# Mirakon - Reference / Rev. 11

Structuring and securing know how
Optimization of manufacturing costs
Knowledge based product design
Planning and control of business processes

# 1 Introduction

## 1.1 What is the Mirakon-system

The Mirakon-System is a Software kit that enables you to assemble business-specific applications in a fast, flexible and integrated manner.

Without having to be a software developer you can build and link knowledge based applications, such as cost estimation of new products, automatic configuration of products or production planning, without redundancies and with interfaces to other systems.

The Mirakon system is particularly suitable for complex applications that require much technical specialized knowledge. You can store and structure company specific know-how in a knowledge base in a way that it can be used in different situations and applications.

Heterogeneous and dynamic data structures can be easily handled. Complex products such as machines or fabrication plants can be modelled in different structures (functional structure, constructional structure, production process, cost structure...) and be stored as only one data record in the Mirakon data base system.

One speciality of the Mirakon-System is the early cost estimation for technical designers and product planners (earlier called the HKB program), equally the Mirakon Sytem is frequently used as configuration system for complex customer dependent products, production planning/simulation and FMEA systems. However, you can also develop simpler applications such as customer administration, financial plan, stock management, etc.

## 1.2 Your advantages with Mirakon

**Flexibility**
Standard software packages for pre-defined areas of application, even if parameterized, offer many functions you do not need, and misses some which are important for you. They are based on standard data structures and algorithms. Result: You must adapt to the software when it should be the reverse! Mirakon however offers to you a language with which you can build your own structures, dialogues, algorithms and output documents. You control your applications and are independent of us (and that´s the way it should be).

**Integration**
Many heterogeneous applications with all possible interfaces provoke untransparent situations and high costs. A new version of application X appears and the interfaces with other applications stop working! The applications in Mirakon are embedded in a common data base and the operating surface requires no interfaces. However it allows output interfaces (ODBC, ASCII and others).

**The treatment of complex structures**
Relational data base systems are unsuitable, when it comes to model complex products or production processes. The data of an object is distributed on many tables and relations and it generates complicated data structures. Mirakon offers, from start, a new approach: Data trees growing freely instead of rigid pre-defined tables.

**The knowledge-based definition**
It is better to store a method (knowledge) that is able to compute the drilling time, depending on the tool, material and drilling parameters, than to store this time (data) for thousands of pieces. If a new type of drilling tool is purchased,

all this data becomes obsolete. The calculation method, however, can be extended within few minutes, and all the data can be recalculated in seconds fast.
Mirakon links data with its generator knowledge in such a way, that this data can be justified and updated at any time.

## 1.3 System Architecture

The Mirakon system consists of 4 components:

1. **The Mirakon program**
   Our Mirakon program and reference (files MIRAKON.EXE and REFE.PIB);
   The Mirakon program makes it possible to you, as an author, to store knowledge and build applications, as well as to structure and manage data. On the other hand, for you as user, Mirakon starts and runs applications interpreting its comands and allowing interactive dialogs.

2. **Your knowledge base**
   The knowledge base (KNO-file) contains knowledge and applications in the form of tables, functions, procedures, documents, dialogues and data structures.
   It is developed in a modular way and allows the plugging of knowledge modules from different authors.

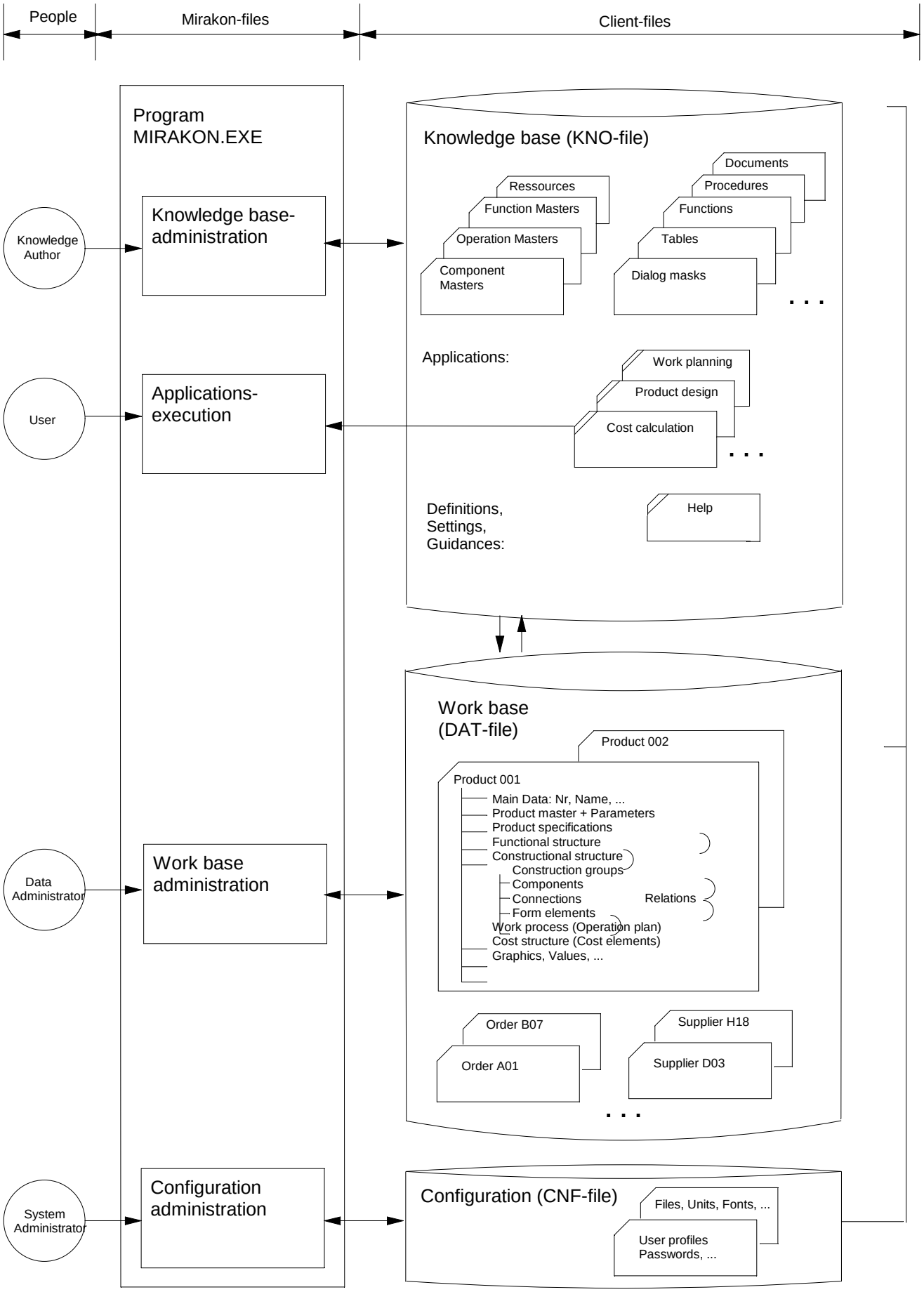
3. **Your work base**
   The work base (DAT-files) contains the data which resulted from the work of all users: Products, customers, schedules, orders, production orders, suppliers, etc.

4. **Your configuration**
   The configuration (CNF file) defines your project and its options and links together all the files involved (knowledge bases and work bases).

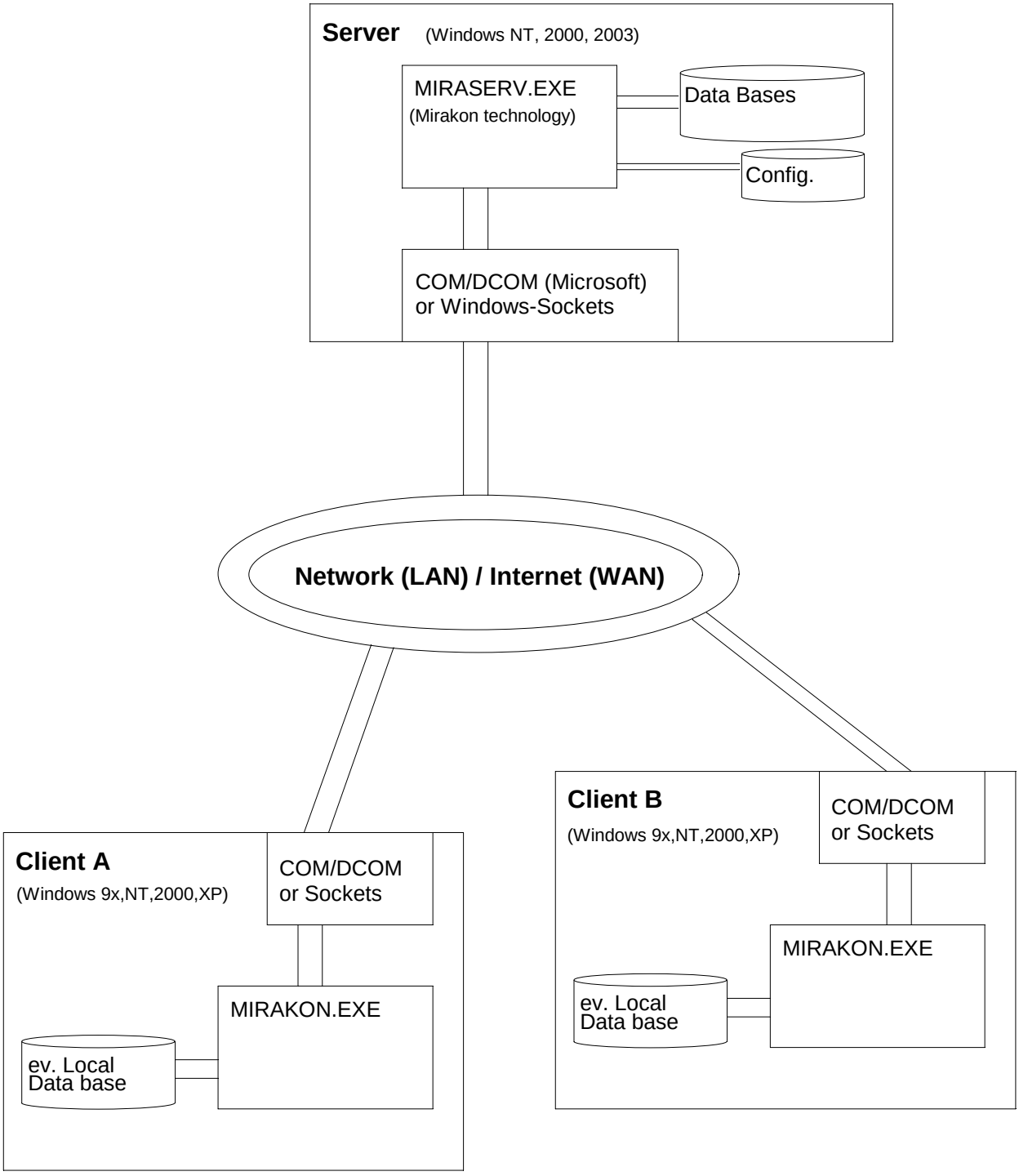
Two kinds of users must be differentiated:

- The **authors** (you, your coworkers or your advisor) develop applications, define data structures and model existing knowledge in accordance with your specifications and the needs of the users. Authors must not necessarily be at the same time administrators.
- The **users** (your coworkers and yourself) run the applications and process the data of your company (calculations, planing, configurations, offers, documentation, etc).

**Architecture of MIRAKON-System**



## Architecture of MIRAKON-System in Client/Server-Version:



## 1.4 Installation

Requirements:

- Operating system: any Windows after NT4 (2000, XP, Vista, 7, ...);
- During installation of client/server version you should log in with administrator rights.

**Software protection**
We protect our software with a USB-Stick containing a small client specific LIC-file (license file) which is unique and works only together with that USB-Device.

Following configurations are possible:

- single place installation (for single user)
- standard network installation (for few users and simple applications)
- client/server installation (for many concurrent users and complex applications)

### 1.4.1 Single place installation

Installation procedure:

**1.** Create a new folder (eg: c:\programs\Mirakon) and copy the Mirakon-files into it.

### 1.4.2 Client/Server installation

The server program, MIRASERV.EXE, runs continuously in the server machine.
The client program, MIRAKON.EXE, is stored in the server, but runs in the CPU of the Client PC.

The configuration file (xxx.CNF) contains the server's path (server ID or IP address) and is called by the client, which then know's where the server is.
Data bases are stored in the server, and exclusively acceeded by the server program.

Two possible technologies can be used for the communication between client and server processes:
- the COM/DCOM technology from Microsoft (not longer supported by Microsoft)
- and the Windows-Sockets method (easier to implement)

Installation in Server:

**1.** Create a new folder (eg: c:\programs\Mirakon) and copy the Mirakon-files into it.
Be sure you only have one CNF-file in this folder.

**2.** Set the Configuration options:
- Run MIRAKON.EXE using a double click (the CNF file will be read).
- Choose: Configuration -> Edit, and set the architecture as follows:
  - architecture = Client/Server
  - server ID = server name or server IP address
  - Connect using = Windows Sockets or COM/DCOM
  - only for Sockets connection:
    port = port number (e.g.: 7575) port number should be configured with your network administrator and no standard TCP/IP Port number should be used (use numbers larger than 3000)
- Save the configuration (use F2 or the Save-Button)
- Close the Mirakon program

**3A.** Start MIRASERV.EXE with double click. This programm is to remain always running.
If you chose COM/DCOM: registration of COM-object in Windows Registry is made automatically.
To connect over the Internet, call MIRASERV with the parameter SRS=1 (eg: miraserv.exe SRS=1)

**3B.**

Installation in each work station:

**1.** Create a shortcut in the Desktop (see chapter program start).
Example: F:\mirakon\MIRAKON.EXE F:\mirakon\client.cnf
For connection through the the Internet, you must indicate the RSID and PORT parameters.
Example1: MIRAKON.EXE RSID=192.190.20.14 PORT=7575
Example2: MIRAKON.EXE RSID=MYSERVER PORT=7575

**2.** only for COM/DCOM connection: start and terminate MIRAKON.EXE once, using the created shortcut (registration of COM-object in Windows is made automatically)

## 1.5 Program start and parameters

The Mirakon system can be started in different manners:
- double clicking MIRAKON.EXE: the program starts and looks for the nearest configuration file;
- creating a shortcut on your Desktop calling MIRAKON.EXE and indicating the path of the configuration file as parameter (eg: c:\folder1\mirakon.exe f:\folder2\myconf.cnf)

In a Client/Server installation via Internet the call to MIRAKON.EXE
must be followed with the indication of the remote server IP address,
and without configuration path: Example: c:\mir\mirakon.exe RSID=192.168.22.12
The following program parameters and options can also be indicated:

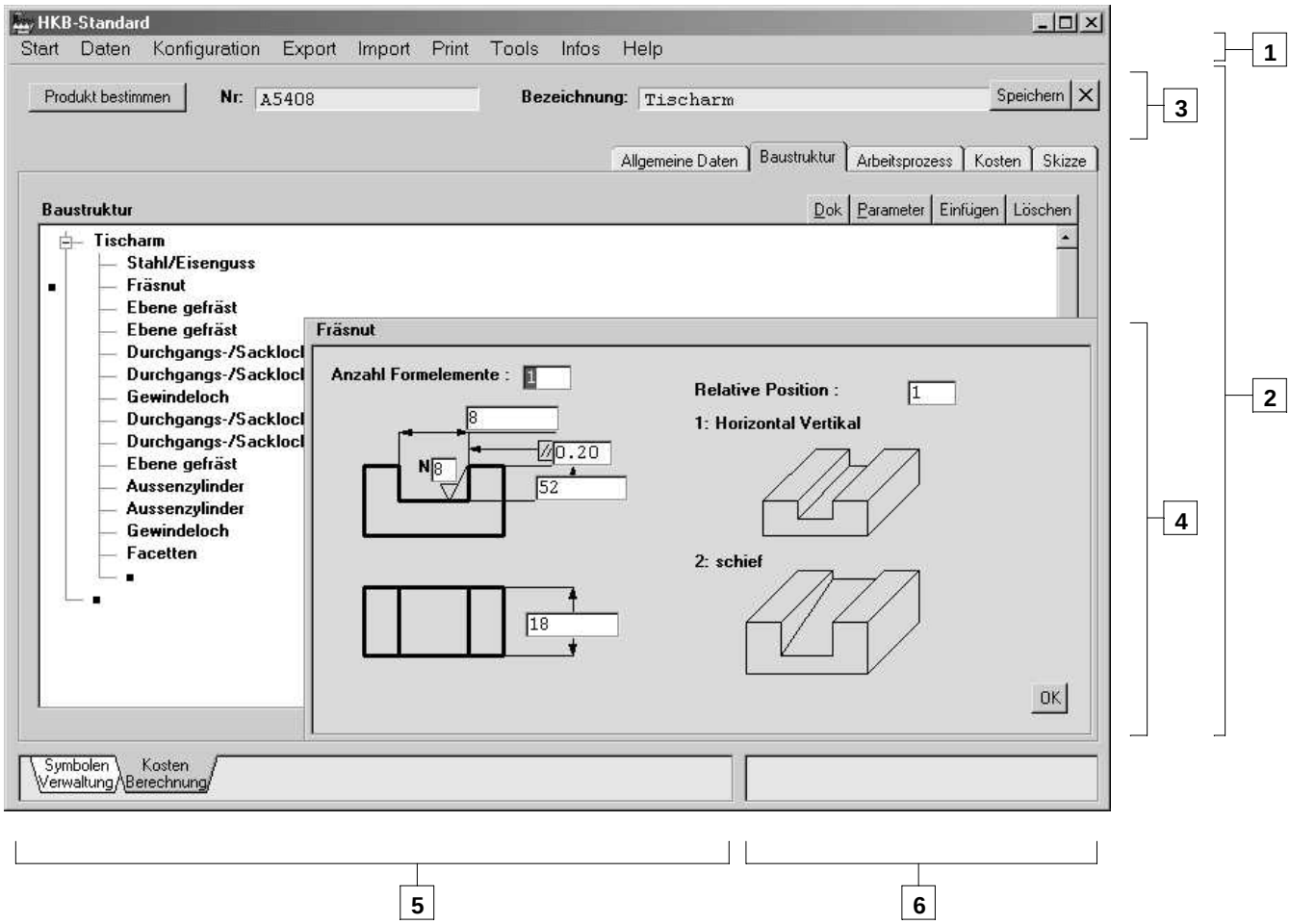
- option /NOLOG - jumps over the user registration (login);
- option /NI - jumps over the execution of the initialization code;
- option /WINMIN - starts Mirakon with minimized window;
- APP=[Application ID ] - starts the indicated application automatically ;
- P1, p2, p3 =... - passes up to 3 parameter to the starting application;
- UID=[user ID] - fills in automatically the first log in field
- UPW=[user password] - fills in automatically the second log in field
- RSID=[IP address] - provides the address for the remote server
- SRS=1 starts server as a remote server (only for Client/Server via Internet)
- PORT=[TCP/IP-Port Number] - indicates the port number for socket connection through Internet

Examples of program calls:
- c:\programs\mirakon\mirakon.exe myconf.cnf APP=HKB P1=120 P2=Peter /NOLOG
- \\server\mirakon\mirakon.exe \\server\mirakon\myconf

## 1.6 The user interface

### 1.6.1 Desktop

Down you see a typical application situation:




The Mirakon windows is divided into the following areas:

**Menu bar [1]:**
It contains general functions, such as Print, Export, etc.

**Application frame [2]:**
Total work surface that is available to an application. Contains one or more dialogue windows. By clicking the button-X (right above, near "Save") the application is closed.

**Status area [3]:**
Upper work surface in an Application frame, which contains application-specific information and buttons, which are always accessible.

**Dialogue mask [4]:**
Dialogue window with its respective dialog fields. These dialogue masks can be arranged by you. With escape Key or OK button the window will close. With TAB key or with Enter key the cursor jumps to the next field, the SHIFT-TAB key jumps the cursor to the previous field.

**Frame register field [5]:**
This field, at the bottom left hand corner, shows all Frames opened at the moment. By clicking the register with the mouse, the user can jump between opened documents or applications.

**Message field [6]:**
This field, down right, shows various messages during an application.

## 1.6.2 Menu bar

**Menu options:**

| | |
|---|---|
| **Start** | Enables to start applications. |
| **Data** | Allows to edit knowledge bases and work bases. |
| **Configuration** | Allows to edit the configuration; |
| **Export** | Allows to export the selected item into the internal clipboard or into an external file. Several items can be exported into the clipboard. |
| **Import** | Allows to import items from the internal clipboard or from an external file. |
| **Print** | Allows to print the selected object. Can also be called with F5. |
| **Tools** | Allows to execute comand lines, debug actions, read external documents, execute external procedures, etc.. |
| **Infos** | Informs the user about implementation data, memory consumption, momentary values of variables and color numbers |
| **Help** | Shows several documents (manuals, references, instructions, etc). Can also be called with F1. |
| **Log Off** | Terminates session and allows new log in with another user. |

## 1.6.3 Function keys

The following keys and/or combinations of keys can accelerate your work:

| | |
|---|---|
| **F1** | Help |
| **F2** | saves the changed objects |
| **F5** | prints the selected item |
| **F11** | starts and/or terminates the debugger |
| **F12** | shows the internal data structure of a structure element |
| **Ctrl-S** | saves the changed objects (same as F2) |
| **Ctrl-P** | prints the selected item (same as F5) |
| **Ctrl-C** | copies the marked text into the Windows clipboard |
| **Ctrl-V** | pastes the item from the Windows clipboard in the current cursor position |
| **Ctrl-Ins** | exports the selected item into the Mirakon clipboard |
| **Shift-Ins** | calls the Mirakon clipboard, to paste items in the current position |

# 2 The configuration of the system

The configuration file (xxx.CNF) contains settings, options and parameters concerning the users and ressources of one implementation.

## 2.1 Title

Determines the title (text and picture) and the user (company) of an implementation.

Title: Text that appear in the program window title.

Application: 4 lines that appear in the welcome window.

User: 3 lines that appear in the welcome window.

Image: Bitmap file (xxx.BMP) that appears in the welcome window.

Title mask: Dialogue mask file (xxx.MSK) that appears in the welcome window.

## 2.2 Project revision

Indication of the project revision number and date (optional).
These data appears into the welcome window of Mirakon.

## 2.3 Language

It sets the system language for all standard texts.
Following languages are supported: English, German, French and Portuguese

## 2.4 Architecture

It determines one of 2 versions: Standard or Client/Server.

For the Client/Server version the server-ID or -IP must be indicated.

To increase the speed, the data Packets can be compressed before they are sent over the network, by enabling the option "Compress Client\Server packets".

## 2.5 Databases

List of the data bases of an installation.

The first data base must be the knowledge base (Id=KNO).
The second data base must be the work base (Id=DAT).
Further data bases can be freely added (archives, test data files, etc..).

The identificator allows the access on these data bases (with LOAD, SAVE, etc.).

The paths can be absolute (c:\mir\myproj.kno) or relative (myproj.kno).
A configuration with relative paths can be easily transported from one path to another.

Only for the Client/Server version: for each data base the access mode must be indicated: Direct or Trough server.

The option "Secure file to CNF" allows data bases to be secured by a password so that they can only be opened with its CNF or by knowing the password.
Thus a stolen data base, without the password, cannot be used.

## 2.6 Start Menu

The start menu is modelled as a hierarchical structure of applications.

Into this structure you can insert groups, dividing lines and application calls.

An application call consists of:

- ID: identification of the application as it is in the knowledge base.

- Name: designation of the Application as it should appear in the start menu.

- Image: Bitmap that appears before the designation in the start menu. You must indicate the Picture-ID from the resources in the module A000 of the knowledge base.

- Register: 2 lines in the lower application register.

- Parameters: declarations of variables for a specific call. They are set before the init procedure and can be queried during the initialization of the application with the Prefix APARS. Example: if APARS.demo: initdemo1;

- Data bases selectors: define which Knowledge base is to be used (default is KNO) and which work base are the objects stored (default is DAT). Access to the defined data bases is made without the data base identification, e.g.: load(PROD,'001',prd), but the access to other data bases, if any, must be made using the data base identification as prefix,e.g.: load(ARC.PROD,'O001',prd).

## 2.7 User profiles

In order to describe users efficiently and without redundancies, you can define user profiles and assign them to several users later.

With the Insert button you can add and identify a new user profile.
Below the profile you can insert permissions and prohibitions, and thus describe the profile (see next chapter).

A permission determines which applications the user can start and with which access restrictions. The Read Only option allows user to see and change objects without saving them.

A prohibition determines which applications a user cannot start.

Filters can be used to describe several applications, e.g.: PROD,G_,B7

## 2.8 Users

List of the users that can login.
With the Insert button you can add individual users or groups.

The following data define a user:

- ID: user identification (1.Login-Question);

- Password: User password (2.Login-Question).
  For safety reasons it is not visible and is doubly queried.

- Name: User name as it should appear in documents;

- Status: there are 3 user Status with the following meanings:

A=user: has neither access to the data nor to the configuration;
V=Author/Manager: has restricted access to the configuration;
S=System administrator: can access everything;

- Code: Tekla instructions executed immediately after the Login.
  Usage examples: to welcome, check the mailbox, show messages,etc.

- Parameters: User specific variables that can be accessed during any application execution with the prefix UP. Example: if up.abt='E7': ...;

Under the user, you can insert user profiles, permissions and prohibitions.

## 2.9 Fonts

This chapter lists all Windows fonts to be used in Mirakon.
Each font gets a number that can be used when programming a document.

The first 3 fonts are already defined:
- Font 1 = Courier new: used for the input fields;
- Font 2 = Ms Sans Serif: used for the output of standard texts
- Font 3 = Arial: assures for transportability of certain documents.

In the configuration you can define 50 fonts.
These fonts apply to all applications into this configuration.

To keep fonts numbers stable, you can only delete the last font, and insert new fonts only at the end.

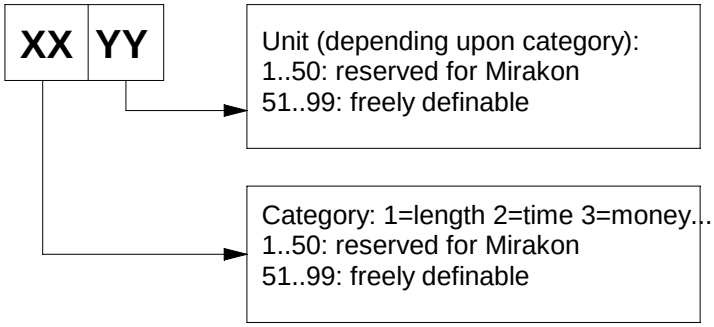
The font name must be indicated accurately as it is written in Windows.
As comment you can, for example, indicate the purpose of usage (optional).

## 2.10 Units

The units table lists all needed units (time, length, money, ...).

| NR | NAME | ABREV | VAL1 |
|---|---|---|---|
| 100 | Length | | |
| 101 | Meter | m | 1 |
| 102 | Millimeter | mm | 0.001 |
| 103 | Centimeter | cm | 0.01 |
| 151 | Inch | inch | 0.0254 |
| 200 | Time | | |
| 201 | Minute | min | 1 |
| 300 | Money | | |
| 301 | Swiss Francs | CHF | 1 |
| 302 | US-Dollar | US$ | 1.38 |
| 304 | Euro | EUR | 1.53 |

Structure of the units numbers:




The column ABREV contains the abbreviation of the unit as it should be written in expression, e.g.: d1:q=80 mm;

The column VAL1 contains the numerical value of the unit in reference to the reference unit (unit number 1). In this way, quantities can be later easily converted.

## 2.11 Calendar

Here you can describe firm-specific holiday calendars,
to be assigned to individual resources during production planning.

## 2.12 Events

Under this chapter you can define jobs for the server to execute automatically,
in given time intervals. Example: Automatic Backups;

## 2.13 Tekla Editor

Following settings are available:

- whether syntax check is always performed when leaving procedure editor (recommended);

- color of system variables (examples: ENR, PI, ENAME)

- color of comments (after /)

- color of certain user variables that are indicated in the following field;

## 2.14 Help Menu

The Help Menu is modelled as an hierarchical structure of documents. Thus, you can write several company-specific documents, and make them available to your users.

Into this structure you can insert groups, dividing lines and document calls.

Help document is a file that can be provided in the following formats:

- Mirakon format (PIB File): viewed by Mirakon Viewer, allows to search, and to print in different formats.

- HTML Compiled Microsoft format (CHM File): showed by Microsoft Viewer;

- Word format (DOC-file): showed by Microsoft Word;

- Power Point format (PPT-file): showed by Microsoft Power Point;

Each entry of a document in the Help Menu needs:

- Name: Designation of the document as it should appear on the Help Menu.

- Image: Bitmap that appears before the designation on the Help menu.
  Use the Picture-ID from the resources in module A000 of the knowledge base.

- File format: PIB, CHM, DOC or PPT;

- File: File path + file name;

## 2.15 Options

Following options can be set:

- Event Protocol: when active, Mirakon writes following events to the file MIRAKON.LOG:
  Login, Logout, Application start, Save and deletion of objects.
  These events (maximum 5000) can be seen selecting the Menu Infos/event protocol.

- Enter jumps to the next field: If this option is active, Mirakon behaves as follows:
  after a Enter in a dialog field with assigned Tekla instructions, the cursor jumps to the next field, except if the author forbids it expressly with the instruction jumpfok:=0;
  If this option is not activated, the author must instruct the jump with JUMPF-comand.
  Dialog fields without Tekla comands are not affected by this option; for these, after Enter, the cursor always jumps to the next field.

- Path for saving objects and error report: here you define the path where the error reports are to be stored, as well as the objects saved during an error.

- Advanced error reporting: when active, Mirakon builds an extended error report containing internal commands and sequences, so that we can better find errors.
  Using this option if you can give us detailed information about an error, allowing us to correct it faster.

# 3 Data and data bases

## 3.1 Data organization

Elementary information, like the address of a customer, must be stored into a file, so that it survives, after the computer has been switched off.

In order to discover this information among million of others, these files must be internaly organized, exactly the same as books in a library.

The Mirakon system organizes the data file in 4 stages:
- data element (field, variable)
- data object (data record, record)
- data module (collection, application)
- data base (file)

Data elements are elementary information units (address of a customer) that must be stored in an object.

Objects (also called records) are numbered and stored into a data base module. Depending upon contents and purpose of the object, we differentiate between:

- work objects are the ones that are changed and processed by the user during an application. e.g.: Products, orders, customer, etc..

- knowledge objects contain knowledge and do not change during an application, but they are thereby used. e.g.: Material tables, Component Masters, procedures, etc..

A data base module is a system of related objects, arranged hierarchically in chapters. It offers mechanisms for the numbering and relationing of its objects.

Depending upon contents and purpose we differentiate between 3 categories of modules:

- collection: contains work objects of the same kind (customers, articles, orders, etc..);

- Application: contains konowledge objects that define one application:
  Procedures, dialogue masks, tables, diagrams, etc.;

- Knowledge module: contains knowledge objects used by different applications like:
  material tables, documents, calculation methods (procedures), pictures, etc.
  The grouping of knowledge objects into a module can follow different criteria, such as:
  - objects who are thematically related, e.g.: Hydraulics
  - objects written by the same author, e.g.: Purchase processes

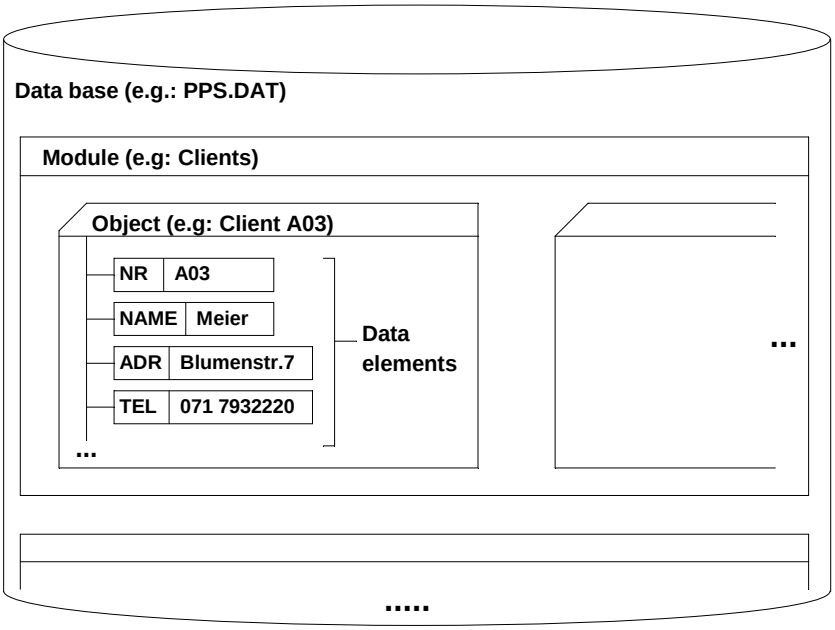
A data base module must be defined and stored in a data base.

A data base is a file. Depending upon contents and purpose we differentiate between 3 kinds of Mirakon data bases:

- work bases (DAT files): contain mostly collections of work objects;

- knowledge bases (KNO files): for knowledge objects (tables, procedures...)

- configuration (CNF file): Data base with a pre-defined structure, which contains information about the implementation (project).




## 3.2 Knowledge base

**Base module A000**
The standard knowledge base (KNO) must contain a basic module A000.
This module, contains central data and procedures, which can be accessed by every other module, without the indication of the prefix A000.

The INIT procedure contains all global variables and definitions, which should be constantly present in the system. These are initialized automatically when the first application is called.

## 3.3 Applications

An application must contain an INIT procedure. This procedure defines all global variables and definitions, which must be constantly present during the application.

The INIT procedure is executed after the start of the application. The INIT procedure of an application X is also executed, if another application Y accesses objects of X. This permits the declaration of variables that eventually are necessary. In this case the INIT procedure of X, must protect the dialogue code with AT START: preventing the start of application X in the middle of application Y.

Example :
```
av1:r; / general-variable
av2:s; / general-variable

AT START: / dialogue started
openstat(STAT);
opentabs(0,0,0,'',MAIN);
```
The variables with the save attribute are supervised by the Save-system. If they are changed during the course of an application, they will be saved by pressing the Save button. The application author can enter an optional procedure name as the second parameter of the save attribute. In that case that procedure will always be executed before the internal save procedure.

Example : `pr:l&save(PROD, check_inputs);  /product`

## 3.4 Collections

**Transactions**
In the course of a project, new needs and developments arise. As a consequence of it, changes in the data structure of the already existing objects in the work base are needed. Such changes can be easily accomplished with transactions. A transaction is implemented by describing a procedure for each object. The system loads each object into the internal variable O:list, afterwards the transaction instructions are executed (those that change the object O) and in the end the object O is saved.
Thus, values can be placed and/or changed, new data elements can be added or old ones deleted or renamed, and relations changed.

## 3.5 Tables + Editor

The table editor is a Frame, in which tables are edited.

A table consists of lines and columns. A line consists of a line ID/Number and several cells. A cell consists of one or several text lines.

A table is edited in two levels:

1. Navigation level for moving from cell to cell, using the cursor keys or mouse.

2. Input level to write within a cell.

Function buttons and keys in the navigation level:

- Edit button, Enter key, letter or number: It enters in the input level. The selected cell is opened to input data.

- + button or + key: If a column or a line is selected, the column width or line height increases by one character or line.

- - button or - key: If a column or a line is selected, the column width or line height decreases by one character or line.

- Insert button or INSERT key: If a column or a line is selected, a new column or line is added right from or above the cursor.
  If a cell is selected, it is zoomed and opened to input data (same as Zoom button).

- Delete button or DELETE key: The selected cell, column or line is deleted.
  If several lines are marked they are all deleted.

- Zoom button: If a column or a line is selected, it is made smaller or bigger, depending upon the actual state. If a cell is selected it is zoomed and opened to input data (same as the Insert button).

- Legend button: shows a text editor where you can write comments for the table.

- Doc button: allows you to write or draw a document for the selected cell.

- SHIFT + down-key: If a line is selected, it is marked or unmarked, depending on the actual state. Marked lines appear with inverted colors.

- Import and Export functions (Mirakon menu): The selected cells, and the selected or

marked lines from the table can be transferred to another table. If the cursor is in the top-left corner (NR field) the whole table will be transferred.

- With Mouse Right-click, other functions/shortcuts are available.
  For example, the option Format Cell, allows to format the selected cells, defining the background color, font style, etc...

Function buttons and keys in the input level:

- Enter key or Esc key: Cell is left and the editor gets back into the navigation level.

- Ctrl+Cursorkeys, Tab-key or SHIFT-Tab key: leaves the actual cell and enters the adjacent cell.

## 3.6 Grafics + Editor

The grafic editor is used to develop grafics and dialogue masks.

With the page selector the desired page can be selected in multi paged documents.

With the mouse, you can select one or more grafic elements:
- by simple Click (left mouse button) on individual grafic elements;
- or by draging a square around several grafic elements.
When a selected grafic element is clicked, it is deselected.

The Delete-key deletes all selected elements.

The buttons [+] and [-] increase and/or decrease the grafic zoom by 20%.

The Insert button offers the following functions:

- **Line** or L-key: for drawing a straight line or a Polyline.
  1. Select the beginning and ending points, with the mouse or arrow keys, by moving the cross and by short clicking the left mouse button or pressing the enter key. This step is repeated until the desired line path is drawn.
  2. Leave this function with the right mouse button or with the Esc key.

- **Text** or T-key: for writing a text of several lines.
  1. Write the desired text in the text editor, terminating with the Esc-key.
  2. Position the text box into the desired position with the mouse or arrow keys, and click the left mouse button or the enter key to terminate.

- **Arc**: it draws an elliptical arc with selectable ray and angle.
  1. input X and Y ray (mm), start and End angle (degree) and press enter or OK.
  2. Position the arc with the mouse or arrow keys.

- **Surface**: fills a line path with a color and/or pattern.
  1. Define the corners of the polygon, which will be filled, positioning the cross and pressing enter or left mouse button (as in the LINE function).
  2. Leave the function with the right mouse button or with the Esc key.

- **Box**: draws and fills a rectangle with a color and/or pattern.
  1. Define the upper left corner of the desired rectangle, positioning the cross and pressing the left mouse button or enter.
  2. Define the lower right corner of the desired rectangle, positioning the cross and pressing the left mouse button or enter.

- **Symbol**: draws a symbol from the symbol catalog.

1. Select the desired catalog.
2. Select the desired symbol.
3. Put symbol into position.

- **Dialog field**: defines and places dialogue field into the dialogue mask
  1. Enter the dialogue field data and press ok button to confirm.
  2. Position the dialogue field with the mouse or arrow keys.

The Settings button offers the following functions:

- **Zoom** or Z-key: zooms a given rectangle. The zoom rectangle is entered as in the box function.

- **View all** or 0-key: it shows the entire grafic.

- **Original view** or 1-key: it shows the grafic in its originally formatted size.

- **Text attributes**: edits text attributes for the texts that will be inserted.

- **Grafic attributes**: edits attributes for the next grafic elements to be inserted.

- **Layout**: for changing size, color and position of the dialogue mask, as well as the margins for printing.

- **Grid**: sets the distance between the orienting points that appear in the grafic window.

- **Resolution** or R-key controls the positioning accuracy (snap).

- **Initialization**: contains instructions, that are executed before the mask opening. e.g.: Variable declarations, menu lists, etc.

- **Opening**: contains instructions, that are executed after the mask fields are drawn. e.g.: to change field status, etc.

- **Finalization**: contains instructions, that are execute before the mask is closed. e.g.: to check coherence, warnings, etc. Usually the system variables EXITOK, OKAYED, ESCAPED are used here.

- **Layers**: sets the active layers (the ones that are visible)

With the right mouse button, grafic elements can be selected and changed.

The following functions are available:

- **Delete**: it deletes all selected elements;

- **Parameters**: edits the contents of a text or a dialog field;

- **Attributes**: edits the appearance of the selected elements;

- **Move**: for moving the selected grafic elements; these can be moved to a new position with the mouse or arrow keys, and be fixed there with the left mouse button or the Enter key.

- **Scale**: increases or decreases the size of the selected elements, by entering

the X-Y scaling factors and press the OK button.

- **Rotate**: rotates the selected elements around a desired angle, by entering the angle and press the ok button; Elements rotate counterclockwise.

- **Copy**: for copying the selected elements.
  1. Position with the mouse or arrow keys, and copy with the left mouse button.
  2. Leave the function with the right mouse button or with the Esc key.

- **Group**: joins several elements into a group.

- **Separate**: separates the individual elements of a group.

- **Remodel/Change points**: edits the position of the points of lines or surfaces.
  1. Select the desired corner and position it again.
  2. Leave the function with the right mouse button or with the Esc key.

- **Remodel/Insert points**: allows to insert new poins in lines or surfaces.
  1. Select the position for the new point and press the left mouse button.
  2. Leave the function with the right mouse button or with the Esc key.

- **Remodel/Delete points**: allows to delete poins of lines or surfaces.
  1. Select the point to delete and press the left mouse button.
  2. Leave the function with the right mouse button or with the Esc key.

- **Mirrow**: mirrors (copies) selected elements by entering the angle of the symetry axe.

- **Order**: changes the order in which graphic elements are drawn.
  - bring to back: the selected element is the last to be drawn.
  - bring to front: the selected element is the first to be drawn.

- **Layer**: it sets the layer in which the element is drawn (0=all layers).

- **Identify**: assigns an ID to the selected element (to use later with SETF,etc)

During the positioning, it is possible to enter the position numerically by pressing the X-key.

Grafic files can be imported with the Import/from file function.
The following formats are supported:

- bitmap pictures: BMP files;
- designs: DXF files;
- compressed pictures: JPG files;
- icons: ICO files;

Procedure:
1. From the Mirakon menu: select Import/from file;
2. Select the file;
3. Position the picture frame with the mouse;
4. Press left mouse button;

Bitmaps from other applications can be imported with the function Import/from Windows clipboard.

Procedure:
1. In another application, copy the Bitmap into the Windows clipboard,

normally with CTRL+C or CTRL+Ins;
2. Activate Mirakon window and open the grafic editor;
3. From the Mirakon menu: Select Import/from Windows clipboard;
4. Position the picture frame with the mouse;
5. Press left mouse button;

## 3.7 Dialog masks

### 3.7.1 Dialog fields

Each dialog field contains:

- ID: Identificator (alphanumeric). This determines the order of the field selection when the user uses the dialogue mask.

- variable ID: Name of the variable that is shown and changed by the field.

- Field dimensions: in Milimeter or text units.

- options: Selectability, colors, etc..

- Commands: to be executed when deselecting of the field: after Enter, UP, Down, tab, shift-Tab or Mouse; With the variable FCHANGED (integer), it can be queried whether the field variable was changed. After the execution of the commands the field will be left by the cursor, except if the system variable JUMPFOK is set to 0;

- Tip: Text that appears when the mouse cursor stops for a second over the field.

### 3.7.2 Question

A question is a dialog field, that allows the input of a text line.

Arrow keys: Left, Right, Home or End, move the cursor inside the field.

DEL-key: deletes the character under the cursor.

BACKSPACE-key: deletes the character left from the cursor.

Shift-F8: The entire dialog field content id deleted.

Mirakon controls each input and will not accept invalid characters.

The cursor jumps between several questions with the cursor keys UP, DOWN, TAB, SHIFT SHIFT-TAB or with ENTER.

### 3.7.3 Menu

A menu is a dialog field, that allows the selection of an option from a hierarchical list of options.

The functioning is similar to the Windows menus.

ENTER key: selects option. If an option contains suboptions (down arrow appears right from option), these are opened and/or closed (zoom).

HOME key: Sets the cursor to the beginning of the menu.

END key: Sets the cursor to the end of the menu.

### 3.7.4 Button

The button is a dialog field, that allows to execute a set of commands.

Selectable buttons (e.g. OK button) must be selected and confirmed, in order to let the commands be executed. Confirmation is made by clicking the left mouse button or pressing the Enter key.

### 3.7.5 Selector

A selector is a dialog field that is used as a switch between few options. Options are selected by clicking the edge buttons or pressing the Left/Right cursor keys.

### 3.7.6 Textdisplay

A display shows a value (numeric or text) and does not allow inputs.

### 3.7.7 Graficdisplay

A display shows a picture and does not allow inputs.

### 3.7.8 Lister

The Lister field allows editing a list of heterogeneous elements.

Functioning mode:

INSERT-key: inserts an element in the list from an associated menu.

ENTER-key: allows the input of parameters for the selected element.

DEL-key: deletes the selected element from the list.

### 3.7.9 Text editor

A text editor allows the input of texts of several lines.

Functioning mode of the keys:

Enter: advances the cursor to a new line.

Backspace: joins 2 text lines if the cursor stands at the begining of the line.

CTRL-Y: deletes the line in which the cursor is located.

Shift-F6: marks/unmarks the line in which the cursor is located.

Shift-F7: inserts an empty line above the cursor.

Shift-F8: deletes the line in which the cursor is located. (same as CTRL Y)

Shift-F9: copies the marked lines to where the cursor stands.

### 3.7.10 Structure editor

The structure editor is a dialogue window, in which data structures are edited.

A structure consists of a hierarchy of elements.
An element is selected from a master table, parameterized and inserted into the structure. Each element can contain sub-elements.

Functioning mode:

- INSERT-key or Insert button: inserts a choden element into the structure.

- ENTER-key or Double click: closes or opens the hierarchic level of the selected element.

- Plus-key (+): opens the hierarchic level of the selected element.

- Minus-key (-): closes the hierarchic level of the selected element.

- Star-key (*): opens all hierarchic levels under the selected element.

- P-key, or Parameter button or Right-Cursor-Key: allows the input of parameters for the selected element.

- DEL-key or Delete button: it deletes the selected element.

- Right mouse button: opens a context menu if the selected element has one assigned.

### 3.7.11 Grid

### 3.7.12 Table

### 3.7.13 Checkbox

### 3.7.14 Combobox

A menu is a dialog field, that allows the selection of an option from a hierarchical list of options.

The functioning is similar to the Windows menus.

ENTER key: selects option. If an option contains suboptions (down arrow appears right from option), these are opened and/or closed (zoom).

HOME key: Sets the cursor to the beginning of the menu.

END key: Sets the cursor to the end of the menu.

### 3.7.15 Radio buttons

### 3.7.16

# 4 The Tekla language

## 4.1 Language elements

The TEKLA language consists of following elements or terms:

**Variable:** Is a logical storage location and possesses a name, a type and a value. In order to allocate storage place for a variable, it must be declared.

**Type:** During the program execution values are entered, computed and stored in variables. These values can be of different natures (type): Numbers, texts, etc.. The variable type specifies, which values can be stored and which operations can be applied to it.

**Expression:** A value expression is a formula, consisting of operands (constants and variables), operators and function calls that return a value. The use of several parentheses as in algebra is allowed.

**Command:** A command is an instruction used to declare variables, calculate, change or transport values, etc.

**Function:** A function is a value generator. It is not a command but a part of an expression. It consists however internally of commands that calculate the return value.

**Object:** An object is a connected list of identified values that can be stored in a data base as a whole (record).

The first three values are always:
- number (NR:string)
- designation (NAME:string)
- status (STAT:recstatus)

An object grows and shrinks with its use and must not have a pre-defined data structure.

Depending upon the purpose, we can differentiate between:

- work objects: are changed and worked by the user during an application. E.G.: Products, orders, customers, etc..

- knowledge objects: contain information that does not change during an application. E.G.: material tables, component masters, procedures, etc..

**Table:** An object, which contains a quantity of expressions, instructions and comments, that are arranged in matrix form. A table cell can contain several text lines.

**Dialog mask:** An object describing a window with dialogue fields and graphics. When called, a dialogue takes place with the user.

**Data base:** A file that contains a quantity of related objects. An Data base is organized into modules for fast data access.
Depending upon purpose of the objects contained in it we differentiate between:
- knowledge bases: for knowledge objects (tables, procedures...)
- work bases: for work objects (clients, products, orders...)

**Module:** A quantity of related objects inside one data base.
We differentiate between:

- knowledge module: contains knowledge objects hierarchically organized that are writen most by one author;

- application: it describes an application and contains: initialization, main menu, procedures, functions and dialogue masks;

- collection: contains work objects of the same kind, ordered by their numbers;

**Configuration:** File with information about the project implementation: title, user, paths of needed data bases, options, etc.

## 4.2 Types

There are elementary types and compound types.

The elementary types cannot be further partitioned.

The compound types can be divided into fields of elementary types.

**Elementary types**

| | |
|---|---|
| **b** | Byte = integer numbers between 0 and 255. |
| **i** | Integer = integer numbers between -32000 and 32000. |
| **a** | Address = 32-bit number. |
| **c** | Currency = real number with exact roundness to 4 decimal places. Important for financial accounting, because real numbers are not always exactly rounded. |
| **r** | Real = real numbers between -1E18 and +1E18. |
| **s** | String = string with a maximum of 255 characters. |
| **n** | Name = continuous character string that identifies one data element (variable, table, structure element). |
| **d** | Date = indication of time with year, month, day, hour, minutes and seconds. |
| **q** | Quantity = real number (value) + unit (name). |
| **l** | List = list with variables of different types. |
| **li** | List of integer = list with integer numbers. |
| **lr** | List of reals = list with real values. |
| **ls** | List of string = list with character strings (text). |
| **ln** | List of name = list with names. |
| **^...** | Pointer to... (examples: pos1:^e; pr2:^r; pl3:^l;) |

**Compound types**

The type ELEMENT is the basic component for building data structures and therefore is here mentioned.

Further compound types are described later with its respective topics.

| | |
|---|---|
| **e** | Element; consists of the following fields:<br>- ID:s; (Module\Table.Line Number).<br>- NAME:s; (designation).<br>- PARS:l; (parameters list). |

- DATA:l; (further attributes: Quantity, resources...).
- SS:l; (substructure).

## 4.3 Variables

There are system variables defined by Mirakon and those that can be freely defined by the application author.

Variables declared with the same name as the system variables overrun the system variables. Therefore it is not recommended.
Relative to their life span the variables can be global or local.

Global variables are defined in the initialization of the modules (init procedure) and live as long as the application.

Local variables are defined in procedures, dialogue masks and tables and live only as long as the procedure, dialogue or table is active.

## 4.4 Situations

Situations are names with fixed meaning. These give the possibility to the Tekla author to implement certain instructions only on certain situations.

Example: Actions that are assigned to a product-master (A-column), which are not to be always implemented. With the AT instruction its possible to write situation specific instructions.

## 4.5 Operators

Arithmetic operators (result in real or integer values):

**+** addition

**-** subtraction

***** multiplication

***/*** division

Boolean operators (result is 0 or 1):

**=** Equality

**#** Inequality

**in** Containment (needs blank space before and afterwards!)
Example: if ID in ('A_','B_'):inf('found');

**out** Exclusion (needs blank before and afterwards!)
Example: if NR out (1,4,12):inf('ok');

**<** Smaller

**>** Bigger

**<=** Smaller or equal

**>=** Bigger or equal

**not** Denial

**and** AND connection
Example: if (A 20) and (b<350):inf('OK');

**or** OR connection

String operators (return string values):

**+** Addition
Example: title:s='Chapter '+nr; /nr is also a string

***** Multiplication
Example: line:s=75 *'-';

**~** Negation: for negative filters:
Example: loader(...,gf='~A _ '); / Filters all objects that do not begin with A

Name operators (the return value is a name):

**[ ]** Evaluation operator
Example: path:n=pr.bs.[i].ss.[id1];
/i:integer and id1:string

Pointer operators (the return value is in the type of the pointed variable):

**^** De-references the pointer and returns the value of the pointed variable
Example: x1:r:=pr2^; (pr2 is a pointer of the type ^R)

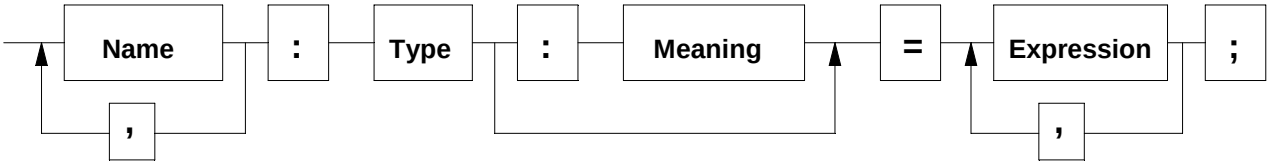
Decision operator (for all types)

**{ }** Returns the value preceeded by the first fulfilled condition.
Syntax: {condition1:value1, condition2:Value2, other value};
Example: x:r={length>10:2.8, width>200:3.5, 5.7};

## 4.6 Commands

### 4.6.1 Variable declaration

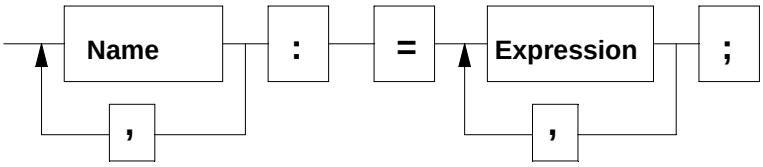
The variable declaration allocates storage place for a variable.




```
Example :  x:r:temperatur in degrees;
           x,y:r=2; / results in x=2 y=2
           x,y:r=2,5;  / results in x=2 y=5
           l1:li=(1,2,3);
```

### 4.6.2 Value assignment

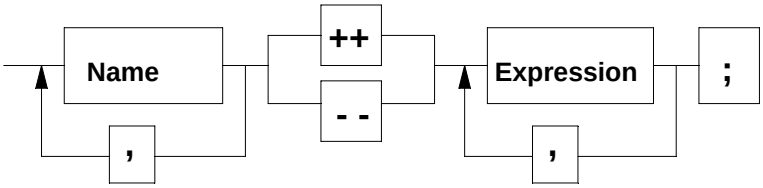
The value assignment fills the storage location of a variable with a value.




Example :
```
s1:='ABC';
x,y:=6,x/2; /results in x=6 y=3
```

### 4.6.3 Reduction and/or increase of a value

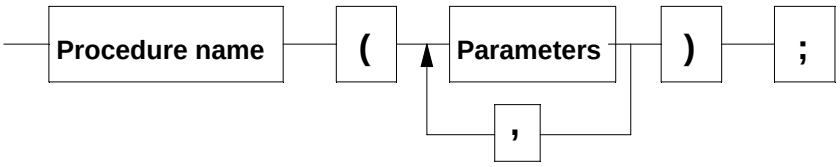
Adds and/or subtracts a value (number or string) to a variable.




Example :
```
z--2.5;
x,y++a;
x,y++a,b;
s1++'ABC';
```

### 4.6.4 Procedure call

The procedure call transfers the parameters to the called procedure and executes the commands contained there.




There are 2 kinds of parameters transfer:

1) value parameter: the procedure expects a value for internal calculations and decisions.

2) variable parameter: the procedure expects a variable to return a computed value. This value is thus exported. These parameters must be defined with the prefix VAR in the procedure header.

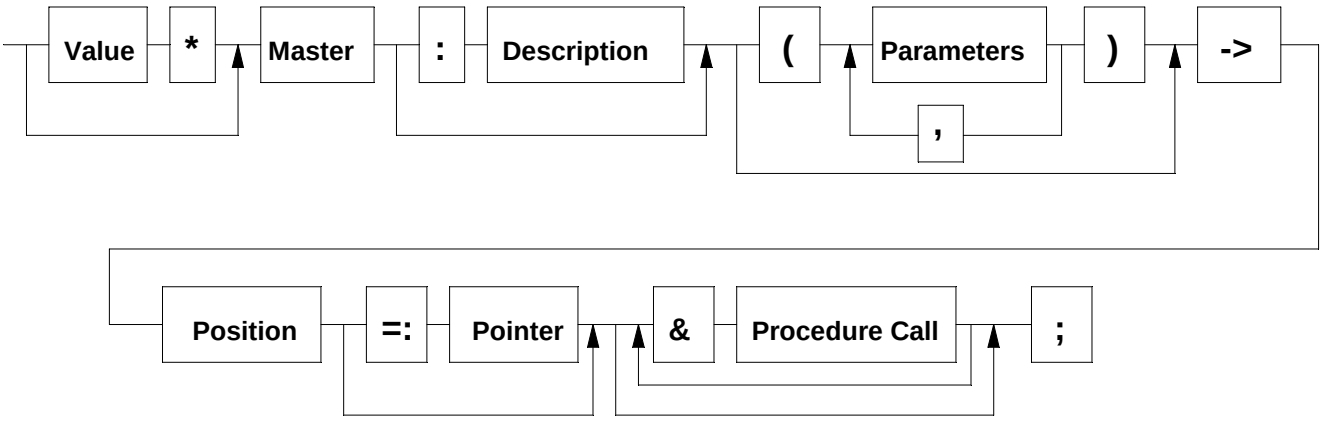
TEKLA offers also the possibility to define optional parameters. All optional parameters must be defined with a / after all fixed parameters. In the procedure call, optional parameters are passed indicating parameter name and = sign before its value.

Example :
```
/ Procedure declaration:

procedure sum(a,b:r; var result:r);
  /...
  end;

procedure calcd(d:d; /var year,mon,day:i);
  /...
  end;
```

```
/ Procedure call:

x,y:r=35,0;
sum(x,20,y);

d1:d=now;
m,d:i;
calcd(d1,mon=m,day=t);
```

### 4.6.5 Build command

The build command inserts an element into a structure.




The inserted element is always an instance (particularization) of a master from a masters table in the knowledge base.

The &-operator, at the end of the build command, enables calling related procedures

- &id=string : assigns an identifier to the inserted element.

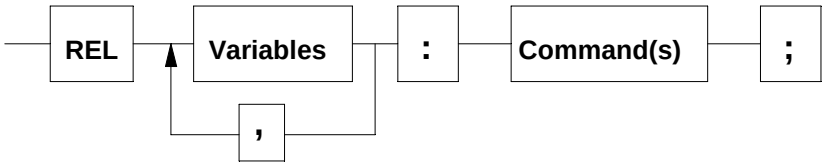
- &nss : inserted element wont have a substructure

Example : `TAB1.A1->pr.bs.1;  /simplest construction instruction`

```
p1:^e;
2*A2(10,'A')->ss.0=:p1;
/p1 now points to the inserted element

MOD5\TEILE.A3:'Left shaft'(p.d,p.l+2)->p1^.ss.0;
/Shaft will be inserted in the substructure pointed by p1

BK.TEIL1->pr.bs.0&id='X1';
```

### 4.6.6 Relation building

The relation building command builds a relation between 2 or more data elements/variables.
The relation consists of commands to be executed when any of the related elements changes.
The related elements are indicated with @1, @2..., in the "variables" part of the command.




Example :
```
rel p.n,pr.lg:@1:=@2;

rel p1,p2,p3:
  begin
  @1.x:=(@2.x+@3.x)/2;
  @1.y:=(@2.y+@3.y)/2;
  end;
```

### 4.6.7 Data structure definition

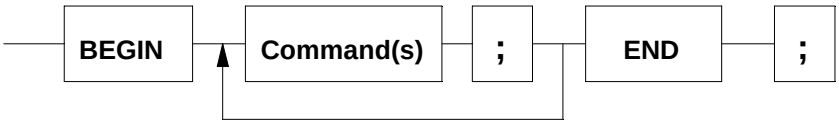
A data structure can be defined with the = sign:

Example :
```
book=
  author:s;
  year:i;
  publisher:s;
  end;

/afterwards a structured variable can be defined

mybook:book;
```

## 4.7 Control structures

### 4.7.1 BEGIN-END command block

With BEGIN... END you can assign a group of commands to a condition or a repetition loop.




Example :
```
if a>b:
  begin
  i:=i+1;
  inf('ok');
  end;
```

### 4.7.2 IF-structure

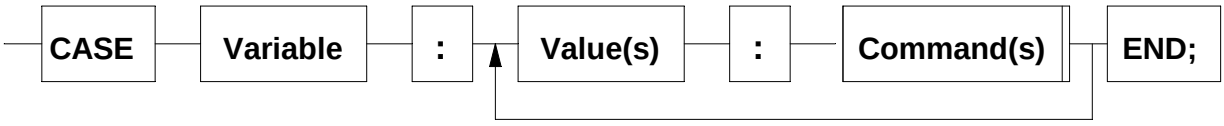
Allows conditioned execution of commands.


Example :
```
if a>b:
```

```
  begin
  a++1;
  inf('ok');
  end;
else inf('error');
```

### 4.7.3 CASE-structure

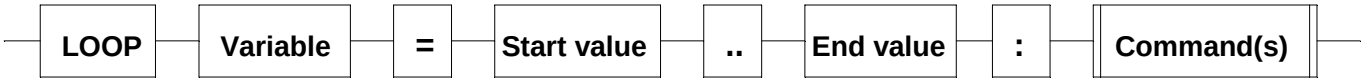
Allows conditioned execution of commands depending upon several value sets of a variable. This variable can be of the type Integer, Currency, Real, Quantity, Date, String or Name.




Example :
```
t:s='A12';
case t:
  'A_': inf('ok');
  'B_,C_': inf('false');
  end;
```

### 4.7.4 LOOP-structure

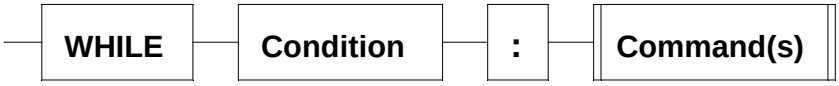
Repeats a command block several times.




Example :
```
t:s; n:i=10;
loop i=1..n:
  begin
  t:=si(i)+'. Line';
  drawr(2,-10-5*i,1,t,2);
  end;
```

### 4.7.5 WHILE-structure

Repeats a command block until a condition is fulfilled.

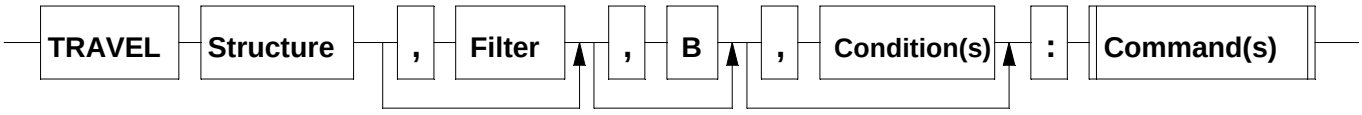



Example :
```
while y>100:
  begin
  drawr(2,y,1,r,2);
  y:=y-5;
  end;
```

### 4.7.6 TRAVEL-structure

Travels in a structure (from top to bottom) and executes the indicated commands for each structure element. In order to allow you to access the data of each element, TRAVEL activates each position before executing your commands. Thus the system variables D, P, SS, ENR, EMOD, ENAME, EP, EH, PE refer allways to the active element.

Optionally you may write:
- an element filter (string) as a second parameter
- options (characters) as a third parameter: B=backwards;
- conditions as a fourth parameter;



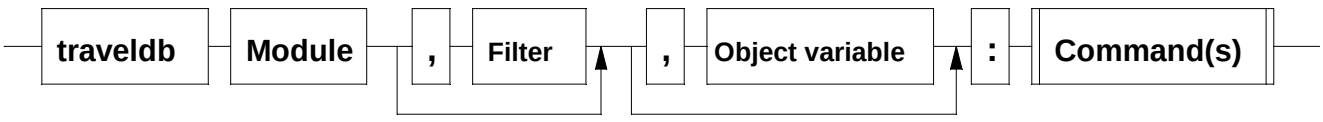

Example :
```
travel pr.bs:
  begin
  writes(1,2,1,ep);
  writeq(0,14,1,d.q);
  writes(0,20,1,ename);
  end;

travel pr.ap,'D_,B_',B: writes(1,2,1,ename);

travel pr.ap,'D_,B_',,p.d>120: writes(1,2,1,ename);
```

### 4.7.7 TRAVELDB-structure

Travels through a work base module in alphabetical order and executes the indicated commands for each object number. The respective object number is contained in the system variable KEYI:string. This variable KEYI is only valid within the TRAVELDB commands.

Optionally an object number filter can be indicated as the second parameter (string) and an object variable (list) as the third parameter.

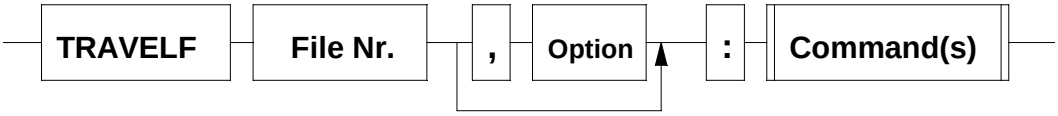
If this object variable is indicated, each object is loaded into it.




Example 1:
```
obj:l;
traveldb PROD:
  begin
  load(PROD,keyi,obj);
  writes(1,1,1,keyi+' '+obj.name);
  end;
```
Example 2:
```
obj:l;
traveldb PROD,'_',obj:
  writes(1,1,1,obj.name);
```

### 4.7.8 TRAVELF-structure

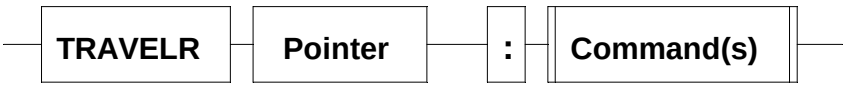
Travels through a file byte for byte and executes the indicated commands for each section. The sections are formed by the indication of a separator in the variable TFS.
Options: L = separator is carriage return line feed.




Example :
```
f:i;
openf(f,'ABC.TXT');
travelf f,L: writes(1,1,1,tfs);
closef(f);
```

### 4.7.9 TRAVELR-structure

Travels through all relations of the element pointed by the pointer variable and executes the indicated commands for each relation.
In order to access each individual relation, the system variables RTYP, RREF, PRE1, PRE2 and RDATA are updated at each position.




Example :
```
p1:^e=^pr.bs.1;
travelr p1:
  begin
  if rtyp=3:continue;
  rdata.ncy:=4;
  end;
```

### 4.7.10 EXIT-Command

Forces the active procedure to be left.

### 4.7.11 CONTINUE-Command

Breaks the actual round in the active LOOP/WHILE/TRAVEL loop and starts the next.
Example :
```
travel pr.bs:
  begin
  if eh>2:continue;
  writes(1,20,1,ename);
  end;
```

### 4.7.12 BREAK-Command

Leaves the active LOOP/WHILE/TRAVEL.
Example :
```
travel pr.bs:
  begin
  if ei>10:break;
  writes(1,20,1,ename);
  end;
```

# 5 The Mirakon library

The Mirakon library contains variables, types, procedures and functions
that Mirakon already programmed and are ready to be called from your applications.

It contains also schemes for the solution of more complex problems.

A scheme is a system of functionally related language elements.

Example: The scheme for the modeling of structures is the EDITST and consists
of the variables ENR, PE, ENAME, etc., the situations I1, I2, etc., the handler
procedure, the master table for the structure elements and the appropriate
dialogue masks.

## 5.1

### procedure getram(var total,avail:a);

total :

avail :

Example :

## 5.2 Program control in general

#### Variables and constants

| | | |
|---|---|---|
| **uid** | n | user id |
| **uname** | s | user name |
| **us** | n | user status |
| **up** | l | user parameters list |
| **errormode** | i | If 1: Calculation errors are not shown. |
| **mirrev** | s | Mirakon-Revision, e.g.: '10.128'; |
| **lang** | i | Defined language (in the configuration):<br>1=English 2=German 3=French 4=Portuguese |

#### Situations for program control

| | |
|---|---|
| **START** | After the start of an application |

### function EX (v:n):r;

Returns 1 if the variable V exists and 0 if it does not exist.

Example : `if ex(pr.status)=0: inf('Status not found');`

### function TYPOF (v:n):s;

Returns the type of the variable V.

Example : `x:r;  t:s=typof(x);  /returns t='R'`

### function SIZEOF (v:n):a;

Returns the memory consumption (in RAM) of the indicated variable V.

Example : `x:r;  t:a=sizeof(x);`

**procedure SETBIT (bitnr,bitval:i; v:n);**

Sets a bit value in a variable.

bitnr : bit position: 1 to 16

bitval : bit value: 0 or 1

v : variable to change; Must be of the Integer type

Example : `a:i=0; setbit(3,1,a); /returns a=4`

**function BIT (bitnr:i; v:n):i;**

Returns one bit value of a variable.

bitnr : bit position: 1 to 16

v : variable

Example : `a:i=4; b:i=bit(3,a); /returns b=1`

**procedure CLOSEPROG;**

Terminates the Mirakon program.

**procedure CLOSEFRAME;**

Terminates the current application and/or Frame.

**procedure WAIT (n:i);**

Waits N hundredth of seconds.

Example : `wait(200); /waits 2 Seconds`

**procedure EXECCODE (var code:ls; /o:n);**

Implements the instructions contained in the variable CODE.

code : List of instructions

o : Options: N: Errors are not shown

**procedure EXECCOM (com:s; /o:n);**

Implements the instructiosn indicated in the variable COM.

com : Instruction

o : Options: N: Errors are not shown

Example : `s1:s='beep;'; execcom(s1);`

**procedure EXECPROG (cmd:s; /o:n; params:s);**

Executes the indicated Windows program (EXE file).
In this way, external programs can be called from inside of Mirakon.
If a document file is indicated (e.g.: DOC file), Mirakon calls
the assigned Windows program to that file (e.g.: Word).

cmd : Program or document file.

o : Options (characters):
- M = program window appears maximized;
- W = Mirakon waits until the program is closed before continuing the execution of the next Tekla commands.

params : Program call parameters

Example : `execprog('C:\win\notepad.exe',params='c:myfile.txt');`

**procedure EXECSHELL (cmd:s);**

Executes the indicated Windows instruction (similar to the windows run option).

cmd : windows instruction

Example : `execshell('net send XYZ message from Meier');`

**procedure DEBUGGER (op:i);**

Starts the debugger with (op=1) and terminates it with (op=2)
showing the executed instructions and the values of used variables.

Example :
```
debugger(1);
calculate1;
debugger(2);
```

**function WINUID:s;**

Returns the memory consumption (in RAM) of the indicated variable V.

Example : `x:r;  t:a=sizeof(x);`

## 5.2.1 Search and find errors

### 5.2.1.1 The Debugger

With the integrated debugger internal operational sequences can be analyzed in detail. This helps enormously with the search for errors and shows as programmed commands work. The debugger is a tool that functions like an execution protocol. It shows all implemented instructions as well as the value of all the variables during their execution.

Procedure:
1. Run the application until the moment before the error will appear;
2. Press the F11-key: The debugger starts;
3. Let the error appear;
4. During the error message: Press the button "Show protocole"
   or (if no error message appears) press F11-key again:
   the Debugger shows detailed execution sequence;

When the program crashes, the debugger should be activated from the Tools menu with the option "builds DEBUG.TXT" switched on. After the crash, the file DEBUG.TXT contains the execution structure and can be seen with any text editor

### 5.2.1.2 See data structures

In the structure editor you can watch the data structure of the selected element with F12.

With the main menu option information/system variables you can see the actual value of all system variables.

With the main menu option information/application variables you can see the actual value of all variables defined in the current application.

With the main menu option information/knowledge base variables you can see the actual value of all the variable defined in the base module (A000).

#### 5.2.1.3 Implementing instructions manually

With the main menu option Tools/Comands you can, during an application, write and execute Tekla-commands. This is useful both to query data (with INFV instruction) as well as to change variables.

### 5.2.2

#### procedure STARTAT (d0:d; p:n; /nextd:n);

Executes the indicated Windows program (EXE file).
In this way, external programs can be called from inside of Mirakon.
If a document file is indicated (e.g.: DOC file), Mirakon calls the assigned Windows program to that file (e.g.: Word).

d0 : Program or document file.

p : Options (characters):
- M = program window appears maximized;
- W = Mirakon waits until the program is closed before continuing the execution of the next Tekla commands.

nextd : Program call parameters

Example : `execprog('C:\win\notepad.exe',params='c:myfile.txt');`

#### procedure INSATODO (id:s; proc:n; hs:r);

Executes the indicated Windows program (EXE file).
In this way, external programs can be called from inside of Mirakon.
If a document file is indicated (e.g.: DOC file), Mirakon calls the assigned Windows program to that file (e.g.: Word).

id : Program or document file.

proc : Options (characters):
- M = program window appears maximized;
- W = Mirakon waits until the program is closed before continuing the execution of the next Tekla commands.

hs : Program call parameters

Example : `execprog('C:\win\notepad.exe',params='c:myfile.txt');`

## 5.3 Numbers and quantities

### 5.3.1 Algebra

#### Variables and constants

| | | |
|---|---|---|
| **E** | r | 2.71828 (constant value) |
| **PI** | r | 3.14159 (constant value) |

#### function ABS (x:r):r;

Returns the absolute value of X.

Example : `x:r=-4.5; a:r=abs(x); /returns a=4.5`

**function ARCTAN (x:r):r;**

Returns the arctangent of X in degrees.

Example :
```
x:r=1;  a:r=arctan(x);  /returns  a=45 (degrees)
```

**function COS (x:r):r;**

Returns the Cosinus of the indicated angle X in degrees.

Example :
```
x:r=60;  a:r=cos(x); /returns a=0.5
```

**function EXP (x:r):r;**

Returns E raised to X.

Example :
```
x:r=3;  y:r=exp(x);  /returns  y=20.09
```

**function FRAC (x:r):r;**

Returns the non integer part of X.

Example :
```
x:r=-24.75;  a:r=frac(x);  /returns a=-0.75
```

**function INTHI (x:r):r;**

Returns the rounded up integer portion of X.

Example :
```
x:r=24.25;  a:r=inthi(x);  /returns a=25
```

**function INTLO (x:r):r;**

Returns the rounded down integer portion of X.

Example :
```
x:r=24.75;  a:r=intlo(x);  /returns a=24
```

**function LG (x:r):r;**

Returns the decimal logarithm of x (basis 10).

Example :
```
x:r=100;  y:r=lg(x);  /returns y=2
```

**function LN (x:r)r;**

Returns the natural logarithm of x (basis e).

Example :
```
x:r=100;  y:r=ln(x);  /returns y=4.61
```

**function RND (x:r):r;**

Returns the rounded value of X.

Example :
```
r1:r=rnd(24.75);  /returns r1=25
r2:r=rnd(24.50);  /returns r2=25
r3:r=rnd(24.45);  /returns r3=24
```

**function ROUND (val:r; /dec,step:i):r;**

Extended rounding function.

val : Value to round

dec : Number of decimal places to be rounded;

step : Rounding step;

Example :
```
x:r=24.762;
y:r=round(x,dec=2,step=5);  /returns y=24.75
```

**function SIN (x:r):r;**

Returns the sine of X (in degrees).

Example : `x:r=30; y:r=sin(x); /returns y=0.5`

**function SQRT (x:r):r;**

Returns the square root of X.

Example : `x:r=9; y:r=sqrt(x); /returns y=3`

**function TAN (x:r):r;**

Returns the tangent of x (in degrees).

Example : `x:r=45; y:r=tan(x); /returns y=1`

**function PWR (a,b:r):r;**

Returns A raised to B.

Example : `a:r=2; b:r=3; y:r=pwr(a,b); /returns y=8`

### 5.3.2 Numbers order

**function RMAX (l:lr):r;**

Returns the biggest number from the number list L.

Example : `zl:lr=(1,5,-8,3.5); y:r=rmax(zl); /returns y=5`

**function RMIN (l:lr):r;**

Returns the smallest number from the list L.

Example : `zl:lr=(1,5,-8,3.5); y:r=rmin(zl); /returns y=-8`

**function NORMHI (r0:r; l:lr):r;**

Returns the largest number from the list L, which is smaller than R0.

Example :
```
zl:lr=(6,8,12,20,30,50);
d:r=normhi(28,zl); / returns d=20
```

**function NORMLO (r0:r; l:lr):r;**

Returns the smallest number from the list L that is bigger than R0.

Example :
```
zl:lr=(6,8,12,20,30,50);
d:r=normlo(28,zl); / results d=30
```

### 5.3.3 Quantities

**function QQ (q:q; u:s):r;**

Returns the quantity Q converted to the unit U. The unit U must be written as the abreviation in the units table in the configuration.

Example : `q1:q=2 m; x:r=qq(q1,'mm'); /returns x=2000`

**function RQ (q:q):r;**

Returns the value (without unit) of the quantity Q.

Example : `q1:q=1.4 kg; w:r=rq(q1); /returns w=1.4`

**function QR (v:r; u:s):q;**

Returns a quantity with value V and unit U. The unit U must be written as the abreviation in the units table in the configuration.

Example : `x:r=20; y:q=qr(x,'mm'); /returns y=20 mm`

**function UQ (q:q):i;**

Returns the unit of the quantity Q.

Example : `x:q=7.8 kg; u:i=uq(x); /returns u=401`

**function QABREV (q:q):s;**

Returns the unit of the quantity Q as abreviation from the units table.

Example : `x:q=7.8 kg; a:s=qabrev(x); /returns a='kg'`

### 5.3.4 Dimensions and tolerances

**function DIM (x:s):r;**

Returns the nominal dimension of X.

x : Measure with or without tolerances

Example :
```
x1:s='8+-0.1'; m1:r=dim(x1); /returns m1=8
x2:s='120H7'; m2:r=dim(x2); /returns m2=120
```

**function ISO (d:s):r;**

Returns the tolerance in ISO-quality number of the dimension X.

Example :
```
x:s='80h6'; t:r=iso(x); /returns t=6
x:s='80+-0.02'; t:r=iso(x); /returns t=8
```

## 5.4 Strings

### 5.4.1 Handling strings

**Variables and constants**

**non** n Empty name (= NO name)

**function SLEN (s:s):i;**

Returns the length the string S.

Example : `i1:i=slen('15.7'); /returns i1=4`

### function CHAR (n:i):s;

Returns the character that has the ANSI value N.

Example : `s1:s=char(65); /returns s1='A'`

### function UPC (s:s):s;

Returns the string S upcased.

Example : `s1:s=upc('abc123'); /returns s1='ABC123'`

### procedure DELS (var s:s; pos,nchars:i);

Delete characters from a string.

s : String variable to be changed
pos : Start position from which charcters will be deleted
nchars : number of characters to be deleted

Example : `s:s='ABCDEF'; dels(s,2,4); /returns s='AF'`

### function POSS (obj,source:s; n:i):i;

Returns the position of N-th appearance of OBJ in SOURCE.

Example : `i1:i=poss('A1','A1B7A1C3',2); /returns i1=5`

### function SUBS (s:s; p,n:i):s;

Deletes N-characters from S beginning in the position P.

Example : `s1:s=subs('ABC123',3,2); /returns s1='C1'`

### procedure INSS (obj:s; var s:s; pos:i);

Adds the string OBJ to the string S starting at the position POS.

Example : `s1:s='1234'; inss('AB',s1,3); /s1='12AB34'`

### procedure PUTS (var s:s; obj:s; pos,adj:i);

Builds the string OBJ in the string S in the position P.
ADJ determines the adjustment of OBJ: 1=left-justified 2=right-justified

Example :
```
s1:s='12345';
puts(s1,'AB',3,1); /s1='12AB5'
puts(s1,'XY',9,2); /s1='12AB5  XY'
```

### procedure CLEANS (var s:s; side:i);

Removes all blanks from S.
SIDE: 0=all; 1=left; 2=right; 3=left and right;

Example :
```
s1:s='  A B C ';
cleans(s1,3); /s1='A B C'
cleans(s1,0); /s1='ABC'
```

### function NS (s,sep:s):i;

Returns the number of string segments in S separated with SEP.

Example : `s:s='ABC/123/XYZ'; n:i=ns(s,'/'); /returns n=3`

### function SS (s,sep:s; n:i):s;

Returns the N-th substring from S separated with SEP.

Example :
```
s:s='ABC/123/XYZ'; t:s=ss(s,'/',2);  /returns t='123'
```

### procedure FRS (var s:s; old,new:s);

Replaces all occurrences of OLD by NEW in the string S.

Example :
```
t:s='AB12AB45';
frs(t,'AB','AB/');  /returns t='AB/12AB/45'
```

### procedure FRSL (var l:ls; old,new:s);

Replaces all occurrences of OLD by NEW in the text L.

Example :
```
frsl(txt,'AB','AB/');
/applies FRS to all lines of TXT
```

### function SV (form:i; var l:l; nr:i):s;

Returns the content of a list element as a string.

form : if 0 only the value is returned;
if 1 is returned the Identifier + Value;

l : List where the variable is;

nr : Position of the variable in the list L;

Example :
```
a:l;
a.temp:r=7.5;
t:s=sv(1,a,1);  /returns t='TEMP=7.5'
```

### function SX (s:s; l,adj:i):s;

Returns the string S plus a blank Sufix/Prefix so that the length is L.
ADJ=Adjustment: 1=left 2=right.

Example :
```
s1:s=sx('ABC',5,1);  /returns  s1='ABC  '
s2:s=sx('ABC',5,2);  /returns  s2='  ABC'
s3:s=sx('ABC',2,1);  /returns  s3='AB'
```

### function MATCHS (obj,source:s; prec:i):i;

Returns the position of N-th appearance of OBJ in SOURCE.

Example :
```
i1:i=poss('A1','A1B7A1C3',2);  /returns i1=5
```

## 5.4.2 Formatting strings

### function SF (/):s;

Returns a formatted string according to the entered parameters.
Each parameter consists of an instruction (1.character) and an operand
(remaining characters).

The following instructions are possible:

| | |
|---|---|
| **L + number N** | Limits the next output to N characters; |
| **V + name N** | Outputs the contents of the variable N. |
| **X + number N** | Moves the cursor to the position N (in number of characters) |
| **A + number N** | Sets the adjustment=N (1=left 2 =right) |

| | |
|---|---|
| **F + number N** | Sets the Format = N (number of decimals, date format, ...) |
| **S + string** | Outputs the string constant |
| **' + text + '** | Writes the text as it is between the apostrophes |
| **M + number N** | Moves the cursor the to position M (indicated in mm);<br>This positioning functions only with left justified text. |
| **CF + number N** | Sets the text color = N (default = 0) |
| **CB + number N** | Sets the background colour = N (default = -1).<br>If this color is set, the following letters are drawn on rectangles<br>with the background colour. With CB-1 this function is deactivated. |
| **CR + number N** | Sets the border color = N; (default is -1);<br>If this color is set, the following letters are drawn with border.<br>With CR-1 the border is deactivated. |

```
Example :  s1:s='ABCDEFGH';  s2:s='123456789';  r1:r=5.3;
           u:s=sf(L3,Vs1,X6,L5,Vs2);       /returns u='ABC  12345'
           v:s=sf('Value=',X15,A2,F2,Vr1); /returns v='Value= 5.30'
```

## 5.4.3 Conversion of types

### function IS (s:s):i;

Converts the string value S into an integer value.

```
Example :  s:s='15'; i:i=is(s);  /returns i=15
```

### function SI (i:i):s;

Converts the integer value I into a string.

```
Example :  i:i=15;  s:s=si(15);  /returns s='15'
```

### function RS (s:s):r;

Converts the string S into an real value.

```
Example :  s:s='15.7';  x:r=rs(s);  /returns x=15.7
```

### function SR (r:r; k:i):s;

Converts the real value R with K decimal places into a string.
If K=0 to 8: SR contains K decimal places
If K=9: SR contains as many decimal places as R has, however the maximum is 9
if K=10 to 15: SR contains K-10 decimal places and thousand separator.

```
Example :  x:r=12555.75;
           s1:s=sr(x,0);  /returns  s1='12556'
           s2:s=sr(x,3);  /returns  s2='12555.750'
           s3:s=sr(x,9);  /returns  s3='12555.75'
           s4:s=sr(x,12); /returns  s4='12'555.75'
```

### function RN (n:n):r;

Converts the name N into an real value.

```
Example :  n:n=15.7; x:r=rn(n);  /returns x=15.7
```

### function NR (r:r; k:i):n;

Converts the real value R with K decimal places into a name.

```
Example :  x:r=15.7; n1:n=nr(x,3);  /returns  n1=15.700
```

**function QS (s:s):q;**

Converts the string S into a quantity value.

Example : `t:s='15.7 mm'; x:q=qs(t); /returns x=15.7 mm`

**function SQ (q:q; k:i):s;**

Converts the quantity value Q with K decimal places into a string.

Example : `q1:q=5.128 m; t:s=sq(q1,2); /returns t='5.13 m'`

## 5.4.4 Coding information in strings

Sometimes it is necessary to compact information, so that it can be easily interpreted later .

Mirakon offers 2 syntax possibilities:

- Character codes (1) (e.g.: ' AB7C3 '):
  Charactercode = letter with a meaning and an associated value as number;
  Advantage: very compact;
  Disadvantage: limited (only numbers as values, only 26 characters)
  procedures: CHV, INSCH, DELCH;

- Characteristic-strings (2) (e.g.: ' A, B2=7, C7=aby '):
  Characteristic = Identificator + value as number or string;
  Characteristics are separated by commas; Values are indicated after "=";
  Advantage: very flexible (limited only by string length of 255 letters)
  Disadvantage: less compact (needs separators)
  procedures: INSC, DELC, SC, IC;

**procedure INSCH (ch:s; var s:s);**

Adds a charactercode CH into the string S.
A charactercode consists of a letter that can
followed by a number (however not obligator).

Example :
```
s1:s='ABC';
insch('R',s1);   /s1='ABCR'
insch('T4',s1);  /s1='ABCRT4'
```

**procedure DELCH (ch:s; var s:s);**

Deletes the charactercode CH from the string S. (see also INSCH)

Example :
```
s1:s='AB7C';
delch('B',s1);   /s1='AC'
```

**function CHV (ch,s:s):i;**

Returns the charactercode value of CH from the string S. (see also INSCH)

Example :
```
s1:s='AB7C';
i1:i=chv('A',s1);  /returns i1=0
i2:i=chv('B',s1);  /returns i2=7
i3:i=chv('F',s1);  /returns i3=-1
```

**procedure INSC (id:n; val:s; var s:s);**

Inserts or changes the characteristic ID with the value VAL in the string S.

Example :
```
s1:s;
insc(RL,'3',s1);   /returns s1='RL=3,'
insc(BK,'A',s1);   /returns s1='RL=3,BK=A,'
insc(RL,'5',s1);   /returns s1='RL=5,BK=A,'
```

**procedure DELC (id:n; var s:s);**

Deletes the characteristic ID from the string S.

Example :
```
s1:s='RL=3,BK=A';
delch('RL',s1);   /returns s1='BK=A,'
```

**function SC (id:n; s:s):s;**

Returns the string value from the characteristic ID of the string S.

Example :
```
s1:s='RL=3,BK=A,';  v:s=sc(RL,s1);   /returns v='3'
```

**function IC (id:n; s:s):i;**

Returns the integer value of the characteristic ID from the string S.

Example :
```
s1:s='RL=3,BK=A,';  v:i=ic(RL,s1);  /returns v=3
```

## 5.5 Dates and time

**function NOW :d;**

Returns the time at the moment (date and time).

Example :
```
d1:d=now;  /d1 contains the actual date and time
```

**function TODAY :s;**

Returns today's date as string.

Example :
```
s1:s=today;  /s1 contians today's Date as String
```

**procedure GETD (d:d; /var year,month,day,week,wday,hour,min,sec:i);**

Examines the date D and extracts the selected information
to the indicated variables.

| | |
|---|---|
| D : | Date to be examined |
| YEAR : | Variable that receive the year of D |
| MONTH : | Variable that receive the month of D |
| DAY : | Variable that receive the day of D |
| WEEK : | Variable that receive the week of D |
| WDAY : | Variable that receive the week day of D |
| HOUR : | Variable that receive the hour of D |
| MIN : | Variable that receive the minute of D |
| SEC : | Variable that receive the seconds of D |

Example :
```
ye,mo,day,h,w:i;
getd(now,year=ye,month=mo,day=day,hour=h);
if h<12:inf('Good Morning');
getd(now,wday=w);
```

```
if w=7:inf('Good Sunday');
```

**procedure MAKED (var d:d; year,month,day,hour,min,sec:i);**

Assigns year, month, day, hour, minute and second to date variable D.

Example : `d1:d; maked(d1,99,1,11,8,0,0); /d1=11.1.99/8:00`

**procedure MAKEDW (var d:d; year,week,wday,hour,min,sec:i);**

Builds date variable D indicating of year and week number,
weekday (1=Monday 2=Tuesday...), hour, minute and second.

Example : `d:d; makedw(d,3,3,2,8,0,0); /returns d=14.1.03/8:00`

**procedure INCD (var d:d; /years,months,days,hours,mins:i);**

Increases or decreases the variable D by the
number of years, months, days, hours or minutes.

Example : `d1:d=now; inc(d1,days=7);  /d1=today plus 7 days`

**function NDAYS (d1,d2:d):i;**

Returns the number of days between starting date D1 and final date D2.

Example :
```
d1,d1:d=30.12.02,4.1.03;
n:i=ndays(d1,d2); /returns n=5
```

**function NHOURS (d1,d2:d):r;**

Returns the number of hours between starting date D1 and final date D2.

Example :
```
d1,d2:d=1.1.03/08:00,2.1.03/09:30;
n:r=nhours(d1,d2);  /returns n=25.5
```

**function NMINS (d1,d2:d):r;**

Returns the number of minutes between starting date D1 and final date D2.

Example :
```
d1,d2:d=1.1.03/08:00,1.1.03/09:30;
n:r=nmins(d1,d2);  /returns n=90
```

**function RD (d:d):r;**

Converts the date D into an real value.
So that time diagrams can be easily computed.

Example : `drawbox(rd(d1),y1,rd(d2),y2);`

**function DR (r:r):d;**

Converts a real value R into a date value.

Example : `d1:d=dr(x);`

**function SD (d:d; f:i):s;**

Converts a date value into a Sring with the format F.

Example :
```
d:d=3.7.01/08:05;
s1:s=sd(d,1);  /returns s1='03.07.01'
s2:s=sd(d,2);  /returns s2='3.7.2001'
s3:s=sd(d,3);  /returns s3='3.7.01'
s4:s=sd(d,4);  /returns s4='03.07.2001/08:05'
```

```
s5:s=sd(d,5);  /returns s5='08:05'
s6:s=sd(d,6);  /returns s6='200107030805'
s7:s=sd(d,7);  /returns s7='010703'
s8:s=sd(d,8);  /returns s8='200127' (Year+Week number)
s9:s=sd(d,9);  /returns s9='20010703'
```

**function DS (s:s):d;**

Converts a string into a date.

Example : `s:s=today;  d1:d=ds(s);`

## 5.6 Lists

**procedure EMPTY (var l:l);**

Empties the contents of the list L.

Example : `empty(pr.ks);`

**procedure DEL (id:n);**

Deletes the list element ID;

Example : `del(pr.x);`

**function LLEN (l:l);**

Returns the number of list components of L.

Example : `loop i=1..llen(text): writes(1,1,1,text.[i]);`

**procedure INSL (id:s; v:n; var l:l; nr:i; /o:n);**

Inserts a variable (new list element) into a list.

| | |
|---|---|
| id : | identifier of the new list element |
| v : | Name of the variable that contains the value to be inserted |
| l : | Destination List |
| nr : | Destination position; if NR=0 V is inserted at the last position |
| o : | Options (letter):<br>- S = elements are sorted alphabetically according to ID<br>- N = if ID already exists in L, it is not inserted. |

Example : `x:r=2.5;  z:l;  insl('A',x,z,0,o=SN);`

**function IDOF (path:n);**

Returns the identifier of the list element in the position indicated by PATH.

Example : `if idof(l.[i])='D1':beep;`

**procedure SETID (var l:l; nr:i; id:s);**

Sets the identifier of NR-th element of L to ID.

```
Example : l:li=(2,4,7);  setid(l,2,'B');
          /now the 2nd element of the list
          /can be also be accessed with L.B
```

### procedure MODID (var l:l; id1,id2:s);

Replaces the identifier ID1 by ID2 in all elements of L that
have ID1 as identifier.

Example : `modid(pr.daten,'DAUSSEN','DA');`

### procedure COPYL (var l1,l2:l; nr:i; /o:n);

Copies the contents of the list L1 into the list L2.

| | |
|---|---|
| l1 : | Source |
| l2 : | Destiny list |
| nr : | Insert position in L2; if NR=0, L1 is attached at the end; |
| O : | Options (letter):<br>- M: only marked elements are copied |

Example : `copyl(pr1.bs,pr2.bs,0);`

## 5.7 Tables

#### Variables for table access

After POST, OPENT, PUTT the following variables are set and/or updated:

| | | |
|---|---|---|
| **trowid** | s | Identifier (NR) of active row |
| **tcolid** | s | Identifier of active column |
| **tc** | l | Content of active table cell (list of strings) |
| **tcels** | s | |
| **trows** | l | List of the tables rows |
| **trowp** | i | Position of active table row |
| **tcols** | i | List of table columns |
| **tcolp** | i | Position of active column |

### function CT (tabid,rowid,colid:name):s;

Returns the contents of a table cell like as it is written.
If one or two parameters are not indicated, the last parameters
are used. This saves time and code and applies to all table access
procedures like CT, IT, RT, ST, NT, DT and QT.

| | |
|---|---|
| tabid : | Table identifier |
| rowid : | Row id |
| colid : | Column id |

Example :
```
s1:s=ct(MIF1\BK,D07,NAME);
p1:r=rt(,,PREIS);
```

### function CTI (row,col:i):s;

Returns the contents of the cell of the active table as it is written.

| | |
|---|---|
| row : | Row position |
| col : | Column position |

Example : `s1:s=cti(2,4);`

**function IT (tabid,rowid,colid:name):i;**

Returns the content of a table cell converted to an INTEGER.
TABID, ROWID and COLID as in the function CT.

Example : `i1:i=it(MUSTER,A01,BED);`

**function ITI (row,col:i):s;**

Returns the contents of a cell from the active table converted to an INTEGER.
ROW, and COL as in the function CTI.

Example : `i1:i=iti(2,4);`

**function RT (tabid,rowid,colid:name):r;**

Returns the contents of a table cell and converted to an REAL.
TABID, ROWID and COLID as in the function CT.

Example : `r1:r=rt(MUSTER,A01,PREIS);`

**function RTI (row,col:i):s;**

Returns the contents of a cell from the active table and converted to an REAL.
ROW, and COL as in the function CTI.

Example : `r1:r=rti(2,4);`

**function ST (tabid,rowid,colid:name):s;**

Returns the contents of a table cell converted to an STRING.
TABID, ROWID and COLID as in the function CT.

Example : `s1:s=st(MUSTER,A01,CODE);`

**function NT (tabid,rowid,colid:name):name;**

Returns the contents of a table cell converted to a NAME.
TABID, ROWID and COLID as in the function CT.

Example : `n1:n=nt(MUSTER,A01,MASK);`

**function LT (tabid,rowid,colid:name):list;**

Returns the contents of a table cell and converted to a LIST.
TABID, ROWID and COLID as in the function CT.

Example : `l1:ls=lt(MUSTER,A01,FILTER);`

**function LTI (row,col:i):list;**

Returns the contents of a cell from the active table converted to a LIST.
ROW, and COL as in the function CTI.

Example : `l1:l=lti(2,4);`

**function DT (tabid,rowid,colid:name):date;**

Returns the contents of a table cell converted to a DATE.
TABID, ROWID and COLID as in the function CT.

Example : `d1:d=dt(MUSTER,A01,START);`

### function QT (tabid,rowid,colid:name):quantity;

Returns the contents of a table cell converted to a QUANTITY.
TABID, ROWID and COLID as in the function CT.

Example : `q1:r=qt(MUSTER,A01,PREIS);`

### function LTAB (tabid:n; rowids:ln; colid,bedid:n; /o:n):ls;

Returns a list of strings from the column COLID of the table TABID.
Only the rows whose number (in NR column) is contained in ROWIDS are taken.
If BEDID is indicated, the condition in the BEDID column is examined.

| | |
|---|---|
| tabid : | Table Identifier |
| rowids : | Line filter (list of line numbers) |
| colid : | Column name; in order to list several columns, the desired column names can be indicated, separated by \|. |
| bedid : | Name of the condition column |
| o : | Options (letter):<br>- T = sequence of LTAB corresponds to table sequence instead of sequence of ROWIDS |

Example :
```
menu1:ls=ltab(MAT,(S_,G_),NAME);
menu2:ls=ltab(MAT,(S_,G_),NAME|PRICE,BED,o=T);
```

### procedure FINDR (tabid,colid:n; rowids:ln; var rowid:s);

Searches for the row in table TABID, whose identifier is contained
in ROWIDS and passes the condition in the column COLID.
The search sequence depends on the ROWIDS list. The identifier of the
first found row is returned in ROWID.

Example : `findr(MOD1\WZ,BED,(FR_,B_),p.wz);`

### procedure FINDT (tabid,colid:n; val:s; var rowids:l);

Searches for the rows in table TABID that contain the value VAL
in the column COLID, and stores their identifiers in the list ROWIDS.

Example : `lines:l;  findt(TAB1,PRICE,'120',lines);`

### procedure POST (/tab,rid,cid:n; r,c:i);

Activates a table, a row or a column depending upon the indicated
parameters. Once a table, row or column is activated, it remains
active until the next POST instruction.
After POST the variables TC, TROWID, TROWP, TCOLID, TCOLP are set.

| | |
|---|---|
| tab : | Table Identifier |
| rid : | Row identifier |
| cid : | Column name |
| r : | Row position (starts at 1) |
| c : | Column position; The first column is not NR but the following (mostly the NAME column). |

Example :
```
post(tab=M1\NT);  /activates table NT in module M1
post(cid=PR,r=3); /activates column PR and the 3rd row
infv(TROWID);     /shows the row ID.
tc.1:s='5';  /inserts 1.line in the active cell = '5'
post(r=2);   /activates 2.row, active column remains PR
tc.1:='7';   /changes 1.line in the active cell to '7'
```

### procedure PUTT (v:n; /tab,rid,cid,o:n; r,c,f:i);

Adds the content of the variable V to the active table.
Tables, rows or columns are activated by PUTT and remain so
until the following PUTT instruction. After the instruction
PUTT the variables TC, TROWID, TROWP, TCOLID, TCOLP are set.

v : Name of the variable containing the value to be inserted

tab : Table identifier

rid : Row identifier

cid : Column name

o : Options (letter):
- I: If line RID it not found, it is inserted
- S: Inserted lines are sorted alphabetically
- A: Values are added

r : Row position (starts at 1)

c : Column position; The first column is not NR
but the following (mostly the NAME column).

f : Format; Depending upon type: Decimal cases or date format

Example :
```
m1:r=230;
putt(m1,tab=MIF1\NT,r=1,c=3,o=A);
putt(m1,rid=[rownr],c=3,o=ISA);
```

### procedure OPENT (t:n; nr,name,cols:s);

Opens a table. After OPENT, T is the active table.

t : Name of the variable (list) that contains the table

nr : Table identifier

name : Table designation

cols : String with the column names and sizes,
each separated with : and ; .

Example :
```
t1:l;
opent(t1,'T','Test','NAME:20;QUANTITY:8;PRICE:12');
```

### procedure INSCOL (p,w:i; id:s);

Inserts a column in the active table.

p : Position of the new column.

w : Width (in number of letters) of the new column.

id : Name of the new column.

Example :
```
inscol(3,20,'PRICE');
```

### procedure MODCOL (/p:i; id:n; w:i; id2:s);

Changes a column in the active table.

p : Position of the column.

id : Name of the column.

w : New width (in number of letters) of the column.

id2 : New name of the column.

Example :
```
modcol(id=NAME,w=20);
modcol(p=3,id2='gew>20');
```

**procedure DELCOL (/p:i; id:n);**

Deletes a column from the active table.

p : Column position

id : Column name

Example :
```
delcol(p=2);
delcol(id=PRICE);
```

**procedure INSROW (p,h,lev:i; id:s);**

Inserts a row in the active table.

p : Position of the new row.

h : Height (number of text lines) of the new row.

lev : Hierarchic level of the new row (1 to 20).

id : Identifier of the new row.

Example :
```
insrow(i,1,1,'A07');
```

**procedure MODROW (/p:i; id:n; h,lev:i; id2:s);**

Changes a row in the active table.

p : Position of the row.

id : Identifier of the row.

h : New height (number of text lines) of the new row.

lev : New hierarchic level of the row (1 to 20)

id2 : New identifier of the row.

Example :
```
modrow(id=A01,w=20);
modrow(p=3,id2='gew>20');
```

**procedure DELROW (/p:i; id:n);**

Deletes a row from the active table.

p : Row position

id : Row identifier

Example :
```
delrow(p=7);
delrow(id=A01);
```

**function EXROW (nr:s):i;**

Returns 1 if the row NR exists in the active table and 0 if it does not.

Example :
```
if exrow('A01')=0: inf('Row not found');
```

**function EXCOL (nr:s):i;**

Returns 1 if the column NR exists in the active table and 0 if it doesn´t.

Example :
```
if excol('NAME')=0: inf('Column not found');
```

**procedure EDITTAB (t,t0:n; /o:n; b1:s);**

Opens the table editor to edit the table T.
If the table T is empty, it is filled with the initial table T0.

T : Table to be edited

T0 : Initial table

O : Options = sequence of characters with the following meaning:
L = all cells are single-line; Enter key jumps on next cell
A = automatic line numbering (1.2.3...)

B1 : Optional button: Syntax = 'button_name:procedure'

Example :
```
edittab(tab1,tab0,o=LA);
edittab(t1,t0,b1='_Control:controltab');
```

## 5.8 Structures

Structures are lists that have elements of the type ELEMENT only.
These elements can contain substructures, as well as parameters
and other assigned data.

### 5.8.1 Structures handling

#### Variable for the access to structure elements

After POSE, POSP,or during the TRAVEL-instruction, the following
variables are set and/or updated:

| | | |
|---|---|---|
| **pe** | ^e | Pointer to the active element. |
| **pe0** | ^e | Pointer to the father of the active element. |
| **enr** | s | Number of the masters of the active element. |
| **enr0** | s | Number of the master of the father of the active element. |
| **etabid** | s | Masters table of the active element. |
| **emod** | s | Knowledge module of the master table of the active element. |
| **eid** | s | Complete indentifier of the active element.<br>Format: emod\etabid.enr |
| **ename** | s | Name of the active element. |
| **p** | l | Parameter list of the active element. |
| **d** | l | DATA field of the active element. |
| **ss** | l | Substructure of the active element. |
| **eh** | i | Hierarchic level (1..40) of the active element. |
| **ei** | i | Position of the active element in its hierarchic level. |
| **ech** | s | characteristic of the masters of the active element (column CH). |
| **etyp** | n | Type of the active element, e.g.: R, l, E, GLINE, etc.. |
| **ep** | s | Path of the active element. |
| **efound** | i | Result of procedure FINDE. |
| **pefound** | ^e | Pointer to the found element with FINDE. |

#### procedure POSE (var e:e);

Activates the element indicated as E and sets the associated system
variables D, P, SS, ENR, EMOD, ENAME, EP, EH, PE.

Example :
```
pose(pr.bs.2.ss.1);  inf('NAME='+ename);
```

#### procedure PUSHST;

Saves the current structure context (variables D, P, SS, ENR...)
This command should be issued before POSE or POSP, and must allaways
be followed by a POPST-command.

Example :
```
pushst;
```

```
pose(pr.ap.1);
ts:s=ename;
popst;
```

### procedure POPST;

Re-establishes the structure context left before with PUSHST.

### procedure MODEID (var st:l; id1,id2:s)

Replaces the identifier ID1 by ID2 in all elements of ST that have ID1 as element master number.

Example : `modeid(pr.bs,'DA1','DA');`

### procedure MODEMOD (var st:l; mod1,mod2:s)

Replaces the module MOD1 by MOD2 in all elements of ST that have MOD1 as module.

Example : `modemod(pr.bs,'MODX','MODY');`

### procedure MODETAB (var st:l; tab1,tab2:s)

Replaces the table TAB1 by TAB2 in all elements of ST that have TAB1 as master table.

Example : `modetab(pr.ap,'OP','AV');`

### function NE (var st:l; nr:s; /h:i; c:s; path:n):i;

Searches elements in a structure and returns the number of elements found.

| | |
|---|---|
| st : | Structure where to search for |
| nr : | Filters for master number |
| h : | Number of hierarchic levels to be searched; if not indicated, it is assumed H=1; |
| c : | Condition function (optional) filters found elements |
| path : | Path delimitation (optional) |

```
Example : if ne(pr.bs,'EL')>0: inf('Electric found');
          p1:n=3.1;
          nab:i=ne(pr.bs,'A_,B_',h=5,c=bed5,path=p1);
```

### procedure FINDE (var st:l; /);

Searches an element in the structure ST that fulfills several conditions. These conditions are entered freely. With the special condition PP, it is made a PUSHST before and a POPST after the positioning. If the element if found, the system variable EFOUND gets the value 1 and the system remains there positioned, i.e. the system variables PE, ENR, ENAME, etc. refere to the found element. In particular the variable PEFOUND is set and should be used if the condition PP was entered.

```
Example : finde(pr.bs,pp,ex(p.width),p.width>120);
          if efound: inf(pefound^.name+' found');
```

**procedure DELE (var st:l; ids:s; /c:s);**

Deletes all elements in the structure ST where the master number is contained in IDS. With C, additional condition for the deletion can be formulated.

Example : `dele(pr.ap,'A000\OP.A_',c='p.l>250');`

**procedure POSST (/h:i; p:n; o:n);**

Activates all elements in the current structure path;

h : maximum hierarchic level

p : Procedure to be executed in each stage

o : Options (characters):
- P: calls PUSHST before positioning and POPST after positioning; thus re-establishing the state before POSST

Example :
```
posst(p=PROC1, o=P);
posst(h=eh-1);

/ in the procedure PROC1 could stand
/ e.g.:  if etabid='MAT': material:=enr;
/ thus raw material position is recognized
/ without having a fixed position
```

**procedure SORTST (var st:l; id:n; /inv:i);**

Sorts the structure ST according to a given parameter.

st : Structure to be sorted;

id : Name of the parameters to serve as sorting criteria;
This parameter can be from the following types: i, r, s, n, d

inv : If INV=1, ST is sorted in descending order;

Example : `sortst(pr.bs,p.width);`

**procedure MODP (var path:n; l,n:i);**

Example :

## 5.8.2 Generate structures

**Variables for generating structures**

The following variables are used in the process of generating structures. PEB, LEB and IEB are set after each build command (->).

| | | |
|---|---|---|
| **peb** | ^e | Pointer to the last inserted element |
| **leb** | l | List that contains de last inserted element |
| **ieb** | i | Position of the last inserted element in LEB |
| **enew** | i | 1 if active element was inserted, otherwise=0 |
| **as** | s | Active situation |

**procedure ACT (as,e0:n; t:s; /d:i; p1,pa1:n; tfp,tfn:s);**

Executes the actions defined in the master of the element (or in the element list) of E0.

AS : Situation for the required actions

| | |
|---|---|
| E0 : | Name of the list or element that start the actions |
| T : | Title for the reporting window |
| D : | Display mode: 0=shows nothing; 1=message; 2=report; |
| P1 : | Procedure that will be executed for each position, before the actions; |
| PA1 : | Id of the path of the producer element. This path is inserted in DATA of the new builded element |
| TFP : | Positive filters: indicate which positions will be searched |
| TFN : | Negative filters: indicate which positions will not be searched |

Example :
```
act(PAI1,pr.m,'Generation process');
act(GENAP,pr.bs,'Generation process',pa1=BK);
act(K2,pr.ap,'Costs',d=0,p1=posop,tfp='1._,2._');
```

### procedure SETAS(s:s);

Sets the active condition.

Example : `setas('PAI1');`

## 5.8.3 Dialogue with structures

#### Variables for dialogues in structures

The following variables are used for formatting and programming the behavior of a structure editor:

| | | |
|---|---|---|
| **eol** | b | 1 if the cursor is at the end of the list. |
| **xlfe** | r | X-left-coordinate of the current line |
| **xrfe** | r | X-right-coordinate of the current line |
| **ytfe** | r | Y-top-coordinate of the current line |
| **ybfe** | r | Y-bottom-coordinate of the current line |
| **copyok** | i | Permission to copy into the clipboard in EDITST; default is 1 (allowed); it can be set in to 0 in the structure editor handler; |
| **cutok** | i | Permission to cut into the clipboard in EDITST; default is 1 (allowed); it can be set in to 0 in the structure editor handler; |
| **pasteok** | i | Permission to insert from the clipboard into the selected position; can be set in the structure editor handler; |
| **delok** | i | Permission to delete in EDITST; default is 1 (allowed); can be set in the structure editor handler; |
| **insok** | i | Permission to insert in EDITST; default is 1 (allowed); can be set in the structure editor handler; |
| **parsok** | i | Permission to open the dialogue to edit the element parameters. Default is 1 (allowed); Can be set in the structure editor handler; |
| **sttabid** | s | Table to be used as source for the structure editor. Used in the handler procedure of EDITST. |
| **stinsnr** | s | Number of the element to be inserted in the structure editor. Thus the insert menu does not appear. |

| | | |
|---|---|---|
| | | Used in the handler procedure of EDITST (AT PREINS). |
| **stinst** | s | Title for the insert menu in the structure editor.<br>Used in the Handler procedure of EDITST. |
| **stinsf** | s | Insertion filter for elements in the structure editor.<br>Set in the Handler procedure of EDITST. |
| **stinsfm** | s | Insertion filter for modules in then structure editor.<br>Set in the Handler procedure of EDITST. |

**Situations for the EDITST-handler**

Following situations can be used in the handler of EDITST

| | |
|---|---|
| **PREINS** | after pressing the INSERT button in the structure editor |
| **POSTINS** | after inserting an element in a structure |
| **PREDEL** | after pressing the DELETE button in the structure editor |
| **POSTDEL** | after the deletion of an element in a structure |
| **PREPARS** | after pressing the PARAMETERS button in the structure editor |
| **POSTPARS** | after the parameter editing of an element in a structure |
| **DRAG** | before starting to drag |
| **DROP** | before dropping the dragged elements |
| **RCLICK** | after right Click with mouse |
| **COPYCLIP** | after pressing CTRL-Ins (copy into the clipboard) |
| **CUTCLIP** | after pressing SHIFT-Del (cut into the clipboard) |
| **PASTECLIP** | after pressing SHIFT-Ins (insert from the clipboard) |

**Situations for the A-column**

Following situations can be used in the A-column of the master table:

| | |
|---|---|
| **I2** | after the manual insertion of an element in a structure<br>(before the dialogue mask) |
| **I3** | after inserting an element in a structure<br>(after the dialogue mask) |

### procedure EDITST (typ:i; var st:l; es,as,opt:n; t1,t2:S /h:n; b:s; id,ef:i);

Opens the structure editor for the structure ST.

| | |
|---|---|
| TYP : | Always 1 (for later use) |
| ST : | Structure to be edited |
| ES : | Source of the elements = Identifier of the master table |
| AS : | Active situation during editing |
| OPT : | Options = letters sequence with the following meaning:<br>A: Editor with zoom all button<br>C: Editor with copy button<br>D: Editor with document button<br>E: Editor without delete button<br>F: Editor without insert button<br>G: Editor with group button (adds masterless groups)<br>H: Hierarchical position is indicated for each element<br>K: suppresses F12-Function (see data structure)<br>L: Editor with relations button<br>N: Editor with rename button<br>P: Editor with parameters button |

Q: Editor with quantity button
R: Editor with resources button
T: Editor with terms button
U: Editor with status button
W: Resources editor with Order button.
Condition: Resources data base module must be defined as DW!
Y: Suppresses the transfer functions (clipboard)
Z: Editor with zoom button to make the text lines thicker

T1 : First title for the Frame register

T2 : Second title for the Frame register

H : Optional control procedure (handler):
So that structure changes can be controled
(insertion, deletion, etc..).

B : Optional buttons: Syntax: 'Button_name1:Procedure1,Button_name2:Procedur2...'

ID : Identification of the dialogue window;

EF : Options for formatting the lines:
bit1 1: with [ doc ] if a document is assigned;
bit2 1: with [ ress ] if resources are assigned;

Example :
```
editst(1,pr.bs,BK,B01,PQ,'Structure analysis',pr.nr);
editst(2,pr.lc,PROD,B07,PQ,'Parts list ',pr.nr);
editst(1,pr.bs,BK,,P,'','',b='Client:inpc,Place:inpp');
editst(1,pr.ap,OP,,PRQ,'Expiration','',h=expcontrol);

Example of control procedure of EXPCONTROL:

AT PREINS: /after pressing the insert button

/in the hierarchy level 1 nothing can be inserted:
if eh=1:insok:=0&exit;
/Insert menu if specified:
if k1='MA':sttabid,stinst:='MA','Material';
if k1='OP':sttabid,stinst,stinsf:='OP','Operations','D_';

at POSTINS: /after the element insertion
if etabid='OP':calculate_operations_nr;

at PREDEL:  /after pressing the delete button
/at hierarchy level 1 nothing can be deleted
if eh=1:delok:=0&exit;
/Element MA cannot be deleted
if enr='MA':delok:=0&exit;

at POSTDEL: / after the element is deleted
if etabid='OP':calculate_operations_nr;

at RCLICK:  / after mouse Right-click
del(d.menu);/  deletes previous menu
d.menu:l;   / builds a new menu
if enr='MA': d.menu.0:s='Material sheet:printmatsheet';
d.menu.0:s='Show costs:showcost';
```

**Formatting the structure lines at the screen**

In the column FORMAT you can flexibly design the appearance of an element in the structure editor. Here you can write commands to change the system variables EFICON or EFSYMB, EFPREFIX, EFNAME, EFPARS, EFSUFIX. The output line consists in the concatenation of these system variables.

In addition you can specify the line color with the variable EFCOLOR.

It is also possible to draw graphic elements in the output line. By using the draw instructions (draw, drawline...). The coordinates of the line rectangle are indicated in the system variables XLFE, YTFE, XRFE, YBFE.

Example :
```
efpars:=sr(p.d,0)+' * '+sr(p.l,2);
if p.status=2: efcolor:=40;
```

### procedure SELP (var st:l; var p:n; /t:s; w,h:r);

Allows to select an element from a structure returning the position path of the selected element.

ST : Structure from which an element is to be selected

P : Name of the variable (of the type name) that will contain the path of the element selected by the user.

t : Window title (default is 'component')

w : Width of dialogue window in number of characters / default=80

h : Height of dialogue window in number of lines / default=24

Example :
```
p1:n; selp(pr.bs,p1,t='Please select !');
```

### procedure SELE (var st:l; var p:^e; /t:s; w,h:r);

Allows to select an element from a structure returning the pointer to the selected element.

ST : Structure from which an element is to be selected

P : Name of the pointer variable (of the type ^e) that will point to the element selected by the user

t : Window title (default is 'component')

w : Width of dialogue window in number of characters (default=80)

h : Height of dialogue window in number of lines (default=24)

Example :
```
p1:^e; sele(pr.bs,p1,t='Please select !');
```

### procedure DESELAE;

Deselects the structure-field active by EDITST.
This is necessary and important when the structure beeing displayed must be changed. Mirakon will allways try to return to the selected position. If this position doesn´t exist, an error will occur.

Example :
```
/ button in structure-editor calls following commands:
/ in order to empty the all structure
deselae;
empty(pr.bs);
rsetm(MAIN);
```

### function EMARK (p:^e):i

Informs whether the element pointed by P is marked (emark=1) or not (emark=0).

Example :
```
if emark(pe): number++1;
```

### procedure MARKE (p:^e; val:i);

Mark (val=1) and/or unmark (val=0) the element pointed by P.

Example :
```
marke(pe,1);
```

#### 5.8.3.1 Structure editor with master table

Structures with elements from master tables are edited with EDITST.

When a new element is added (INSERT key), the parameters defined in the PARS column are created, and edited with the dialogue mask indicated in the MASK column.

After leaving the mask with OK, Mirakon executes all assigned actions for the active situation.

When the parameters of an existing element are changed with the PARS button, the system behaviour can be affected with the characteristics indicated in the CH column:

- E = the sub-structure of the element concerned is emptied, before executing the assigned actions. If this sub-structure was manually changed, the program asks the user whether he wants to overwrite or keep the manually inserted sub-elements.

## 5.9 Relations

**Variables for the handling of relations**

During TRAVELR the following variables are set and/or updated.

| | | |
|---|---|---|
| **rtyp** | i | Type of the active relation. |
| **rref** | i | Reference of the active relation.<br>1=Element is reference 2=Element is referenced (Slave) |
| **pre1** | ^e | Pointer to the reference element of the active relation. |
| **pre2** | ^e | Pointer to the dependent element of the active relation. |
| **rdata** | l | Data of the active relation. |

## 5.10 Processes

Prozessen sind Listen von Elementen die grafisch auf eine "Papier"-Fläche frei darstellbar und positionierbar sind. Diese Elemente lassen sich beliebig logisch wie grafisch miteinander relationieren.
Prozesse sind geeignet um z.B. Fabrikationsprozesse darzustellen.

### 5.10.1 Grafic objects and editors

**procedure EDITP (var proc:l; tab:n; /relt,h:n; b:s);**

Opens the grafic editor for the grafic G.

proc : Grafic object

tab : Options (letters sequence):
S = grafic editor is opened as a son window of the current dialogue window (small window)

relt :

h :

b :

Example : `editp(fp.tp,TPE,relt=RELTAB,b='Berechnen:ber1',mult=1);`

Table TPE:

| NR | NAME | PARS | MASK | G | A |
|---|---|---|---|---|---|
| T | Teile | | | | |
| TB | Blechteil | tnr,tbez:s;<br>mk:r;<br>b:r:breite mm=40;<br>h:r:höhe mm=20;<br>bild:s; | TPTB | draw(ab10,gb(0,0,p.b,-p.h));<br>draw(tf3,th12,x2,y-5,tw1,vp.tb<br>h2,b2:r=p.h-7,1.2*h2;<br>bild:s='FPR1\'+p.bild;<br>draw(gi(bild,2,-7,2+b2,-7-h2)) | |
| V | Vorgänge | | | | |
| VK | Kleben | nl:r;<br>bez:s='Kleben';<br>b:r:breite mm=35; | TPVK | draw(ab14,gb(0,0,p.b,-p.h));<br>draw(tf3,th12,x2,y-5,tw1,'Kleb<br>pars:s=si(p.nl)+' mm'; | |
| VP | Punkt-Schweiße | np:i;<br>bez:s='Punktschwei<br>b:r:breite mm=35; | TPVP | draw(ab3,gb(0,0,p.b,-p.h));<br>draw(tf3,th12,x2,y-5,tw1,'Punk<br>pars:s=si(p.np)+' Punkte'; | |
| Z | Zellen | | | | |
| ZM | Montage-Zelle | znr:i;<br>zbez:s;<br>flaech:r;<br>takt:r; | TPZM | draw(ab7,gb(0,0,p.b,-p.h));<br>titel:s='Zelle '+p.znr+' '+p.z<br>draw(tf3,th12,x2,y-5,tw1,vtite | |

Table RELTAB:

| NR | NAME | PARS | MASK | G |
|---|---|---|---|---|
| BKOP | Input Relation | funk:s;<br>farbe:i=40;<br>ap1:i=2;<br>ap2:i=4; | RELBKOP | p1,p2:point2d;<br>geted(pie1,p2d=p1);<br>geted(pie2,p2d=p2);<br>x1,y1,x2,y2:r=p1.x,p1.y,p2.x,p2.y;<br>b1,h1:r=pie1^.pars.b,pie1^.pars.h;<br>b2,h2:r=pie2^.pars.b,pie2^.pars.h;<br>draw(lcp.farbe,gl(x1,y1,x1b,y1b),<br>gl(x1b,y1b,x2b,y2b),le2,<br>gl(x2b,y2b,x2,y2),le1,lc0);<br>xm,ym:r=(x1+x2)/2,(y1+y2)/2;<br>draw(tf3,th12,tw1,ta3,tcp.farbe,<br>xxm+1,yym+1,vp.funk,tw0,ta1,tc0); |

## 5.11 Documents

### 5.11.1 Listings

Text documents are unformated lists of strings. Pages cannot be indicated nor numbered. The text is always written in the Font Courier-New.

We recommend the use of grafic documents instead.

**procedure OPENLST (var lst:l);**

Opens the text document LST.

Example : `openlst(tdoc);`

**procedure EDITLST (var lst:l; t:s; /w,h:r);**

Opens the text editor for the document LST, with the title T.
The window width and/or height can be specified with W and H,
in number of characters. If W and H are not specified, the
whole work surface is used.

Example : `editlst(tdoc,'Parts list');`

**procedure WRITES (dr,c,adj:i; val:s);**

Writes a string value in the opened text document.

dr : Line jump related to the last written line .

c : Column position.

adj : Adjustment: 1=left-justified 2=right-justified.

val : String value to be written

Example : `writes(1,25,1,pr.name);`

**procedure WRITEI (dr,c,adj,val:i);**

Writes an integer value in the opened text document.
Parameters as in WRITES.

Example : `writei(0,42,2,i1);`

**procedure WRITER (dr,c,adj:i; val:r; f:i);**

Writes an real value in the opened text document.
Parameters as in WRITES.
F=Format=Number of decimal places

Example : `writer(0,25,2,price,2);`

**procedure WRITED (dr,c,adj:i; val:d; f:i);**

Writes a date value in the opened text document.
Parameters as in WRITES.
F = date format (see function SD)

Example : `writed(1,2,1,d1,1);`

## 5.11.2 Grafic-Documents

**Types for the production of grafic documents**

**GIMAGE** Picture (bitmap) in BMP or JPG format

**Variables for the production grafic documents**

**page** i Current page number of the active document.

### 5.11.2.1 Grafic objects and editors

**procedure OPENG (var g:l; nr,name:s; w,h:r; /xm,ym:r; m,np,ep:n; nd:i);**

Opens the (formatted) grafic G.

g : Grafic object

nr : Number of the grafic object

name : Designation of the grafic object

w : Grafic width in mm

h : Grafic height in mm

xm : Left and Right margin in mm (optional)

ym : Top and bottom margin in mm (optional)

m : Page layout = Mask-Id

np : new page procedure, called after a new page is inserted

ep : End of page procedure, called after a new page is inserted,
but still refering to the last page.

nd :

Example :
```
g1:l;
openg(g1,'A1','Sketch',210,290,xm=20,ym=10,m=M1,np=ns1);
editg(g1);
```

### procedure EDITG (var g:l; /o,build:n; zf,w0,h0:r; id:i);

Opens the grafic editor for the grafic G.

g : Grafic object

o : Options (letters sequence):
S = grafic editor is opened as a son window of the
current dialogue window (small window)

build : Name of the procedure that (re)builds the grafic; this parameter
causes a BUILD-button to appear above the grafic window;

zf : Initial zoom factor; Default = 1.0;

w0 : Initial paper width (only if G is empty)

h0 : Initial paper height (only if G is empty)

id : Identification of the dialogue window;

Example : `editg(pr.doc1,build=gendoc,zf=1.2);`

### procedure VIEWG (var g:l; /o,build:n; zf:r; id,nar:i);

Opens the Grafic viewer for the grafic G.
Parameters: equal as in EDITG.

NAR : Without active regions

## 5.11.2.2 Paper coordinates, fields and frames

### procedure SETMARGIN (x,y:r);

Sets the left and top margin, therefore shifting the absolute zero.

x : Left side margin in mm (always positive value)

y : Top margin in mm (always positive value)

Example : `setmargin(20,10);`

### procedure SETZERO (x,y:r);

Sets a new zero point with coordinates X,Y from the absolute paper zero point
which is the top left corner of the paper sheet.

Example : `setzero(20,-110);`

**procedure DEFGF (id:n; xlp,ytp,xrp,ybp,xli,yti,xri,ybi:r);**

Defines a rectangular grafic field in the active document with its own inner coordinate system.

id : Identifier of the field;
xlp : X-left coordinate on the paper
ytp : Y-top coordinate on the paper
xrp : X-right coordinate on the paper
ybp : Y-bottom coordinate on the paper
xli : X-left coordinate within the field
yti : Y-top coordinate within the field
xri : X-right coordinate within the field
ybi : Y-bottom coordinate within the field

Example : `defgf(BAR,20,-50,150,-130,10000,0,0,200);`

**procedure SETGF (id:n);**

Activates the grafic field ID.

Example : `setgf(BAR);`

### 5.11.2.3 Drawing texts

**procedure SETFONT(font,h,sty,w,col:i);**

Sets the font attributes for the following DRAW procedures.

font : Font number in the configuration.
h : Font size in points
sty : 0=normal 1=italic
w : 0=normal 1=bold
col : Text color NR.

Example : `setfont(4,10,0,0,0);`

**procedure DRAWI (x,y:r; adj:i; i:i);**

Writes the Integer I at the position X,Y (in mm).

adj : Adjustment: 1=left justified 2=right-justified 3=center

Example : `drawi(150,-8,1,i1);`

**procedure DRAWR (x,y:r; adj:i; r:r; f:i);**

Writes the real R at the position X,Y (in mm).

adj : Adjustment: 1=left justified 2=right-justified 3=center
f : Format=number of decimal places

Example : `drawr(120,-40,2,price,2);`

**procedure DRAWD (x,y:r; adj:i; d:d; f:i);**

Writes the date D at the position X,Y (in mm).

adj : Adjustment: 1=left justified 2=right-justified 3=center

f : Date format (see function SD)

Example : `drawd(150,-8,1,now,1);`

**procedure DRAWS (x,y:r; adj:i; t:s);**

Writes the string T at the position X,Y (in mm).

adj : Adjustment: 1=left justified 2=right-justified 3=center

Example :

**procedure DRAWQ (x,y:r; adj:i; q:q; f:i);**

Writes the quantity Q at the position X,Y (in mm).

adj : Adjustment: 1=left justified 2=right-justified 3=center

f : Format=number of decimal places

Example :

**procedure DRAWTEXT (x,y,dy:r; adj:i; var txt:l);**

Writes all lines from the list of strings TXT starting at the position X,Y (in mm).

dy : Line space in mm

adj : Adjustment: 1=left justified 2=right-justified 3=center

Example :
```
txt1:ls=('AAA','BBB','CCC');
drawtext(50,-25,4,1,txt1);
```

### 5.11.2.4 Drawing lines and surfaces

**procedure SETLINE (sty,wid,end:b; col:i);**

Sets the line attributes for the following DRAWLINE procedures.

sty : Line type: 1=full 2 = - - - 3 =... 4 = -. -

wid : line thickness: 1=0.25mm 2=0.35mm 3=0.5mm 4=0.75mm 5=1mm

end : line end: 1=normal 2 = -> 3 = <-> 4 = -< 5 = -o

col : 3 = Line color number

Example : `setline(3,1,1,3);`

**procedure DRAWLINE (x1,y1,x2,y2:r);**

Draws a line from the point (x1, y1) to the point (x2, y2).

Example : `drawline(150,-8,200,-40);`

**procedure DRAWARC (x0,y0,rx,ry,a1,a2:r);**

Draws a circular or eliptical arc.

x0 : X-coordinate of the center

y0 : Y-coordinate of the center

rx : Radius in x-direction

ry : Radius in y-direction

a1 : Initial angle in degrees

a2 : Final angle in degrees

Example :
```
drawarc(40,-50,25,25,0,360); /arc with ray=25
drawarc(40,-50,5,20,0,180);  /top half of an Elipse
```

**procedure SETBOX (form,pat:b; bcol,fcol:i);**

Sets the graphic attributes for DRAWBOX.

form : Box format: 0= without border 1=normal 3=3-D-Box

pat : Filling pattern: 0=empty 1=full 2=points 3=points 4=///
5 = \\\ 6 = # # # , ...

bcol : Bbackground color number

fcol : Foreground color number

Example : `setbox(1,2,14,2);`

**procedure DRAWBOX (xl,yt,xr,yb:r);**

Draws a rectangular box.

xl : X-coordinate of the left edge of the box

yt : Y-coordinate of the upper edge of the box

xr : X-coordinate of the right edge of the box

yb : Y-coordinate of the lower edge of the box

Example : `drawbox(150,-8,200,-40);`

### 5.11.2.5 Drawing lines and surfaces

Example :
```
x1,x2,y1,y2:r=20,40,-10,-15;
draw(X0,Y0,W120,H50,GH0,GHy2,GVx2,GL(x1,y1,x2,y2));
draw(X20,Y-60,TF4,TH12,Vename,X60,'ready !');
```

**path2d**

**segline2d**

**segarcc2d**

**segarce2d**

### 5.11.2.6 Using existing grafics

**procedure DRAWIMAGE (id:s; xl,yt,xr,yb:r);**

Draws an image from the images resources, on the active document.
The image is ajusted to the specified window.

id : Image ID as it is indicated in the resources

xl : X-coordinate of the left edge of the window

yt : Y-coordinate of the upper edge of the window

xr : X-coordinate of the right edge of the window

yb : Y-coordinate of the lower edge of the window

Example : `drawimage('LOGO',150,-8,170,-20);`

**procedure DRAWGRAFIC (var g:l; xl,yt,xr,yb:r; /dm:i);**

Draws the grafic G in the specified window on the active document.

g : Grafic object to be drawn

xl : X-coordinate of the left edge of the window

yt : Y-coordinate of the upper edge of the window

xr : X-coordinate of the right edge of the window

yb : Y-coordinate of the lower edge of the window

dm :

Example :
```
drawgrafic(g1,150,-8,200,-40);
```

### procedure DRAWMASK (id:n);

Draws an dialogue mask in the active document.

id : Mask identifier

Example :
```
drawmask(A000\M1);
```

### procedure IMPGIMAGE (file:s; var img:gimage; /fmt,compr:i);

Imports an external image file to a variable.
If desired, the file format can be converted.

file : Image file path

img : Image variable where to the picture will be copied

fmt : Image format if it is to be converted: 1=BMP, 2=JPG

compr : Compression factor in %; Default=100; (only for JPG)

Example :
```
mylogo:gimage;
impgimage('C:\LOGO.BMP',mylogo);
drawgimage(mylogo,150,-8,170,-20);
```

### procedure DRAWGIMAGE (var img:gimage; xl,yt,xr,yb:r; /dm:i);

Draws an image from a variable into the active document.
The image is ajusted to the specified window.
(see example in the procedure IMPGIMAGE)

img : Image variable

xl : X-coordinate of the left edge of the window

yt : Y-coordinate of the upper edge of the window

xr : X-coordinate of the right edge of the window

yb : Y-coordinate of the lower edge of the window

dm :

### procedure CONVGIMAGE (var img1,img2:gimage; typ2:i; /compr:i);

Converts an image variable into the indicated format.
The compression factor may be specified (when converting to JPG).

img1 : Source image variable

img2 : Image variable where to the image is converted

typ2 : Image format of img2: 1=BMP, 2=JPG

compr : Compression factor in %; Default=100;

Example :
```
logo1,logo2:gimage;
convgimage(logo1,logo2,2,compr=75);
```

## 5.11.2.7 Drawing value and time scales

### procedure DRAWSCALE (x1,y1,x2,y2,v1,v2,s1,s2:r);

Draws a value scale from X1, y1 to X2, y2 with values from V1 to V2.

s1 : Step size between written values

s2 : Step size between drawn lines

Example : `drawscale(20,-100,20,-30,0,100,10,5);`

### procedure DRAWTSCALE (tu:i; xl,yt,xr,yb:r; t1,t2:d);

Draws a time scale.
Character size should be adjusted before with SETFONT or DRAW.

tu : Scale format, determines what is to be written:
1 = months, weeks and days; 2 = months and year;
3 = weeks; 4 = days; 5 = hours;

xl,yt,xr,yb : Coordinates of the square on the paper:
X-Left, y-Top, x-right, y-Bottom

t1,t2 : Initial and final date of the scale

Example :
```
d1:d=21.3.05; d2:d=4.5.05;
drawtscale(3,0,0,200,-20,d1,d2);
```

## 5.11.2.8 General drawing comands DRAW

### procedure DRAW (/)

Draws according to the entered parameters.
Each parameter consists of a command (first or first two characters)
and an operand (remaining characters, usualy numbers).

Commands:

X + number : sets the current x-coordinate

Y + number : sets the current Y-coordinate

W + number : sets the current width

H + number : sets the current height

AP + number : sets the filling pattern for areas:
0=empty 1=full 2=points 3=points 4=/// 5 = \ \ \
6 = # # #, ...

AC + number : sets the foreground color

AB + number : sets the background color

AE + number : sets the edge of surfaces: 1=with outline;

BF + number : sets the box format: 0=without border 1=normal 3=3-D-Box

LW + number : sets the line weight

LS + number : sets the line style

LC + number : sets the line color

LE + number : sets the line ending: 1=normal 2=with one arrow
3=with two arrows ,...

TA + number : sets the text adjustment: 1=left 2=right

TF + number : sets the text font

TH + number : sets the text size in points

TW + number : sets the text thickness: 0=normal 1=bold

TS + number : sets the text style: 0=normal 1=italic

| | |
|---|---|
| TC + number : | sets the text color |
| TL + number : | sets the maximum text length in mm |
| TV + number : | sets the vertical text adjustment: 1=Baseline 2=Top 3=Middle |
| TR + number : | sets the text angle (in degrees) |
| GH + Y-value : | draws an horizontal line from the actual x-position and the y-position indicated, with the actual width |
| GV + X-value : | draws an vertical line from the actual y-position and the indicated x-position, with the actual height |
| GL(x1,y1,...) : | draws a polygon through the indicated corner points X1, y1, etc. |
| GA(x1,y1,...) : | draws a surface with the indicated corner points X1, y1, X2, Y2, etc. |
| GB(xl,yt,xr,yb) : | draws a filled box |
| GR(xl,yt,xr,yb) : | draws a rectangle and sets the actual values for X, Y, W and H; (ideal for tables) |
| 'Text' : | writes the indicated text |
| V + name : | writes the content of the indicated variable |
| F + number : | sets the format = number |
| M0 : | sets the mirror function off |
| MX : | reflects all elements using the x axis |
| MY : | reflects all elements using the y axis |
| NR : | next line |
| NP : | next page |
| NX : | next X |
| NY : | next Y |
| ZX + number : | sets the X coordinate of the zero point |
| ZY + number : | sets the Y coordinate of the zero point |
| ON1 : | switches ON the automatic page change. (switch OFF with ON0) When, during text writing, the maximum Y-value (defined with RB parameter in openg command), if exceeded, a new page is inserted. |
| OP0 : | Option P0: suspends printing: after this instruction the drawn elements will not be printed, they only appear on the screen (eg.: usefull when using paper with letter heads); |
| OP1 : | Option P1: Enables printing (after using OP0); after this instruction the drawn elements will be printed again; |
| OZ0 : | Option Z0: Numbers with value 0 will not be written |
| OZ1 : | Option Z1: Numbers with value 0 will be written |

Example :
```
x1,x2,y1,y2:r=20,40,-10,-15;
draw(X0,Y0,W120,H50,GH0,GHy2,GVx2,GL(x1,y1,x2,y2));
draw(X20,Y-60,TF4,TH12,Vename,X60,'ready !');
```

### 5.11.2.9 Page change

#### Variable for controlling the page change

| | | |
|---|---|---|
| **page** | i | Current page NR; updated with INSPAGE/SETPAGE |
| **ryt** | r | Y-Top-coordinate of the current line in the active document |
| **ryb** | r | Y-Bottom-coordinate of current line in the active document |
| **rh** | r | Line height in the active document |

### procedure INSPAGE (nr:i);

Adds a new page in the position NR.
OPENG already produces the first page, therefore INSPAGE
is only needed for further pages.
If NR=0, the page is inserted at the end.
The system variable PAGE is updated.

Example : `inspage(0);`

### procedure SETPAGE (nr:i);

Activates the page NR. If NR=0, the last page is activated.
The system variable PAGE is updated.

Example : `setpage(1);`

### procedure NEXTPAGE;

Inserts a new page and executes the new page procedure defined in OPENG;

## 5.11.2.10 Text adjustment

### function SWMM (f,h,w,i:i; s:s):r;

Returns the length of the string S in mm;

| | |
|---|---|
| f : | Font number |
| h : | Text height in points |
| w : | text thickness: 0=normal 1=bold |
| i : | text style: 0=normal 1=italic |
| s : | text which length will be determined |

Example : `if swmm(4,12,1,0,t)>20: nextrow;`

### procedure ADJSTXT (f,h,w,i:i; wmax:r; s:s; var txt:l);

Divides a text line in several lines so that the individual
line width (in mm) in TXT does not exceed WMAX.

| | |
|---|---|
| f : | Font number |
| h : | Font height |
| w : | Font thickness: 0=Normal; 1=bold; |
| i : | Font style: 0=Normal; 1=Italic; |
| wmax : | Maximum width in mm |
| s : | Text line |
| txt : | List with the result of the separation |

Example : `adjstxt(4,11,0,0,50,s,tx1);`

## 5.11.2.11 Active document

You may build grafic elements in such a way, that they react to a mouse right click.
For this you use the system variables GACTIV, DRAWN, GEPARS and GEMENU.

#### Variables to shape the active document

| | | |
|---|---|---|
| **gactiv** | i | if 1: next grafic element are active;<br>if 2: next elements are not active; |

| | | |
|---|---|---|
| **gemenu** | ls | Right-click menu of an active grafic element;<br>Each string contains: Option-Text:Procedure; (see 6.7.8) |
| **drawn** | i | is 1 when the last draw command was not cliped,<br>that is, the grafic element is visible |
| **gsel** | l | selected grafic in the grafic editor; |

Example :
```
1) drawing the document:

/... openg(...);...
gactiv:=1; /annonces the active element
drawbox(100,-50,200,70);
if drawn: / if the box is visible
  begin
  gepars.nr:s=n1; /builds parameter NR in grafic element
  gemenu.1:s='Process:showproc'; /first menu option
  gemenu.2:s='Terms:term1';      /second menu option
  end;
gactiv:=0; /next elements are not active any longer
/ ..... draw instructions...  editg(...)

2) in the procedure SHOWPROC:

if exobj(ordf,gepars.nr)=0:inf('Order is missing ');
else load(ordf, gepars.nr, of);
/....
```

**procedure SETAREG (xl,yt,xr,yb:r; h:n; /bc:i; p1:s);**

Defines an active area (active region) in the document.

| | |
|---|---|
| xl : | X-left-coordinate of the square area |
| yt : | Y-top-coordinate of the square area |
| xr : | X-right-coordinate of the square area |
| yb : | Y-bottom-coordinate of the square area |
| h : | Handler procedure |
| bc : | background color |
| p1 : | String parameter of the square area |

Example :
```
setareg(12,-20,32,-35,goto1,p1=path);
```

### 5.11.2.12 Grafic components

You can define parameterizable grafic components in the GCM table (Grafic Component Masters) in the resources of any module. Then, the grafic editor offers the possibility to insert and parameterize these predefined grafic components.

Grafic components can have TEKLA commands assigned (right Mouse-click/commands), which are executed when the user clicks them with the left mouse button.

| NR | NAME | PARS | MASK | G |
|---|---|---|---|---|
| FE | Form element | | | |
| FEAZ | External cylinder | d,l:r; | GCFEAZ | x:r=p.l;<br>y:r=p.d/2;<br>draw(gl(0,0,0,y,x,y,x,0)); |
| MB | Mechanical enginee | | | |
| MBTO | Tolerances | form:i=1;<br>tol:s='0.01';<br>ref:s:reference;<br>h:r:height in mm= | GCMBTO | drawmbto; |
| MBWN | Company standards | | | |

#### 5.11.2.13 Grafic buffers

With the command INSGBUF it is possible to attach several grafic symbols to a grafic without drawing them yet. The grafic editor offers then the possibility to insert these symbols one after the other in the grafic G.

```
Example :  /pr.g is a grafic thal will have attached symbols
           insgbuf(pr.g,'E1','Element 1',100,100);
           draws(10,-20,1,'TEXT1 ');
           drawbox(10,-30,30,50);
           insgbuf(pr.g,'E2','Element 2',100,100);
           opengroup;
           draws(40,-20,1,'TEXT2');
           drawbox(40,-30,60,50);
           closegroup;
```

**procedure INSGBUF (var g:l; id,name:s; w,h:r);**

Opens the definition of a symbol grafic buffer for the grafic G.

| | |
|---|---|
| g : | Grafic |
| id : | Symbol identifier (not always necessary) |
| name : | Symbol designation (appears in grafic buffer menu) |
| w : | Width in mm of the surface that contains the symbol; |
| h : | Height in mm of the surface that contains the symbol; |

```
Example :  insgbuf(pr.g,'','Station A1',100,100);
           drawline(2,-5,20,-5);
           draws(2,-10,1,'STATION A1');
```

**procedure OPENGROUP;**

Opens a group of grafics in the active grafic.
All grafic elements that are after drawn with DRAWxxx commands, are stored into this group and thus appear grouped.

```
Example :  opengroup;
           drawline(x1,y1,x2,y2);
           drawline(x3,y3,x4,y4);
           closegroup;
```

**procedure CLOSEGROUP;**

Deactivates the group opened before with OPENGROUP.

### 5.11.3 Flow text documents

#### 5.11.3.1 Document objects and editors

**procedure OPENDOC (var doc:l; nr,name:s; w,h:r; /lm,tm,rm,bm:r);**

Opens the (formatted) text flow document DOC.

| | |
|---|---|
| doc : | Document object |
| nr : | Number of the document object |
| name : | Designation of the document |
| w : | document width in mm |
| h : | document height in mm |

lm : left margin in mm (optional)

tm : top margin in mm (optional)

rm : right margin in mm (optional)

bm : bottom margin in mm (optional)

Example :
```
d1:l;
opendoc(d1,'A1','Report',210,290,lm=20,tm=10);
editdoc(d1);
```

**procedure EDITDOC (var doc:l; /mode:i);**

Opens the document editor for the document DOC.

doc : Document object

mode : Work mode: 1=Text frame 2=Total Area (Default=1)

Example : `editdoc(pr.doc1,mode=2);`

**procedure VIEWDOC (var doc:l; /mode:i);**

Opens the Document viewer for the document DOC.

doc : Document object

mode : Work mode: 1=Text frame (default); 2=Total Area

Example : `viewdoc(pr.doc1,mode=2);`

#### 5.11.3.2 General draw command DOC

**procedure DOC (/)**

Writes in according to the entered parameters.
Each parameter consists of a command (first or first two characters)
and an operand (remaining characters, usualy numbers).

Commands:

X + Number : sets the current column

Y + Number : sets the current paragraph

TF + Number : sets the text font

TH + Number : sets the text size in points

TW + Number : sets the text thickness: 0=normal 1=bold

TS + Number : sets the text style: 0=normal 1=italic

TC + Number : sets the text color

TP + String : sets the paragraph format

'Text' : writes the indicated text

V + Name : writes the content of the indicated variable

F + Number : sets the format = number

Example : `doc(X1,Y1,TF3,TH12,'Title',Y5,TH9,'Date:');`

### 5.11.4 Book-documents

#### 5.11.4.1 Book objects and editors

**procedure OPENBOOK (var doc:l; nr,name:s);**

Opens the (formatted) book DOC.

doc : Book object

nr : Book Ident. NR

name : Book designation

Example :
```
b1:l;
openbook(b1,'A1','Report');
editbook(d1);
```

**procedure EDITBOOK (var doc:l; /zf:r);**

Opens the book editor for the book DOC.

doc : Book object

zf : Initial zoom factor; Default=1.0;

Example : `editbook(report,zf=1.2);`

**procedure VIEWBOOK (var doc:l; /zf:r);**

Opens the book viewer for the book DOC.
Parameters: same as in EDITBOOK.

### 5.12 Files and folders

**function EXFILE (f:s):i;**

Informs whether a file exists or not.
Returns 1 if the file exists and 0 if it does not exist.

F : File path

Example : `if exfile('C:\temp\test.doc')=0: inf('Document missing!');`

**procedure COPYFILE (f1,f2:s);**

Copies a file.

f1 : File to be copied

f2 : Destination path for the file copy

Example : `copyfile('IMPORT.TXT','D:\TEMP\IMPORT03.TXT');`

**procedure RENFILE (f1,f2:s);**

Renames a file.

f1 : File to be renamed.

f2 : New name.

Example : `renfile('C:\IMPORT.TXT','C:\IMPORT2.TXT');`

**procedure DELFILE (f:s);**

Deletes the file F.

Example : `delfile('C:\IMPORT.TXT');`

**procedure CREATEDIR (dir:s);**

Inserts a new folder.

dir : Folder path.

Example : `createdir('C:\MIRAKON\ARCHIVE');`

**procedure REMOVEDIR (dir:s; /o:n);**

Deletes an empty folder.

dir : Folder path.

o : Options (letters sequence):
- E = The folder content is deleted

Example : `removedir('C:\MIRAKON\ARCHIVE',O=E);`

**function EXDIR (dir:s):i;**

Informs whether a folder exists.

dir : Folder path

Example : `if exdir('C:\temp'): inf('Folder found');`

## 5.13 Data bases

In order to store or load objects from a data base you must indicate the data base module. You do this indicating of the data base ID, one dot and the module ID.
Example: save(KNO.A000,obj);
/saves the object OBJ in the module A000 of the knowledge base KNO

Accessing modules of the standard work base DAT (most frequent case) does not need the indication of the data base ID:
Example: save(ART, obj);

### 5.13.1 Handling of objects

**function EXOBJ (wm:n; nr:s; /db:i):i;**

Informs whether an object is present in a data base.
Returns = 1 if exists or 0 if does not exist.

wm : Data base module

nr : Object number

db : Number of an external data base (opened with OPENDB)

Example :
```
if exobj(ART,'123')=0: inf('Product 123 does not exist');
if exobj(KNO.A000,'RS')=0: inf('Table RS does not exist');
```

**procedure LOAD (wm:n; key:s; var wobj:l; /o,st0:n; max,db:i);**

Loads an object from a data base into a variable.

wm : Data base module

key : Object number

wobj : Variable that will contain the loaded object

o : options: Letters sequence with the following meaning:
- C: Creates a new object if NR does not exist;
- N: Formats the object again if found;

- U: Loads only if the object is not unlocked;
- L: Locks the object after loading it;

st0 : Procedure with the definition of the initial structure;

max : Number of object fields to be loaded;

db : Number of the external data base (opened with OPENDB)

Example :
```
load(ART,'123',a);
load(ARC02.ART,nr1,a1,o=C);
```

**procedure SAVE (wm:n; var obj:l; /i,db,compress:i);**

Saves an object in a data base.

wm : Data base module

obj : Object to be saved

i : if 1: informs if the data base is locked.

db : Number of an external data base (opened with OPENDB)

compress : compress=1 The object is compressed before the save.

Example :
```
save(ART,a);
```

**procedure LOADE (wm:n; key,varname:s; varid:n; /db:i);**

Loads a field of an object from a data base into a variable.
This field must be on the first hierarchic level of the object.

wm : Data base module

key : Object number

varname : Name of the field to be loaded.

varid : Variable that will contain the loaded object field

db : number of an external data base (opened with OPENDB)

Example :
```
cnr:s;
loade(ART,'123','CLIENTNR',cnr);
```

**procedure SAVEE (wm:n; key,varname:s; varid:n; /db:i);**

Changes the value of the field VARNAME of the object KEY with the value contained in VARID.
This field must be on the first hierarchic level of the object.

wm : Data base module

key : Object number

varname : Name of the field to be changed.

varid : Variable that contains the value.

db : number of an external data base (opened with OPENDB)

Example :
```
cnr:s='C025';
savee(ART,'123','CLIENTNR',cnr);
```

**procedure DELWOBJ (wm:n; nr:s; /db:i);**

Deletes one or several objects from a data base.

wm : Data base module

nr : Number of the object to be deleted. This can also be a filter, for the deletion of several objects (see example).

db : Number of an external data base (opened with OPENDB)

```
Example :  delwobj(ART,'123'); /deletes the object '123' from ART
           delwobj(ART,'M_');  /deletes all objects that start with M
```

### function OBJNAME (wm:n; nr:s; /db:i):s;

Returns the name of an object from a data base without loading him.
Advantage: Speed.

wm : Data base module

nr : Object number

db : Number of an external data base (opened with OPENDB)

```
Example :  t:s=objname(ART,a.nr);
```

### function OBJINF (wm:n; nr:s; /db:i):s;

Returns the characteristic of an object from a data base without loading it.
Advantage: Speed. The characteristic is a string that is identified with
INF and is always at the 4th position in the object structure.

wm : Data base module

nr : Object number

db : Number of an external data base (opened with OPENDB)

```
Example :  ch:s=objinf(ART,a.nr);
           if poss('V',ch,1)>0: inf('validated');
```

### procedure PUTOBJINF (var obj:l; inf:s);

Inserts the characteristic INF in the object OBJ.
If the object has already a characteristic, it is replaced.
See also function OBJINF.

```
Example :  putobjinf(pr,'A4,V,X=20');
```

### procedure FMTOBJ (var obj:l; nr,name:s; st0:n);

Formats an object with an initial structure.
The procedure FMTWOBJ was used in earlier versions and is only
implemented for compatibility reasons (not recommended).

obj : Object to be formatted

nr : Object number

name : Object name

st0 : Identifier of the procedure that contains the initial structure

```
Example :  fmtobj(product,'123','Screw',ARTST0);
```

### procedure LOCKOBJ (wm:n; nr:s; /db:i);

Locks an object for all the other users. If it is already locked,
the procedure is not executed and the system variable DBRES gets the value 2.

wm : Data base module

nr : Object number

db : Number of an external data base (opened with OPENDB)

```
Example :  lockobj(ART,'A1'); /locks the  Object A1
           if dbres>0:inf('Object is already locked');
```

### procedure UNLOCKOBJ (wm:n; nr:s; /db:i);

Unlocks an object for all the other users. If it is already locked
for another user , the procedure is not executed and the system
variable DBRES gets the value 2.

wm : Data base module

nr : Object number

db : Number of an external data base (opened with OPENDB)

Example :
```
unlockobj(ART,'A1'); /unlocks the object A1
```

### function OBJLOCKED (wm:n; nr:s; /db:i):i;

Informs whether an object is locked (Returns 1).

wm : Data base module

nr : Object number

db : Number of an external data base (opened with OPENDB)

Example :
```
if objlocked(ART,nr)=1: inf('Object locked');
```

### procedure GENKEY (var key:s; wm:n; prefix:s; ndigits:i; /db:i; startnr:a; wm2:n);

Generates an object number that does not yet exists. This number consists
of a prefix and a sequencial number.
Functioning mode: The sequencial number is incremented, starting from 0;
for each number, it is examined whether an object with that number already exists;

key : Variable that will contain the generated number

wm : data base module

prefix : prefix of the object number

ndigits : number of digits of the generated sequence

db : number of an external data base (opened with OPENDB)

startnr : starting number, if it is not 0

wm2 : second data base module that should also be consulted;
Example: archives

Example :
```
/assuming that the objects A01 and A03 already exist
genkey(k,PROD,'A',2); / returns K = 'A02'
```

### procedure SETSAVE (objid:n; dm:n);

Assigns a new data base module to an object.
Purpose: If the user switches between different work modules
the object to be saved must be rerouted.

objid : Variable that contains the object

dm : New data base module

Example :
```
pr:l&save(PROD)
moduleid:s='PROD';
/after moduleid is changed ...
setsave(pr,[moduleid]);
```

### function WBL (wm:n; form:i; keys:s; /db:i):ls;

Returns a list of numbers and/or names of the objects inside a data base module.

wm : Data base module

form : 1: only numbers 2: only names 3: Numbers+Names

keys : Filter for the desired numbers

db : Number of an external data base (opened with OPENDB)

Example : `l1:l=wbl(PROD,3,'A_,B_');`

### function WBML (db:i):l;

Returns a list of numbers and/or names of the objects inside a data base module.

db : Data base module

Example : `l1:l=wbl(PROD,3,'A_,B_');`

### procedure FILTERWB (wm:n; var kl:l; kpat:s; /f:n; h,db:i);

Filters all object numbers from the work base module WM with the filter KPAT
and returns them in the list KL. With F you can indicate a function
to examine further conditions. This function should be of the type REAL,
if it does not return 0, the respective object number is returned in KL.
By setting the variables SORTID1 and SORTID2 (strings) in the function F,
the list KL is multi-level sorted.

wm : Data base module

kl : List to return the filtered object numbers

kpat : Object number filter

f : Filter function

h : Number of sort stages (maximum 2)

db : Number of an external data base (opened with OPENDB)

Example 1:
```
keys:l;
filterwb(PROD,keys,'A_',f=C7);  /without sorting

funktion C7:i;
  c7:=0;
  load(PROD,keyi,pr);
  if pr.lenght>200:c7:=1;
  end;
```
Example 2:
```
filterwb(PERSONAL,keys,'_',f=C8,h=1); /1-stage sorting

function C8:i;
  c8:=1;
  load(PERSONAL,keyi,pers);
  sortid1:=pers.name;  /Sort criteria=name of the person
  end;

possible result:

  KEYS:list
  |
  +-- MEIER PETER:s='7803';
  +-- MÜLLER HANS:s='3354';
  +-- WILLIAMS JOHN:s='5072';
```
Example 3:
```
filterwb(PROD,keys,'_',f=C9,h=2); /2-stages sorting

Funktion C9:i;
  c9:=1;
  load(PROD,keyi,pr);
  sortid1:=pr.client; /1st. Sort criteria=Client Name
```

```
  sortid2:=pr.name;   /2nd. Sort criteria=Product name
  end;

possible result:

  KEYS:list
  |
  +- BMW:list;     1st. Client
  |    |
  |    +- DRIVE:s='G870012';
  |    +- SHAFT:s='A201456.1';
  |
  +- MERCEDES:list; 2nd. Client
  |    |
  |    +- BOLT:s='A048765B';
  |    +- SHAFT:s='A338945.0';
```

**procedure SECOBJ (var obj:l; level,cond:i);**

Secures an object against certain operations by certain users.

obj : Object that will be secured

level : safety stage:
1 = allows to load and see/change the object, but not
to save, delete or rename it;
2 = does not allow to see, edit, copy, export or print the object
3 = does not allow to load (i.e. use) the object.

cond : Protection condition:
1 = protection against all users with unequal password
2 = protection against all users with unequal status
3 = protection against all users with unequal user ID

Example : `secobj(pr,1,1);`

## 5.13.2 Handling of data bases

**procedure MAKEDB (file:s);**

Produces a new Mirakon data base (Dat or KNO File).
This is only necessary if the data base is not defined in the configuration.

file : File (path + name)

**procedure OPENDB (var db:i; file:s; /accmode,compress:i);**

Opens a Mirakon data base (DAT or KNO File) and returns the assigned
data base number in the variable DB.
This is only necessary if the data base is not defined in the configuration.

db : Variable that receives the data base number

file : File (path + name)

accmode : 1=Direct; 2=Through server (only for Client\Server version)

compress : compress=1: compress the object before saving it in the data base

Example : `archive:i; opendb(archive,'c:\arc\DATA02.DAT');`

**procedure CLOSEDB (db:i);**

Closes the data base indicated by DB.

db : Data base number (in accordance with OPENDB command)

Example :
```
closedb(db1);
```

**procedure LOCKWB (wb:n; /i:i);**

Locks the work base WB for all other users.
If it is already locked the procedure is not executed
and the system variable DBRES gets the value 1.

wb : Data base ID

i : with I=0 information about a possible failure is suppressed.
The default is I=1

Example :
```
lockwb(PROD,i=0);
if dbres>0:inf('Products file is locked');
```

**procedure UNLOCKWB (wb:n; /i:i);**

Unlocks the work base WB for all the other users.
If it is already unlocked the procedure is not executed
and the system variable DBRES gets the value 1.

wb : Data base ID.

i : with I=0 information about a possible failure is suppressed.
The default is I=1

Example :
```
unlockwb(PROD);
```

**function WBLOCKED (wb:n):i;**

Returns 1 if the data base is locked and 0 if it is unlocked.

wb : Data base ID.

Example :
```
if wblocked(PROD)=1: inf('Products file is locked');
```

**function EXWMOD (id:s; /db:i):i;**

Informs whether a module exists in a data base.

id : Module ID.

db : Number of an external data base (opened with OPENDB)

Example :
```
if exwmod('PROD')=0: inf('Module PROD does not exist');
```

**procedure INSWMOD (id,name:s; /p,db:i);**

Inserts a module into a data base, if it does not already exist.

id : Module ID

name : Module designation

p : Position in the modules list;

db : Number of an external data base (opened with OPENDB)

Example :
```
inswmod('PROD','Products',p=3);
```

### procedure DELWMOD (id:s; /db:i);

Deletes a module from a data base, if it exists.

id : Module ID

db : Number of an external data base (opened with OPENDB)

Example : `delwmod('PROD');`

### procedure REBUILDDB (/file:s; id:n; refnr:i);

Rebuilds a data base.

file : Data base file name (alternative to the ID)

id : Data base ID in accordance with the configuration.

refnr : if 1, internal reference numbers are rebuilded (default=0)

Example :
```
rebuilddb(id=PROD);
rebuilddb(file='C:\ARCHIVE\PROD.DAT',refnr=1);
```

### function NKEYS (wm:n):a;

WM :

### procedure COPYDB (id,dir:s; /fname:s);

Opens a Mirakon data base (DAT or KNO File) and returns the assigned
data base number in the variable DB.
This is only necessary if the data base is not defined in the configuration.

id : Variable that receives the data base number

dir : File (path + name)

fname : 1=Direct; 2=Through server (only for Client\Server version)

Example : `archive:i; opendb(archive,'c:\arc\DATA02.DAT');`

## 5.13.3 Interactive access to data bases

#### Variables for the data base access with LOADER

| | | |
|---|---|---|
| **dbok** | i | Permission to load/delete/insert |
| **dbkey** | s | Number of the selected object |
| **dbname** | s | Name of the selected object |
| **dbkey0** | s | Default object NR for LOADER with the option A. |
| **dbobjid** | s | Determines the object line in LOADER |
| **dbobjcolor** | i | Color of the object line in LOADER. |
| **idxnr** | s | Index number of the loaded object. |
| **idxname** | s | Index name of the loaded object |
| **idxtab** | l | Index table that was last used . |
| **lmask** | i | Switches for the LOADER Dialogue (default=1):<br>when lmask=0, the standard dialogue for insert, copy<br>or rename are replaced by your own dialogue procedures.<br>After the dialogue, you must communicate to Mirakon<br>how the object is to be numbered and designated using the<br>system variables DBKEY and DBNAME. At the end of the dialogue,<br>if DBOK=1, Mirakon continues the LOADER procedure (insert,<br>copy or rename). See LOADER procedure, exampl 1, at L5:... |

**Situations for the data bases access**

During the access to data bases with the LOADER-command, following situations can be used:

| | |
|---|---|
| **L1** | Before loading an object |
| **L2** | After loading an object |
| **L3** | Before the deletion of an object |
| **L4** | After the deletion of an object |
| **L5** | Before the insertion dialogue |
| **L6** | After closing the loader |
| **L7** | Before renaming an object |
| **L8** | After renaming an object |
| **L9** | Before copying an object |
| **L10** | After copying an object |
| **ARC** | When archiving |
| **RST** | When restoring |

### procedure LOADER (wm,hp:n;var obj:l; /o,sp,st0,idx,flt,st:n;gf,of,ti,b:s;db:i);

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

wm : Data base module

hp : Handler procedure: this procedure defines a specific behaviour in the different loader situations L1, L2, L3, etc..
By setting DBOK:=0 the loading or the deletion are prevented.

obj : Variable where the loaded object is returned

o : Options: Sequence of letters with the following meaning:

A: Automatic allocation of numbers. When the user wants to insert a product, the number field of the insertion dialogue will contain the content of the variable DBKEY0.

G + number: With button "Generate" for automatic allocation of numbers. When this button is pressed, the program looks for a new number, consisting of the existing prefix in the number field and the smallest number sequence with as many digits as the number in the G option. This new number appears in the number field.

J: Automatic jump to the insert mask, without the user having to press the INSERT key/button.

H + number: Certain standard buttons do not appear:
H1: INSERT button (i.e. the user cannot insert an object)
H2: DELETE button (i.e. the user cannot delete an object)
H3: RENAME button (i.e. the user cannot rename an object)
H4: COPY button (i.e. the user cannot copy an object)
H6: FILTER button (i.e. the user cannot filter)
H7: LOAD button (i.e. the user cannot load an object)
H0: All standard buttons (H1 to H7)

M: Automatic jump to the ok button in the insert mask,

without the user having to press the enter key twice.

O: With ARCHIVE-button.

P: With RESTORE-button.

Q: Enables export/import of external objects (OBJ-Files)

R: Keeps the last position in group index.

S: Allows the input of the object number.

T: With TRANSFER-button,allowing objects to be transfered between 2 data bases.

U: Object will not be locked after beeing loaded.

X: The index prefix does not appears in the number input field but in a separate display field.

W + number: Width of the index menu (default = 28). If you enter W0, the index does not appear.

Z: Index menu without the option "ALL"

sp : String format procedure allows formatting the object list (see example 4)

st0 : Procedure with the definition of the initial structure

idx : Table with data base module group index

flt : Table with the filter definitions

st : Sorting table

gf : Group filter for limiting the group index

of : Object filter for limiting the object list

ti : Title for the loader window.

b : Optional buttons:
Syntax: Button_name1:Procedure1, Button_name2:Procedure2,...

db : Number of an external data base (opened with OPENDB)

Example 1:
```
loader(PROD,hp1,pr);

procedure HP1;
at L2:
  prname:=pr.nr+' '+pr.name;
  dspsf(D1);
at L3:
  if pr.status=2:
    begin
    inf('Deletion not allowed');
    dbok:=0;
    end;
at L5:  /before the insertion of a new object
  lmask:=0;
  ournr,ourprefix,ourname:s='';
  mask(NEWPRODUCT,o=A);
  if escaped: dbok:=0&exit;
  failure:i=0;
```

```
            control(ournr,ourprefix,failure);
            if failure: dbok:=0&exit;
            dbkey:=ourprefix+'.'+ournr;
            dbname:=ourname;
            dbok:=1;
```

Example 2:
```
dbkey0:='A-'+today;
loader(PROD,hp1,pr,o=AJG3);
```
Example 3:
```
loader(ORD,hp2,order,o=W0,b1='_Terms');

procedure HP2;
at LB1:
load(ORD,dbkey,order);
mask(TERMS);
save(ORD,order);
```
Example 4:
```
loader(PROD,lp,pr,SP=fprod);

procedure FPROD;
  obj:l;
  load(PROD,dbkey,obj);
  dbobjid:=obj.nr+' '+obj.client;
  /or:  dbobjid:=sf(Vdbkey,M15,Vdbname);
  end;
```
Example 5:
```
loader(PROD,lp,pr,gf='A_,B_');
/only the groups starting by A or B appear
```

### function LASTOBJS (wm:n; / db:i);

Returns the list of the last loaded objects from module WM.

wm : Data base module

db : Number of an external data base (opened with OPENDB)

Example :
```
nr:s;
options:l=lastobjs(prod);
selid(nr,options,'Last loaded products',20,-5,20,6);
load(prod,nr,pr);
```

## 5.13.3.1 Table of contents for LOADER

**Data base access with table of contents**
The access to big data bases can be much more friendly if you use an indexed table of contents.

Procedure:

- create the table of contents (also called group index).
- Call the Loader with optional parameter IDX=Table ID

Example: Table of contents for a product data base module:

| NR | NAME |
|---|---|
| IP | Single parts |
| IPC | Casting parts |
| IPCB | Big casting parts |
| IPCS | Small casting pars |
| IPR | Rotation parts |
| I | (all single parts) |
| CG | Constructive groups |

The hierarchical grouping in the table of contents is achieved by the indentation of the numbers in the NR column, and helps the user to find his object as fast as possible, without knowing the object number. The group number in the NR column must be the prefix of the object number, only then the object is contained in the group.

Example: The group IPCG contains the objects IPCG001, IPCG07A, IPCGTEST, etc..

### 5.13.3.2 Initial structures in LOADER

If an object is inserted in a work base, Mirakon must know its initial structure. This is described in form of a procedure and communicated to the LOADER.

This procedure contains declarations of variables (object fields) that are attached to the object list when a new object is created .

Example :

```
/Procedure CLIST0, the initial structure of a client:

adr:l;     / address
tel:s;     / telephone
pcz:s;     / postal and zip code
rem:l;     / remarks
evs:l;     / list of events
chr:l;     / list of characteristics
cnv:r;     / conversion


/During the application, the list of the fields EVS and CHR
/can be filled with new elements as well as new
/fields can be inserted into the object structure.
/The initial structure is only a "skeleton"
/that is filled with data during the applications.
```

### 5.13.3.3 Structure of filters in LOADER

The friendly search of objects in an extensive work base can be made with user defined filters.

**Procedure:**

**1** LOADER must be called with the optional parameter FLT (filter table);
Example: loader(PROD,LP,pr,flt=PRODFLT);

**2.** Create global filter criteria variables (in the INIT procedure);
E.G.: fltnr:s:filter for product number='_';
fltname:s:filter for product name='_';
fltcli:filter for client='_';

**3.** Create a Filter table with the columns NAME, MASK and A:
The filter table contains a filter per row;
Each filter can call a dialogue (dialogue mask in MASK column);
The A-column contains the Tekla commands that decide whether an object if filtered or not.

| NR | NAME | MASK | A |
|---|---|---|---|
| F1 | General filter | MF1 | if o.nr out (fltnr):exit;<br>if o.name out (fltname):exit;<br>load(PROD,o.nr,o);<br>ok:=o.ad.client in (fltcli); |
| F2 | Technical Data | MF2 | ..... |

Create the dialogue masks, at best beside the filter table;
Here you ask the values for the different filter criteria,
e.g.: Product number filter in FLTNR, product name filter in
FLTNAME and client filter in FLTCLI;

Function mode:

1. The user presses the button FILTER (during the loader execution);
2. Mirakon opens the filter menu in accordance with the filter table;
3. The user selects one filter;
4. Mirakon opens the assigned dialogue mask;
5. The user answers to the questions about the filter criteria and presses OK;
6. Mirakon starts the filter process:
   It loads the first 4 fields of each object in the variable O.
   The variable OK is set to 0;
   The commands in the A-column are executed;
   If the variable OK = 0, the object is filtered;
   If a filter criteria needs further object fields, the complete object must be loaded using the LOAD-command.
7. As a result of this process, only the objects that pass the filter criteria appear in the LOADER-list.

## 5.13.4 Fast access to objects data

Fast access variables are object "key" variables, normally used to filter, search and/or sort objects, so they need to be fast acceded, otherwise, the filter, search and/or sort process will be slow.

To use fast access variables, the option Fast Access Variables in a collection status (in the data base editor), must be checked, and the fast access variables table columns must be defined. Each column corresponds to an object variable. The access to the fast access variables table is made by right click in the collection, and choosing the option fast access variables.

To further optimizations, the fast access variables can be splitted in several tables. To do so, the conditions for splitting must be defined, by pressing the button "conditions" in the table editor, and filling the conditions table.

### procedure FINDINFT(wm:n; var colids,exprs,rowids:l; /mode,sort,msg,db:i; rows,sep:s)

Returns in ROWIDS the rows that fulfil the expression in EXPRS for each column in COLIDS.

| | |
|---|---|
| wm : | Data base module |
| colids : | Columns to be filtered.<br>Eg. colids:ls=('PRICE','SUPPLIER'); |
| exprs : | Expression to use for each column<br>Eg.: Exprs:ls=('rs(tc.1)>100;','tc.1='S001';'); |
| rowids : | Rows that had satisfied the expressions |
| mode : | 1=AND (all columns must fulfill the expressions)<br>0=OR (only one column must fulfill the expression) |
| sort : | 1=Sort, 0=Without sort |
| msg : | 1= shows a message for each row (in the bottom-right corner) |

db : Number of an external data base (opened with OPENDB)

rows : Columns to be returned in ROWIDS. If not indicated, only the column NR will be returned.
Eg.: rows='NR|PRICE|SUPPLIER';

sep : Separator to be used when to return more than the column NR.
Used only when parameter ROWS is indicated. (default=#)

asle :

asls :

elid :

leid :

acid :

Example :

```
colids:ls=('PRICE');
exprs:ls=('rs(tc.1)>100');
rows:s='SUPPLIER|PRICE|NR';
result:l;
findinft(PRODS,colids,exprs,result,sort=1,rows=rows);

/ returns the supplier, price and object nr for all object
/ were the price is bigger than 100, sorted by supplier+price+nr
```

| NR | PRICE | SUPPLIER | QUANTITY |
|---|---|---|---|
| P01 | 100 | S001 | 90 |
| P02 | 120 | S001 | 32 |
| P03 | 50 | S002 | 4 |
| P04 | 200 | S003 | 66 |

## 5.13.5

Fast access variables are object "key" variables, normally used to filter, search and/or sort objects, so they need to be fast acceded, otherwise, the filter, search and/or sort process will be slow.

To use fast access variables, the option Fast Access Variables in a collection status (in the data base editor), must be checked, and the fast access variables table columns must be defined. Each column corresponds to an object variable. The access to the fast access variables table is made by right click in the collection, and choosing the option fast access variables.

To further optimizations, the fast access variables can be splitted in several tables. To do so, the conditions for splitting must be defined, by pressing the button "conditions" in the table editor, and filling the conditions table.

### procedure OPENQUERYTABLE (var qth:i; modid:n; tabid:s);

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

qth :

modid : Data base module

tabid : Optional buttons:
Syntax: Button_name1:Procedure1, Button_name2:Procedure2,...

Example 1: `loader(PROD,hp1,pr);`

```
procedure HP1;
at L2:
  prname:=pr.nr+' '+pr.name;
  dspsf(D1);
at L3:
  if pr.status=2:
    begin
    inf('Deletion not allowed');
    dbok:=0;
    end;
at L5:  /before the insertion of a new object
  lmask:=0;
  ournr,ourprefix,ourname:s='';
  mask(NEWPRODUCT,o=A);
  if escaped: dbok:=0&exit;
  failure:i=0;
  control(ournr,ourprefix,failure);
  if failure: dbok:=0&exit;
  dbkey:=ourprefix+'.'+ournr;
  dbname:=ourname;
  dbok:=1;
```

### procedure NEWQUERYTABLE (var qth:i; tabid:s);

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

qth :

tabid : Optional buttons:
Syntax: Button_name1:Procedure1, Button_name2:Procedure2,...

Example 1:
```
loader(PROD,hp1,pr);

procedure HP1;
at L2:
  prname:=pr.nr+' '+pr.name;
  dspsf(D1);
at L3:
  if pr.status=2:
    begin
    inf('Deletion not allowed');
    dbok:=0;
    end;
at L5:  /before the insertion of a new object
  lmask:=0;
  ournr,ourprefix,ourname:s='';
  mask(NEWPRODUCT,o=A);
  if escaped: dbok:=0&exit;
  failure:i=0;
  control(ournr,ourprefix,failure);
  if failure: dbok:=0&exit;
  dbkey:=ourprefix+'.'+ournr;
  dbname:=ourname;
  dbok:=1;
```

### procedure CLOSEQUERYTABLE (qth:i);

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

qth :

Example 1:
```
loader(PROD,hp1,pr);

procedure HP1;
at L2:
  prname:=pr.nr+' '+pr.name;
  dspsf(D1);
at L3:
  if pr.status=2:
    begin
    inf('Deletion not allowed');
    dbok:=0;
    end;
at L5:  /before the insertion of a new object
  lmask:=0;
  ournr,ourprefix,ourname:s='';
  mask(NEWPRODUCT,o=A);
  if escaped: dbok:=0&exit;
  failure:i=0;
  control(ournr,ourprefix,failure);
  if failure: dbok:=0&exit;
  dbkey:=ourprefix+'.'+ournr;
  dbname:=ourname;
```

```
  dbok:=1;
```

### function NQTROWS (qth:i):a;

Returns the list of the last loaded objects from module WM.

qth : Number of an external data base (opened with OPENDB)

### function NQTCOLS (qth:i):i;

Returns the list of the last loaded objects from module WM.

qth : Number of an external data base (opened with OPENDB)

### procedure GETQUERYTABLE (qth1,qth2:i; colids,filter:ls; keys:s);

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

qth1 :

qth2 :

colids : Data base module

filter : Optional buttons:
Syntax: Button_name1:Procedure1, Button_name2:Procedure2,...

keys :

Example 1: `loader(PROD,hp1,pr);`

```
procedure HP1;
at L2:
  prname:=pr.nr+' '+pr.name;
  dspsf(D1);
at L3:
  if pr.status=2:
    begin
    inf('Deletion not allowed');
    dbok:=0;
    end;
at L5:  /before the insertion of a new object
  lmask:=0;
  ournr,ourprefix,ourname:s='';
  mask(NEWPRODUCT,o=A);
  if escaped: dbok:=0&exit;
  failure:i=0;
  control(ournr,ourprefix,failure);
  if failure: dbok:=0&exit;
  dbkey:=ourprefix+'.'+ournr;
  dbname:=ourname;
  dbok:=1;
```

### procedure VIEWQT (qth:i);

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

qth :

### procedure EXPQT2TXT (qth:i; var colids:ls; var txt:l; sep:s);

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

qth :

colids : Data base module

txt : Optional buttons:
Syntax: Button_name1:Procedure1, Button_name2:Procedure2,...

sep :

### procedure EXPQT2ST (qth:i; var colids:ls; var st:l; / grpid,grpenr:s; nss:i);

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

qth :

colids : Data base module

st : Optional buttons:
Syntax: Button_name1:Procedure1, Button_name2:Procedure2,...

grpid :

grpenr :

nss :

### procedure EXPQT2XLS (qth:i; var colids:ls; xlc,xlr:i);

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

qth :

colids : Data base module

xlc : Optional buttons:
Syntax: Button_name1:Procedure1, Button_name2:Procedure2,...

xlr :

Example 1: `loader(PROD,hp1,pr);`

```
procedure HP1;
at L2:
  prname:=pr.nr+' '+pr.name;
  dspsf(D1);
at L3:
  if pr.status=2:
    begin
    inf('Deletion not allowed');
    dbok:=0;
    end;
at L5:  /before the insertion of a new object
  lmask:=0;
  ournr,ourprefix,ourname:s='';
  mask(NEWPRODUCT,o=A);
```

```
if escaped: dbok:=0&exit;
failure:i=0;
control(ournr,ourprefix,failure);
if failure: dbok:=0&exit;
dbkey:=ourprefix+'.'+ournr;
dbname:=ourname;
dbok:=1;
```

**procedure EXPLORER (wm:n; var obj:l; /colst,st0,idx,o,hp:n; ti,b:s; us:i);**

Opens the loader dialogue so the the user can select and load an object from a data base into a variable. The loader allows also creating, deleting, renaming and copying of objects.

wm : Data base module

obj :

hp : Handler procedure: this procedure defines a specific behaviour in the different loader situations L1, L2, L3, etc..
By setting DBOK:=0 the loading or the deletion are prevented.

o : Options: Sequence of letters with the following meaning:

A: Automatic allocation of numbers. When the user wants to insert a product, the number field of the insertion dialogue will contain the content of the variable DBKEY0.

G + number: With button "Generate" for automatic allocation of numbers. When this button is pressed, the program looks for a new number, consisting of the existing prefix in the number field and the smallest number sequence with as many digits as the number in the G option. This new number appears in the number field.

J: Automatic jump to the insert mask, without the user having to press the INSERT key/button.

H + number: Certain standard buttons do not appear:
H1: INSERT button (i.e. the user cannot insert an object)
H2: DELETE button (i.e. the user cannot delete an object)
H3: RENAME button (i.e. the user cannot rename an object)
H4: COPY button (i.e. the user cannot copy an object)
H6: FILTER button (i.e. the user cannot filter)
H7: LOAD button (i.e. the user cannot load an object)
H0: All standard buttons (H1 to H7)

M: Automatic jump to the ok button in the insert mask, without the user having to press the enter key twice.

O: With ARCHIVE-button.

P: With RESTORE-button.

Q: Enables export/import of external objects (OBJ-Files)

R: Keeps the last position in group index.

S: Allows the input of the object number.

T: With TRANSFER-button,allowing objects to be transfered between 2 data bases.

U: Object will not be locked after beeing loaded.

X: The index prefix does not appears in the number input field but in a separate display field.

W + number: Width of the index menu (default = 28). If you enter W0, the index does not appear.

Z: Index menu without the option "ALL"

st0 : Procedure with the definition of the initial structure

idx : Table with data base module group index

colst : Sorting table

ti : Title for the loader window.

b : Optional buttons:
Syntax: Button_name1:Procedure1, Button_name2:Procedure2,...

us :

Example 1: `loader(PROD,hp1,pr);`

```
procedure HP1;
at L2:
  prname:=pr.nr+' '+pr.name;
  dspsf(D1);
at L3:
  if pr.status=2:
    begin
    inf('Deletion not allowed');
    dbok:=0;
    end;
at L5:  /before the insertion of a new object
  lmask:=0;
  ournr,ourprefix,ourname:s='';
  mask(NEWPRODUCT,o=A);
  if escaped: dbok:=0&exit;
  failure:i=0;
  control(ournr,ourprefix,failure);
  if failure: dbok:=0&exit;
  dbkey:=ourprefix+'.'+ournr;
  dbname:=ourname;
  dbok:=1;
```

| NR | PRICE | SUPPLIER | QUANTITY |
|---|---|---|---|
| P01 | 100 | S001 | 90 |
| P02 | 120 | S001 | 32 |
| P03 | 50 | S002 | 4 |
| P04 | 200 | S003 | 66 |

### procedure QUERY (var sql,result:l; /qt,qtdest:i);

Sends an SQL inquiry string (CODE) and receives the result in RES.

SQL :

RESULT :

QTDEST :

QT :

Example :

```
result:l;
sqlodbc('SELECT NAME, AGE FROM PERSONAL',result);
```

## 5.14 Dialog control

**Variables for dialog control**

| | | |
|---|---|---|
| **fid** | s | identifier of the selected field |
| **stageid** | i | identifier of the current dialogue window<br>(assigned by the author in EDITG, EDITST or EDITPLAN); |
| **fchanged** | i | this variable informs whether the deselected dialog field<br>was changed (fchanged=1) or not (fchanged=0) |
| **jumpfok** | i | setting this variable to 0, in the dialog field commands,<br>causes the cursor to remain in the same field |
| **okayed** | i | becomes 1 if the user presses the OK button leaving a<br>dialogue mask, otherwise OKAYED remains 0 |
| **escaped** | i | becomes 1 if the user abandons a dialogue mask with the ESC-key<br>otherwise ESCAPED remains 0 |
| **exitok** | i | when set to 0, in the dialogue finalisation, prevents<br>Mirakon from closing the dialogue mask (default = 1) |
| **xlfa** | r | x-left-coordinate of the last selected field |
| **xrfa** | r | x-right-coordinate of the last selected field |
| **ytfa** | r | y-top-coordinate of the last selected field |
| **ybfa** | r | y-bottom-coordinate of the last selected field |
| **infmode** | i | 1 = informations appear interrupting the application<br>2 = informations are written in INFL and the application is<br>not interrupted |
| **infl** | l | information container when INFMODE=2 |

### 5.14.1 Cursor, keyboard, speaker

**procedure SETCURSOR (mode:i);**

Changes the cursor form.

mode : Cursor form: 1=normal 2=Hourglass

Example : `setcursor(2); dosomething; setcursor(1);`

**procedure BEEP;**

Makes a BEEP sound over the PC speaker.

**function INPUTKEY:i;**

Returns the value of the last key pressed (ANSI-Code).

Key values:
Back=8, Tab=9, Home=371, End=379, PgUp=373, PgDn=381, Enter=13, Esc=27,
Left=375, Del=383, Right=377, Ins=382, BTab=315, Up=372, Down=380,
F1..F10=359..368, Shift-F1..Shift-F10=384..393

Example : `if inputkey=27:exit;`

## 5.14.2 Showing informations

### procedure OPENMSG (s1,s2:s; /cm,wait:i);

Opens a message box with two text lines s1 and s2 on the right bottom corner of the screen.

s1 : first line

s2 : second line

cm : changes the cursor form: 1:normal 2:hourglass

wait : waits 1/100-seconds

Example :
```
openmsg('Document is to be build',cm=2,wait=50);
```

### procedure CLOSEMSG;

Closes a message box.

### procedure INF (s:s; /z,pos:i);

Shows an information box with the text S and an OK button.

s : Information text

z : With Z you can use a standard Sufix :
1 = ' does not exist '
2 = ' already exist!'
3 = ' not valid!'
4 = ' not compatible!'
5 = ' not allowed!'

pos : box position: 1=top left, 2=bottom right (default) 3=center

Example :
```
x:r=a+b;  pnr:s='123456';
inf('X='+sr(x,2));
if x>1200:inf('Weight to high !');
inf('Product '+pnr,z=1);
```

### procedure INFV (v:n; /bars,pos:i; w,h:r);

Informs the value of the indicated variable V.
If bars=1 is indicated, the storage space (in bytes) is shown as a grafic bar, for all sub-positions.

v : Variable name

bars : bars=1 shows a diagram of the memory consumption of the individual positions

pos : infv position: 1=top left, 2=bottom right (default) 3=center

w : window width in text characters

h : window height in text lines

Example :
```
x:r=a+b; infv(x);
load(mod1,'123',obj);
infv(obj,bars=1,w=50,h=20);
```

### procedure OPENREPORT (t:s; /w,h:r);

Opens a report window where you can write text lines using the procedure REPORT.

t : Window title

w : Window width in text units

h : Window height in text units

Example : `openreport('Protocol');`

**procedure REPORT (dr,c:i; s:s; /w:i);**

Writes a text line in the opened report window.

dr : Line jump, related to the last written line

c : Column position

s : Text line

w : Waiting period in 0,01 seconds

Example : `report(1,3,'Product '+p.nr,w=50);`

**procedure CLOSEREPORT;**

Closes the opened report window.

## 5.14.3 Handling of dialog fields

**procedure DSPF (/);**

Redraws the indicated dialog fields of the current dialogue mask.

Example : `dspf(A1,A2,B1,C1);`

**procedure DSPSF (ID:N);**

Redraws the indicated dialog field of the status surface.

ID : Field identifier

Example : `dspf(D1);`

**procedure SELF (ID:N);**

Selects the indicated dialog field, cause the cursor to jump there.
The standard fields OK and ABANDON can be selected with the id ZOK and/or ZESC.

ID : Field identifier

Example :
**procedure JUMPF (J:I);**

Jumps to another field.

J : Number of the destiny field

Example :
```
jumpf(1);    /jumps to next field
jumpf(-1);   /jumps to the previous field
```

**procedure DELF (id:n);**

Deletes the indicated dialog field or grafic element.
ID can also be a filter.

id : Identifier or filter

Example : `delf(A1); delf(B_);`

**procedure SETF (id:n; stat:i; /bc,fc:i);**

Changes the status of the dialog fields or grafic elements in the current dialogue. ID can also be a filter.

id : Field identifier or filter

stat : Status value:
1=normal;
2=locked: appears in grey and cannot be selected;
4=invisible: does not appear;

bc :

fc :

Example :
```
setf(A_,2);
/all field with ID starting with A are locked
```

**procedure SETFCOL (id:n; bcol,fcol:i; /pbcol,pfcol:i);**

Changes the colors of a dialog field.

id : Identifier of the field (can also be a filter).

bcol : Background color

fcol : Text color

pbcol : Background color in locked status

pfcol : Text color in locked status;

Example :
```
setfcol(A_,3,0,pbcol=5);
```

**procedure RSETF (id:n; /path:s; var lo:l);**

Rebuilds a dialog field. This is necessary if the structure of a field (lister, menu or structure editor) changes. If ID is not indicated, the selected field is rebuilded.

id : Field ID

path : new structure position (in EDITST)

lo : menu options list; must be indicated when the options list of a menu field has been changed.

Example :
```
menu1:ls=('Option1','Option2 new')
rsetf(M1,lo=menu1);
```

**procedure SETTIP (id:s; tips:ls);**

Assigns a Windows tip to the field ID. Purpose: If the user stays with the mouse stopped over a field for one second, a window appears with additional information.

id : Field ID

tips : Information text

Example :
```
info:ls=('This field','means ...');
settip(A1,info);
```

## 5.14.4 Handling of dialogue masks

### procedure MASK (id:n; /ok,o,vl:n; mode,nr:i);

Opens a dialogue mask and starts a dialogue with the user. A dialogue mask can also contain several pages. This opens one register for each page in the top of the dialogue.

id : Dialogue mask ID

ok : Procedure that is called, when the OK key is pressed

o : Options: N=without OK button; A=with ABANDON button

vl : Visible Layers

mode : work mode: 1=Fields of other masks cannot be selected

nr : page NR (if the dialogue mask has several pages)

Example :
```
mask(DATA,ok=controldata,mode=1,o=A);
```

### procedure OPENSTAT (id:n);

Opens a status mask on the top-left surface of the application.

id : Dialogue mask ID

Example :
```
openstat(STAT);
```

### procedure OPENMAIN (xl,yt,w,h:r; id:n);

Opens a main menu in the application surface.

xl : X-left-coordinate of the menu (text units) in the work surface

yt : Y-top-coordinate of the menu (negative value!) in text units;
The program ensures that the status surface is not covered;

w : Width of the menu field (in text units);

h : Height of the menu field (in text units);

id : Main menu table

Example :
```
openmain(1,-3,30,40,MAIN);
```

### procedure OPENTABS (p:i; x,y:r; t:s; id:n);

Openas a register menu in the application surface.

p : Positioning:
0: fills the entire work surface (recommended value);
In this case the parameters X and Y do not have influence;
1: Centered in the work surface:
X: left margin = right margin (in text units)
Y: top margin = lower margin (in text units)
2: Centered in the work surface:
X: Width (in text units)
Y: Height (in text units) A
7: Bound to the right bottom corner of the work surface;
X: X-left coordinate (in text units);
Y: Y-top coordinate, negative value! (in text units);

x : Data for positioning: margin or width

y : Data for positioning: margin or height

t : Title for the register menu; appears left on the top;

id : ID of the table containing the register options

Example :
```
opentabs(0,0,0,'',MAIN);
opentabs(7,0,-3,'Main menu',MAIN);
```

### procedure RSETM (id:n);

Rebuilds the dialogue window ID. This is necessary if the addresses or values of the shown variables are changed. If ID is not indicated, the selected window is rebuilded.

id : Dialog mask ID

Example :
```
rsetm(MAIN);
```

### procedure SELM (id:n; /fnr:i);

Selects a dialogue window.

id : Dialogue mask ID

fnr : Field to be selected (optional);

Example :
```
selm(MAIN,fnr=2);
```

### procedure DELM (id:n);

Deletes the indicated dialogue window.

Example :
```
delm(MAIN);
```

### procedure SETML (layer,mode:i; /fnr,rset:i);

Sets a layer visible or invisible in the active dialogue mask.

layer : Layer number

mode : 0:invisible; 1:visible;

fnr :

rset :

Example :
```
setml(2,1);
```

### procedure SECDATA (/);

Allows to secure data in the opening of a mask. This command is called in the initialization of the mask and causes the appearance of a RESET button (beside the OK button). This button allows the user, after having changed data, to restore the original situation.

/ : Variables to secure (separated with comma)

Example :
```
secdata(p,v1);
/secures all actual parameters and the variable V1
```

## 5.14.5 Standard Dialogues

### procedure ASK (t:s; var txt:ls; but:s; var a:i);

Opens a dialogue mask with a question and the respective answers as buttons. When the user presses a button, Mirakon returns in A the number of the pressed button. The dialogue mask appears in center of the screen.

t : Title of the dialogue mask

txt : Text in dialogue mask (list of strings)

but : Button texts (separated with comma)

a : Variable that receives the answer (button number);

Example :
```
answer:i;
txt:ls=('Do you want','to delete ?');
```

```
ask('Question',txt,'Yes,No',answer);
if answer=2: exit; / if NO aborts
```

**procedure SELID (var id:s; var opts:l; t:s; xl,yt,w,h:r);**

Openas a dialogue mask with a menu, lets the user select an option
and returns the answer in the variable ID.

id : Variable that receives the answer

opts : List of the options (list of strings)

t : Title of the dialogue mask

xl : X-Left-position of the dialogue mask in text units

yt : Y-Top-position of the dialogue mask in text units

w : Width of the dialogue mask in text units

h : Height of the dialogue mask in text units

Example :
```
answer:s;
options:ls=(B='Big',M='Medium',S='Small');
selid(answer,options,'Size',20,-5,20,6);
if answer='B': inf('Big was selected !');
```

**procedure CHOOSEFILE (var fp:s; sp:s);**

Start the Windows dialogue to select a file;

fp : Variable that receives the selected file path

sp : Initial path

Example :
```
file:s;  choosefile(file,'C:\MIRAKON');
```

**procedure CHOOSEFOLDER (var dir:s);**

Start the Windows dialogue to select a folder;

dir : Variable that indicates the initial path and receives
the selected path

Example :
```
folder:s='C:\MIRAKON';
choosefolder(folder);
```

**procedure SAVEFILE (var fn:s; /ltxt:ls);**

Start a Windows dialogue to save a file.

fn : Variable that indicates the initial file path
and returns the selected file path

ltxt : Text to be stores in the selected file (optional)

Example :
```
file:s='c:\mirakon\new.txt';
savefile(file);
```

**procedure CHOOSEKEY(wm,fid:n; var key:s; /o,sp,idx,flt,st:n;gf,of,b,ti:s;w,h:r;db:i)**

Opens the loader dialogue allowing the user to select an object from a data base.
In contrast to the LOADER only the number of selected object is loaded.
Parameters O, SP, IDX, FLT, ST, GF, OF, TI, B, DB are equal as in LOADER.
The system variable DBKEY informs about the selected object number.
(if needed you can build programmable buttons in the parameter B).

wm : Data base module

fid : If FID contains a dialog field ID, that dialog field is redrawn

key : Variable that receives the selected object number

w : Window width in text units (Default=65)

h : Window height in text units (Default=22)

Example :
```
pnr:s;  choosekey(PROD,,pnr,o=W35,w=75);
```

**procedure CHOOSEKEYL(wm:n; var keys:l; /o,sp,idx,flt,st:n;gf,of,b,ti:s;w,h:r;db:i)**

Functions analog as CHOOSEKEY, it returns however a list of the numbers of those objects marked by the user.

wm : Data base module

keys : List that receives the list of marked object numbers

Example :
```
lnr:ls; choosekeyl(PROD,lnr);
```

**procedure ASKME (var m:e; tab,as:n; t1,q1:s);**

Opens a dialogue mask for the master selection and parameter input.

m : The selected master element

tab : Table with all existing masters

as : Active situation during the master choice

t1 : Title for the dialogue mask

q1 : Text appearing left from the master menu

Example :
```
askme(pr.pm,PR,PRE,'Specifications','Product master');
```

## 5.14.6 Programmable Dialogue

**procedure OPENDIALOG (x,y,w,h:r; t:s; cb:i);**

Opens a dialogue window (without ok button).

x : X-coordinate of the left edge of the window in text units

y : Y-coordinate of the upper edge of the window in text units (always a negative value!)

w : Window width in text units

h : Window height in text units

t : Title of the window

cb : Background color

Example :
```
opendialog(10,-8,30,10,'Input',3);
/...openstring, openquestion, etc.
rundialog(ok=closedialog);
```

**procedure RUNDIALOG (/ok:n);**

Starts a dialogue opened with OPENDIALOG.

ok : Procedure to be called when the OK button is pressed.

**procedure CLOSEDIALOG;**

Closes the current dialogue window;

**procedure OPENSTRING (x,y:r; s:s; /color,font,size,bold,ital,adj:i);**

Draws a text line in the dialogue window, opened with OPENDIALOG.

x : X-coordinate of the text line in mm
y : Y-coordinate of the text line in mm
s : Text line
color : Color number (default 0)
font : Font number (default 2)
size : Font size in points (default 12)
bold : Font width: 0:normal 1:bold (default 1)
ital : Font style: 0:normal 1:italic (default 0)
adj : Adjustment: 1:left, 2:right, 3:midle (default 1)

Example : `openstring(2,-1,'Input',color=3,font=5,size=20);`

**procedure OPENGRAFIC (var g:l; x,y:r; /id:n);**

Draws a grafic in the opened dialogue window.

g : Grafic
x : X-coordinate of the upper left corner in mm
y : Y-coordinate of the upper left corner in mm
id : Identifier of the grafic in the dialogue mask (optional);

**procedure OPENDISPLAY (id:n; x,y,w,h:r; v:n; f:i);**

Opens a display field in the opened dialogue window.

id : Field Identifier
x : X-coordinate of the left edge of the field in mm
y : Y-coordinate of the upper edge of the field in mm
w : Field width in text units
h : Field height in text units
v : Name of the variable to be shown
f : Display format: depending upon type: Decimal places or date format

Example : `opendisplay(A1,15,-2,8,1,value1,3);`

**procedure OPENCHECKBOX (id:n; x,y:r; t:s; v:n; /p:n);**

Openas a checkbox field in the dialogue window opened with OPENDIALOG.

id : Field Identifier
x : X-coordinate of the left edge of the field in mm
y : Y-coordinate of the upper edge of the field in mm
t : Text line; appears on the right of the field;
v : Name of the variable to be asked
p : Procedure that is called when leaving the field

Example : `opencheckbox(A1,15,-2,'With border',o1);`

**procedure OPENQUESTION (id:n; x,y,w:r; v:n; f:i; /p:n);**

Opens a question field in the dialogue window opened with OPENDIALOG.

id : Field Identifier

x : X-coordinate of the left edge of the field in mm

y : Y-coordinate of the upper edge of the field in mm

w : Width of the field in text units

v : Name of the variable to be asked

f : Display format: depending upon type: Decimal places or date format

p : Procedure that is called when leaving the field

Example :
```
openquestion(A2,15,-2,8,value1,3,p=controlvalue1);
```

**procedure OPENBUTTON (id:n; x,y,w,h:r; t:s; p:n);**

Opens a button field in the dialogue window opened with OPENDIALOG.

id : Field identifier

x : X-coordinate of the left edge of the field in mm

y : Y-coordinate of the upper edge of the field in mm

w : Field width in mm

h : Field height in mm

t : Text that appears in the button center

p : Procedure that is called, when the button is pressed

Example :
```
openbutton(B1,10,-6,20,2,'Calculate',calcvalue1);
```

**procedure OPENSELECTOR (id:n; x,y,w:r; v:n; var opts:l; p:n);**

Opens a selector field in the dialogue window opened with OPENDIALOG.

id : Field Identifier

x : X-coordinate of the left edge of the field in mm

y : Y-coordinate of the upper edge of the field in mm

w : Width of the field in text units

v : Name of the variable to be asked

opts : List of the selector options

p : Procedure that is called when leaving the field

Example :
```
l1:ls=('Light','Normal','Eavy');
openselector(M1,10,-4,20,answer,l1);
```

**procedure OPENMENU (id:n; x,y,w,h:r; v:n; var opts:l; p:n);**

Opens a menu field in the dialogue window opened with OPENDIALOG.

id : Field Identifier

x : X-coordinate of the left edge of the field in mm

y : Y-coordinate of the upper edge of the field in mm

w : Width of the field in text units

h : Height of the field in text units

v : Name of the variable to be asked

opts : List of the menu options

p : Procedure that is called when leaving the field

Example :
```
l1:ls=('Light','Normal','Eavy');
openmenu(M1,10,-4,40,12,answer,l1);
```

### procedure OPENCOMBO (id:n; x,y,w,h:r; v:n; var opts:l; /p:n);

Opens a combobox field in the dialogue window opened with OPENDIALOG.

id : Field Identifier
x : X-coordinate of the left edge of the field in mm
y : Y-coordinate of the upper edge of the field in mm
w : Width of the field in text units
h : Height of the field in text units
v : Name of the variable to be asked
opts : List of the combobox options
p : Procedure that is called when leaving the field

Example :
```
l1:ls=('Light','Normal','Eavy');
opencombo(M1,10,-4,40,12,answer,l1,p=control01);
```

### procedure OPENEDITOR (id:n; x,y,w,h:r; var txt:l; /l:i);

Opens a text editor field in the dialogue window opened with OPENDIALOG.

id : Field Identifier
x : X-coordinate of the left edge of the field in mm
y : Y-coordinate of the upper edge of the field in mm
w : Width of the field in text units
h : Height of the field in text units
txt : Text to be edited
l : Maximal length of one text line

Example :
```
openeditor(E1,15,-2,8,1,value1,l=40);
```

## 5.14.7 Selfrunning demos

### procedure INPUT (/);

Adds keyboard entries or commands into the input buffer
(for self running demos).

Instructions:

K + Number : Keyboard entry: Number=key code
B + Name : Calls a mask
P + Number : Sets the general break between keys pressing Number in 0.01-Seconds
W + Number : Inserts an unique break; Number in 0.01-Seconds
'text' : Inserts text input from the keyboard
M(x,y) : Moves the mouse to X,Y in pixels
L : Left mouse button is pressed
R : Right mouse button is pressed

Example :
```
input(K13,'ABC',W100,BM02);
```

| NR | NAME | WERT |
|---|---|---|
| 0 | Back | 8 |
| 1 | Tab | 9 |
| 2 | Enter | 13 |
| 3 | Escape | 27 |
| 4 | Btab | 315 |
| 5 | Cursor Down | 320 |

## 5.15 Configuration

### procedure GETFILEPARS (id:n; /var file:s);

Returns information (file path) about a configured data base.

id : Data base ID as it is indicated in the configuration

file : Variable that receives the file path

Example :
```
file:s;
getfilepars(DAT,file=file);
if exfile(file)=0: inf(file+' not found');
```

### procedure SETFILEPARS (id:n; /file:s; accmode:i);

Changes the file paths and/or access mode of a configured data base. This command can be called at the user login, (in the configuration, on the user code, the situation AT LOGIN) in order to change the data bases according to users.

id : Data base ID as it is indicated in the configuration

file : File path

accmode : Access mode:
1 = direct (via client)
2 = via server (only for Client/Server version)

Example :
```
/ in User-CODE
at LOGIN:
localdat:s='C:\MIRAKON\MYDATA.DAT';
setfilepars(DAT,file=localdat,accmode=1);
```

### procedure GETDBASES (var dbs:l);

Returns the configured data base parameters in DBS (list of elements):

Data base ID in: files.[i].id
Data base path in: files.[i].pars.fname
Access mode in: files.[i].pars.accmode
Data base number in: files.[i].pars.dbnr

Example :
```
dbs:l;
getdbases(dbs);
travel dbs:
  begin
  if exfile(p.fname)=0:
  inf('Data base'+pe^.id+' not found');
  end;
```

### function LISTUSERS(/filt:s; ulevel,active:i):l

Returns the list of users.

filt : User id filter

ulevel : User level filter
1:User
2:Author
3:Administrator

active : When 1 only returns the active (logged) users

Example :
```
l1:l=listusers(ulevel=3); /returns all administrators
```

### procedure GETUPERMS(uid:s; var uperms:l)

Returns in UPERMS the permissions of user UID.

### procedure GETUSERS (var users:l);

Reads the users list from the configuration data into the variable USERS.
This command may only be executed by administrators.

users : List that receives the configuration data

Example :
```
users:l;
getusers(users);
infv(users);
```

### procedure SAVEUSERS (var users:l);

Saves the configuration data (users list) from the variable USERS.
This command may only be executed by administrators.

users : List that contains the configuration data (users list).

Example :
```
saveusers(users);
```

### procedure SAVEUSER;

Saves the actual user object with last user settings.

### procedure GETLFC (var lfcs:ls; /form:i)

Returns the list of calendars from the configuration.

lfcs : Variable that receives the list of calendars.

form : Result format:
0 = calendar Identifier (Default)
1 = calendar Identifier + '|' + calendar description.
2 = calendar description

Example :
```
lfc:ls; getlfc(lfc,form=2);
```

### procedure GETFC(id:s; var fc:l)

Returns the calendar ID in the list FC.

Example :
```
fc:l; getfc('ALLG',fc);
```

### procedure GETSTARTMENU(var sm:l)

Returns the start menu list from the configuration in the variable SM.

### procedure SAVEUNITSTAB

Saves the units table, in the system variable UNITSTAB, into the configuration.

### procedure GETFONTS(var fnts:l)

Returns the list of fonts from the configuration.

## 5.16 Printer

### procedure PRINT (mode:i; var doc:l);

Prints the document DOC.

mode : 1:Text 2:Grafic

doc : Document

Example :
```
print(2,gdoc);
```

### procedure SETPRINTER (/lm,rm,tm,bm:r; sc,size,scale,copies,orient,psource:i; pname,doc:s);

Sets printer parameters.

lm : Left margin in mm

rm : Right margin in mm

tm : Top margin in mm

bm : Bottom margin in mm

sc : Automatic scaling: 1:On 0:Off

size : Paper size: 1=A4; 2=A3;

scale : Scaling in % (default 100)

copies : Number of copies

orient : Paper adjustment: 1=Portrait; 2=Landscape;

psource : Paper source number: 1=Auto 2=Cassette 3=Envelope
4=Env. manual 5=Formsource 6=Large capacity
7=LargeFmt 8=Lower 9=Manual 10=Middle 11=Onlyone
12=tractor 13=smallFmt

pname : Printer designation, as it is defined in Windows

doc : Document name

Example :
```
setprinter(size=2,scale=50,pname='HP Laserjet 5P');
```

### procedure GETPRINTER (/var lm,rm,tm,bm:r; var sc,size,scale,copies,orient,psource:i; var pname, doc:s);

Gets the printer parameters.
Parameters are the same as in SETPRINTER.

Example :
```
pn:s;
getprinter(pname=pn);
inf('The actual printer is'+pn);
```

## 5.17 Interfaces

### 5.17.1 Import images

You can import BMP, JPEG, ICO and TIFF image files into the grafic editor.

Procedure:
1) select the option import/from a file from the main menu.
2) select a file with BMP, JPG, ICO or TIFF extension.
3) position the square frame of the picture with the mouse at the desired position and click the left mouse button. If the picture is too large for the grafic, you should first zoom the grafic.

## 5.17.2 Files read and write

**Variable for reading from external files**

| | | |
|---|---|---|
| **TFS** | s | File section in TRAVELF |
| **TFP** | a | File position in TRAVELF |

### procedure READTEXT (file:s; var text:ls);

Opens a text file and writes its contents in a list.

file : Text file (path + name)

text : Variable that receives the text

Example : `data:ls; readtext('IMPORT.TXT',data);`

### procedure WRITETEXT (file:s; var text:ls);

Writes a text in a text file.

file : Text file (path + file name)

text : List with the text lines.

Example : `writetext('EXPORT.TXT',data);`

### procedure MAKEF (file:s; size:a);

Creates a file and fills it with blank characters.

file : File (path + name)

size : Number of blank characters

Example : `makef('DATA.DAT',512);`

### procedure OPENF (var f:i; file:s);

Opens a file (of any type) and writes the assigned file number (handle) in the variable F.

f : Variable that receives the file number

file : File (path + name

Example : `f1:i; openf(f1,'c:\temp\DATA.DAT');`

### procedure CLOSEF (f:i);

Closes the file indicated by F.

f : File number (returned by OPENF command)

Example : `closef(f1);`

### procedure READFS (f:i; pos:a; n:i; var s:s; /c:i);

Reads the a number of bytes from one file, starting at a given position.

f : File number (returned by OPENF command)

pos : Start position for reading from the file
Important: The first position is 1, not 0

n : Number of byte to be read

s : Variable that receives the characters read

c : String clearing:

0: removes all blanks from S;
1: removes all blanks on the left side;
2: removes all blanks on the right side;
3: removes all blanks on the left and right side;

Example : `name1:s; readfs(f1,1250,20,name1,c=3);`

### procedure READFI (f:i; pos:a; n:i; var i:i);

Reads a number of bytes, starting at the given position, from one file, and interprets it as an integer value.

f : File number (returned by OPENF command)

pos : Start position for reading from the file
Important: The first position is 1, not 0

n : Number of byte to be read

i : Variable that receives the number read

Example : `v1:i; readfi(f1,1250,20,v1);`

### procedure READFR (f:i; pos:a; n:i; var r:r);

Reads a number of bytes, starting at a given position, from one file, and interprets it as an real value.

f : File number (returned by OPENF command)

pos : Start position for reading from the file
Important: The first position is 1, not 0

n : Number of byte to be read

r : Variable that receives the number read

Example : `v1:r; readfr(f1,1250,20,v1);`

### procedure READFD (f:i; pos:a; n:i; var d:d);

Reads a number of bytes, starting at a given position, from one file, and interprets it as a date-value.

f : File number (returned by OPENF command)

pos : Start position for reading from the file
Important: The first position is 1, not 0

n : Number of byte to be read

d : Variable that receives the date read

Example : `d1:d; readfd(f1,1250,20,d1);`

### procedure WRITEFS (f:i; pos:a; n:i; s:s);

Writes a number of bytes starting from a position into a file.

f : File number (returned by OPENF command)

pos : Start position for writing into the file
Important: The first position is 1, not 0

n : Number of byte to be written

s : Variable that contains the characters to be written

Example : `writefs(f1,1250,3,'ABC');`

**procedure FINDFSEQ (f:i; headl,sepl,recl,keyp:i; key:s; var recp:a);**

Searches for a data record in a sequential file.

| | |
|---|---|
| f : | File number (returned by OPENF command) |
| headl : | File header length in bytes |
| sepl : | Separator length in bytes |
| recl : | Data record length in bytes |
| keyp : | Key position |
| key : | Searched record primary key |
| recp : | Variabel that receives the found position.<br>If RECP=0 the key was not found; |

Example :
```
recpos:a; name1:s;  f:i;
openf(f,'TEST.DAT');
findfseq(f,128,2,730,1,'A007X',recpos);
readfs(f,recpos+12,20,name1);
```

## 5.17.3 ODBC- interface

With the following procedures data can be read and/or written directly from/to ODBC compatible data bases (ACCESS, Excel, Oracle, and so on) without interference of the owner program.

**Conditions for the functioning of the interface:**

1. ODBC driver manager (Microsoft program) must be present.
2. ODBC driver for the respective data base must be available and installed. This is normally supplied by the data base creator and installed on your computer.
3. The data base to which you would like to access must be announced as a data source. You can set this with the ODBC Admnistrator (Microsoft program). The data source name assigned by you is later used.

**Applications examples:**

Example 1:
```
/ inquires which drivers + data sources
/ are installed

info:l;
openodbc;
infodbc(info);
editlst(info,'INFO');
closeodbc;
```
Example 2:
```
/ inquires what tables are installed in the
/ data source D1

tabs:l;
openodbc;
connectodbc('DSN=D1');
tabsodbc(tabs);
editlst(tabs,'Tables');
disconnectodbc;
closeodbc;
```
Example 3:
```
/ executes an SQL inquiry:
```

```
          / names and age of all coworker from
          / the table PERSONAL from the data source COMPANYDATA.

          sql:s='SELECT NAME, AGE FROM PERSONS';
          result:l;
          openodbc;
          connectodbc('DSN=COMPANYDATA');
          sqlodbc(sql,result);
          editlst(result,'SQL Result');
          disconnectodbc;
          closeodbc;
```

Example 4:

```
          / questions a value withou having to
          / write an SQL inquiry: Age of Lisa Simpson

          age:r;
          openodbc;
          connectodbc('DSN=COMPANYDATA');
          getvalodbc(PERSONAL,AGE,NAME,'Lisa Simpson',age);
          infv(age);
          disconnectodbc;
          closeodbc;
```

**procedure OPENODBC;**

Initializes the ODBC interface system.

**procedure CLOSEODBC;**

Terminates the ODBC interface system.

**procedure INFODBC (var info:l);**

Writes in the list INFO all installed ODBC drivers and data sources.

**procedure CONNECTODBC (code:s);**

Establishes a connection to a data source.
In CODE the data source name be indicated with 'DSN='.
Optionally the user ID (UID) and/or password (PWD) can be indicated.

Example :

```
connectodbc('DSN=ADRESSES')
connectodbc('DSN=SOURCE1;UID=USR;PWD=XY')
```

**procedure DISCONNECTODBC;**

Closes the active connection to a data source.

**procedure SQLODBC (code:s; var res:l);**

Sends an SQL inquiry string (CODE) and receives the result in RES.

Example :

```
result:l;
sqlodbc('SELECT NAME, AGE FROM PERSONAL',result);
```

**procedure GETVALODBC (tabid,colid,keyid:n; key:s; varid:n);**

Reads a value from a table.

tabid : Table ID

colid : Column containing the value searched

keyid : Column containing the search criteria

key : Value of the search criteria word

varid : Variable that receives the value found

Example :
```
getvalodbc(PERSONAL,AGE,NAME,'Lisa Simpson',age);
infv(age);
```

**procedure TABSODBC (var tabs:l);**

Writes in list TABS the list of tables of the connected data sources.

## 5.17.4 ODBC- interface

With the following procedures data can be read and/or written directly
from/to ODBC compatible data bases (ACCESS, Excel, Oracle, and so on)
without interference of the owner program.

**Applications examples:**

**procedure CONNECTADODB (dbpath:s; /passw,server,user:s; xlfirstl:i);**

Establishes a connection to a data source.
In CODE the data source name be indicated with 'DSN='.
Optionally the user ID (UID) and/or password (PWD) can be indicated.

dbpath :

passw :

server :

user :

xlfirst :

**procedure DISCONNECTADODB;**

Closes the active connection to a data source.

**procedure RUNADODBSQL (var sql:l; var res:l; /fmt:i);**

Sends an SQL inquiry string (CODE) and receives the result in RES.

sql :

res :

fmt :

**procedure GETADODBTABLES (var tabs:l);**

Writes in list TABS the list of tables of the connected data sources.

**procedure READXLFILE (fname:s; var res:l; /line1:i);**

Sends an SQL inquiry string (CODE) and receives the result in RES.

fname :

res :

line1 :

Example : `data:ls; readtext('IMPORT.TXT',data);`

### 5.17.5 Excel-Interface

Example :
**procedure OPENEXCEL (file:s; /wbtmp:s);**

file :
wbtmp :

**procedure CLOSEEXCEL (save:i);**

save :

**procedure EXCELVISIBLE (vis:i);**

vis :

**procedure CELLS (/);**

R + Zahl :
C + Zahl :
S + String :
M :
N1 :
N2 :
GR :
GC :
L + Name :
V + Name :
E + String :
W + Zahl :
WA :
H + Zahl :
AH + Zahl :
AV + Zahl :
IC + Zahl :
IR(r,g,b) :
BI + Zahl :
BC + Zahl :
BR(r,g,b) :
BS + Zahl :
TF + Zahl :
TH + Zahl :
TW + Zahl :
TS + Zahl :
TC + Zahl :
TR(r,g,b) :
TO + Zahl :

TI + Zahl :

Fxyz :

F + String :

X + String :

K0 :

DR :

DC :

UC :

UX :

UP :

OA :

OF + 0/1 :

OOR + 0/1 :

OOX + 0/1 :

**procedure ACTIVATECELL (r,c:i);**

r :

c :

Example :

**procedure SETWORKSHEET (/wsnr:i; wsname:s; var retnr:i; var retname:s; var vis:i);**

wsnr :

wsname :

retnr :

retname :

vis :

Example :

**procedure NEWWORKSHEET (pos:i; wsname:s);**

pos :

wsname :

Example :

**procedure DELWORKSHEET (/wsnr:i; wsname:s;);**

wsnr :

wsname :

Example :

**procedure IMPXLIMAGE (file:s; /il,it,io:r);**

file :

il :

it :

io :

Example :

**procedure XLSAVEAS (file:s);**

file :

**procedure COPYWORKSHEET (pos:i) ;**

pos :

Example :
**procedure MOVEWORKSHEET (pos:i) ;**

pos :

Example :
**procedure PROTECTWORKSHEET (pwd:s);**

Example :
**procedure READCLIPTAB (var tab:l);**

tab :

Example :
**function function COUNTWS:i;**

## 5.17.6 MS-Project-Interface

Example :

```
/ Create a Project file

openmsproj('c:\proj.mpp');
setprojsettings(stg=1,prior=2,hpd=8,hpw=40,dpm=20,start=now);
if exprojcal('RESCAL')=0:
  begin
  newprojcal('RESCAL',basecal='STANDARD');
  defcalendar('RESCAL',wd=6,iswork=0);
  defcalendar('RESCAL',wd=7,iswork=0);
  d1:d=ds('09:00');
  d2:d=ds('12:00');
  d3:d=ds('13:00');
  d4:d=ds('18:00');
  defcalendar('RESCAL',wd=5,shiftn=1,ds=d1,de=d2);
  defcalendar('RESCAL',wd=5,shiftn=2,ds=d3,de=d4);
  end;

if exprojres('DEFRES')=0:
  begin
  newprojres('DEFRES',initials='DFR',basecal='MIRCAL',type=1,maxunits=200);
  newprojres('DEFRES2',initials='DFR',basecal='MIRCAL',type=1,maxunits=200);
  defcalendar('DEFRES',resc=1,wd=1,shiftn=1,ds=ds('10:00'),de=ds('12:00'));
  defcalendar('DEFRES',resc=1,wd=1,shiftn=2,ds=ds('14:00'),de=ds('19:00'));
  end;

if exprojtask('TASKG')=0:
  begin
  newprojtask('TASKG','50m',prior=1,type=1,effdr=1,cal='MIRCAL',ignorerc=1,miles=
  newprojtask('TASKG1','95m',prior=2,type=1,percw=90,indent=1,miles=1);
  newprojtask('TASKG2','95m',prior=2,type=1,percw=50,indent=-1);
  assignrestotask(90,tid='TASKG1',rid='DEFRES');
  assignrestotask(70,tid='TASKG2',rid='DEFRES2');
  reltasks(1,'10m',tids='TASKG1',tidm='TASKG2');
  end;

closemsproj(1);
```

### procedure OPENMSPROJ(file:s)

Opens an MS. Project project file. When the given file does not exist, a new project is created. CLOSEMSPROJ must be called at the end.

file : Project filename

### procedure CLOSEMSPROJ(save:i)

Closes the active project.

save : when -1 all changes are ignored, project isn't saved;
when 0 the user is prompted to save the project;
when 1 the project is automaticaly saved;

### procedure PROJVISIBLE(vis:i)

Shows or hides the active project.

vis : 1 = show, 0 = hide

### function EXPROJCAL(id:s)

Informs if a calendar exists in the active project.
Returns 1 if found and 0 if not found

id : Calendar Id.

### procedure NEWPROJCAL(id:s; /basecal:s)

Inserts a new project calendar.

id : Calendar Id.

basecal : Base calendar Id.

Example :
```
if exprojcal('MIRCAL')=0:
  newprojcal('MIRCAL',basecal='STANDARD');
```

### procedure DEFCALENDAR(id:s; /rset,wd,d,m,y,iswork,shiftn,resc:i; pstart,pend,ds,de:d)

Defines the calendar properties.

id : Calendar Id. or Resource Id. when RESC=1

rset : Resets the calendar.

wd : Week day (1=monday to 7=sunday)

d : Day

m : Month

y : Year

iswork : 1 if period is working period, 0 if period is non-working period

shiftn : Shift number (1 to 5)

resc : If 1 then we are referring to an resource calendar, and ID is the resource ID.

pstart : Period start date

pend : Period end date

ds : Shift start

de : Shift end

Example :
```
defcalendar('MIRCAL',wd=6,iswork=0);
defcalendar('MIRCAL',wd=7,iswork=0);
d1,d2,d3,d4:d;
maked(d1,0,0,0,09,00);
maked(d2,0,0,0,12,00);
maked(d3,0,0,0,13,00);
maked(d4,0,0,0,17,00);
defcalendar('MIRCAL',wd=1,shiftn=1,ds=d1,de=d2);
defcalendar('MIRCAL',wd=1,shiftn=2,ds=d3,de=d4);
defcalendar('MIRCAL',wd=2,shiftn=1,ds=d1,de=d2);
defcalendar('MIRCAL',wd=2,shiftn=2,ds=d3,de=d4);
```

### procedure SETPROJSETTINGS(/stg,prior,defdu,defwu:i; hpd,hpw,dpm:r; start,end:d)

Sets the active project parameters.

stg : Strategy:
1:forward;
2:backwards;

prior : Priority: 1 to 10 (1 Highest priority, 10 lowest priority)

defdu : Sets the default duration unit.
1=Minutes
2=Hours
3=Days
4=Weeks

defwu : Sets the default work unit.
1=Minutes
2=Hours
3=Days
4=Weeks

hpd : Sets the number of hours per day for tasks in a project.

hpw : Sets the number of hours per week for tasks in a project.

dpm : Sets the number of days per month for tasks in a project.

start : Sets the start date for a project.
Setting the start date also causes the project to be scheduled from its start date

end : Sets the finish date for a project.
Setting the finish date also causes the project to be scheduled
from its finish date.

Example :
```
setprojsettings(stg=1,prior=2,hpd=8,hpw=40,dpm=22,start=now,defdu=2,defwu=2);
```

### function EXPROJRES(id:s; /var ruid,index:a)

Informs whether the Resource ID exists in the active project.

id : Resource ID.

ruid : Returns the Resource unique identifier.

index : Returns the Resource position in the Resources Sheet.

### procedure NEWPROJRES(id:s; /initials,basecal,email:s; type,pos:i; maxunits:r; var ruid:a)

Inserts a new Resource in the active project.

id : Resource ID.

initials : Resource initials

basecal : Resource Calendar Id.

email : Resource E-Mail

type : Resource type: (1=Work (default), 2=Material)

pos : Resource position in the Resources Sheet.

maxunits : Maximal resource units (in %)

ruid : When indicated, returns the resource unique identifier

Example :
```
if exprojres('DEFRES')=0:
  begin
  newprojres('DEFRES',initials='DFR',basecal='MIRCAL',type=1,maxunits=200);
  defcalendar('DEFRES',resc=1,wd=1,shiftn=1,ds=ds('10:00'),de=ds('12:00'));
  defcalendar('DEFRES',resc=1,wd=1,shiftn=2,ds=ds('14:00'),de=ds('19:00'));
```

```
end;
```

**function EXPROJTASK(id:s; /var tuid,index:a)**

Informs whether the task ID exists in the active project.

id : Task Id.

tuid : Returns the Task unique identifier.

index : Returns the Task position in the tasks list.

**procedure NEWPROJTASK(id:s; dur:s; /pos,prior,constrt,type,percw,indent,effdr,ignorerc,miles,est:i; deadl,constrd,ds,de:d; cal:s; var tuid:a)**

Inserts a new task in the active project.

id : Task Id.

dur : Task duration (eg.: '10 mins')

pos : Task position in the tasks list

prior : Priority: 1 to 10 (1 Highest priority, 10 lowest priority)

constrt : Constraint type:
1=As late as possible
2=As soon as possible
3=Finish no earlier than constraint Date
4=Finish no later than constraint Date
5=Must finish on constraint date
6=Must start on constraint date
7=Start no earlier than constraint date
8=Start no later than constraint date

type : Task type:
1=Fixed units
2=Fixed duration
3=Fixed work

percw : Sets the percent complete.

indent : Task indentation (X = down X levels, -X = up X levels)

effdr : 1 if the task is effort-driven

ignorerc : 1 if the resource calendar is ignored when scheduling the task

miles : 1 if the task is a milestone

est : 1 if a task's duration is an estimate

deadl : Sets a deadline for a task

constrd : Sets the constraint date.

ds : Sets the start date.

de : Sets the finish date.

cal : Sets the calendar to be used when scheduling the task

tuid : When indicated, returns the task unique identifier

Example :
```
if exprojtask('TASKG')=0:
  newprojtask('TASKG','50 mins',prior=1,type=1,effdr=1,cal='MIRCAL',ignorerc=1,mi
```

**procedure ASSIGNRESTOTASK(nunits:r; /tid,rid:s; tuid,ruid:a)**

Assign NUNITS of an Resource (rid or ruid) to a task (tid or tuid).

nunits : Number of Resource units (in %)

tid : Task Id.

rid : Resource Id.

tuid : Task unique identifier

ruid : Resource unique identifier

Example :
```
assignrestotask(90,tid='TASKG1',rid='DEFRES');
assignrestotask(100,tid='TASKG1',rid='DEFRES2');
```

**procedure RELTASKS(type:i; lag:s; /tids,tidm:s; tuids,tuidm:a)**

Creates a relation between two tasks.

type : Relation type:
1=Finish to Start
2=Start to Start
3=Finish to Finish
4=Start to Finish

lag :

tids :

tidm :

tuids :

tuidm :

Example :
```
reltasks(1,'10 mins',tids='TASKG1',tidm='TASKG2');
```

## 5.17.7 Find and scan files

**procedure GETFILES (dir:s; var lst:ls);**

Returns the list of the files from a folder.

dir : Folder path

lst : List that receives the file names (list of strings)

Example :
```
files:ls;
getfiles('C:\MIRAKON\',files);
```

**procedure GETDIRST (dir:s; var st:l; /maxl,netw,head:i);**

Returns the sub-structure of a folder as an hierarchical list of elements.

The element type (pe^.typ) informs about the meaning of each position:
0=Drive; 1=File; 2=Folder; 10=Network drive

The element name (ename) contains the name of the Drive, File or Folder.

Each element contains the following parameters :
- p.dcreated:d; Date of creation (file or folder)
- p.dlastacc:d; Date of the last access (file or folder)
- p.dlastwrite:d; Date of the last write (file or folder)
- p.filesize:r; File size in bytes (only files)
- p.sdir:s; Folder

dir : Folder path

st : Folder structure (hierarchical list of elements).
Depending upon type of element (pe^.typ) it can contain:
- Drives (Typ=0)
- Folders (Typ=1)
- Files (Typ=2)
- Network drives (Typ=10)

maxl : Maximum number of hierarchic levels to be searched.
Allows you to avoid unnecessary time-consuming searches.

netw : 0=Network is not searched (default)
1=Network is search (can take very long time)

head : 0 = ST contains only the sub-structure of DIR
1 = ST contains the initial position as header element,
and in its sub-structure, the sub-structure of DIR

Example :
```
dir:l;
getdirst('C:\MIRAKON\',dir);
travel dir:
  begin
  if pe^.typ=2: inf('Folder='+ename);
  if pe^.typ=1: inf('File='+ename);
  end;
```

### procedure CHOOSEPATH (dir:s; var path:s; /netw:i);

Examines the sub-structure of a folder and shows an hierarchical menu to the user so he can select a position.

dir : Folder path

path : Path of the selected position

netw : 0=Network is not searched (default)
1=Network is search (can take very long time)

Example :
```
docfile:s;
choosepath('C:\DOCUMENTS\',docfile);
```

## 5.18 Costs calculation

#### Types for cost calculation

**COSTELEM** Cost element. Contains the following fields:
- VC:q; calculated value
- QR:q; reference quantity (number of operations during process)
- CPU:q; costs per unit of value (e.g.: € per Hour)
- REF:s; references = ka;fepath;bkpath;oppath;rsnrs;wpnrs;atts

#### Variables for cost calculation

| | | |
|---|---|---|
| **bkm** | q | Component quantity accumulated over all hierarchic levels<br>only valid in connection with EDITBS, GENAP and CALCKS. |
| **kla** | l | List of the cost elements that belong to the selected position<br>in the cost editor (only valid in the KA table) |
| **ks** | l | List of all cost elements during CALCCOST |
| **cpucolid** | s | Active cost rate column in the RS table (for CALCCOST) |

#### Situations for cost calculation

In the cost calculation and structure generator procedures CALCCOST, GENBS, GENAP, the following situations can be used:

| | |
|---|---|
| **K0** | Cost calculation phase 1: actions from product master |
| **PAA0** | Cost calculation phase 2: work process analysis |
| **K1** | Cost calculation phase 3: building of cost elements |

| | |
|---|---|
| **PBA0** | Work process generation phase 1: analysis of constructional structure |
| **PAI0** | Work process generation phase 2: formating work process from product master |
| **PAI1** | Work process generation phase 3: components actions build operations in the work process |
| **PAI2** | Work process edition |
| **PFI0** | Function structure generation out of product master actions |
| **PFI1** | Function structure edition |
| **PFA0** | Constructional structure generation phase 1: function structure analysis |
| **PBI0** | Constructional structure generation phase 2: formating structure from product master |
| **PBI1** | Constructional structure generation phase 3: functions actions build components in the constructional structure |
| **PBI2** | Constructional structure edition |
| **PBA1 bis PBAn** | Constructive structure completion phase 1, runs 1 to n |
| **PBC1 bis PBCn** | Constructive structure complete phase 2, runs 1 to n |

### procedure INSR (rs:n; /nr:s; need:q; o:n);

Assigns a resources master to the active operation.

| | |
|---|---|
| rs : | Number of the resource master in the RS table; |
| nr : | Possible resource numbers for the active operation. Allows you to assign a work place before the time calculation. |
| need : | Work need (for planning) |
| o : | options: Letters with the following meaning: E: Resources structure is emptied before the new assignment; |

Example : `insr(D01,o=E);`

### procedure INSRS (rss:ln; /nr:s; need:q);

Assigns resources to the active operation. RSS contains all possible resources numbers for the active operation. Mirakon examines then the operating conditions for each resource in the B-column of the RS table. The first valid resource is assigned and the search is stopped.

| | |
|---|---|
| rss : | list the possible resource masters |
| nr : | fix work place to assign before the time planning |
| need : | work need (for planning) |

Example :
```
insrs((B_,F_));
rsl1:ln=(A1,A2,A5,B4,B8,C3,C4,C7);
insrs(rsl1);
insrs((X),nr='1209');
```

### procedure INSCE (ka:n; val:r; /t:s; rs:n; rsi:i; u,supl,country:s);

Inserts a cost element into the cost structure of the product.

| | |
|---|---|
| ka : | Cost category number according to the KA table |
| val : | Cost value in the unit defined in KA table |
| t : | Description text |
| rs : | Resource NR from the RS table |
| rsi : | Resource position in D.RS list |

u : Cost unit if not equal to the one defined in the KA table

supl :

country :

Example :
```
insce(TR,50);
insce(TE,p.te,);
```

### procedure CALCCOST (var obj,opl:l; /o,cpu:n; d,n:i; var ks:l);

Calculates and generates the cost structure of an object from an operations list.
The cost structure is stored in obj.KS.

obj : Calculation object (product, order or plan)

opl : Operations list

o : options:
H: HKB-compatibel

cpu : Procedure that calculates the costing rate

d : Display mode: 1:no display

n : Number of runs (at K1,K2...) during the cost calculation;
Default=1;

ks : Cost structure (optional) if not obj.ks

Example :
```
calccost(pr,pr.ap);
calccost(pr,pr.ap,d=1,n=2);
```

The conditions for CALCOST are:

- Resources list for each operation in operation.DATA.RS.
- Cost categories in the KA table on the basis module.
  This table must contain the cost category unit in the U or UN column.
- Ressource masters in the RS table.
- Work unit of each resources defined in column WU of RS tables.
  When this column does not exist the unit HOUR (202) is selected.
- Costs per work unit of each resource defined in the column
  CCU or KS of the RS table. If the cost unit is not indicated,
  the unit selected units is CHF (301).

The cost calculation consists in the execution of actions of the knowledge base with the purpose of build all the necessary cost elements into the cost structure.

This happens in 3 phases:

1. Phase / Situation = K0 / product master actions:

   The actions assigned to the product masters are executed.
   At this time (normally) the global variables are set:

   Example: Action of product master "Part":

   AK K0: A000\v1:=p.lg; /batch quantity becomes global in variable V1

2. Phase / Situation = PAA0 / work process analysis:

   The work process is examined from top to bottom.
   For each operation the defined actions of its master are executed.
   The purpose of these actions is to recognize the product characteristics,

which are important for its manufacturing. This recognition consists of the assignment of values to global variables in the knowledge base, which can be saved as indicators in the product structure.

3. Phase / Situation = K1 / building of cost elements:

The work process is examined again from top to the bottom.
For each operation the defined actions of its master are executed.
The purpose of these actions is to insert and evaluate cost elements into the cost structure of the product.
This is done with the INSCE command (early also with INSKE and INSKEL)

Example: AK K1: insce(TH, 120);

CALCCOST needs following tables in the basis module A000:
- KA table containing the definitions of the cost categories;
- RS table containing the resource masters.

### procedure EDITCOST (fmt:n; var obj:l; /o:n; n:i);

Shows the cost structure of an object.

fmt : Cost format = row number of the KOF table

obj : Calculation object (product, order or plan)

o : options:
C = with calculate button;
H = HKB-compatible;

n : Number of runs (at K1, k2...) during cost calculation;
Default=1;

Example : `editcost(OP,pr,o=C,n=2);`

| NR | NAME | ST | FW | KAS | CODE |
|---|---|---|---|---|---|
| CG | By components | 2 | 90 | ('MK/5/0','TG/5/0','HK/8/2') | |
| OP | By operations | 3 | 90 | ('MK/5/0','TG/5/0','HK/8/2') | |
| FN | By functions | 1 | 90 | ('MK/5/0','TG/5/0','HK/8/2') | |

ST = structure: 1=functional structure 2=constructional structure 3=work process

FW = formular width in characters

KAS = cost category columns (list of strings).
one string = one cost category column.
string format: cost category/column width/decimal places

CODE = commands for co-designing the formular. At situation LST.4 you can add headlines using the system variabel TXI.
Example:
AT LST.4:
txi.1:=pr.nr+' ' +pr.name+' ' +pr.client;
txi.2:='Lot size='+sr(pr.ls, 0);

### function SUMKE (calc:i; var kel:l; kas:s; /u:n):q;

Returns the sum (quantity) of all cost elements in the list KEL that belong to the cost category KAS.

calc : Calculation method:

1 = adds all cost element values
2 = adds all cost element values multiplied by the number of operations
3 = adds all cost element values multiplied by the appropriate cost rate
4 = adds all cost element values multiplied by the appropriate cost rate and number of operations

kel : List with the cost elements

kas : Cost category filter

u : Desired unit

Example :
```
tgsum:=sumke(1,pr.ks,'TH,TN');
matsum:=sumke(2,pr.ks,'M_',u=USD);
```

**procedure FILTERKE (crit:i; filt:s; var l1,l2:l; /o:n; nh:i);**

Copies all the cost elements from the list l1 to the list l2 that fulfill the indicated criteria/filters.

crit : Filter criteria:
1: Cost elements that belong to a functional unit.
FILT = path of this functional unit.
2: Cost elements that belong to a component.
FILT = path of this component.
3: Cost elements that belong to an operation.
FILT = path of this operation.
4: Cost elements that belong to a cost category.
FILT = filter of cost categories.
5: Cost elements that belong to a function master.
FILT = filter of function master numbers.
6: Cost elements that belong to a component master.
FILT = filter of component master numbers.
7: Cost elements that belong to an operation master.
FILT = filter of operation master numbers.
8: Cost elements that belong to a cost element master.
FILT = filter of cost element master numbers.
9: Cost elements that belong to a resource master.
FILT = filter of resources master numbers.

filt : Filter value dependent of the filter criteria (see above).

l1 : Input list that contain the cost elements to be filtered

l2 : List that receives the filtered cost elements

o : Options:
- E: empties L2 before the filter process
- R: resets/initializes the cost calculation variables; necessarily if the object was replaced since the last FILTERKE.
- N: Filter with contrary meaning: all cost elements that fulfill the criteria are filtered (not returned).

nh : Number of hierarchic levels to be scanned based on the indicated path.

Example :
```
l1,l2:l;
filterke(3,'2.7',pr.ks,l1);
filterke(7,'D_',l1,l2,o=E);
```

### procedure GENAP (/o:n; d:i);

Generates a work process based on a constructional structure.

o : Sequence of letters with the following meaning:
L = the existing structure is deleted without
further inquiry (used for 'batch' processes).

d : Displaymode: 1:no display

**How GENAP works**
The generating of the work process consists of the execution of actions in the knowledge base with the purpose to build all necessary operations in the product structure.

This happens in three phases:

1. Phase / Situation = PBA0 / analysis of the constructive structure:

   The constructive structure is examined from the top to the bottom.
   For each component the appropriate actions of the masters are executed.
   Purpose of these actions is to recognize product characteristics, which are important for its manufacturing. This recognition consists of the assignment of values to knowledge base variables, which may be saved as indicators in the product structure.

   Example: Action for "flat surface":

   ```
   AT PBA0:  bknd++1; /inscreases the number of turned elements
   ```

2. Phase / Situation = PAI0 / formats work process:

   The actions defined in the assigned product master are executed.
   At this time, groups of operations (the process skeleton) are usualy inserted:

   Example: Action of the product master "locomotive":

   ```
   AT PAI0: K01->ap.0; /inserts the assembly operation group
   ```

3. Phase / Situation = PAI1 / building new operations in the work process:

   The constructional structure is examined again from the top to the bottom.
   For each component the defined actions of its master are executed.
   Purpose of these actions is to insert operations and sub-operations and to assign resources to them. When an operation is added, Mirakon examines whether the master of this new operation has subsequent actions at the situation PAI1, and executes them (chain reaction).

   Examples:

   - actions for the constructive component master "flat surface":

   ```
   AT PAI1: D7(p.l,p.d)->popd^.ss.0;
   /inserts sub-operation Roughing under operation Turning
   ```

   - actions of the operation boring:

   ```
   AT PA _: AR01->ss.0; / inserts the sub-operation Preparation
   AT PAI_: insrs((B_, F _)); /assigns ressource B_ or  F_
   ```

### procedure GENBS (/o:n; nfa:i; d:i);

Generates a constructional structure based on a functional structure.

o : Sequence of letters with the following meaning:
L = the existing structure is deleted without
further inquiry (used for 'batch' processes).

nfa : Number of runs during funtional structure analysis.
If not indicated, there is only 1 run with the condition PFA0.
If nfa=2, there are 2 runs with the conditions PFA0 and PFA1.
With nfa=3, there are 3 runs with PFA0, PFA1 and PFA2.
And so on.

d : Display mode: 1:no display

Example : `genbs(o=L);`

### procedure COMPLBS (n:i; /o:n);

Complets (supplements) the constructional structure within N runs.
A run contains 2 phases:
- constructive structure analysis: situation PBA1 to PBAn
- constructive structure construction: situation PBC1 to PBCn

n : Sequence of letters with the following meaning:
L = the existing structure is deleted without
further inquiry (used for 'batch' processes).

o : Display mode: 1:no display

Example : `complbs(2);`

### procedure GENFS (/o:n; d:i);

Generates a function structure output from the product master.

o : Sequence of letters with the following meaning:
L = the existing structure is deleted without
further inquiry (used for 'batch' processes).

d : Display mode: 1:no display