************************)

PROCEDURE GenADot;
BEGIN
  x:= curU();  curGetZ(z1); y:= curU(); curGetZ(z2);
  SelectForOutput(graphW);
  UCDot(x,y);
  DocDotData(dataW);
END GenADot;


(*****************************************)
(* Menu command: "Lösche und setze zurück" *)
(*****************************************)

PROCEDURE ClearResetAndScale;
  VAR wf: WindowFrame;
BEGIN
  IF curDMState = withWindsAndRandGen THEN
    SetDMState(withWindsNoRandGen);
  END(*IF*);
  Clear(graphW);
  ScaleGraph;
  Clear(dataW);
  ResetRandGen;
  DocDotData(dataW);
END ClearResetAndScale;


(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)


(*********************************************)
(* Menu command: "Wähle Zufallszahlengenerator" *)
(*********************************************)

VAR
  usePreDefGen: BOOLEAN;

PROCEDURE SetGenerator;
  VAR bf: FormFrame; ok: BOOLEAN; A,M: CARDINAL;
    genSet,adHocGen,preDefGen: RadioButtonID;
```

```
BEGIN
  WriteLabel(2,5,"Linearer Kongruenter Zufallszahlengenerator:");
  WriteLabel(3,5,"           z(k+1) = A * z(k) MOD M");
  DefineRadioButtonSet(genSet);
  WriteLabel(5,6,"Vordefinierter multiplikativer Generator der Form");
  RadioButton(preDefGen,6,6,"z(k+1) = 950706376 * z(k) MOD (2**31 - 1)");
  WriteLabel(8,6,"Definierbarer Generator:");
  RadioButton(adHocGen,9,6,"z(k+1) = A * z(k) MOD M");
  IF usePreDefGen THEN genSet:= preDefGen ELSE genSet:= adHocGen END;
  GetParams(A,M);
  WriteLabel(10,9,"A = ");
  CardField(10,13,7,A,useAsDeflt,0,MAX(CARDINAL));
  WriteLabel(10,25,"M = ");
  CardField(10,29,7,M,useAsDeflt,0,MAX(CARDINAL));
  bf.x:= 0; bf.y:= -1 (*display dialog window in middle of screen*);
  bf.lines:= 13; bf.columns:= 50;
  UseEntryForm(bf,ok);
  IF ok THEN
    IF genSet = preDefGen THEN
      usePreDefGen:= TRUE; curU:= U; curGetZ:= GetZ
    ELSE
      usePreDefGen:= FALSE; curU:= AdHocU; curGetZ:= AdHocGetZ;
      SetParams(A,M);
    END;
    IF curDMState <> noWind THEN
      ClearResetAndScale
    ELSE
      ResetRandGen;
    END(*IF*);
  END(*IF*);
END SetGenerator;


(*********************************)
(* Menu command: "Setze 'Seed'" *)
(*********************************)

PROCEDURE SetSeed;
  VAR bf: FormFrame; ok: BOOLEAN; seed: CARDINAL;
BEGIN
  WriteLabel(2,10,"seed = ");
  IF usePreDefGen THEN seed:= seed0 ELSE seed:= adHocSeed0 END;
  CardField(2,18,7,seed,useAsDeflt,1,MAX(CARDINAL));
  bf.x:= 0; bf.y:= -1 (*display dialog window in middle of screen*);
  bf.lines:= 6; bf.columns:= 40;
  UseEntryForm(bf,ok);
  IF ok THEN
    IF usePreDefGen THEN Seed(seed) ELSE AdHocSeed(seed) END;
    ResetGlobVars;
    IF curDMState <> noWind THEN
      Clear(dataW);
      DocDotData(dataW);
    END(*IF*);
  END(*IF*);
END SetSeed;


(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)


(***********************************)
(* Menu command: "Programm beenden" *)
(***********************************)

PROCEDURE Quitting(VAR reallyQuit: BOOLEAN);
BEGIN
  reallyQuit:= TRUE;
  SetDMState(noWind)
END Quitting;


(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)




(*****************)
(* MouseHandlers *)
(*****************)

PROCEDURE EnableMenuIfWindowCloses(u: Window);
BEGIN
  SetDMState(noWind);
END EnableMenuIfWindowCloses;

PROCEDURE RescaleIfWindowIsRedefined(u: Window);
```

```
  BEGIN
    ClearResetAndScale;
  END RescaleIfWindowIsRedefined;



  (****************************************)
  (* Initialization when DM starts to run *)
  (****************************************)

  PROCEDURE SettingUp;
  BEGIN
    curU:= U; curGetZ:= GetZ;
    usePreDefGen:= TRUE; ResetRandGen;
    graphW:= notExistingWindow;
    dataW:= notExistingWindow;
    SetDMState(noWind);
  END SettingUp;



  (*********************************************)
  (* Initialization of DM before it is running *)
  (*********************************************)

  PROCEDURE DMInitialization;
  BEGIN
    InstallAbout("Über |  RANDOM ...",300,140,AboutProc);
    InstallMenu(myMenu,'Kontrolle',enabled);
    InstallCommand(myMenu, makeWindows,"Öffne Fenster", MakeWindows,
                   enabled, unchecked);
    InstallSeparator(myMenu,line);
    InstallCommand(myMenu,randGens,"Starte kont. Zufallszahlengeneration",
                   ToggleRandGen,disabled,unchecked);
    InstallAliasChar(myMenu,randGens,"S");
    InstallCommand(myMenu,oneDot,"Erzeuge zwei Zufallszahlen",
                   GenADot,disabled,unchecked);
    InstallAliasChar(myMenu,oneDot,"p");
    InstallCommand(myMenu,clear,"Lösche und setze zurück",
                   ClearResetAndScale,disabled,unchecked);
    InstallSeparator(myMenu,line);
    InstallCommand(myMenu,setPars,"Wähle Zufallszahlengenerator",
                   SetGenerator,disabled,unchecked);
    InstallCommand(myMenu,seed,"Setze 'Seed'",
                   SetSeed,disabled,unchecked);
    InstallQuitCommand("Programm beenden",Quitting,0C);
    InstallSetUpProc(SettingUp);
    InstallMouseHandler(CloseWindow,EnableMenuIfWindowCloses);
    InstallMouseHandler(RedefWindow,RescaleIfWindowIsRedefined);
  END DMInitialization;



BEGIN
  DMInitialization;
  RunDialogMachine
END Random.
```

5 References

APPLE COMPUTER INC. 1985. Inside Macintosh, Volume I. Addison-Wesley.

DOMEISEN, H., FORSTER, A., SCHAUFELBERGER, W. & WEGMANN, P. 1992. Computer im Unterricht an der ETH Zürich - Bericht über das Projekt IDA 1986-1991. vdf, Zürich.

FISCHLIN, A. & SCHAUFELBERGER, W., 1987. Arbeitsplatzrechner im technisch-naturwissenschaftlichen Hochschulunterricht. *Bulletin des Schweizerischen Elektrotechnischen Vereins (SEV/VSE)*, **78**(1): 15-21.

FISCHLIN, A. 1986. Simplifying the usage and the programming of modern working stations with Modula-2: The Dialog Machine. Department of Automatic Control and Industrial Electronics, Swiss Federal Institute of Technology Zurich (ETHZ).

FISCHLIN, A. 1988. The ‘Dialog Machine’ for the Macintosh. Projekt-Zentrum IDA, ETH Zürich.

FISCHLIN, A., GYALISTRAS, D., ROTH, O., ULRICH, M., THÖNY, J., NEMECEK, T., BUGMANN, H. & THOMMEN, F. 1994. ModelWorks 2.2 – An Interactive Simulation Environment for Personal Computers and Workstations. Internal Reports of Systems Ecology ETHZ No. 14, Switzerland.

FISCHLIN, A., MANSOUR, M.A., RIMVALL, M. & SCHAUFELBERGER, W., 1987. *Simulation and computer aided control system design in engineering education*. In: Troch, I., Kopacek, P. & Breitenecker, F. (eds.), *Simulation of Control Systems*. Pergamon Press, Oxford a.o., pp. 51-60.

HOUGH, D., 1981. Applications of the proposed IEEE 754 standard for floating-point arithmetic. *Computer*, **14**(3): 70-74.

IEEE Std 754-1985, 1985. *IEEE standard for binary floating-point arithmetic*. IEEE, Inc., New York.

KELLER, D., 1989. *Introduction to the Dialog Machine*. Interner Bericht Nr. 5 (Nov.), Projekt-Zentrum IDA, Swiss Federal Institute of Technology Zürich (ETHZ), Switzerland, 37pp.

THOENY, J., FISCHLIN, A. & GYALISTRAS, D., 1994. *RASS*[6]*: Towards bridging the gap between interactive and off-line simulation*. Halin, J. (ed.), 1995, Proc. CISS 94, Springer, in prep.

WIRTH, N., GUTKNECHT, J., HEIZ, W., SCHäR, H., SEILER, H., VETTERLI, C. & FISCHLIN, A., 1992. *MacMETH. A fast Modula-2 language system for the Apple Macintosh. User Manual. 4th, completely revised ed.* Department of Computer Sciences (ETH), Zürich, Switzerland, 116 pp.

[6]RASS is an acronym for RAMSES Simulation Server.

6 Appendix

6.1 How to Get the “Dialog Machine”

The “Dialog Machine” is distributed via the INTERNET. For the Macintosh it comes as part of the modeling and simulation package RAMSES[7]. For the IBM PC it is available as part of the software package "ModelWorks for Windows". Visit our home page

http://www.ito.umnw.ethz.ch/SysEcol/

where you can also learn about the latest updates. Alternatively You can get the *Macintosh version* it via anonymous ftp from

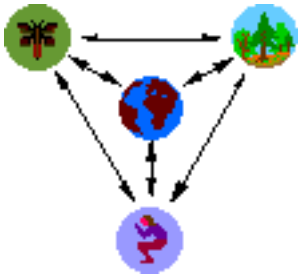
ftp.ito.umnw.ethz.ch

in directory “/pub/mac/RAMSES” (Password: ftp; User: Your e-mail address). The Windows version is available from the same server (“*ftp.ito.umnw.ethz.ch*”) in “/pub/pc/MW-Windows”. Finally, for the Macintosh it is also available on the

"CD Apprentice"

as an excellent bargain from this company:

Celestin Company, Inc., 5652 NE Meadow Road, Kingston, WA 98346-9505, Telephone 360 297 8091 -- Fax 360 297 8092 -- http://www.celestin.com/, send a blank message to info@celestin.com for an index of products.



6.2 Hard- and Software Requirements

The “Dialog Machine” is currently available for the whole Apple® Macintosh® computer family (except for machines with less than 512KByte RAM and the 64KByte ROM, i.e. the very first Mac and the Fat Mac are no longer supported; every machine from the Reflex on are supported, including Power Macs) and in a version with exactly the same interfaces on IBM PCs running under “Windows 3.1”.If Windows can run on your IBM PC, the “Dialog Machine” will run easily

[7] Research Aids for Modeling and Simulation of Environmental Systems.

The latest version 3.0 of the “Dialog Machine” has been implemented using the newest Modula-2 implementation MacMETH[8] Version 3.2.7. MacMETH was developed at the department of Computer Science at the Swiss Federal Institute of Technology Zürich ETHZ under Prof. Dr. N. Wirth, the creator of Pascal and Modula-2 (WIRTH *et al.*, 1992). MacMETH is a fast Modula-2 language system for the Macintosh; it consists of a 1-pass compiler, symbolic debugger, editor (displaying compilation errors), merge utility (merges compilation error messages into source), linker respectively application maker, loader, plus a shell to quickly switch between compiler, editor, and debugger or execute a program. There is also a special source-code compatible compiler available, which supports the 68020 resp. 68030 CPU and the floating point coprocessor 68881 resp. 68882. It enhances the computing performance for some mathematical functions like transcendental functions (e.g. Ln, Exp) considerably (direct instructions bypassing SANE)).

Macintosh. In order to use the “Dialog Machine” (any version) you need a Macintosh computer with at least 512K of memory, the 64K ROM (satisfied by all current models), and a hard disk or two 800 KB floppy disks. Every System software later than 2.4 is sufficient to run a “Dialog Machine” program. However, it is recommended to have at least 1 MB RAM and a hard disk (although you can develop large “Dialog Machine” programs on a floppy based computer with only 512 KB RAM; the only thing you need in addition to this hardware is a bit more of patience).

IBM PC. The current “Dialog Machine” 3.0 needs at least Windows 3.1 or higher and the hardware required by Windows. Moreover you need to buy the license for the Stonybrook Modula-2 windows developing system (formerly Logitech Modula-2). A hard disk is a must and because of Windows you need also lots of memory. For further details visit <http://www.ito.umnw.ethz.ch/SysEcol/SimSoftware/SimSoftware.html#ModelWorks for Windows>

6.3 Disclaimer

The “Dialog Machine” version 3.0 is currently distributed as freeware (but not public domain) on an “as is” basis. No support, e.g. hot line etc. is offered. Documentation is provided together with the software in electronic form only. No warranty for correctness of the programs is provided. The distribution of this software is primarily for academic usage, i.e. teaching and research. The demonstration programs explained below are public domain software and may be copied or altered freely as long as they are not used for commercial purposes. All copyrights remain with the authors of the “Dialog Machine” and the Swiss Federal Institute of Technology Zurich, Switzerland.

You may use the “Dialog Machine” to develop programs in any way you desire. However, we require that you

give us some credit by mentioning in your software, the software documentation, and/or scientific publication in a well visible fashion

that you have developed your software by means of the Dialog Machine. In particular do we ask you to leave in your software the copyright notice window of the “Dialog Machine” intact. If you copy the software for someone else, you may do so, but please, copy everything exactly as you have received it from us.

[8] Macintosh Modula ETH

7 Quick Reference

```
Dialog Machine Version 3.0 (9/Juni/2001)
(c) 1988-2002 Andreas Fischlin, Systems Ecology, and Swiss Federal Institute of Technology Zurich ETHZ

| Marks lines which contain objects which have been added since Version 1.0


                (*****************************************************************************)
                (*#####                      K E R N E L   M O D U L E S                  #####*)
                (*****************************************************************************)


(*=======================================        DMConversions          =======================================*)

  TYPE RealFormat = (FixedFormat, ScientificNotation);

  PROCEDURE StringToCard(str: ARRAY OF CHAR; VAR card: CARDINAL; VAR done: BOOLEAN);
  PROCEDURE CardToString(card: CARDINAL; VAR str: ARRAY OF CHAR; length: CARDINAL);
  PROCEDURE StringToLongCard(str: ARRAY OF CHAR; VAR lcard: LONGCARD; VAR done: BOOLEAN);
  PROCEDURE LongCardToString(lcard: LONGCARD; VAR str: ARRAY OF CHAR; length: CARDINAL);
  PROCEDURE StringToInt(str: ARRAY OF CHAR; VAR int: INTEGER; VAR done: BOOLEAN);
  PROCEDURE IntToString(int: INTEGER; VAR str: ARRAY OF CHAR; length: CARDINAL);
  PROCEDURE StringToLongInt(str: ARRAY OF CHAR; VAR lint: LONGINT; VAR done: BOOLEAN);
  PROCEDURE LongIntToString(lint: LONGINT; VAR str: ARRAY OF CHAR; length: CARDINAL);
  PROCEDURE StringToReal(str:ARRAY OF CHAR; VAR real: REAL; VAR done: BOOLEAN);
  PROCEDURE RealToString(real: REAL; VAR str: ARRAY OF CHAR; length, dec: CARDINAL; f: RealFormat);
  PROCEDURE StringToLongReal(Str:ARRAY OF CHAR; VAR longReal: LONGREAL; VAR done: BOOLEAN);
  PROCEDURE LongRealToString(longreal: LONGREAL; VAR str: ARRAY OF CHAR; length, dec: CARDINAL; f: RealFormat);
| PROCEDURE HexStringToBytes(hstr: ARRAY OF CHAR; VAR x: ARRAY OF BYTE; VAR done: BOOLEAN);
| PROCEDURE BytesToHexString(x: ARRAY OF BYTE; VAR hstr: ARRAY OF CHAR);   PROCEDURE SetHexDigitsUpperCase(upperC: BOOLEAN);
| PROCEDURE IllegalSyntaxDetected(): BOOLEAN;

| PROCEDURE UndefREAL(): REAL; (* = NAN(017) *)     PROCEDURE UndefLONGREAL(): LONGREAL; (* = NAN(017) *)
| PROCEDURE IsUndefREAL(x: REAL): BOOLEAN;          PROCEDURE IsUndefLONGREAL(x: LONGREAL): BOOLEAN;


(*=======================================          DMLanguage           =======================================*)

  CONST
|   allOk = 0;

    (* General ('Dialog Machine' and other software layers) *)
|     badProgState (* should not occurr *) = -4;
|     onlyAnInsert = -3;                  unknownErr = -2;                  insuffMem = -1;
|     tooOldMac = 9;                      tooManyTermProc = 10;             notImplemented = 100;

    (* DMWindIO arithmetic *)
|     intOverflow = 1;                    lowUpSame = 2;

    (* User Input (DMEntryForms etc.) *)
|     numExpected = 5;                    outOfRange = 7;                   wrongChar = 3;
|     wrongCharOrNone = 14;               only1Char = 4;                    only1CharOrNone = 15;
|     stringTooLong = 16;

    (* Object access *)
|     unknownWindow = 8;                  unknownEditField = 6;             unknownGraph = 11;

    (* DM2DGraphs *)
|     noLogScale = 12;                    graphTooSmall = 17;

    (* DMFiles: Subsequent message order fits DMFiles.Response, i.e. code = fileResBase+ORD(f.res) *)
|     fileResBase = 20;
|     fileNotFound = 21;                  volNotFound = 22;                 fileDlgCancelled = 23;
|     unknownFile = 24;                   tooManyFiles = 25;                diskFull = 26;
|     insuffMemForFileFct = 27;           fileAlreadyOpen = 28;             fileIsBusy = 29;
|     fileIsLocked = 30;                  fileFctNotDone = 31;

    (* Error constants beyond userBase for free use *)
|     userBase = 300;

  (* See also module Errors from AuxLib for further constants and error messages display *)

  TYPE Language = (English, German, French, Italian, MyLanguage1, MyLanguage2);

  PROCEDURE SetLanguage(l: Language);                    PROCEDURE CurrentLanguage(): Language;
| PROCEDURE GetMsgString(msgNr: INTEGER; VAR str: ARRAY OF CHAR);

(*=======================================            DMMaster           =======================================*)

  TYPE MouseHandlers = (WindowContent, BringToFront, RemoveFromFront, RedefWindow, CloseWindow);
    MouseHandler = PROCEDURE (Window);       KeyboardHandler = PROC;              SubProgStatus = (normal, abnormal);

  VAR MasterDone: BOOLEAN;

| PROCEDURE AddSetupProc(sup: PROC; priority: INTEGER);      PROCEDURE RemoveSetupProc(sup: PROC);
| PROCEDURE AddMouseHandler(which: MouseHandlers; mhp: MouseHandler; priority: INTEGER);
| PROCEDURE RemoveMouseHandler(which: MouseHandlers; mhp: MouseHandler);

| PROCEDURE AddKeyboardHandler(khP: KeyboardHandler; priority: INTEGER);          PROCEDURE RemoveKeyboardHandler(khP: KeyboardHandler);

| PROCEDURE InspectKey(VAR ch: CHAR; VAR modifiers: BITSET); PROCEDURE KeyAccepted;  PROCEDURE DoTillKeyReleased(p: PROC);
| PROCEDURE SetKeyboardHandlerMode(    readGetsThem: BOOLEAN;     maxPriority: INTEGER);     PROCEDURE Read(VAR ch: CHAR);
| PROCEDURE GetKeyboardHandlerMode(VAR readGetsThem: BOOLEAN; VAR maxPriority: INTEGER);     PROCEDURE BusyRead(VAR ch: CHAR);

  PROCEDURE ShowWaitSymbol;          PROCEDURE HideWaitSymbol;                     PROCEDURE Wait(nrTicks: LONGCARD); (* 1 tick = 1/60 second *)
| PROCEDURE SoundBell;                PROCEDURE PlayPredefinedMusic(fileName: ARRAY OF CHAR; musicID: INTEGER);

  PROCEDURE InitDialogMachine;               PROCEDURE RunDialogMachine;           PROCEDURE DialogMachineIsRunning(): BOOLEAN;
```

```
  PROCEDURE QuitDialogMachine;                PROCEDURE AbortDialogMachine;          PROCEDURE DialogMachineTask;
  PROCEDURE CallSubProg(module: ARRAY OF CHAR; VAR status: SubProgStatus);
| PROCEDURE ForceDialogMachineIntoBatchMode(bm: BOOLEAN);    PROCEDURE DialogMachineIsInBatchMode(): BOOLEAN;

(*=========================================             DMMenus             =========================================*)

  TYPE Menu;   Command;    AccessStatus = (enabled, disabled);   Marking = (checked, unchecked);   Separator = (line, blank);
    QuitProc = PROCEDURE(VAR BOOLEAN);  SeparatorPosition = (beforeCmd, afterCmd);

| VAR MenusDone: BOOLEAN;    notInstalledMenu: Menu;    notInstalledCommand: Command;

  PROCEDURE InstallAbout(s: ARRAY OF CHAR; w,h: CARDINAL; p: PROC);
  PROCEDURE NoDeskAccessories;
  PROCEDURE InstallMenu(VAR m: Menu; menuText: ARRAY OF CHAR; ast: AccessStatus);
| PROCEDURE InstallSubMenu (inMenu: Menu; VAR subMenu: Menu; menuText: ARRAY OF CHAR; ast: AccessStatus);
  PROCEDURE InstallCommand(m: Menu; VAR c: Command; cmdText: ARRAY OF CHAR; p: PROC; ast: AccessStatus; chm: Marking);
  PROCEDURE InstallAliasChar(m: Menu; c: Command; ch: CHAR);
| PROCEDURE InstallSeparator(m: Menu; s: Separator);          PROCEDURE RemoveSeparator(m: Menu; s: CARDINAL);
| PROCEDURE RemoveSeparatorAtCommand(m: Menu; cmd: Command; sp: SeparatorPosition);

  PROCEDURE InstallQuitCommand(s: ARRAY OF CHAR; p: QuitProc; aliasChar: CHAR);
| PROCEDURE HideSubQuit(onLevel: CARDINAL);                  PROCEDURE ShowSubQuit(onLevel: CARDINAL);
  PROCEDURE UseMenu(m: Menu);                                PROCEDURE UseMenuBar;
  PROCEDURE RemoveMenu(VAR m: Menu);                         PROCEDURE RemoveMenuBar;
| PROCEDURE RemoveCommand(m: Menu; cmd: Command);
  PROCEDURE EnableDeskAccessories;                           PROCEDURE DisableDeskAccessories;
  PROCEDURE EnableMenu(m: Menu);                             PROCEDURE DisableMenu(m: Menu);
  PROCEDURE EnableCommand(m: Menu; c: Command);              PROCEDURE DisableCommand(m: Menu; c: Command);
  PROCEDURE CheckCommand(m: Menu; c: Command);               PROCEDURE UncheckCommand(m: Menu; c: Command);
| PROCEDURE SetCheckSym(m: Menu; c: Command; ch: CHAR);      PROCEDURE IsCommandChecked(m: Menu; c: Command): BOOLEAN;
| PROCEDURE ChangeCommand(m: Menu; c: Command; p: PROC);     PROCEDURE ChangeCommandText(m: Menu; c: Command;
                                                                                        newCmdText: ARRAY OF CHAR);
| PROCEDURE ChangeAliasChar(m: Menu; c: Command; newCh: CHAR);  PROCEDURE ChangeQuitAliasChar(onLevel: CARDINAL; newAliasCh: CHAR);
| PROCEDURE ExecuteCommand(m: Menu; c: Command);            PROCEDURE ExecuteAbout;
| PROCEDURE MenuExists(m: Menu): BOOLEAN;                   PROCEDURE CommandExists(m: Menu; c: Command): BOOLEAN;
| PROCEDURE MenuLevel(m: Menu): CARDINAL;                   PROCEDURE CommandLevel(m: Menu; c: Command): CARDINAL;
| PROCEDURE GetMenuAttributes(m: Menu; VAR menuNr: CARDINAL; VAR menuText: ARRAY OF CHAR; VAR ast: AccessStatus;
                          VAR isSubMenu: BOOLEAN; VAR parentMenu: Menu);
| PROCEDURE GetCommandAttributes(m: Menu; c: Command; VAR cmdNr: CARDINAL; VAR cmdText: ARRAY OF CHAR; VAR p: PROC;
                            VAR ast: AccessStatus; VAR chm: Marking; VAR chmCh, aliasCh: CHAR);

| PROCEDURE InstallPredefinedMenu       (fileName: ARRAY OF CHAR; menuID: INTEGER; VAR m: Menu);
| PROCEDURE InstallPredefinedSubMenu    (fileName: ARRAY OF CHAR; menuID: INTEGER; inMenu: Menu; VAR subMenu: Menu);
| PROCEDURE InstallPredefinedCommand    (fileName: ARRAY OF CHAR; menuID, itemNr: INTEGER; m: Menu; VAR c: Command; p: PROC);
| PROCEDURE InstallPredefinedSeparator  (fileName: ARRAY OF CHAR; menuID, itemNr: INTEGER; m: Menu);
| PROCEDURE SaveAsPredefinedMenu        (fileName: ARRAY OF CHAR; menuID: INTEGER; m: Menu);
| PROCEDURE SaveAsPredefinedMenuSection (fileName: ARRAY OF CHAR; menuID: INTEGER; m: Menu; maxItemNr: INTEGER);

(*=========================================            DMMessages           =========================================*)

| CONST LNBREAK = 15C; undefMsgNr = -1;  toScreen = 0;  toJournalFile = 1;

| TYPE MsgRetrieveProc = PROCEDURE ( INTEGER , VAR ARRAY OF CHAR );
|   MsgDevice = [toScreen..toJournalFile]; MsgWriteProc = PROCEDURE ( CHAR ); MsgWriteLnProc = PROC;

| PROCEDURE Ask(question: ARRAY OF CHAR; butTexts: ARRAY OF CHAR; butWidth: CARDINAL; VAR answer: INTEGER);
| PROCEDURE DisplayBusyMessage(msg: ARRAY OF CHAR);     PROCEDURE DiscardBusyMessage;
| PROCEDURE Inform   (paragraph1, paragraph2, paragraph3: ARRAY OF CHAR);
| PROCEDURE DoInform (msgnr: INTEGER; modIdent, locDescr, insertions: ARRAY OF CHAR);
| PROCEDURE Warn     (paragraph1, paragraph2, paragraph3: ARRAY OF CHAR);
| PROCEDURE DoWarn   (msgnr: INTEGER; modIdent, locDescr, insertions: ARRAY OF CHAR);
| PROCEDURE Abort    (paragraph1, paragraph2, paragraph3: ARRAY OF CHAR);
| PROCEDURE DoAbort  (msgnr: INTEGER; modIdent, locDescr, insertions: ARRAY OF CHAR);

| PROCEDURE SetMsgRetrieveProc(rp: MsgRetrieveProc);    PROCEDURE GetMsgRetrieveProc(VAR rp: MsgRetrieveProc);
| PROCEDURE UseForMsgJournaling(wp: MsgWriteProc; wlnp: MsgWriteLnProc);
| PROCEDURE SetMaxMsgs (max: INTEGER);
| PROCEDURE SetMsgDevice     (forAsk,forInform,forWarn,forAbort: MsgDevice);
| PROCEDURE GetMsgDevice (VAR forAsk,forInform,forWarn,forAbort: MsgDevice);
| PROCEDURE AskPredefinedQuestion(fileName: ARRAY OF CHAR; alertID: INTEGER;
                              str1,str2,str3,str4: ARRAY OF CHAR; VAR answer: INTEGER);

(*=========================================             DMStorage           =========================================*)

| PROCEDURE Allocate(VAR p: ADDRESS; size: LONGINT);  PROCEDURE AllocateOnLevel(VAR adr: ADDRESS; size: LONGINT; onLevel: INTEGER);
| PROCEDURE Deallocate(VAR p: ADDRESS);               PROCEDURE DeallocateOnLevel(VAR p: ADDRESS; onLevel: INTEGER);

(* IBM PC compatibility: *)
| PROCEDURE ALLOCATE(VAR p: ADDRESS; size: CARDINAL); PROCEDURE DEALLOCATE(VAR p: ADDRESS; size: CARDINAL);

(*=========================================             DMStrings           =========================================*)

  TYPE String; StringRelation = (smaller, equal, greater);

| VAR notAllocatedStr: String; ResourceStringsDone: BOOLEAN;

| PROCEDURE AllocateStr(VAR strRef: String; s: ARRAY OF CHAR);      PROCEDURE DeallocateStr(VAR strRef: String);
| PROCEDURE SetStr(VAR strRef: String; s: ARRAY OF CHAR);          PROCEDURE GetStr(strRef: String; VAR s: ARRAY OF CHAR);
| PROCEDURE StrLevel(strRef: String): CARDINAL;                   PROCEDURE StrLength(strRef: String): INTEGER;
  PROCEDURE Length(VAR string: ARRAY OF CHAR): INTEGER;
  PROCEDURE AssignString(source: ARRAY OF CHAR; VAR d: ARRAY OF CHAR);
| PROCEDURE Append(VAR dest: ARRAY OF CHAR; source: ARRAY OF CHAR); PROCEDURE AppendCh(VAR dest: ARRAY OF CHAR; ch: CHAR);
| PROCEDURE AppendStr(VAR strRef: String; s: ARRAY OF CHAR);       PROCEDURE AppendChr(VAR strRef: String; ch: CHAR);
| PROCEDURE Concatenate(first,second: ARRAY OF CHAR; VAR result: ARRAY OF CHAR);
| PROCEDURE CopyString (VAR from: ARRAY OF CHAR; i1,nrChs: INTEGER; VAR to: ARRAY OF CHAR; VAR i2: INTEGER);
| PROCEDURE Copy(from: ARRAY OF CHAR; startIndex, nrOfChars: INTEGER; VAR to: ARRAY OF CHAR);
| PROCEDURE ExtractSubString(VAR curPosInSrcS: INTEGER; VAR srcS,destS: ARRAY OF CHAR; delimiter: CHAR);
| PROCEDURE FindInString (VAR theString: ARRAY OF CHAR; searchStr: ARRAY OF CHAR; VAR firstCh,lastCh: INTEGER): BOOLEAN;
| PROCEDURE CompareStrings(s1,s2: ARRAY OF CHAR): StringRelation;
| PROCEDURE CompVarStrings( VAR a, b: ARRAY OF CHAR): StringRelation;
```

```
| PROCEDURE CompStr( VAR a: ARRAY OF CHAR;  bS: String): StringRelation;
  PROCEDURE LoadString(fileName: ARRAY OF CHAR; stringID: INTEGER; VAR string: ARRAY OF CHAR);
  PROCEDURE StoreString(fileName: ARRAY OF CHAR; VAR stringID: INTEGER; string: ARRAY OF CHAR);
  PROCEDURE GetRString(stringID: INTEGER; VAR str: ARRAY OF CHAR);
| PROCEDURE SetRStringName (fileName: ARRAY OF CHAR; stringID: INTEGER; name: ARRAY OF CHAR);
| PROCEDURE GetRStringName (fileName: ARRAY OF CHAR; stringID: INTEGER; VAR name: ARRAY OF CHAR);
  PROCEDURE NewString(VAR s: ARRAY OF CHAR): String;              PROCEDURE PutString(VAR strRef: String; VAR s: ARRAY OF CHAR);

(*=======================================            DMSystem            =======================================*)

  CONST startUpLevel = 1;   maxLevel = 5;

  PROCEDURE CurrentDMLevel(): CARDINAL;   PROCEDURE LevelisDMLevel(l: CARDINAL): BOOLEAN;
| PROCEDURE LevelIsTerminating(): BOOLEAN;
| PROCEDURE TopDMLevel(): CARDINAL;       PROCEDURE DoOnSubProgLevel(l: CARDINAL; p: PROC);
| PROCEDURE ForceDMLevel(extraLevel: CARDINAL);              PROCEDURE ResumeDMLevel(normalLevel: CARDINAL);

| PROCEDURE InstallInitProc(ip: PROC; VAR done: BOOLEAN);       PROCEDURE ExecuteInitProcs;
| PROCEDURE InstallTermProc(tp: PROC; VAR done: BOOLEAN);       PROCEDURE ExecuteTermProcs;

| PROCEDURE GetDMVersion(VAR vers,lastModifDate: ARRAY OF CHAR);    PROCEDURE SystemVersion(): REAL;
| PROCEDURE GetComputerName(VAR name: ARRAY OF CHAR);
| PROCEDURE RunsOnAMac(): BOOLEAN;
| PROCEDURE RunsOnAnIBMPC(): BOOLEAN;
| PROCEDURE RunsOnAUnixMachine(): BOOLEAN;
| PROCEDURE GetCPUName(VAR name: ARRAY OF CHAR);        PROCEDURE GetFPUName(VAR name: ARRAY OF CHAR);
| PROCEDURE FPUPresent(): BOOLEAN;                      PROCEDURE GetROMName(VAR name: ARRAY OF CHAR);

  PROCEDURE ScreenWidth(): INTEGER;       PROCEDURE ScreenHeight(): INTEGER;
| PROCEDURE MainScreen(): INTEGER;
| PROCEDURE SuperScreen(VAR whichScreen, x,y,w,h, nrOfColors: INTEGER; colorPriority: BOOLEAN);

 (* low level routines *)
  PROCEDURE MenuBarHeight(): INTEGER;   PROCEDURE TitleBarHeight(): INTEGER;    PROCEDURE ScrollBarWidth(): INTEGER;
| PROCEDURE GrowIconSize(): INTEGER;
  PROCEDURE NumberOfColors(): INTEGER;  (* supported by DM regardless of currently used screen *)
| PROCEDURE HowManyScreens(): INTEGER;  PROCEDURE GetScreenSize(screen: INTEGER; VAR x,y,w,h: INTEGER);
| PROCEDURE NumberOfColorsOnScreen(screen: INTEGER): INTEGER;

  CONST unknown = 0;
|   Mac512KE     = 3;   MacSE30     =  9;   MacLC            = 19;   MacPowerBook140 = 25;       SUN      = 101;   IBMPC       = 201;
|   MacPlus      = 4;   MacPortable = 10;   MacQuadra900     = 20;   MacLCII         = 19;       SUN3     = 102;   IBMAT       = 202;
|   MacSE        = 5;   MacIIci     = 11;   MacPowerBook170  = 21;   MacQuadra950    = 26;       SUNSparc = 103;   IBMPS2      = 203;
|   MacII        = 6;   MacIIfx     = 13;   MacQuadra700     = 22;                                                 IBMRisc6000 = 300;
|   MacIIx       = 7;   MacClassic  = 17;   MacClassicII     = 23;
|   MacIIcx      = 8;   MacIIsi     = 18;   MacPowerBook100  = 24;
  PROCEDURE ComputerSystem(): INTEGER;

  CONST CPU68000 = 1;    CPU8088 = 201;    CPU80186 = 203;              FPU68881 = 1;
        CPU68010 = 2;    CPU8086 = 202;    CPU80286 = 204;              FPU68882 = 2;
        CPU68020 = 3;                      CPU80386 = 205;              FPU68040 = 3;
|       CPU68030 = 4;                      CPU80486 = 206;
|       CPU68040 = 5;
| PROCEDURE CPUType(): INTEGER;         PROCEDURE FPUType(): INTEGER;

| CONST MacKeyboard  = 1;      AExtendKbd  = 4;     PortableISOKbd = 7;     ADBKbdII    = 10;    PwrBkISOKbd    = 13;
|       MacKbdAndPad = 2;      ADBKeyboard = 5;     EastwoodISOKbd = 8;     ADBISOKbdII = 11;
|       MacPlusKbd   = 3;      PortableKbd = 6;     SaratogaISOKbd = 9;     PwrBkADBKbd = 12;
  PROCEDURE Keyboard(): INTEGER;

| CONST  ROM64k = 1;    ROM128k = 2;    ROM256k = 3;        ROM512k = 4;        ROM1024k = 5;  (* ROM types *)
| PROCEDURE ROMType(): INTEGER;         PROCEDURE ROMVersionNr(): INTEGER;      PROCEDURE QuickDrawVersion(): REAL;

(*=======================================            DMWindIO            =======================================*)

  TYPE MouseModifiers = (ordinary, cmded, opted, shifted, capsLock, controlled);         ClickKind = SET OF MouseModifiers;
    DragProc = PROCEDURE (INTEGER, INTEGER);

  VAR WindowIODone: BOOLEAN;

  PROCEDURE PointClicked(x,y: INTEGER; maxDist: INTEGER): BOOLEAN;
  PROCEDURE RectClicked(rect: RectArea): BOOLEAN;
  PROCEDURE PointDoubleClicked(x,y: INTEGER; maxDist: INTEGER): BOOLEAN;
  PROCEDURE RectDoubleClicked(rect: RectArea): BOOLEAN;
| PROCEDURE GetLastClick(VAR x,y: INTEGER; VAR click: ClickKind): BOOLEAN;
| PROCEDURE GetLastDoubleClick(VAR x,y: INTEGER; VAR click: ClickKind): BOOLEAN;
  PROCEDURE GetCurMousePos(VAR x,y: INTEGER);
  PROCEDURE GetLastMouseClick(VAR x,y: INTEGER; VAR click: ClickKind);
  PROCEDURE DoTillMButReleased(p: PROC);
  PROCEDURE Drag(duringDragP,afterDragP: DragProc);
  PROCEDURE SetContSize(u: Window; contentRect: RectArea);   PROCEDURE GetContSize(u: Window; VAR contentRect: RectArea);
  PROCEDURE SetScrollStep(u: Window; xStep,yStep: INTEGER);  PROCEDURE GetScrollStep(u: Window; VAR xStep, yStep: INTEGER);
  PROCEDURE GetScrollBoxPos(u: Window; VAR posX,posY: INTEGER);
| PROCEDURE SetScrollBoxPos(u: Window; posX,posY: INTEGER);
  PROCEDURE GetScrollBoxChange(u: Window; VAR changeX,changeY: INTEGER);
  PROCEDURE AutoScrollProc(u: Window);
  PROCEDURE SetScrollProc(u: Window; scrollP: RestoreProc);  PROCEDURE GetScrollProc(u: Window; VAR scrollP: RestoreProc);
  PROCEDURE ScrollContent(u: Window; dx,dy: INTEGER);        PROCEDURE MoveOriginTo(u: Window; x0,y0: INTEGER);
  PROCEDURE SelectForOutput(u: Window);                      PROCEDURE CurrentOutputWindow(): Window;

  TYPE PaintMode = (replace, paint, invert, erase);
    Hue = [0..359];     GreyContent = (light, lightGrey, grey, darkGrey, dark);         Saturation = [0..100];
    Color = RECORD hue: Hue; greyContent: GreyContent; saturation: Saturation; END;
    PatLine = BYTE;     Pattern = ARRAY [0..7] OF PatLine;

  VAR pat: ARRAY [light..dark] OF Pattern;   black, white, red, green, blue, cyan, magenta, yellow: Color;

  PROCEDURE SetMode(mode: PaintMode);                  PROCEDURE GetMode(VAR mode: PaintMode);
  PROCEDURE SetBackground(c: Color; pat: Pattern);     PROCEDURE GetBackground(VAR c: Color; VAR pat: Pattern);
  PROCEDURE SetColor(c: Color);                        PROCEDURE GetColor(VAR c: Color);
```

```
  PROCEDURE SetPattern(p: Pattern);                    PROCEDURE GetPattern(VAR p: Pattern);
  PROCEDURE IdentifyPos(x,y: INTEGER; VAR line,col: CARDINAL);
  PROCEDURE IdentifyPoint(line,col: CARDINAL; VAR x,y: INTEGER);
  PROCEDURE MaxCol(): CARDINAL;                        PROCEDURE MaxLn(): CARDINAL;
  PROCEDURE CellWidth(): INTEGER;                      PROCEDURE CellHeight(): INTEGER;
| PROCEDURE StringArea (s: ARRAY OF CHAR; VAR a: RectArea; VAR baseLine,sepSpace: INTEGER);
| PROCEDURE StringWidth (VAR s: ARRAY OF CHAR): INTEGER;
  PROCEDURE BackgroundWidth(): INTEGER;                PROCEDURE BackgroundHeight(): INTEGER;
  PROCEDURE SetEOWAction(u: Window; action: PROC);     PROCEDURE GetEOWAction(u: Window; VAR action: PROC);
  PROCEDURE EraseContent;                              PROCEDURE RedrawContent;
  PROCEDURE SetClipping(cr: RectArea);                 PROCEDURE GetClipping(VAR cr: RectArea);
  PROCEDURE RemoveClipping;

  TYPE WindowFont = (Chicago, Monaco, Geneva, NewYork);        FontStyles = (bold, italic, underline);
|   LaserFont = (Times, Helvetica, Courier, Symbol);          FontStyle  = SET OF FontStyles;

  PROCEDURE SetWindowFont(wf: WindowFont; size: CARDINAL; style: FontStyle);
  PROCEDURE GetWindowFont(VAR wf: WindowFont; VAR size: CARDINAL; VAR style: FontStyle);
| PROCEDURE SetLaserFont(lf: LaserFont; size: CARDINAL; style: FontStyle);
| PROCEDURE GetLaserFont(VAR lf: LaserFont; VAR size: CARDINAL; VAR style: FontStyle);
  PROCEDURE SetPos(line,col: CARDINAL);                PROCEDURE GetPos(VAR line,col: CARDINAL);
  PROCEDURE ShowCaret(on: BOOLEAN);                    PROCEDURE Invert(on: BOOLEAN);
  PROCEDURE Write(ch: CHAR);                           PROCEDURE WriteString(s: ARRAY OF CHAR);
| PROCEDURE WriteLn;                                   PROCEDURE WriteVarString(VAR s: ARRAY OF CHAR);
| PROCEDURE WriteCard(c,n: CARDINAL);                  PROCEDURE WriteLongCard(lc: LONGCARD; n: CARDINAL);
| PROCEDURE WriteInt(c: INTEGER; n: CARDINAL);         PROCEDURE WriteLongInt(li: LONGINT; n: CARDINAL);
  PROCEDURE WriteReal(r: REAL; n,dec: CARDINAL);       PROCEDURE WriteRealSci(r: REAL; n,dec: CARDINAL);
| PROCEDURE WriteLongReal(lr:LONGREAL; n,dec:CARDINAL); PROCEDURE WriteLongRealSci(lr: LONGREAL; n,dec: CARDINAL);
  PROCEDURE SetPen(x,y: INTEGER);                      PROCEDURE GetPen(VAR x,y: INTEGER);
| PROCEDURE SetBrushSize(width,height: INTEGER);       PROCEDURE GetBrushSize(VAR width,height: INTEGER);
  PROCEDURE Dot(x,y: INTEGER);                         PROCEDURE LineTo(x,y: INTEGER);
  PROCEDURE Circle(x,y: INTEGER; radius: CARDINAL; filled: BOOLEAN; fillpat: Pattern);
  PROCEDURE Area(r: RectArea; pat: Pattern);           PROCEDURE CopyArea(sourceArea: RectArea; dx,dy: INTEGER);
  PROCEDURE MapArea(sourceArea,destArea: RectArea);
  PROCEDURE DisplayPredefinedPicture (fileName: ARRAY OF CHAR; pictureID: INTEGER;     f: RectArea);
| PROCEDURE GetPredefinedPictureFrame(fileName: ARRAY OF CHAR; pictureID: INTEGER; VAR f: RectArea);
  PROCEDURE StartPolygon;                              PROCEDURE CloseAndFillPolygon(pat: Pattern);
| PROCEDURE DrawAndFillPoly(nPoints: CARDINAL; VAR x, y: ARRAY OF INTEGER; VAR withEdge: ARRAY OF BOOLEAN;
                           VAR edgeColors: ARRAY OF Color; isFilled: BOOLEAN; fillColor: Color; fillPattern: Pattern);

  TYPE QDVHSelect = (v,h);   QDVHSelectR = [v..h];
    QDPoint = RECORD CASE:INTEGER OF 0: v,h: INTEGER; | 1: vh: ARRAY QDVHSelectR OF INTEGER; END; END;
    QDRect = RECORD CASE:INTEGER OF 0: top,left,bottom,right: INTEGER; | 1: topLeft,botRight: QDPoint; END; END;

  PROCEDURE XYToQDPoint(x,y: INTEGER; VAR p: QDPoint);       PROCEDURE RectAreaToQDRect(r: RectArea; VAR qdr: QDRect);
  PROCEDURE SelectRestoreCopy(u: Window);                    PROCEDURE SetRestoreCopy(u: Window; rcp: ADDRESS);
  PROCEDURE Turn(angle: INTEGER);       PROCEDURE TurnTo(angle: INTEGER);       PROCEDURE MoveBy(distance: CARDINAL);
  PROCEDURE ScaleUC(r: RectArea; xmin,xmax,ymin,ymax: REAL); PROCEDURE GetUC(VAR r: RectArea; VAR xmin,xmax,ymin,ymax: REAL);
  PROCEDURE ConvertPointToUC(x,y: INTEGER; VAR xUC,yUC: REAL); PROCEDURE ConvertUCToPoint(xUC,yUC: REAL; VAR x,y: INTEGER);
  PROCEDURE UCFrame;            PROCEDURE EraseUCFrame;      PROCEDURE EraseUCFrameContent;
  PROCEDURE SetUCPen(xUC,yUC: REAL);                         PROCEDURE GetUCPen(VAR xUC,yUC: REAL);
  PROCEDURE UCDot(xUC,yUC: REAL);                            PROCEDURE UCLineTo(xUC,yUC: REAL);
  PROCEDURE DrawSym(ch: CHAR);

(*=======================================          DMWindows          =======================================*)

  TYPE Window;
|   WindowKind = (GrowOrShrinkOrDrag, FixedSize, FixedLocation, FixedLocTitleBar);
|   ModalWindowKind = (DoubleFrame, SingleFrameShadowed);
    ScrollBars = (WithVerticalScrollBar, WithHorizontalScrollBar, WithBothScrollBars, WithoutScrollBars);
    CloseAttr = (WithCloseBox, WithoutCloseBox);             ZoomAttr = (WithZoomBox, WithoutZoomBox);
    RectArea = RECORD x,y,w,h: INTEGER END;                  WindowFrame = RectArea;
    WFFixPoint = (bottomLeft, topLeft);                      RestoreProc = PROCEDURE (Window);
    WindowProc = PROCEDURE (Window);
|   WindowHandlers = (clickedInContent, broughtToFront, removedFromFront,
                     redefined, onlyMoved, disappeared, reappeared, closing);

| VAR background: Window;          WindowsDone: BOOLEAN;       notExistingWindow: Window;

| PROCEDURE NoBackground;                                    PROCEDURE ReshowBackground;
  PROCEDURE OuterWindowFrame(innerf: WindowFrame; wk: WindowKind; s: ScrollBars; VAR outerf: RectArea);
  PROCEDURE InnerWindowFrame(outerf: WindowFrame; wk: WindowKind; s: ScrollBars; VAR innerf: RectArea);
  PROCEDURE CreateWindow(VAR u: Window; wk: WindowKind; s: ScrollBars; c: CloseAttr; z: ZoomAttr;
                       fixPoint: WFFixPoint; f: WindowFrame; title: ARRAY OF CHAR; Repaint: RestoreProc);
| PROCEDURE CreateModalWindow(VAR u: Window; wk: ModalWindowKind; s: ScrollBars; f: WindowFrame; Repaint: RestoreProc);
  PROCEDURE UsePredefinedWindow(VAR u: Window; fileName: ARRAY OF CHAR; windowID: INTEGER;
                       fixPoint: WFFixPoint; Repaint: RestoreProc);
| PROCEDURE CreateTitledModalWindow(VAR u: Window; title: ARRAY OF CHAR; f: WindowFrame);     CONST DoubleFrameTitled = 3;
  PROCEDURE RedefineWindow(u: Window; f: WindowFrame);       PROCEDURE RedrawTitle(u: Window; title: ARRAY OF CHAR);
| PROCEDURE MakeWindowInvisible(u: Window);                 PROCEDURE MakeWindowVisible(u: Window);
| PROCEDURE IsNowVisible(u: Window): BOOLEAN;
| PROCEDURE WindowLevel(u: Window): CARDINAL;
| PROCEDURE GetWindowCharacteristics(u: Window; VAR wk: INTEGER; VAR modalKind: BOOLEAN; VAR s: ScrollBars; VAR c: CloseAttr;
                                    VAR z: ZoomAttr; VAR fixPoint: WFFixPoint; VAR f: WindowFrame; VAR title: ARRAY OF CHAR);
  PROCEDURE DummyRestoreProc(u: Window);                     PROCEDURE AutoRestoreProc(u: Window);
  PROCEDURE SetRestoreProc(u: Window; r: RestoreProc);       PROCEDURE GetRestoreProc(u: Window; VAR r: RestoreProc);
  PROCEDURE StartAutoRestoring(u:Window; r: RectArea);       PROCEDURE StopAutoRestoring(u: Window);
  PROCEDURE AutoRestoring(u: Window): BOOLEAN;               PROCEDURE GetHiddenBitMapSize(u: Window; VAR r: RectArea);
  PROCEDURE UpdateWindow(u: Window);                         PROCEDURE InvalidateContent(u: Window);
| PROCEDURE UpdateAllWindows;
| PROCEDURE AddWindowHandler(u: Window; wh: WindowHandlers; wpp: WindowProc; prio: INTEGER);
| PROCEDURE RemoveWindowHandler(u: Window; wh: WindowHandlers; wpp: WindowProc);
  PROCEDURE GetWindowFrame(u: Window; VAR f: WindowFrame);   PROCEDURE GetWFFixPoint(u: Window; VAR loc: WFFixPoint);
  PROCEDURE DoForAllWindows(action: WindowProc);
| PROCEDURE UseWindowModally(u: Window; VAR terminateModalDialog, cancelModalDialog: BOOLEAN);
  PROCEDURE PutOnTop(u: Window);                             PROCEDURE FrontWindow(): Window;
  PROCEDURE RemoveWindow(VAR u: Window);                     PROCEDURE RemoveAllWindows;
  PROCEDURE WindowExists(u: Window): BOOLEAN;                PROCEDURE RedrawBackground;
| PROCEDURE AttachWindowObject(u: Window; obj: ADDRESS);     PROCEDURE WindowObject(u: Window): ADDRESS;
```

```
                (*****************************************************************************)
                (*#####                    O P T I O N A L   M O D U L E S                #####*)
                (*****************************************************************************)


(*=======================================          DM2DGraphs          =======================================*)

  TYPE Graph;   Curve;
    LabelString = ARRAY[0..255] OF CHAR;              GridFlag = (withGrid, withoutGrid);
    ScalingType = (lin, log, negLog);                 PlottingStyle = (solid, slash, slashDot, dots, hidden, wipeout);
    Range = RECORD  min,max: REAL  END;               GraphProc = PROCEDURE(Graph);
    AxisType = RECORD range: Range; scale: ScalingType; dec: CARDINAL; tickD: REAL; label: LabelString; END;

| VAR DM2DGraphsDone: BOOLEAN;                       notExistingGraph: Graph;    notExistingCurve: Curve;

  PROCEDURE DefGraph(VAR g: Graph; u: Window; r: RectArea; xAxis, yAxis: AxisType; grid: GridFlag);
  PROCEDURE DefCurve(g: Graph; VAR c: Curve; col: Color; style: PlottingStyle; sym: CHAR);
  PROCEDURE RedefGraph(g: Graph; r: RectArea; xAxis,yAxis:AxisType; grid: GridFlag);
  PROCEDURE RedefCurve(c: Curve; col: Color; style: PlottingStyle; sym: CHAR);
  PROCEDURE ClearGraph(g: Graph);                     PROCEDURE DrawGraph(g: Graph);
  PROCEDURE DrawLegend(c: Curve; x,y: INTEGER; comment: ARRAY OF CHAR);
  PROCEDURE RemoveGraph(VAR g: Graph);                PROCEDURE RemoveAllGraphs(u: Window);
  PROCEDURE RemoveCurve(VAR c: Curve);
  PROCEDURE GraphExists(g: Graph ): BOOLEAN;          PROCEDURE CurveExists(g: Graph; c: Curve): BOOLEAN;
  PROCEDURE DoForAllGraphs(u: Window; gp: GraphProc);
  PROCEDURE SetNegLogMin(nlm: REAL);                  PROCEDURE SetGapSym(ch: CHAR);      PROCEDURE GetGapSym(VAR ch: CHAR);

  PROCEDURE Move(c: Curve; x,y: REAL);                PROCEDURE Plot(curve: Curve; newX,newY: REAL);
  PROCEDURE PlotSym(g: Graph; x,y: REAL; sym: CHAR);  PROCEDURE PlotCurve(c: Curve; nrOfPoints: CARDINAL; x,y: ARRAY OF REAL);
  PROCEDURE GraphToWindowPoint(g: Graph; xReal,yReal: REAL; VAR xInt,yInt: INTEGER);
  PROCEDURE WindowToGraphPoint(g: Graph;  xInt,yInt: INTEGER; VAR xReal,yReal: REAL);

(*=======================================           DMAlerts           =======================================*)

  PROCEDURE WriteMessage(line,col: CARDINAL; msg: ARRAY OF CHAR);
  PROCEDURE ShowAlert(height,width: CARDINAL; WriteMessages: PROC);
  PROCEDURE ShowPredefinedAlert(fileName: ARRAY OF CHAR; alertID: INTEGER; str1,str2,str3,str4: ARRAY OF CHAR);

(*=======================================          DMClipboard         =======================================*)

  TYPE EditCommands = (undo, cut, copy, paste, clear);

| VAR ClipboardDone: BOOLEAN;

  PROCEDURE InstallEditMenu(UndoProc, CutProc, CopyProc, PasteProc, ClearProc: PROC);
| PROCEDURE RemoveEditMenu;                                  PROCEDURE UseEditMenu;
  PROCEDURE EnableEditMenu;                                 PROCEDURE DisableEditMenu;
  PROCEDURE EnableEditCommand(whichone: EditCommands);      PROCEDURE DisableEditCommand(whichone: EditCommands);

| PROCEDURE PutPictureIntoClipboard;
| PROCEDURE GetPictureFromClipboard(simultaneousDisplay: BOOLEAN; destRect: RectArea);
| PROCEDURE PutTextIntoClipboard;
| PROCEDURE GetTextFromClipboard(simultaneousDisplay: BOOLEAN; destRect: RectArea; fromLine: LONGINT);

(*=======================================           DMClock            =======================================*)

| CONST  Jan = 1; Feb = 2; Mar = 3; Apr = 4; Mai = 5; Jun = 6;  Jul = 7; Aug = 8; Sep = 9; Oct =10; Nov =11; Dec = 12;
|        Sun = 1; Mon = 2; Tue = 3; Wed = 4; Thu = 5; Fri = 6; Sat = 7;

| PROCEDURE Today(VAR year, month, day, dayOfWeek: INTEGER);    PROCEDURE Now(VAR hour, minute, second: INTEGER);
| PROCEDURE NowInSeconds(): LONGINT;
| PROCEDURE InterpreteSeconds(secs: LONGINT; VAR year, month, day, hour, minute, second, dayOfWeek: INTEGER);
| PROCEDURE ConvertDateToSeconds(year, month, day, hour, minute, second: INTEGER; VAR secs: LONGINT);

(*=======================================          DMEditFields        =======================================*)

  TYPE EditItem;        RadioBut;       EditHandler = PROCEDURE(EditItem);
    ItemType = (charField, stringField, textField, cardField, intField, realField,
                pushButton, radioButtonSet, checkBox, scrollBar);   Direction = (horizontal, vertical);

| VAR EditFieldsDone: BOOLEAN;   notInstalledEditItem: EditItem;    notInstalledRadioBut: RadioBut;

| PROCEDURE MakeCharField(u: Window; VAR ei: EditItem; x,y: INTEGER; ch: CHAR; charset: ARRAY OF CHAR);
| PROCEDURE MakeStringField(u: Window; VAR ei: EditItem; x,y: INTEGER; fw: CARDINAL; string: ARRAY OF CHAR);
| PROCEDURE MakeTextField(u: Window; VAR ei: EditItem; x,y: INTEGER; fw,lines: CARDINAL; string: ARRAY OF CHAR);
| PROCEDURE MakeCardField(u: Window; VAR ei: EditItem; x,y: INTEGER; fw: CARDINAL; card: CARDINAL; minCard,maxCard: CARDINAL);
| PROCEDURE MakeLongCardField(u: Window; VAR ei: EditItem; x,y: INTEGER; fw: CARDINAL;
                            card: LONGCARD; minCard,maxCard: LONGCARD);
| PROCEDURE MakeIntField(u: Window; VAR ei: EditItem; x,y: INTEGER; fw: CARDINAL; int: INTEGER; minInt,maxInt: INTEGER);
| PROCEDURE MakeLongIntField(u: Window; VAR ei: EditItem; x,y: INTEGER; fw: CARDINAL;
                           int: LONGINT; minInt,maxInt: LONGINT);
| PROCEDURE MakeRealField(u: Window; VAR ei: EditItem; x,y: INTEGER; fw: CARDINAL; real: REAL; minReal,maxReal: REAL);
| PROCEDURE MakeLongRealField(u: Window; VAR ei: EditItem; x,y: INTEGER; fw: CARDINAL;
                           real: LONGREAL; minReal,maxReal: LONGREAL);
| PROCEDURE MakePushButton(u: Window; VAR ei: EditItem; x,y: INTEGER;
                          buttonWidth: CARDINAL; buttonText: ARRAY OF CHAR; pushButtonAction: PROC);
| PROCEDURE UseAsDefaultButton(pushButton: EditItem);
  PROCEDURE BeginRadioButtonSet(u: Window; VAR ei: EditItem);
| PROCEDURE AddRadioButton(VAR radButt: RadioBut; x,y: INTEGER; text: ARRAY OF CHAR);
  PROCEDURE EndRadioButtonSet(checkedRadioButton: RadioBut);
| PROCEDURE MakeCheckBox(u: Window; VAR ei: EditItem; x,y: INTEGER; text: ARRAY OF CHAR; boxChecked: BOOLEAN);
| PROCEDURE MakeScrollBar(u: Window; VAR ei: EditItem; x, y, length: INTEGER; sbd: Direction; minVal,maxVal: REAL;
                         smallStep, bigStep: REAL; curVal: REAL; actionProc: PROC);

  PROCEDURE SetChar(ei: EditItem; newCh:CHAR);                  PROCEDURE SetString(ei: EditItem; newStr: ARRAY OF CHAR);
| PROCEDURE SetText(ei: EditItem; VAR text: ARRAY OF CHAR);
| PROCEDURE SetCardinal(ei: EditItem;  newValue: CARDINAL);     PROCEDURE SetLongCardinal(ei: EditItem;  newValue: LONGCARD);
```

```
| PROCEDURE SetInteger(ei: EditItem;  newValue: INTEGER);       PROCEDURE SetLongInteger(ei: EditItem;  newValue: LONGINT);
| PROCEDURE SetReal(ei: EditItem; newValue: REAL);              PROCEDURE SetLongReal(ei: EditItem; newValue: LONGREAL);
  PROCEDURE SetRadioButtonSet(ei: EditItem; checkedRadioButton: RadioBut);
  PROCEDURE SetCheckBox(ei: EditItem; boxChecked: BOOLEAN);
| PROCEDURE SetScrollBar(ei: EditItem; newValue: REAL);

  PROCEDURE IsChar(ei: EditItem; VAR ch:CHAR): BOOLEAN;         PROCEDURE GetString(ei: EditItem; VAR str: ARRAY OF CHAR);
| PROCEDURE GetText(ei: EditItem; VAR text: ARRAY OF CHAR);
| PROCEDURE IsCardinal(ei: EditItem; VAR c: CARDINAL): BOOLEAN; PROCEDURE IsLongCardinal(ei: EditItem; VAR c: LONGCARD): BOOLEAN;
| PROCEDURE IsInteger(ei: EditItem; VAR i: INTEGER): BOOLEAN;   PROCEDURE IsLongInteger(ei: EditItem; VAR i: LONGINT): BOOLEAN;
| PROCEDURE IsReal(ei: EditItem; VAR r: REAL): BOOLEAN;         PROCEDURE IsLongReal(ei: EditItem; VAR r: LONGREAL): BOOLEAN;
  PROCEDURE GetRadioButtonSet(ei: EditItem; VAR checkedRadioButton: RadioBut);
  PROCEDURE GetCheckBox(ei: EditItem; VAR boxChecked: BOOLEAN);
| PROCEDURE GetScrollBar(ei: EditItem; VAR r: REAL);

| PROCEDURE InstallEditHandler(u: Window; eh: EditHandler);     PROCEDURE GetEditHandler(u: Window; VAR eh: EditHandler);
| PROCEDURE SelectField(ei: EditItem);                         PROCEDURE ClearFieldSelection (u: Window);

| PROCEDURE EnableItem(ei: EditItem);   PROCEDURE DisableItem(ei: EditItem);   PROCEDURE IsEnabled(ei: EditItem): BOOLEAN;

  PROCEDURE EditItemExists(ei: EditItem): BOOLEAN;              PROCEDURE GetEditItemType(ei: EditItem; VAR it: ItemType);
| PROCEDURE RadioButtonExists(rb: RadioBut): BOOLEAN;
| PROCEDURE EditItemLevel(ei: EditItem): CARDINAL;             PROCEDURE RadioButtonLevel(rb: RadioBut): CARDINAL;
  PROCEDURE RemoveEditItem(VAR ei: EditItem);                   PROCEDURE RemoveAllEditItems(u: Window);
| PROCEDURE AttachEditFieldObject(ei: EditItem; obj: ADDRESS); PROCEDURE EditFieldObject(ei: EditItem): ADDRESS;

(*=======================================          DMEntryForms          =======================================*)

  TYPE FormFrame = RECORD x,y: INTEGER; lines,columns: CARDINAL END;    DefltUse = (useAsDeflt, noDeflt);    RadioButtonID;

| VAR FieldInstalled: BOOLEAN;  notInstalledRadioButton: RadioButtonID;

  PROCEDURE WriteLabel(line,col: CARDINAL; text: ARRAY OF CHAR);
  PROCEDURE CharField(line,col: CARDINAL; VAR ch: CHAR; du: DefltUse; charset: ARRAY OF CHAR);
  PROCEDURE StringField(line,col: CARDINAL; fw: CARDINAL; VAR string: ARRAY OF CHAR; du: DefltUse);
  PROCEDURE CardField(line,col: CARDINAL; fw: CARDINAL; VAR card: CARDINAL; du: DefltUse; minCard,maxCard: CARDINAL);
| PROCEDURE LongCardField (line,col: CARDINAL; fw: CARDINAL; VAR longCard: LONGCARD; du: DefltUse; minLCard,maxLCard: LONGCARD);
  PROCEDURE IntField(line,col: CARDINAL; fw: CARDINAL; VAR int: INTEGER; du: DefltUse; minInt,maxInt: INTEGER);
| PROCEDURE LongIntField (line,col: CARDINAL; fw: CARDINAL; VAR longInt: LONGINT; du: DefltUse; minLInt,maxLInt: LONGINT);
  PROCEDURE RealField(line,col: CARDINAL; fw: CARDINAL; VAR real: REAL; du: DefltUse; minReal,maxReal: REAL);
| PROCEDURE LongRealField (line,col: CARDINAL; fw,dig: CARDINAL; fmt: RealFormat; VAR longReal: LONGREAL; du: DefltUse;
                          minLReal,maxLReal: LONGREAL);
  PROCEDURE PushButton(line,col: CARDINAL; buttonText: ARRAY OF CHAR; buttonWidth: CARDINAL; pushButtonAction: PROC);
  PROCEDURE DefineRadioButtonSet(VAR radioButtonVar: RadioButtonID);
  PROCEDURE RadioButton(VAR radButt: RadioButtonID; line,col: CARDINAL; text: ARRAY OF CHAR);
  PROCEDURE CheckBox(line,col: CARDINAL; text: ARRAY OF CHAR; VAR checkBoxVar: BOOLEAN);
  PROCEDURE UseEntryForm(bf: FormFrame; VAR ok: BOOLEAN);

(*=======================================            DMFiles            =======================================*)

  CONST EOL = 36C;

  TYPE Response = (done, filenotfound, volnotfound, cancelled, unknownfile, toomanyfiles, diskfull, memfull,
                   alreadyopened, isbusy, locked, notdone);
    HiddenFileInfo;         IOMode = (reading, writing);
    TextFile = RECORD
                 res:        Response;
                 filename:   ARRAY [0..255] OF CHAR;
                 path:       ARRAY [0..63] OF CHAR;
                 curIOMode:  IOMode;
                 curChar:    CHAR;
                 fhint:      HiddenFileInfo;
               END;

  VAR
|   legalNum: BOOLEAN; (* read only *)                          PROCEDURE LastResultCode(): INTEGER;
    neverOpenedFile: TextFile; (* read only *)

  PROCEDURE GetExistingFile(VAR f: TextFile; prompt: ARRAY OF CHAR);
  PROCEDURE CreateNewFile(VAR f: TextFile; prompt, defaultName: ARRAY OF CHAR);
  PROCEDURE Lookup(VAR f: TextFile; pathAndFileName: ARRAY OF CHAR; new: BOOLEAN);
| PROCEDURE ReadOnlyLookup(VAR f: TextFile; pathAndFileName: ARRAY OF CHAR);
| PROCEDURE Close(VAR f: TextFile);                             PROCEDURE IsOpen(VAR f: TextFile): BOOLEAN;
| PROCEDURE FileExists(VAR f: TextFile): BOOLEAN;               PROCEDURE FileLevel(VAR f: TextFile): CARDINAL;

  PROCEDURE Delete(VAR f: TextFile);                            PROCEDURE Rename(VAR f: TextFile; filename: ARRAY OF CHAR);
  PROCEDURE Reset(VAR f: TextFile);                             PROCEDURE Rewrite(VAR f: TextFile);
| PROCEDURE AppendAtEOF(VAR f: TextFile);                       PROCEDURE FileSize(VAR f: TextFile): LONGINT;

  PROCEDURE EOF(VAR f: TextFile): BOOLEAN;
  PROCEDURE ReadByte(VAR f: TextFile; VAR b: BYTE);             PROCEDURE WriteByte(VAR f: TextFile; b: BYTE);
  PROCEDURE ReadChar(VAR f: TextFile; VAR ch: CHAR);            PROCEDURE WriteChar(VAR f: TextFile; ch: CHAR);
  PROCEDURE ReadChars(VAR f: TextFile; VAR string: ARRAY OF CHAR);  PROCEDURE WriteChars(VAR f: TextFile; string: ARRAY OF CHAR);
| PROCEDURE WriteEOL(VAR f: TextFile);                          PROCEDURE WriteVarChars(VAR f: TextFile; VAR string: ARRAY OF CHAR);
| PROCEDURE SkipGap(VAR f: TextFile);                          PROCEDURE SkipGapWithinLn(VAR f: TextFile);
  PROCEDURE Again(VAR f: TextFile);
  PROCEDURE GetCardinal(VAR f: TextFile; VAR c: CARDINAL);      PROCEDURE GetLongCard(VAR f: TextFile; VAR c: LONGCARD);
  PROCEDURE PutCardinal(VAR f:TextFile; c:CARDINAL; n:CARDINAL);    PROCEDURE PutLongCard(VAR f: TextFile; lc: LONGCARD;
                                                                                          n: CARDINAL);
  PROCEDURE GetInteger(VAR f: TextFile; VAR i: INTEGER);        PROCEDURE GetLongInt(VAR f: TextFile; VAR i: LONGINT);
  PROCEDURE PutInteger(VAR f: TextFile; i: INTEGER; n: CARDINAL);   PROCEDURE PutLongInt(VAR f: TextFile; li: LONGINT;
                                                                                          n:CARDINAL);
  PROCEDURE GetReal(VAR f: TextFile; VAR x: REAL);              PROCEDURE GetLongReal(VAR f: TextFile; VAR x: LONGREAL);
  PROCEDURE PutReal(VAR f: TextFile; x: REAL; n, dec: CARDINAL);    PROCEDURE PutRealSci(VAR f: TextFile; x: REAL; n: CARDINAL);
  PROCEDURE PutLongReal(VAR f: TextFile; lr: LONGREAL; n,dec: CARDINAL);
  PROCEDURE PutLongRealSci(VAR f: TextFile; lr: LONGREAL; n,dec: CARDINAL);

| PROCEDURE AlterIOMode (VAR f: TextFile; newMode: IOMode);
| PROCEDURE SetFilePos( VAR f: TextFile; pos: LONGINT );        PROCEDURE GetFilePos( VAR f: TextFile; VAR pos: LONGINT );
| PROCEDURE ReadByteBlock ( VAR f: TextFile; VAR buf: ARRAY OF BYTE; VAR count: LONGINT );
```

```
| PROCEDURE WriteByteBlock( VAR f: TextFile; VAR buf: ARRAY OF BYTE; VAR count: LONGINT );

| PROCEDURE SetFileFilter(f1,f2,f3,f4: ARRAY OF CHAR);              PROCEDURE GetFileFilter(VAR f1,f2,f3,f4: ARRAY OF CHAR);
| PROCEDURE UseAsTypeAndCreator(filetype,creator: ARRAY OF CHAR);   PROCEDURE UsedTypeAndCreator(VAR filetype,creator: ARRAY OF CHAR);
| PROCEDURE HasTypeAndCreator(VAR f: TextFile; VAR filetype,creator: ARRAY OF CHAR);

(*=======================================          DMFloatEnv          =======================================*)

  CONST invalid = 0; underflow = 1; overflow = 2; divideByZero = 3; inexact = 4;
    haltIfInvalid = 0; haltIfUnderflow = 1; haltIfOverflow =  2; haltIfDivideByZero =  3; haltIfInexact =  4;
    flagIfInvalid = 8; flagIfUnderflow = 9; flagIfOverflow = 10; flagIfDivideByZero = 11; flagIfInexact = 12;
    IEEEFloatDefaultEnv = {};                                    DMFloatDefaultEnv = {haltIfInvalid, haltIfOverflow, haltIfDivideByZero};

  TYPE Exception = [invalid..inexact];                    FloatEnvironment = BITSET;
    RoundDir = (toNearest, upward, downward, towardZero);  RoundPre = (extPrecision, dblPrecision, sglPrecision);

  PROCEDURE HaltEnabled(which: Exception): BOOLEAN;
  PROCEDURE EnableHalt(which: Exception);                   PROCEDURE DisableHalt(which: Exception);
  PROCEDURE ExceptionPending(which: Exception): BOOLEAN;
  PROCEDURE RaiseException(which: Exception);               PROCEDURE ClearException(which: Exception);
  PROCEDURE SetPrecision(p: RoundPre);                      PROCEDURE GetPrecision(VAR p: RoundPre);
  PROCEDURE SetRound(r: RoundDir);                          PROCEDURE GetRound(VAR r: RoundDir);
  PROCEDURE GetEnvironment(VAR e: FloatEnvironment);        PROCEDURE SetEnvironment(e: FloatEnvironment);
  PROCEDURE ProcEntry(VAR savedEnv: FloatEnvironment);      PROCEDURE ProcExit(savedEnv: FloatEnvironment);

(*=======================================          DMKeyChars          =======================================*)

| CONST mouse=0;    command=1;  alt=1;  option=2;   shift=3;    capslock=4;     control=5;
| VAR cursorUp, cursorDown, cursorLeft, cursorRight, homeKey, endKey, pageUp, pageDown, helpKey, enter, return, delete,
|     backspace, tab, esc, hardBlank: CHAR; (* READ ONLY! *)

| VAR BestCH: PROCEDURE(CHAR): CHAR; (* READ ONLY! *)
| TYPE ComputerPlatform=( Mac, IBMPCCompatible, UNIXMachine);        PROCEDURE ProgrammedOn( c: ComputerPlatform);

| PROCEDURE PCCHAR( macCh: CHAR): CHAR;     PROCEDURE MacCHAR( pcCh: CHAR): CHAR;
| PROCEDURE PCASCII( pcCh: CHAR): CHAR;     PROCEDURE MacASCII( macCh: CHAR): CHAR;

(*=======================================      DMMathLib/DMMathLF      =======================================*)

  PROCEDURE Sqrt  (x: REAL):    REAL;
  PROCEDURE Exp   (x: REAL):    REAL;   PROCEDURE Ln    (x: REAL):    REAL;
  PROCEDURE Sin   (x: REAL):    REAL;   PROCEDURE Cos   (x: REAL):    REAL;
  PROCEDURE ArcTan(x: REAL):    REAL;
  PROCEDURE Real  (x: INTEGER): REAL;   PROCEDURE Entier(x: REAL):    INTEGER;

| PROCEDURE Randomize;                  PROCEDURE RandomInt(upperBound: INTEGER): INTEGER;    PROCEDURE RandomReal(): REAL;

(*=======================================          DMLongMathLib       =======================================*)

| PROCEDURE LongSqrt   (x: LONGREAL): LONGREAL;
| PROCEDURE LongExp    (x: LONGREAL): LONGREAL;   PROCEDURE LongLn     (x: LONGREAL): LONGREAL;
| PROCEDURE LongSin    (x: LONGREAL): LONGREAL;   PROCEDURE LongCos    (x: LONGREAL): LONGREAL;
| PROCEDURE LongArcTan (x: LONGREAL): LONGREAL;
| PROCEDURE LongReal   (x: LONGINT) : LONGREAL;   PROCEDURE LongEntier (x: LONGREAL): LONGINT;

(*=======================================          DMOpSys             =======================================*)

| CONST noError = 0;     notDone = -2;  inexistent = -1;  notOpen = 0;  readOnly = 1;  alreadyWrite = 2;  (* codes returned by CurrentFileUse*)

| TYPE ProgStatus = (regular, moduleNotFound, fileNotFound, illegalKey, readError, badSyntax, noMemory, alreadyLoaded,
|                    killed, tooManyPrograms, continue, noApplication);
|   DirectoryProc = PROCEDURE (INTEGER, ARRAY OF CHAR, BOOLEAN, VAR BOOLEAN);
|   MessageResponder = PROCEDURE (ARRAY OF CHAR, ARRAY OF CHAR, INTEGER);
|   InitDocuHandlingProc = PROCEDURE (INTEGER);     DocuHandler = PROCEDURE (INTEGER, ARRAY OF CHAR, ARRAY OF CHAR, VAR BOOLEAN);

| VAR profileFName: ARRAY [0..127] OF CHAR;

| PROCEDURE CurWorkDirectory(VAR path: ARRAY OF CHAR);      PROCEDURE GetLastResultCode(): INTEGER;
| PROCEDURE CreateDir( path, dirN: ARRAY OF CHAR;  VAR done: BOOLEAN );
| PROCEDURE DeleteDir( path, dirN: ARRAY OF CHAR;  VAR done: BOOLEAN );
| PROCEDURE RenameDir( path, oldDirN, newDirN: ARRAY OF CHAR; VAR done: BOOLEAN );
| PROCEDURE DirInfo( path, dirN: ARRAY OF CHAR; VAR dirExists, containsFiles : BOOLEAN );
| PROCEDURE DoForAllFilesInDirectory(path: ARRAY OF CHAR; dp: DirectoryProc);
| PROCEDURE CurrentFileUse (path,fileName: ARRAY OF CHAR): INTEGER;
| PROCEDURE GetFileDialog(prompt,fileTypes: ARRAY OF CHAR; VAR path,fileName: ARRAY OF CHAR): BOOLEAN;
| PROCEDURE GetApplication(VAR path, applName: ARRAY OF CHAR): BOOLEAN;
| PROCEDURE GetFileTypeAndCreator(path,fn: ARRAY OF CHAR; VAR type,creator: ARRAY OF CHAR);
| PROCEDURE SetFileTypeAndCreator(path,fn: ARRAY OF CHAR; type,creator: ARRAY OF CHAR);
| PROCEDURE HasCustomIcon (path,fn: ARRAY OF CHAR): BOOLEAN;
| PROCEDURE SetCustomIconFlag (path,fn: ARRAY OF CHAR; cif: BOOLEAN);
| PROCEDURE GetFileDates(path,fn: ARRAY OF CHAR; VAR creationDate,modificationDate: LONGINT);
| PROCEDURE SetFileDates(path,fn: ARRAY OF CHAR; creationDate,modificationDate: LONGINT);
| PROCEDURE NowSeconds(): LONGINT;                          PROCEDURE TouchFileDate(path,fn: ARRAY OF CHAR);
| PROCEDURE CopyResourceFork(sourcePath,sourceFn, destPath, destFn: ARRAY OF CHAR; VAR done: BOOLEAN);
| PROCEDURE CopyDataFork    (sourcePath,sourceFn, destPath, destFn: ARRAY OF CHAR; VAR done: BOOLEAN);
| PROCEDURE InstallInitDocuOpening (idhp: InitDocuHandlingProc);    PROCEDURE InstallOpenDocuHandler (dh: DocuHandler);
| PROCEDURE InstallInitDocuPrinting(idhp: InitDocuHandlingProc);    PROCEDURE InstallPrintDocuHandler(dh: DocuHandler);
| PROCEDURE SubLaunch(path, prog: ARRAY OF CHAR);           PROCEDURE Transfer(path, prog: ARRAY OF CHAR);
| PROCEDURE IsForegroundProgram(): BOOLEAN;
| PROCEDURE SetMessageResponder(    mr: MessageResponder);  PROCEDURE GetMessageResponder(VAR mr: MessageResponder);
| PROCEDURE SignalMessageToApplication(creatorOfAppl, eventClass, eventID: ARRAY OF CHAR;
                                       msgVal: INTEGER; VAR resultCode: INTEGER);
| PROCEDURE EmulateKeyPress(ch: CHAR; modifier: BITSET);    PROCEDURE EmulateMenuSelection(aliasChar: CHAR);
| PROCEDURE EmulateMouseDown(x,y: INTEGER; modifier: BITSET);
| PROCEDURE TurnMachineOff;                                 PROCEDURE RestartMachine;

| PROCEDURE SetNewPaths;                                    PROCEDURE EmulateMacMETHCopyProtection;
| PROCEDURE CallDMSubProg(prog: ARRAY OF CHAR; leaveLoaded: BOOLEAN; VAR st: ProgStatus);
| PROCEDURE CallM2SubProg(prog: ARRAY OF CHAR; leaveLoaded: BOOLEAN; VAR st: ProgStatus);
| PROCEDURE IncludeLibModules(prog: ARRAY OF CHAR; VAR st: ProgStatus);
```

```
| PROCEDURE UnLoadM2Progs;                                    PROCEDURE AbortM2Prog(st: ProgStatus);
| PROCEDURE SetCompilerFileTypes(creator, sbmType, obmType, rfmType: ARRAY OF CHAR);
| PROCEDURE GetCompilerFileTypes(VAR creator, sbmType, obmType, rfmType: ARRAY OF CHAR);

(*=========================================            DMPortab            =========================================*)

| VAR zero,   one,   two,   ten,   hundred,   thousand:   LONGREAL; (* read only *)
|     zeroLI, oneLI,twoLI, tenLI, hundredLI, thousandLI: LONGINT;  (* read only *)

| PROCEDURE SameProc( p1, p2: ARRAY OF BYTE): BOOLEAN;
| PROCEDURE LongTRUNC( x: LONGREAL): LONGINT;                PROCEDURE LongFLOAT( x: LONGINT): LONGREAL;
| PROCEDURE LONGINTConst( str: ARRAY OF CHAR): LONGINT;      PROCEDURE LONGREALConst( str: ARRAY OF CHAR): LONGREAL;
| PROCEDURE LR(x: REAL): LONGREAL;                                                    PROCEDURE LI(x: INTEGER): LONGINT;

(*=========================================           DMPrinting           =========================================*)

| TYPE PrinterFont = (chicago, newYork, geneva, monaco, times, helvetica, courier, symbol);

| VAR PrintingDone: BOOLEAN;

  PROCEDURE PageSetup;                              PROCEDURE SetHeaderText(h: ARRAY OF CHAR);
  PROCEDURE SetSubHeaderText(sh: ARRAY OF CHAR);    PROCEDURE SetFooterText(f: ARRAY OF CHAR);
  PROCEDURE PrintPicture;
| PROCEDURE PrintText(font: PrinterFont; fontSize: INTEGER; tabwidth: INTEGER);

(*=========================================            DMPTFiles           =========================================*)

| VAR PTFileDone: BOOLEAN;

| PROCEDURE DumpPicture(VAR f: TextFile);
| PROCEDURE LoadPicture (VAR f: TextFile; simulDisplay: BOOLEAN; destRect: RectArea);
| PROCEDURE DumpText(VAR f: TextFile);
| PROCEDURE LoadText (VAR f: TextFile; simulDisplay: BOOLEAN; destRect: RectArea; fromLine: LONGINT);

(*=========================================           DMResources          =========================================*)

| CONST nulCh = 21C;

| TYPE ResourcePointer = POINTER TO Resource;   Resource = ARRAY [0..32000] OF CHAR;     Padding = (noPadding, padToEven, padToOdd);

| VAR theResource: ResourcePointer;   ResourcesDone: BOOLEAN;

| PROCEDURE StartResourceComposition;       PROCEDURE CurPosition(): INTEGER;
| PROCEDURE AddBoolean(b: BOOLEAN);
| PROCEDURE AddInt(int: INTEGER);           PROCEDURE AddLongInt(lint: LONGINT);
| PROCEDURE AddHexInt(int: INTEGER);        PROCEDURE AddHexLongInt  (lint: LONGINT);
| PROCEDURE AddBinInt(int: INTEGER);        PROCEDURE AddBinLongInt(lint: LONGINT);
| PROCEDURE AddReal(r: REAL);               PROCEDURE AddLongReal(lr: LONGREAL);
| PROCEDURE AddHexReal(r: REAL);            PROCEDURE AddHexLongReal(lr: LONGREAL);
| PROCEDURE AddBinReal(r: REAL);            PROCEDURE AddBinLongReal(lr: LONGREAL);
| PROCEDURE AddChar(ch: CHAR);              PROCEDURE AddString(s: ARRAY OF CHAR);
| PROCEDURE AddString255(s: ARRAY OF CHAR; pad: Padding);
| PROCEDURE OverWriteAtPos (VAR x: ARRAY OF BYTE; VAR theResource: ARRAY OF CHAR; VAR curPos: INTEGER);
| PROCEDURE StoreResource(filename: ARRAY OF CHAR; resID: INTEGER);

| PROCEDURE RetrieveResource(filename: ARRAY OF CHAR; resID: INTEGER);
| PROCEDURE FetchBoolean(VAR b: BOOLEAN);
| PROCEDURE FetchInt(VAR int: INTEGER);     PROCEDURE FetchLongInt(VAR lint: LONGINT);
| PROCEDURE FetchHexInt(VAR int: INTEGER);  PROCEDURE FetchHexLongInt(VAR lint: LONGINT);
| PROCEDURE FetchBinInt(VAR int: INTEGER);  PROCEDURE FetchBinLongInt(VAR lint: LONGINT);
| PROCEDURE FetchReal(VAR r: REAL);         PROCEDURE FetchLongReal(VAR lr: LONGREAL);
| PROCEDURE FetchHexReal(VAR r: REAL);      PROCEDURE FetchHexLongReal(VAR lr: LONGREAL);
| PROCEDURE FetchBinReal(VAR r: REAL);      PROCEDURE FetchBinLongReal(VAR lr: LONGREAL);
| PROCEDURE FetchChar(VAR ch: CHAR);        PROCEDURE FetchString(VAR s: ARRAY OF CHAR);
| PROCEDURE FetchString255(VAR s: ARRAY OF CHAR; pad: Padding);

| PROCEDURE DeleteResource(filename: ARRAY OF CHAR; resID: INTEGER);
| PROCEDURE SetResourceName(fileName: ARRAY OF CHAR; resID: INTEGER; name: ARRAY OF CHAR);
| PROCEDURE GetResourceName(fileName: ARRAY OF CHAR; resID: INTEGER; VAR name: ARRAY OF CHAR);
| PROCEDURE SetResourceType(type: ARRAY OF CHAR);
| PROCEDURE GetResourceType(VAR type: ARRAY OF CHAR);

(*=========================================           DMTextFields          =========================================*)

| TYPE TextPointer = POINTER TO TextSegment;  TextSegment = ARRAY [0..32000] OF CHAR;

| PROCEDURE RedefineTextField(textField: EditItem; wf: WindowFrame; withFrame: BOOLEAN);
| PROCEDURE WrapText(textField: EditItem; wrap: BOOLEAN);
| PROCEDURE CopyWTextIntoTextField(textField: EditItem; VAR done: BOOLEAN);
| PROCEDURE CopyTextFromFieldToWText(textField: EditItem);

| PROCEDURE SetSelection(textField: EditItem; beforeCh,afterCh: INTEGER);
| PROCEDURE GetSelection(textField: EditItem; VAR beforeCh,afterCh: INTEGER);
| PROCEDURE GetSelectedChars(textField: EditItem; VAR text: ARRAY OF CHAR);
| PROCEDURE DeleteSelection(textField: EditItem);
| PROCEDURE InsertBeforeCh(textField: EditItem; VAR text: ARRAY OF CHAR; beforeCh: INTEGER);

| PROCEDURE GetTextSizes(textField: EditItem; VAR curTextLength, nrLns, charHeight, firstLnVis,lastLnVis: INTEGER);
| PROCEDURE GrabText(textField: EditItem; VAR txtbeg: TextPointer; VAR curTextLength: INTEGER);
| PROCEDURE ReleaseText(textField: EditItem);
| PROCEDURE FindInText(textField: EditItem; stringToFind: ARRAY OF CHAR; VAR firstCh,lastCh: INTEGER): BOOLEAN;
| PROCEDURE ScrollText(textField: EditItem; dcols,dlines: INTEGER);
| PROCEDURE ScrollTextWithWindowScrollBars(textField: EditItem);
| PROCEDURE AddScrollBarsToText(textField: EditItem; withVerticalScrollBar, withHorizontalScrollBar: BOOLEAN);

(*=========================================            DMWPictIO           =========================================*)

| VAR PictIODone: BOOLEAN;

  PROCEDURE StartPictureSave;                                   PROCEDURE StopPictureSave;
```

```
  PROCEDURE PausePictureSave;                                    PROCEDURE ResumePictureSave;
| PROCEDURE DisplayPicture(ownerWindow: Window; destRect: RectArea);   PROCEDURE DiscardPicture;

  PROCEDURE SetPictureArea(r: RectArea);                         PROCEDURE GetPictureArea(VAR r: RectArea);
| PROCEDURE SetHairLineWidth(f: REAL);                           PROCEDURE GetHairLineWidth(VAR f: REAL);

(*=======================================          DMWTextIO          =======================================*)

| VAR TextIODone: BOOLEAN;

| PROCEDURE StartTextSave;                                                  PROCEDURE StopTextSave;
| PROCEDURE PauseTextSave;                                                  PROCEDURE ResumeTextSave;
| PROCEDURE DisplayText(ownerWindow: Window; destRect: RectArea; fromLine: LONGINT);   PROCEDURE DiscardText;

| PROCEDURE GrabWText(VAR txtbeg: ADDRESS; VAR curTextLength: LONGINT);    PROCEDURE ReleaseWText;
| PROCEDURE AppendWText(txtbeg: ADDRESS; length: LONGINT);               PROCEDURE SetWTextSize(newTextLength: LONGINT);

(================================================       - E N D -      ================================================)
```


| Different from Version 1.0

Index

“Windows 3.1” 33

AddWindowHandler 16

Anonymous ftp 33

Apple® Macintosh® 33

Area 19

AutoScrollProc 16

CellHeight 17

CellWidth 17

CELTIA 2, 3

Character cell coordinates 17

CheckBox 20

Circle 19

Coordinate system 17

- CARDINAL (character cell) type 17
- INTEGER (pixel) type 17
- INTEGER (turtle, polar coordinate) type 18
- REAL (user, graph) type 18

CreateNewFile 21

CreateWindow 14, 15, 16

Current output window 17

DisableCommand 13

DisplayPredefinedPicture 19

DMBase 10

DMConversions 9, 21

DMEditFields 8, 9, 21

DMEntryForms 8, 9, 20

DMFiles 9, 21

DMLanguage 10

DMMaster 6

DMMathLF 9

DMMathLib 9

DMMenus 8

DMMessages 9

DMStorage 9

DMStrings 9, 21

DMSystem 10

DMWindIO 8, 16

DMWindows 8, 14, 15, 16

DMWPictIO 10

DMWTextIO 10

Dot 19

edit fields 21

EnableCommand 13

Entry forms 20

EraseContent 16

Fat Mac 33

Files 21

Frame part 15

GEM Desktop 3

GetExistingFile 21

Global coordinate system 14

Graph panel 18

Hierarchical file system 21

IDA 2, 3

InnerWindowFrame 15

INTERNET 33

LineTo 19

MacMETH 34

MakePushButton 21

MapArea 19

Matrix coordinate system 17

Method 16

Modal dialog 19

Modal dialog boxes 20

Modal dialogs 15

Modeless dialog 19, 21

Modeless dialog boxes 21

MS-DOS 3

Multiple screens 14

Object oriented programming 16

OOP 16

Outer frame 15

OuterWindowFrame 15

Pen drawing 18

Pen position 18

Program events 4

Push button

- “CANCEL” 20

“OK” 20
RadioButton 20
RAMSES 33
RealField 20
RedefineWindow 14
Reflex 33
RemoveWindow 16
ResEdit 19
Resource file 19
RestoreProc 16
Restoring window content 16
Scrolling 16
SelectForOutput 17
Sending messages 16
SetPattern 19
SetPen 18
SetPos 18
SetRestoreProc 16
Size of working area 14
StringField 20
Turtle graphics 18
Update event handler 16
UseEntryForm 20
User events 4
Window event class 16
closing 16
redefine 16
Window font 17
Window management 13
Window output 18
Window placement 14
Window size 14
WindowFrame 14
WindowHandlers 16
Windows 33
Windows 3.1 3
Working area 14, 15, 17