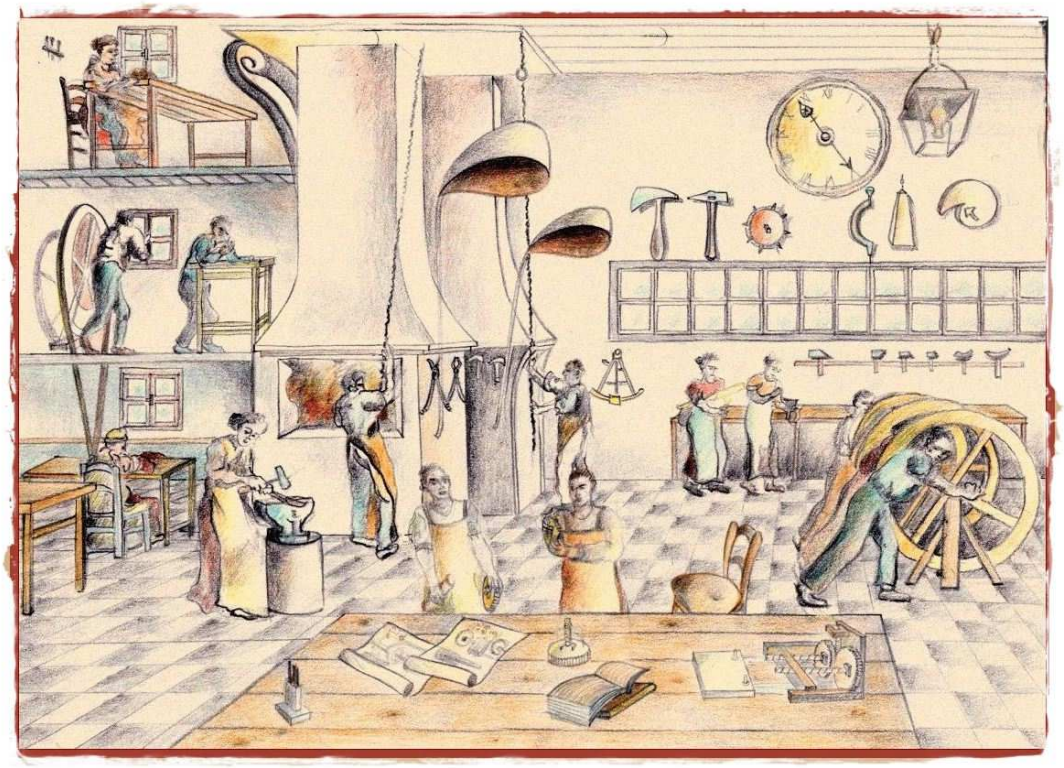
Mirakon - Referenz / Rev. 11



Know-How strukturieren und sichern
Herstellkosten optimieren
Produkte wissensbasiert konstruieren
Geschäftsprozesse planen und steuern

1 Einführung

1.1 Was ist das Mirakon-System

Das Mirakon-System ist ein Softwarebaukasten welches Ihnen ermöglicht betriebsspezifische Anwendungen schnell, flexibel und integriert zu entwickeln.

Als Nicht-Programmierer können Sie wissensbasierte Applikationen wie zum Beispiel die Vorkalkulation neuer Produkte, die automatische Konfiguration von Produkten oder die Produktionsplanung entwickeln und miteinander verknüpfen ohne Redundanzen und Schnittstellen zu schaffen.

Besonders geeignet ist das Mirakon-System für komplexe Anwendungen die viel innerbetriebliches Fachwissen erfordern. Sie können firmenspezifisches Know-how in einer Wissensbank speichern und strukturieren, so daß dieses in verschiedenen Situationen und Anwendungen genutzt werden kann.

Heterogene und dynamische Datenstrukturen lassen sich leicht handhaben. Somit können komplexe Produkte wie Maschinen und Anlagen in verschiedene Strukturen modelliert werden (Funktionsstruktur, Baustruktur, Herstellungsprozess, Kostenstruktur, ...). Dieses Modell lässt sich dann als einen einzigen Datensatz im Mirakon-Datenbanksystem speichern.

Eine Spezialität des Mirakon-System ist die Kostenfrüherkennung für Konstrukteure und Produktplaner (früher das HKB-Programm), ebenso wird das Mirakon-Sytem häufig als Konfigurationssystem für komplexe kundenspezifische Produkte, für die Produktionsplanung und Simulation und als FMEA-System eingesetzt. Sie können jedoch auch einfachere Applikationen wie Kunden- oder Lieferantenverwaltung, Finanzplanung, Lagerverwaltung selber entwickeln.

1.2 Ihre Vorteile mit Mirakon

Flexibilität:
Viele Standardpakete für vordefinierte Einsatzbereiche, auch wenn Sie parametrisiert sind, bieten viele Funktionen, die Sie nicht brauchen, andererseits vermissen Sie einige Funktionen, die Ihre realen Geschäftsprozesse abbilden. Standardpaket basieren auf schon vorgegebenen Datenstrukturen und Algorithmen. Das Resultat: Sie müssen sich an der Software anpassen und nicht umgekehrt. Mirakon bietet Ihnen eine Sprache, mit deren Hilfe Sie Ihre Strukturen, Dialoge, Algorithmen und Ausgabedokumente selber bestimmen und gestalten können. Sie beherrschen Ihre Anwendungen und sind von uns als Softwarelieferant unabhängig (und das ist gut so).

Integration:
Viele heterogene Applikationen mit allen möglichen Schnittstellen sorgen für unübersehbaren Situationen und hohen Informatik-Kosten. Eine neue Version von Applikation X erscheint und schon geht das Theater mit der Schnittstelle los ! Die Applikationen von Mirakon sind in einer gemeinsamen Datenbank und Bedieneroberfläche eingebettet und bedürfen keiner Schnittstelle. Sehr wohl erlauben sie aber Schnittstellen nach außen (ODBC oder ASCII). Häufig setzen unsere Kunden Mirakon dort ein wo die Standardsoftware ihre Grenzen hat, es gibt in der Praxis etliche leicht anzupassende Schnittstellen zu PPS-Systemen und anderen kaufmännischen Softwarepaketen.

Die Behandlung komplexer Strukturen:
Relationale Datenbanksysteme sind ungeeignet, wenn es darum geht, komplexe Produkte oder Geschäftsprozesse zu modellieren. Die Daten eines Objekts werden auf vielen Tabellen und Relationen verteilt und es entstehen starre und komplizierte Datenstrukturen. Mirakon bietet einen, von Grund auf, neuen Ansatz: frei wachsende Bäume statt starre vordefinierte Tabellen. Sie können zu jedem Zeitpunkt einem Objekt völlig neue Strukturen oder Substrukturen mit konsistenten Daten in einer Objektfamilie zuordnen.

Der wissensbasierte Ansatz:
Es ist besser eine Methode (Wissen) anzulegen, die fähig ist Bohrzeiten abhängig von Werkzeug, Werkstoff und Bohrparameter jeder Zeit zu berechnen, als tatsächliche gemessenene Bohrzeiten (Daten) von Tausenden verschiedenen Teilen zu speichern. Ein neuer Bohrwerkzeugtyp wird angeschafft und schon sind alle die Daten veraltet. Eine Methode aber, läßt sich in ein paar Minuten erweitern. Mirakon verbindet die Daten mit dem dazu gehörigen Wissen, so daß sie jederzeit neu generiert und dennoch logisch nachvollzogen werden können.

1.3 Systemarchitektur

Das Mirakon-System besteht aus 4 Komponenten:

1. **Das Mirakon-Programm**
   Unserem Mirakon-Programm und Help (Dateien MIRAKON.EXE und REF.PIB);
   Das Mirakon-Programm ermöglicht Ihnen einerseits als Autor, Wissen und Applikationen zu gestalten, zu strukturieren und zu verwalten.
   Andererseits werden für Sie als Anwender Applikationen gestartet, interpretiert und interaktiv ausgeführt.

2. **Ihrer Wissensbank**
   Die Wissensbank (KNO-Datei) enthält Wissen und Applikationen in Form von Tabellen, Funktionen, Prozeduren, Dokumenten, Dialogen und Datenstrukturen.
   Sie ist modular aufgebaut und erlaubt das "Einstecken" von Wissensmodulen von verschiedenen Autoren.

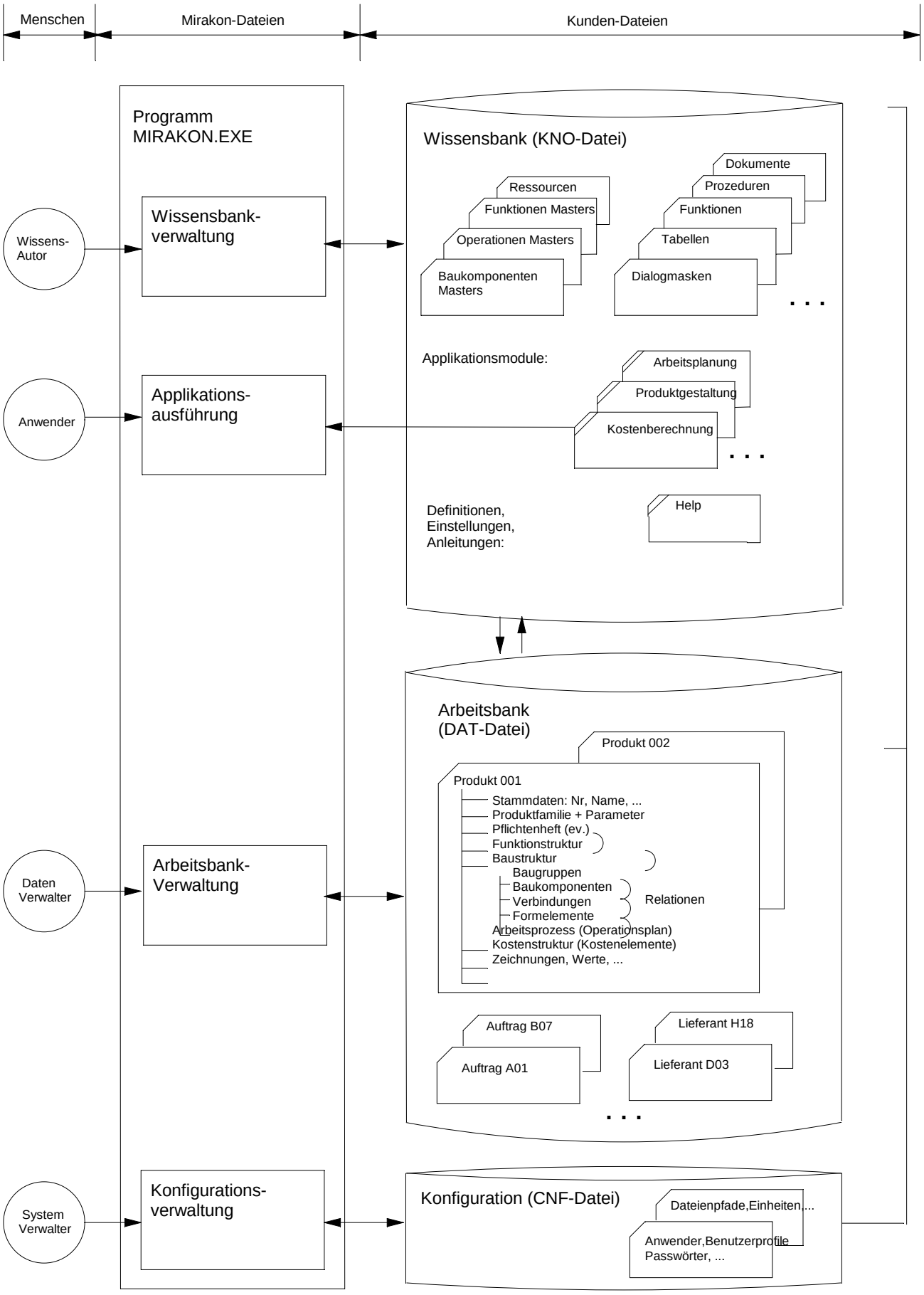
3. **Ihrer Arbeitsbank**
   Die Arbeitsbanken (DAT-Dateien) enthalten die Daten, die als Resultat der Arbeit aller Anwender entstehen: Produkte, Kunden, Arbeitspläne, Bestellungen, Fertigungsaufträge, Lieferanten, usw.

4. **Ihre Konfiguration**
   Die Konfiguration (CNF-Datei) definiert Ihr Projekt und beinhaltet den Zusammenhang zwischen allen beteiligten Dateien (Wissensbank und Arbeitsbanken).

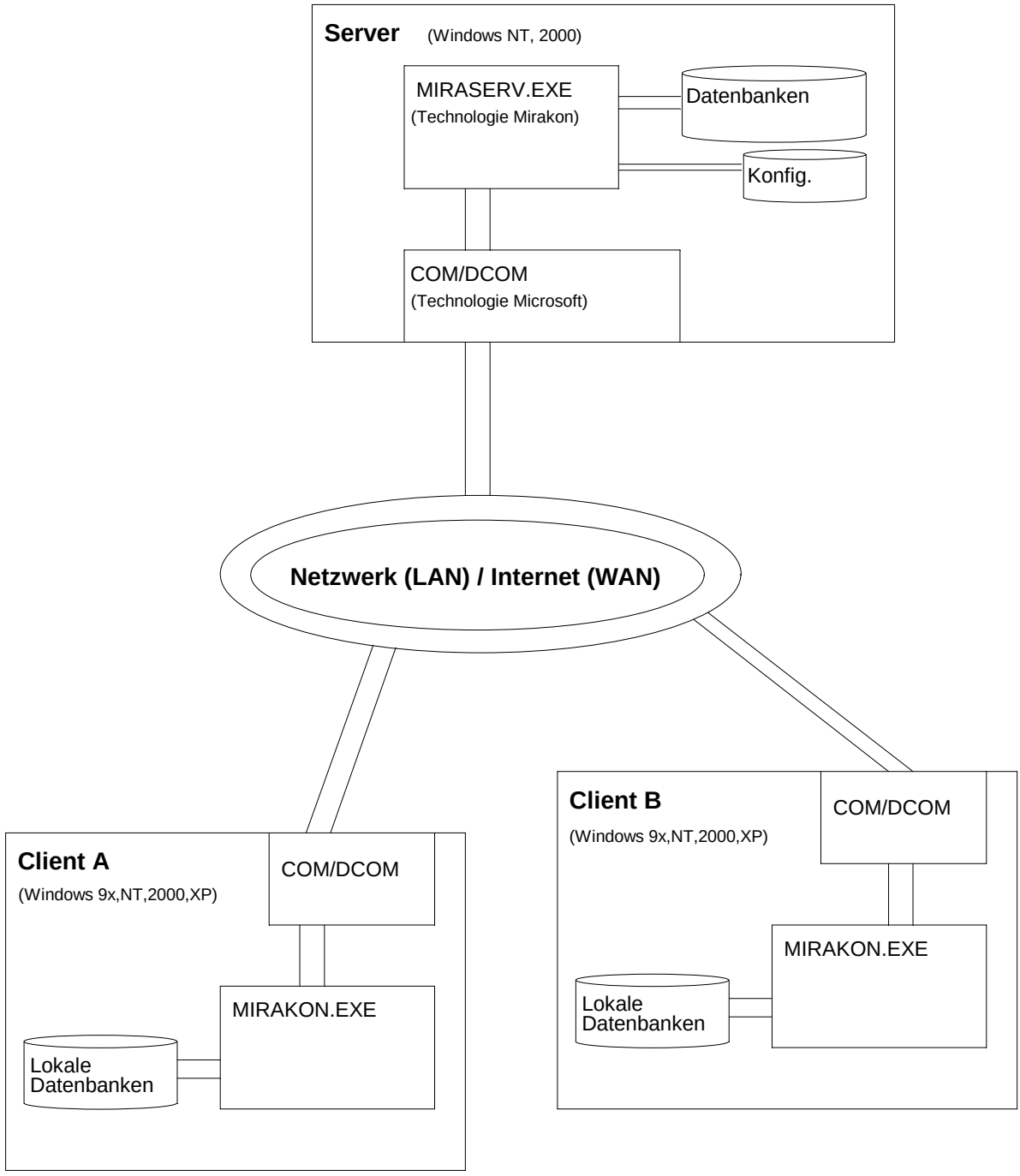
Zwei Arten von Benutzern sind zu unterscheiden:

- Die **Autoren** (Sie, Ihre Mitarbeiter oder Berater) entwickeln Applikationen, definieren Datenstrukturen und erfassen vorhandenes Wissen gemäß Ihren Spezifikationen und den Wünschen der Anwender. Autoren müssen nicht unbedingt gleichzeitig Administratoren sein.
- Die **Anwender** (Ihre Mitarbeiter und Sie) führen die Applikationen aus und verarbeiten Ihre Betriebsdaten (berechnen, planen, gestalten, offerieren, dokumentieren, ...).

Architektur vom MIRAKON-System



DV-Architektur von MIRAKON-System in Client/Server-Version



1.4 Installation

Voraussetzungen:

- Arbeitsplatz-Betriebssystem: Windows 2000, 2003, 2008, XP, Vista, 7;
- Während der Installation vom Client/Server-Version
  sollten Sie sich mit Administratorenrechte einloggen.

Softwareschutz
Wir schützen unsere standardmässig Software mit einem USB-Stick das eine kleine kundenspezifische LIC-Datei (Lizenzdatei) enthält. Diese Datei funktioniert nur mit dem gelieferten USB-Stick.
Der USB-Stick sollte vor der Installation beim Server eingesteckt sein.
Die Kunden mit einem unbegrenzten Lizenzvertrag brauchen keine Softwareschutz bzw. kein USB-Stick.

Folgende Konfigurationen bzw. Architekturen sind möglich:

1.4.1 Einzelplatz-Installation

Installationsvorgehen:

1. Mirakon-Verzeichnis anlegen: z.B: c:\programme\mirakon
und die gelieferte Dateien dorthin kopieren.

1.4.2 Client/Server-Installation

Das Server-Program läuft dauernd im Server als Windows-Programm MIRASERV.EXE,
oder als Windows-Dienst MIRASERVSRV.EXE

Das Client-Program, MIRAKON.EXE, ist im Server gespeichert, läuft aber auf dem Client-PC.

Die Konfigurationsdatei (xxx.CNF) enthält den Pfad zur Server (Server ID
oder IP-Addresse und Port-Nummer) und wird vom Client geöffnet. Somit erfährt der Client
wo sich der Server befindet.
Die Datenbanken (DAT- und KNO-Dateien) sind im Server gespeichert, und werden ausschliesslich
vom Server-Programm zugegriffen.

Drei mögliche Technologien können verwendet werden für die Kommunikation zwischen Client und Server:
- die COM/DCOM-Technologie von Microsoft (nicht weiter von Microsoft unterstützt)
- die Windows-Sockets-Methode (am einfachsten zu implementieren)
- die HTTP-Methode (für Web-Anwendungen)

Installation beim Server:

1. Mirakon-Verzeichnis anlegen (z.B: f:\programme\mirakon) und
die gelieferte Mirakon-Dateien dorthin kopieren.
Nur eine CNF-Datei darf in diesem Verzeichnis sein.

2. Konfigurationsoptionen einstellen:
- MIRAKON.EXE mit Doppelklick starten (CNF-Datei wird gelesen);
- Menuauswahl: Konfiguration -> editieren und Architektur wie folgt einstellen:
  - Architektur = Client/Server
  - Server-Id = Server-Name oder Server-IP-Adresse
  - Verbinden mit = Windows Sockets
  - Port = Portnummer (z.B: 7575)
    Portnummer sollte mit Ihrem EDV-Administrator abgestimmt sein
    und kein standard TCP/IP-Portnummer sein (grösser als 3000)
- Konfiguration speichern
- Mirakon-Programm verlassen

3A. Falls Mirakon-Server als Windows-Programm laufen soll:

MIRASERV.EXE mit Doppelklick starten: dieses Programm soll immer aktiv bleiben.

Nur bei COM/DCOM: Registrierung des COM-Objects in Windows-Registry geschieht automatisch.
Für Internet-Verbindungen, MIRASERV mit dem Parameter SRS=1 aufrufen.
z.B.: miraserv.exe SRS=1

3B. Falls Mirakon-Server als Windows-Dienst laufen soll:

MIRASERVSRV.EXE mit Parameter -INSTALL starten: Dienst wird installiert und gestartet.
(mit Parameter -UNINSTALL wird der Dienst deinstalliert)

Noch einfacher: Doppelklick auf INSTALL.BAT
(mit UNINSTALL.BAT wird der Dienst deinstalliert)

Installation bei jeder Arbeitstation:

1. Verknüpfung in Desktop anlegen (siehe Kapitel Programmstart).
Beispiel: F:\mirakon\MIRAKON.EXE F:\mirakon\kunde.cnf

Sollte MIRAKON den Server über Internet angesprechen, muss es mittels
RSID- und PORT-Parameter den Weg zur Server angeben:
Beispiele:
MIRAKON.EXE RSID=192.190.20.14 PORT=7575
MIRAKON.EXE RSID=MEINSERVER PORT=7435

2. Nur bei COM/DCOM: MIRAKON.EXE, mit der angelegten Verknüpfung,
einmal starten und schliessen (um das COM-Object in Windows zu registrieren)

1.5 Programmstart und Programmparameter

Das Mirakon-System kann auf verschiedenen Arten gestartet werden:

- Doppelklicken auf Datei MIRAKON.EXE: das Programm startet und sucht die nächstgelegene CNF-Datei als Konfiguration;

- Verknüpfung auf Ihren Desktop einrichten; diese soll MIRAKON.EXE mit Angabe der CNF-Datei als Paramater aufrufen; Beispiel: C:\mir9\mirakon.exe myconf

Bei einer Client/Server-Installation via Internet muss die Aufruf von MIRAKON.EXE
mit der Angabe des Remote-Servers-IP-Adresse und ohne Konfigurationspfad gefolgt werden:
Beispiel: c:\mir\mirakon.exe RSID=192.168.22.12

Neben der Konfiguration, können auch folgende Programmparameter und Optionen
eingegeben werden:

- Option /NOLOG überspringt die Anwenderanmeldung (login);
- Option /NI überspringt die Ausführung der Anwender-Initialisierungsbefehle;
- Option /WINMIN startet Mirakon mit minimalisierten Fenster;
- APP=[Applikations-Id] startet automatisch die angegebene Applikation;
- P1,P2,P3=... übergibt bis zu 3 Parameter an die Startapplikation;
- UID=user-Id füllt automatisch das erste Login-Feld
- UPW=user-Passwort füllt automatisch das zweite Login-Feld (Sicherheitsrisiko)
- XGA=1: Programmfenster wird in XGA-Auflösung gestartet (1024x743)

- RSID=IP-Adresse erstellt Verbindung mit Remote-Server (siehe Client/Server-Installation)
- SRS=1 startet Server als Remote Server (nur für Client/Server via Internet)
- PORT= TCP/IP-Port nummer - TCP/IP-Portnummer für Windows-Sockets verbindung
  (nur für Client/Server via Internet)

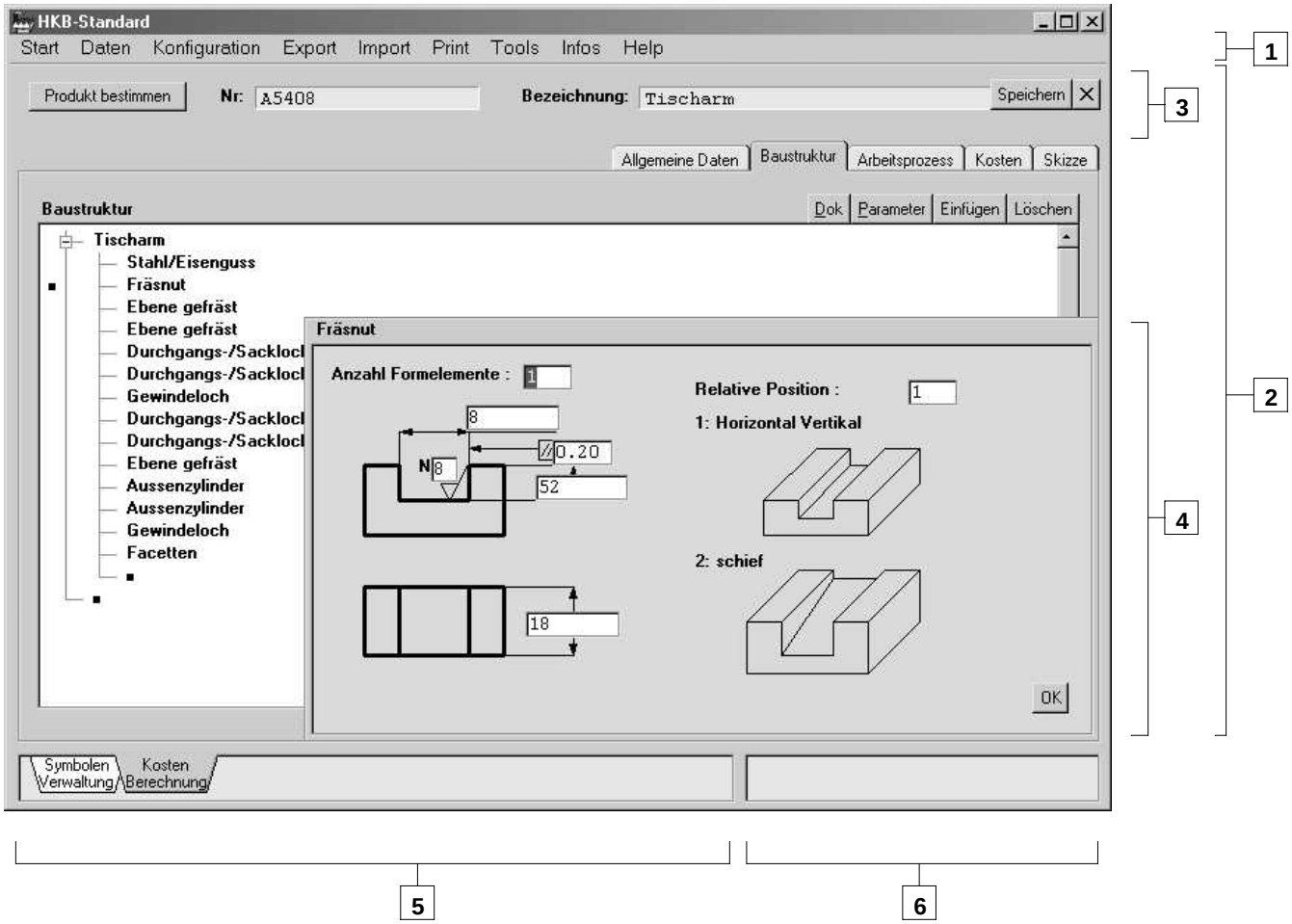
Beispiele von Programmaufrufe:

- c:\programme\mirakon\mirakon.exe myconf APP=HKB P1=120 P2=Peter /NOLOG
- \\server\mirakon\mirakon.exe \\server\mirakon\myconf

1.6 Die Bedienungsoberfläche

1.6.1 Desktop

Unten sehen Sie eine typische Anwendungs-Situation:



Die Mirakon-Fenster unterteilt sich in folgenden Bereichen:

Menuleiste [1]:
enthält allgemeine Funktionen wie Drucken, Datenexport, usw.

Applikationsframe [2]:
Gesamtarbeitsfläche die für eine Applikation zur Verfügung steht.
Enthält ein oder mehrere aufeinander folgenden Dialogfenster.
Durch das Anklicken des X-Knopfes (oben rechts neben "Speichern")
wird das Frame, und somit die Applikation geschlossen.

Statusfläche [3]:
Obere Arbeitsfläche in einem Applikationsframe, die Applikations-spezifischen Informationen und Schaltknöpfe enthält, die immer zugänglich sind.

Dialogmaske [4]:
Dialogfenster mit zusammenhändenden Dialogfelder.
Diese Dialogmasken können von Ihnen gestaltet werden.
Mit Escape-Taste oder OK-Knopf wird das Fenster verlassen.
Mit TAB-Taste oder auch Enter-Taste springt der Cursor zum nächsten Feld
Mit Shift-TAB-Taste springt der Cursor zum vorherigen Feld

Frame-Register-Feld [5]:
Dieses Feld unten links zeigt alle zur Zeit geöffneten Frames.
Durch das Anklicken des Registers mit der Maus kann der Anwender zwischen geöffneten Dokumenten oder Applikationen springen.

Nachrichten-Feld [6]:
Dieses Feld unten rechts zeigt diverse Nachrichten während einer Anwendung.

1.6.2 Menuleiste

Menuoptionen:

Start	startet eine Anwendungen;
Daten	ermöglicht das Anlegen und Gestalten von Wissensbanken und Arbeitsbanken;
Konfiguration	ermöglicht das Anlegen und Gestalten von Konfigurationen; Die Kopie-Option ermöglicht einem Client, in Client/Server-Architektur, Die Konfigurationsdatei herauszukopieren. Dabei wird die Architektur in der Konfigurationskopie auf Standard umgeschaltet.
Export	ermöglicht das Exportieren eines selektierten Objekts in die interne Ablage oder in eine externe Datei. Es können mehrere Objekte in die Ablage exportiert werden.
Import	ermöglicht das Importieren von Objekten aus der internen Ablage oder aus einer externen Datei.
Print	ermöglicht das Ausdrucken des selektierten Objekts. Kann auch mit F5 aufgerufen werden.
Tools	erlaubt das Ausführen von Befehlen, Debuggen von Abläufen, Betrachten von externen Dokumenten, Ausführen von externen Prozeduren, usw.
Infos	informiert den Anwender über Implementationsdaten, Speicher-verbrauch, momentane Werte von Variablen und Farbennummern.
Help	zeigt mehrere Anleitungen. Kann auch mit F1 aufgerufen werden.
Log Off	Erlaubt ein neuen Einloggen ohne das Mirakon-Prograsmm zu verlassen. Dies ist nur möglich wenn alle Anwendungen geschlossen sind.

1.6.3 Funktionstasten

Folgenden Tasten bzw. Tastenkombinationen können Ihre Arbeit beschleunigen:

F1	Help
F2	Speichert veränderte Objekte

F5	Print-Funktion
F11	Startet bzw. beendet Debugger
F12	Zeigt interne Datenstruktur eines Strukturelements
Ctrl-S	Speichert veränderte Objekte (gleich wie F2);
Ctrl-P	Druckt selektiertes Objekt (gleich wie F5);
Ctrl-C	Kopiert markierten Text in die Windows-Ablage;
Ctrl-V	Fügt Text aus der Windows-Ablage in der aktuellen Position ein
Ctrl-Ins	exportiert das selektierte Objekt direkt in die Mirakon-Ablage
Shift-Ins	ruft die Mirakon-Ablage um Objekte zu importieren

2 Die Konfiguration des Systems

Die Konfigurationsdatei (xxx.CNF) enthält Einstellungen und Angaben
über Anwender und Ressourcen einer Installation.

2.1 Titel

Bestimmt den Titel (Text und Bild) und Anwender einer Installation.

Titel: Text der als Windows-Fenster-Titel erscheint.

Anwendung: 4 Zeilen die in Begrüssungsfenster erscheinen.

Anwender: 3 Zeilen die im Begrüssungsfenster erscheinen.

Titelbild: Bitmap-Datei (xxx.BMP) die im Begrüssungsfenster erscheint.

Titelmaske: Dialogmaske-Datei (xxx.MSK) die das Begrüssungsfenster ersetzt.

2.2 Projektrevision

Angabe einen Projektrevisions-Nummer und -Datum (optional).
Diese Angaben erscheinen in das Begrüssungsfenster unter Anwendung.

2.3 Sprache

Setzt die Systemsprache für allen Standardtexten.
Foldende Sprachen werden unterstützt: Englisch, Deutsch, Französisch und Portuguiesisch

2.4 Architektur

Bestimmt eine von 2 Versionen: Standard oder Client/Server.

Bei der Client/Server-Version muss den Server-Id bzw. Server-IP angegeben werden.

Um die Geschwindigkeit zu steigern können die Daten-Packete, bevor sie über die
Leitung geschickt werden, komprimiert werden.

2.5 Datenbanken

Liste der Datenbanken einer Installation.

Die erste Datenbank muss die Wissensbank sein (Id=KNO).
Die zweite Datenbank muss die Arbeitsbank sein (Id=DAT).
Die weitere Datenbanken sind frei wählbar (Archive, Testdateien, usw.)

Die Identifikatoren erlauben den Zugriff auf diesen Datenbanken (mit LOAD, SAVE, usw.).

Die Pfade können absolut (c:\mir\myproj.kno) oder relativ (myproj.kno) angegeben werden.
Eine Konfiguration mit relativen Pfade lässt sich leichter transportieren.

Bei der Client/Server-Version wird für jeden Datenbank den Zugriff
angegeben: Direkt oder über Server.

Die Option "Datei mit CNF sichern" erlaubt Datenbanken mittels Passwort, so zu sichern
dass sie nur mit denselben CNF-Datei geöffnet werden können. Somit kann eine
gestohlene Datenbank, ohne Passwortangabe, nicht verwendet werden.

2.6 Startmenu

Das Startmenu wird als hierarchische Struktur von Applikationen modelliert.

In diese Struktur können Sie Gruppen, Trennlinien und Applikationsaufrufe einfügen.

Eine Applikationsaufruf besteht aus:

- ID: Identifikator der Applikation so wie es in der Wissensbank vorkommt.

- Name: Bezeichnung der Application so wie es in den Startmenu erscheinen soll.

- Bild: Bitmap dass vor der Bezeichnung in den Startmenu erscheinen kann.
  Angegeben wird die ID eines Bildes aus den Ressourcen des Moduls A000 der Wissensbank.

- Register: 2 Zeilen des unteren Applikations-Registers.

- Parameter: enthählt Variablendeklarationen die spezifisch für einen Aufruf sind.
  Sie werden vor der INIT-Prozedur angelegt und können dann während der Applikationsablauf
  mit dem Prefix APARS abgefragt werden. Beispiel: if apars.demo1: initdemo1;

- Datenbanken selektoren: definieren welche Wissensbank wird benutzt (Vorbelegung ist KNO),
  und in welche Arbeitbank werden die Objekte gespeichert.
  Der Zugriff auf die definierten Datenbanken braucht keine Identifikator anzugeben,
  z.B.: load(PROD,'001',prd). Der Zugriff auf andere Datenbanken, muß den Identifikator
  als Präfix angeben, z.B.: load(ARC.PROD,'O001',prd).

2.7 Benutzerprofile

Um Benutzer effizient und ohne Redundanzen zu beschreiben, können Sie
Benutzerprofile definieren und später mehreren Benutzer zuweisen.

Mit Knopf Einfügen können Sie Benutzerprofile anlegen und identifizieren.
Unterhalb einen Profil können Sie dann Erlaubnisse und Verbote einfügen, und somit
den Profil beschreiben (siehe nächstes Kapitel).

Ein Erlaubnis bestimmt welche Applikationen ein Benutzer starten darf und mit
welcher Zugriffsbeschränkungen. Mit der Option "Nur lesen" kann der Benutzer
Objekte zwar lesen, sehen und ändern, jedoch nicht speichern.

Ein Verbot bestimmt welche Applikationen ein Benutzer nicht starten darf.

Als Applikations-Ids können auch Filter eingesetzt werden, z.B: ART,G_,B7

2.8 Benutzer

Liste der Benutzer die sich einloggen können.
Mit Einfügen-Knopf können Sie einzelne Benutzer wie auch Gruppen einfügen.

Folgende Angaben definieren einen Benutzer:

- Id: eindeutiger Benutzer-Identifikation (1.Login-Abfrage);

- Passwort: Benutzer-Passwort (2.Login-Abfrage). Aus Sicherheitsgründen
  ist sie nicht sichtbar und wird doppelt abgefragt.

- Name: Benutzer-Name sowie es in Dokumenten erscheinen soll;

- Status: es gibt 3 Benutzer-Stati mit festen zugeordneten Bedeutungen:
  A=Anwender: hat weder Zugriff zur Daten noch zur Konfiguration;
  V=Autor/Verwalter: hat kein Zugriff zur Konfiguration;
  S=Systemadministrator: kann auf alles zugreifen;

- Code: Tekla-Befehle die unmittelbar nach dem Login ausgeführt werden.
  Beispiele: Begrüssung, Überprüfung von Mailbox, Nachrichten zeigen, usw.

- Parameter: Benutzerspezifischen Variablen die während der Applikationsausführung mit dem Präfix UP zugegriffen werden können. Beispiel: if up.abt='E7': ...;

Darüber hinaus, können Sie unterhalb einen Benutzer, Benutzerprofile, Erlaubnisse und Verbote einfügen. (siehe voriger Kapitel).

2.9 Schriftarten

Unter diesen Konfigurationskapitel befindet sich das Verzeichnis aller Windows-Schriften die in Mirakon verwendet werden sollen.
Jeder Schrift bekommt ein Nummer. Über diesen Nummer kann dann ein Schrift, bei der Programmierung eines Dokuments, bestimmt werden.

Die ersten 3 Schriften sind fest definiert:
- Font 1 = Courier New: für die Eingabefelder verwendet;
- Font 2 = Ms Sans Serif: für die Ausgabe von Standard-Texte verwendet
- Font 3 = Arial: sichert die Transportierbarkeit von bestimmten Dokumenten.

In der Konfiguration können Sie 50 Schriften definieren bzw. nummerieren.
Diese gelten für alle Applikationen in diese Konfiguration.

Damit die Schrift-Nummern stabil bleiben, dürfen Sie nur die jeweils letzte Schrift löschen, und nur zuletzt neue einfügen.

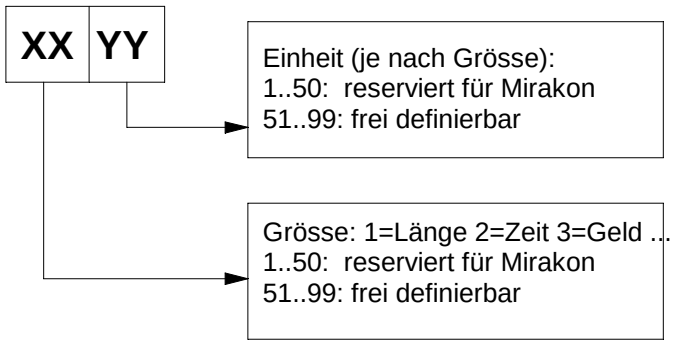
Die Schrift-Name muss exakt wie in Windows angegeben werden.
Als Kommentar können Sie z.B: den Zweck angeben (optional).

2.10 Einheiten

Die Einheiten-Tabelle enthält ein internes Verzeichnis aller vorkommende Einheiten (Zeiten, Längen, Geld, ...).

NR	NAME	ABREV	VAL1
100	Länge		
101	Meter	m	1
102	Millimeter	mm	0.001
103	Zentimeter	cm	0.01
151	Zoll	inch	0.0254
200	Zeit		
201	Minute	min	1
300	Geld		
301	Schweizer Franken	CHF	1
302	US-Dollar	US$	1.38
304	Euro	EUR	1.53

Aufbau der Einheitennummern:



Die Spalte ABREV enthält die Abkürzung der Einheit so wie diese in den Formeln und Eingaben vorkommen sollen (z.B: d1:q=80 mm;)

Die Spalte VAL1 enthält den Zahlenwert der entsprechenden Einheit in der Referenzeinheit (Einheit Nummer 1). Somit können später Quantitäten konvertiert werden.

2.11 Kalender

Unter diesen Punkt können Firmenspezifischen Ferienkalender eingefügt werden, die dann, bei der Produktionsplanung, einzelnen Ressourcen zugewiesen werden können.

2.12 Ereignisse

Unter diesen Punkt können Ereignisse definiert werden die den Server in bestimmten Zeitintervalle, automatisch ausführen soll.

Beispiel: Automatische Backups;

2.13 Tekla-Editor

Folgendes Einstellungen sind möglich:

- Ob ein Syntaxkontrolle immer beim Verlassen einen Prozedur-Editor ausgeführt wird (stark empfohlen);

- Farbe der Systemvariablen (Beispiele: ENR, PI, ENAME)

- Farbe der Kommentaren (nach /)

- Farbe der Anwendervariablen die in den folgenden Feld angegeben werden;

2.14 Helpmenu

Das Helpmenu wird als hierarchische Struktur von Help-Dokumenten modelliert.
Unsere Kunden können somit mehrere Helpdokumente, firmenspezifisch schreiben,
und den Anwender zur Verfügung stellen.

In diese Struktur können Sie Gruppen, Trennlinien und Helpdokumente einfügen.

Eine Help-Dokument ist eine Datei die in folgenden Formaten erstellt werden kann:

- Mirakon-Format (.PIB-Datei): wird von Mirakon-Viewer gezeigt und erlaubt das Suchen und Drucken in verschiedenen Formaten.

- HTML-Compiled-Microsoft-Format (.CHM-Datei): wird von Microsoft-Viewer gezeigt;

- Word-Format (.DOC-Datei): wird von Microsoft Word gezeigt;

Jede Eintragung eines Dokuments in Helpmenu benötigt:

- Name: Bezeichnung des Dokuments so wie es in den Helpmenu erscheinen soll.

- Bild: Bitmap dass vor der Bezeichnung in den Startmenu erscheinen kann.
  Angegeben wird den ID eines Bildes aus der Ressourcen des Moduls A000 der Wissensbank KNO.

- Dateiformat: PIB, CHM oder DOC;

- Datei: Dateipfad + Dateiname;

2.15 Optionen

Hier können verschiedene Optionen eingestellt werden:

- Ereignis-Protokol: wenn diese Option aktiv ist, wird eine Datei MIRAKON.LOG erstellt, und folgende Ereignisse werden dort gespeichert: Logins und Logouts, Applikationsstarts, Laden und Löschen von Objekten.
  Diese Ereignisse (maximal 5000) können dann über Infos / Ereignisprotokoll gesehen werden.

- Enter springt zum nächsten Feld:
  Wenn diese Option aktiv ist, verhaltet sich Mirakon wie folgt: nach einem Enter bei einem Dialogfeld mit zugeordneten Tekla-Befehle, springt der Cursor zum nächsten Feld, ausser der Autor es ausdrücklich mit jumpfok:=0 verbietet;
  Wenn diese Option nicht aktiviert ist, muss der Autor den Sprung mit JUMPF ausdrücklich befehlen.
  Felder ohne Tekla-Befehle sind von dieser Option nicht betroffen. Bei diesen, wird nach Enter, immer zum nächsten Feld gesprungen.

- Pfad für Objektsicherung und Fehlerberichte: hier geben Sie das Verzeichnis an, wo die Fehlerberichte und die zu sichernden Objekten, nach einen Fehler, gespeichert werden.

- Erweiterte Fehlerbericht: wenn aktiv, schreibt Mirakon ein erweiterten Fehlerbericht mit der Reihenfolge der zuletzt ausgeführten internen Befehle. Wenn Sie uns diese Dateien

senden, sind wir besser in der Lage, Fehler und Störungen schneller zu beheben.

3 Datenstrukturen

3.1 Datenorganisation

Eine elementare Information, wie z.B: die Adresse eines Kunden, muss in eine Datei, auf einen Datenträger, gespeichert werden, damit sie das Ausschalten des Computers wo sie angegeben wurde, überlebt.

Damit diese Information aber leicht wieder auffindbar bleibt, neben millionen andere, müssen diese Dateien organisiert sein, genauso wie die Bücher in einer Bibliothek.

Das Mirakon-System organisiert die Datenablage in 4 Stufen:
- Datenelement (Feld, Variable)
- Datenobjekt (Datensatz, Record)
- Datenmodul (Sammlung, Applikation)
- Datenbank (Datei)

Datenelemente sind elementare Informationseinheiten (Adresse eines Kunden) die in einem Objekt gespeichert werden müssen.

Objekte (auch Datensätze oder Records genannt) sind eindeutig nummeriert und müssen in einen Datenbankmodul gespeichert werden.

Je nach Inhalt und Zweck eines Objekts, unterscheiden wir zwischen:

- Arbeitsobjekten werden vom Anwender während einer Anwendung verändert und bearbeitet. Z.B.: Produkte, Aufträge, Kunden, usw.

- Wissensobjekte enthalten Wissen und verändern sich nicht während einer Anwendung, sondern sie werden dabei genutzt. Z.B.: Materialtabellen, Komponentenmasters, Prozeduren, usw.

Ein Datenbankmodul ist ein System von zusammenhängenden Objekte die hierarchisch in Kapiteln geordnet werden können.
Er bietet Mechanismen für die Nummerierung und Relationierung seinen Objekten unabhängig von anderen Module.

Je nach Inhalt und Zweck unterscheiden wir zwischen 3 Kategorien von Modulen:

- Sammlung: Menge von gleichartigen Objekte (Kunden, Artikeln, Aufträge, usw.);

- Applikation: Menge von (programmier-)Objekten die eine Applikation definieren: Prozeduren, Dialogmasken, Tabellen, Grafiken, usw;

- Wissensmodul: Menge von Objekten die Wissen enthalten: Materialtabellen, Dokumenten, Berechnungsmethoden (Prozeduren), Bilder, usw.
  Die Gruppierung von Wissensobjekten in einen Modul kann nach verschiedenen Kriterien geschehen, meistens sind es:
  - Objekte die thematisch zusammenhängen, z.B: Hydraulik
  - Objekte die von gleichen Autor stammen, z.B: Einkaufsprozesse

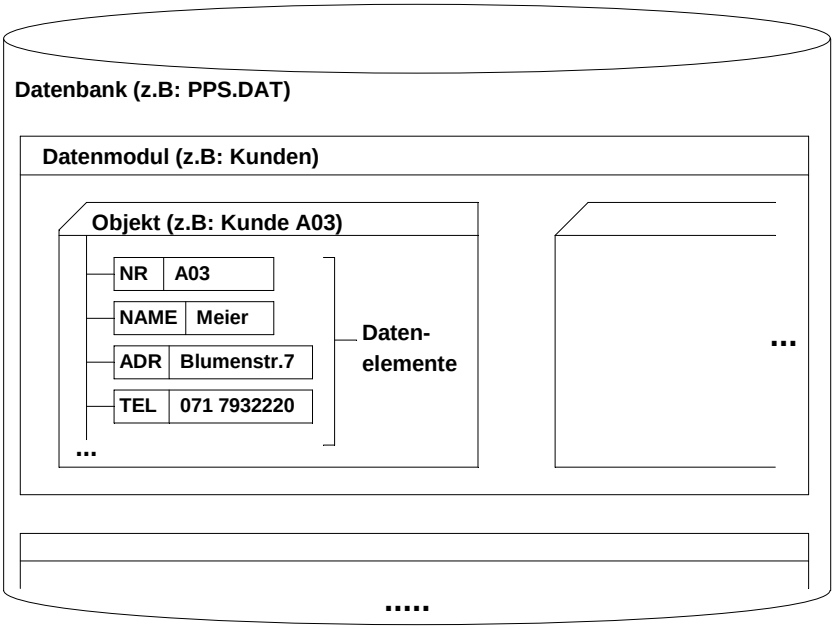
Ein Datenbankmodul muss in einer Datenbank angelegt und gespeichert sein.

Eine Datenbank ist eine Datei. Je nach Inhalt und Zweck unterscheiden wir zwischen 3 Kategorien von Mirakon-Datenbanken:

- Arbeitsbanken (DAT-Dateien): enthalten meistens Sammlungen von Arbeitsobjekten;

- Wissensbanken (KNO-Dateien): für Wissensobjekte (Tabellen, Prozeduren, ...)

- Konfiguration (CNF-Datei): Datenbank mit vordefinierte Struktur, die Informationen über die Installation (Projekt) enthalten.



3.2 Wissensbank

Basismodul A000
Die Standard-Wissensbank (KNO) muss ein Basismodul A000 enthalten.
Dieser enthält zentralen Daten und Prozeduren, die von jedem anderen Modul, ohne Angabe von 'A000' zugegriffen werden können.

Die INIT-Prozedur enthält alle globalen Variablen und Definitionen, die im System ständig vorhanden sein müssen. Diese werden bei der ersten Anwendung automatisch deklariert und initialisiert.

3.3 Applikationen

Eine Applikation muss eine INIT-Prozedur enthalten. Diese deklariert alle globalen Variablen und Definitionen, die während der Anwendung ständig vorhanden sein müssen.
Die INIT-Prozedur wird als erstes nach dem Start der Applikation ausgeführt.
Die INIT-Prozedur eine Applikation X wird auch ausgeführt, wenn eine andere Applikation Y, auf Objekte von X zugreift. Dies erlaubt die Deklaration von Variablen die eventuel nötig sind. In diesem Fall sollte die INIT-Prozedur von X, den Dialogstart mit "at START:" schützen.

Beispiel :
```
av1:r; / allg.Variable
av2:s; / allg.Variable

at START:        / Dialog wird gestartet
openstat(STAT);
```

```
opentabs(0,0,0,'',MAIN);
```

Variablen mit dem SAVE-Attribut werden vom Speichersystem überwacht. Wenn sie verändert werden, können sie mit dem Speichern-Knopf gespeichert werden. Der Applikationsautor kann optionell eine Prozedurname als zweiten Parameter des SAVE-Attributs eingeben. In diesem Fall wird diese Prozedur immer vor dem Speichern ausgeführt.

Beispiel : `pr:l&save(PROD,eingabesichern); /Produkt`

3.4 Sammlungen

Transaktionen
Im Laufe der Zeit entstehen neue Bedürfnisse und, als Folge davon Veränderungen in der Datenstruktur bei den neuen, wie auch bei den bereits vorhandenen Objekten in der Arbeitsbank. Solche Veränderungen lassen sich mit Transaktionen leicht durchführen.

Eine Transaktion wird durch eine Prozedur beschrieben die für jedes Objekt ausgeführt wird. Das System lädt jedes Objekt in die interne Variable o:list, danach werden die Transaktionsbefehle ausgeführt (die das Objekt o:list verändern) und zum Schluss wird das Objekt o:list wieder gespeichert.

Somit können Werte verändert, neue Felder angelegt bzw. alte gelöscht, Felder umbenannt und Relationen verändert werden.

3.5 Tabellen

Der Tabelleneditor ist ein Frame, in dem Tabellen bearbeitet werden.

Eine Tabelle besteht aus Zeilen und Spalten. Eine Zeile besteht aus einer Zeilen-Nummer und mehreren Zellen. Eine Zelle kann aus mehreren (!) Textzeilen bestehen.

Eine Tabelle wird in zwei Ebenen bearbeitet:

1. Bewegungsebene für das Springen von Zelle zu Zelle. Dazu werden die Cursortasten oder der Maus benutzt.

2. Eingabeebene für das Schreiben innerhalb einer Zelle.

Funktionsknöpfe und Tasten in der Bewegungsebene:

- Edit-Knopf, Enter-Taste, Buchstabe oder Zahl: Eintritt in der Eingabeebene. Die selektierte Zelle wird zur Dateneingabe geöffnet.

- + Knopf oder + Taste: Wenn eine Spalte oder Zeile selektiert ist, vergrössert die Spalte bzw. Zeile um ein Zeichen bzw. Zeile.

- - Knopf oder - Taste: Wenn eine Spalte oder Zeile selektiert ist, verkleinert die Spalte bzw. Zeile um ein Zeichen bzw. Zeile.

- Einfügen-Knopf oder INSERT-Taste:
  Wenn eine Spalte oder Zeile selektiert ist, fügt eine neue Spalte bzw. Zeile rechts bzw. oberhalb vom Cursor.
  Wenn eine Zelle selektiert ist wird sie gezoomt und zur Dateneingabe geöffnet (gleich wie Zoom-Knopf).

- Löschen-Knopf oder DELETE-Taste: Löscht die selektierte Zelle, Spalte oder Zeile. Wenn mehrere Zeilen markiert sind werden sie alle gelöscht.

- Zoom-Knopf oder INSERT-Taste:
  Wenn eine Spalte oder Zeile selektiert ist, wird sie, je nach Zustand verkleinert oder vergrössert.
  Wenn eine Zelle selektiert ist wird sie gezoomt und zur Dateneingabe geöffnet (gleich wie Ins-Knopf).

- Legende-Knopf: Zeigt einen editierbaren Kommentar zur gesamten Tabelle.

- Dok-Knopf: Zeigt eine editierbare Grafik. Damit kann eine Zelle dokumentiert werden.

- Shift-Down-Taste: Wenn eine Zeile selektiert ist, wird sie, je nach Zustand markiert oder unmarkiert. In markierten Zustand erscheint die Zeile mit invertierten Farben.

- Import und Export-Funktionen (Menuleiste): Damit können selektierte Zellen, selektierte oder markierten Zeilen von Tabelle zur Tabelle transferiert werden. Wenn der Cursor ganz oben links steht (NR-Feld) kann die ganze Tabelle transferiert werden.

- Mit Rechtsklicken der Maus, sind andere Funktionen abrufbar, z.B. die Option Zellenformat erlaubt Hintergrundfarbe und Schriftart einer Zelle zu definieren.

Funktionsknöpfe und Tasten in der Eingabeebene:

- Enter-Taste oder Esc-Taste: Zelle wird verlassen und Editor kehrt zurück in der Bewegungsebene.

- Ctrl+Cursortasten, Tab-Taste oder Shift-Tab-Taste: Zelle wird verlassen und die Nachbarzelle geöffnet.

3.6 Grafiken

Der Grafikeditor dient zur Gestaltung von Grafiken und Dialogmasken.

Mit dem Seiten-Selektor kann bei mehrseitigen Dokumenten die gewünschte Seite eingestellt werden.

Mit der Maus können ein oder mehrere Grafikelemente selektioniert werden:
- durch einfaches Ancklicken (linke Maustaste) auf einzelne Grafikelemente;
- oder durch Ziehen einen Viereck um mehrere Grafikelemente.
Wenn ein selektiertes Grafikelement angecklickt wird, wird es deselektiert.

Die DELETE-Taste löscht alle selektionierte Elemente.

Die Knöpfe [+] und [-] vergrössern bzw. verkleinern die Grafik um 20%.

Der Einfügen-Knopf bietet folgende Funktionen:

- Linie oder L-Taste: zeichnet eine gerade Linie oder Polyline.
  1. Fadenkreuz mit Maus oder Pfeiltasten bewegen und Anfangs- und Endpunkte durch kurzen Anklicken der linken Maustaste oder Enter-Taste setzen.
     Dieser Schritt wiederholen bis der gewünschte Linienzug gezeichnet ist.

2. Funktion verlassen mit der rechten Maustaste oder mit der Esc-Taste.

- Text oder T-Taste: schreibt einen mehrzeiligen Text.
  1. Gewünschter Text im Texteditor schreiben und mit Esc-Taste beenden.
  2. Textbox mit Maus oder Pfeiltasten in die gewünschte Position bringen und linken Maustaste kurz anklicken oder Enter-Taste drücken.

- Bogen: zeichnet einen eliptischen Bogen mit wählbaren Radien und Winkel.
  1. X- und Y-Radien (mm), Start- und Enwinkel (grad) eingeben und OK drücken.
  2. Bogen mit der Maus oder Pfeiltasten positionieren.

- Fläche: füllt einen Linienzug mit einen Farbmuster.
  1. Die Ecken des zu füllenden Polygons mit der Fadenkreuz eingeben
  2. Funktion verlassen mit der rechten Maustaste oder mit der Esc-Taste.

- Pfad: zeichnet einen Linienpfad bestehend aus geraden Segmenten und/oder Kreisbogen.
  1. Fadenkreuz mit Maus oder Pfeiltasten bewegen und Anfangs- und Endpunkte durch kurzen Anklicken der linken Maustaste oder Enter-Taste setzen.
  2. Um ein Kreisbogen zu zeichnen, vor der Eingabe des letzen Punktes, rechte Maustaste drucken und Option Kreisbogen wählen, letzter Punkt eingeben und danach Kreismittelpunkt eingeben. Um wieder auf gerade Segmente umzuschalten: nochmals rechte Maustaste drucken und Option Linie wählen.
     Diese Schritte wiederholen bis der gewünschte Linienzug gezeichnet ist.
  3. Funktion verlassen mit der rechten Maustaste oder mit der Esc-Taste.
  4. Zu einem Pfad kann einen inneren Pfad zugeordnet werden (eine Insel oder Kern). Dies ist zweckmässig wenn der äusseren Pfad, ohne Kern, gefüllt werden soll. Vorgehen: aussere Pfad selektieren, rechts klicken, Umformen, Innere Pfad einfügen, Pfad eingeben gemäss Schritte 1 bis 3.
  5. Pfad füllen: Pfad selektieren, rechts klicken, Attribute, Flächenparameter eingeben.

- Box: zeichnet und füllt einen Rechteck mit einen Farbmuster.
  1. Oberen linken Ecke des gewünschten Rechtecks mit der Fadenkreuz eingeben. Nicht Ziehen, sondern kurz anklicken.
  2. Unteren rechten Ecke des gewünschten Rechtecks eingeben.

- Symbol: zeichnet Symbole aus dem Symbolkatalog.
  1. Gewünschter Katalog auswählen.
  2. Gewünschter Symbol selektieren.
  3. Gewähltes Symbol positionieren.

- Dialogfeld/(Feld): zeichnet ein Dialogfeld.
  1. Dialogfeld-Daten eingeben und OK-Knopf bestätigen.
  2. Dialogfeld mit der Maus oder Pfeiltasten positionieren.

Der Knopf Einstellen bietet folgende Funktionen:

- Zoomen oder Z-Taste: vergrössert einen Zeichnungsauschnitt.
  Der Ausschnitt-Rechteck wird wie bei der Box-Funktion eingegeben.

- Alles sehen oder 0-Taste: zeigt die gesamte Grafik.

- Ursprungsansicht oder 1-Taste: zeigt die Grafik in der ursprünglich

formatierte Grösse.

- Textattribute: ändert die Voreinstellung für die einzufügenden Texte.

- Grafikattribute: ändert die Voreinstellung für die einzufügenden Elemente.

- Seitenformat: ändert Grösse, Farbe und Position der Dialogmaske, sowie die Seitenränder für den Druck.

- Raster: definiert den Abstand der Orientierungspunkte, die im grafic Fenster erscheinen.

- Auflösung oder R-Taste steuert die Genauigkeit der Tastatur- und Mausbewegungen (snap).

- Initialisierung: enthält Befehle, die vor dem Maskeneröffnung ausgeführt werden, wie z.B: Variablen deklarationen, Menulisten, u.s.w.

- Eröffnung: enthält Befehle, die nach dem zeichnung des Dialogfelde ausgeführt werden, wie z.B.: um zu dialogfeld Status ändern, usw.

- Finalisierung enthält Befehle, die vor dem Maskenverlassen ausgeführt werden, wie z.B: Plausibilitätskontrollen, Warnungen, u.s.w.
Üblich werden hier die Systemvariablen EXITOK, OKAYED, ESCAPED verwendet.

- Layers: definiert die sichtbaren Layers.

Mit der rechten Maustaste können Grafikelemente selektioniert und verändert werden. Folgende Funktionen bieten sich an:

- Löschen: löscht alle selektionierte Elemente;

- Parameter: verändert den Inhalt eines Textes oder Dialogfelds;

- Attribute: ändert das Aussehen der vorher selektierten Elementen;

- Bewegen: bewegt die selektierten grafischen Elemente; diese können mit der Maus oder Pfeiltasten an die neue Position bewegt werden und dann mit der linke Maustaste oder Enter-Taste fixiert werden.

- Skalieren vergrössert oder verkleinert ein oder mehreren Elemente;
Vorgehen: X-Y-Skalierungsfaktoren eingeben und OK-Taste drücken.

- Rotieren dreht ein oder mehrere Elemente um einen gewünschten Winkel.
Vorgehen: Winkel eingeben und OK-Taste drücken; Elemente drehen sich im Gegenuhrzeigersinn.

- Kopieren: kopiert ein oder mehreren Elemente.
  1. Kopien mit der Maus oder Pfeiltasten mehrmals positionieren.
  2. Funktion verlassen mit der rechten Maustaste oder mit der Esc-Taste.

- Gruppieren: verbindet mehrere Elemente zu einer Gruppe.

- Trennen: trennt die einzelnen Elemente einer Gruppe.

- Umformen / Punkte ändern: ändert die Position einzelner Ecken von Linien oder Flächen.
  1. Gewünschte Eckpunkte selektieren und neu positionieren.

2. Funktion verlassen mit der rechten Maustaste oder mit der Esc-Taste.

- Umformen / Punkte einfügen: fügt neue Eckpunkte in Linien oder Flächen.
  1. Position für den neuen Punkt mit dem Maus (linke Taste) selektieren
  2. Funktion verlassen mit der rechten Maustaste oder mit der Esc-Taste.

- Umformen / Punkte löschen: löscht Eckpunkte von Linien oder Flächen.
  1. Gewünschte Eckpunkte selektieren.
  2. Funktion verlassen mit der rechten Maustaste oder mit der Esc-Taste.

- Ordnung: ändert die Ordnung in welcher die graphischen Elemente gezeichnet werden.
  - Nach hinten: das selektierte Element wird zuletzt gezeichnet.
  - Nach vorn: das selektierte Element wird zuerst gezeichnet.

- Spiegeln: kopiert die selektierte Element spiegelverkehrt nach Eingabe des gewünschten Spiegelachsenwinkels.

- Identifizieren: ordnet einen Identifikator dem selectierten Element zu. (um später z.B. mittels SETF zugegriffen werden zu können)

Während des Positionierens, kann man mit der X-Taste, eine Position numerisch eingeben.

Mit Import/aus Textdatei können Grafikdateien importiert werden. Folgende Formate werden unterstützt:

- Bitmap-Bilder: BMP-Dateien;
- Zeichnungen: DXF-Dateien;
- Komprimierte Bilder: JPG-Dateien;
- Ikonen: ICO-Dateien;

Vorgehen:
1. Aus der Menuleiste: Import/aus Textdatei wählen;
2. Datei wählen;
3. Bildrahmen mit der Maus positionieren;
4. Linke maustasten drucken;

Mit Import/aus Windows-Ablage können Bitmaps aus anderen Applikationen transferiert werden.

Vorgehen:
1. Bitmap aus eine andere Applikation in die Windows-Ablage kopieren, meistens mit Ctrl-C oder Ctrl-Ins;
2. Mirakonfenster aktivieren und Grafikeditor öffnen;
3. Aus der Menuleiste: Import/aus Windows-Ablage wählen;
4. Bildrahmen mit der Maus positionieren;
5. Linke maustasten drucken;

3.7 Dialogmasken

3.7.1 Dialogfelder

Jedes Dialogfeld enthält:

- ID: Identifikator (alphanumerisch). Dieser bestimmt die Reihenfolge der Feldsprünge wenn der Anwender die Dialogmaske bedient.

- Variable-ID: Name der Variable die vom Feld gezeigt bzw. verändert wird.

- Ev. Feld-Abmessungen: in Milimeter oder Texteinheiten.

- Optionen: Selektierbarkeit, usw.

- Befehle: werden beim Deselektieren immer ausgeführt: nach Enter, Up, Down, Tab, Shift-Tab oder Mouse;
  Mit der Variable FCHANGED (integer) kann abgefragt werden ob die Feldvariable geändert wurde.
  Nach der Ausführung der Befehle wird das Feld verlassen, ausser die Systemvariable JUMPFOK = 0;

- Tip: Text, der erscheint, wenn die Maus für eine Sekunde über das Feld stoppt.

3.7.2 Frage

Eine Frage ist ein Dialogfeld, das die Eingabe einer Textzeile erlaubt

Pfeiltasten: Left, Right, Home oder End-Taste bewegt den Cursor.

DELETE-Taste: löscht das Zeichen unter dem Cursor.

Backspace-Taste: löscht das Zeichen links vom Cursor und der Cursor rückt an diese Stelle.

Shift-F8: löscht das gesamte Dialogfeld.

Das System kontrolliert jede Eingabe und sorgt dafür, dass ungültige Zeichen nicht akzeptiert werden.

Zwischen mehreren Fragen wird mit den Cursortasten Up, Down, Tab, Shift-Tab oder mit Enter gewechselt.

Frage-Feld-Optionen:

- SEL=0: Frage ist nicht selektierbar
- A=Ausrichtung: bestimmt Textausrichtung: 1=linksbündi 2=rechtsbündig
- L=Anzahl Zeichen: maximale Textlänge (wenn es länger als Feldbreite sein soll)
- TC=Textfarbe: bestimmt Textfarbe
- BC=Backgroundfarbe: bestimmt Hintergrundfarbe

3.7.3 Menu

Ein Menu ist ein Dialogfeld, das die Auswahl einer Option aus einer hierarchischen Liste von Optionen erlaubt.

Die Bedienung erfolgt ähnlich wie die Windows-Menus.

ENTER-Taste: Enthält eine Option untergeordnete Optionen (Pfeil am Schluss der Option), werden diese geöffnet bzw. geschlossen (Zoomen).

HOME-Taste: setzt den Cursor an den Anfang des Menus.

END-Taste: setzt den Cursor an den Schluss des Menus.

Menu-Feld-Optionen:

- ATPOS=Prozedurname: Prozedur wird immer ausgeführt wenn Option selektiert wird

3.7.4 Knopf

Der Knopf ist ein Dialogfeld, der das Ausführen einer Operation bewirkt.

Selektierbare Knöpfe (z.B. OK-Knopf) müssen selektiert und bestätigt
werden, um die Funktion ausführen zu lassen.

Die Bestätigung erfolgt durch Anklicken mit der linken Maustaste oder
durch drücken der ENTER-Taste.

Knopf-Feld-Optionen:

- SEL=0: Knopf ist nicht selektierbar
- KEY=Tastennummer: Knopf kann mit angegebenen Taste aktiviert werden; z.B: key=27;
- TF=Fontnummer: bestimmt Zeichensatz
- TH=Fonthöhe: bestimmt Zeichensatz grösse
- TS=Fontstyle: bestimmt Zeichensatzstyl: 0=Normal, 1=Italic
- TW=Fontdicke: bestimmt Zeichensatzdicke: 0=Normal, 1=fett
- A=Ausrichtung: bestimmt Textausrichtung: 1=linksbündig 2=rechtsbündig
- IHA=Zahl: Horizontale Bildausrichtung: 1=links, 2=rechts, 3=zentriert
- IVA=Zahl: Vertikale Bildausrichtung: 1=oben, 2=unten, 3=zentriert
- IW=Zahl: Bildbreite in mm
- IH=Zahl: Bildhöhe in mm

3.7.5 Selektor

Ein Selektor ist ein Dialogfeld und wird als Umschalter verwendet.
Durch das Anklicken der Randknöpfe bzw. Drücken der Links/Rechts-Tasten
wird die nächste Option selektiert.

3.7.6 Textdisplay

Ein Display zeigt einen bestimmten Wert in Textform und erlaubt keine Eingaben.

Display-Feld-Optionen:

- TC=Textfarbe: bestimmt Textfarbe
- BC=Backgroundfarbe:bestimmt Hintergrundfarbe
- TF=Fontnummer: bestimmt Zeichensatz
- TH=Fonthöhe: bestimmt Zeichensatz grösse
- TS=Fontstyle: bestimmt Zeichensatzstyl: 0=Normal, 1=Italic
- TW=Fontdicke: bestimmt Zeichensatzdicke: 0=Normal, 1=fett
- A=Ausrichtung: bestimmt Textausrichtung: 1=linksbündi 2=rechtsbündig
- FTYP=Variablentyp: gezeigte Wert ist das Ergebnis einer Funktion (=Variablenname)

3.7.7 Grafikdisplay

Ein Grafikdisplay zeigt einen Bild und erlaubt keine Eingaben.

3.7.8 Lister

Ein Lister ist ein Dialogfeld, das die Eingabe einer Liste von heterogenen Elementen erlaubt.

Funktionsweise:

INSERT-Taste: Fügt ein Element aus einen assoziierten Menu in der Liste ein.

ENTER-Taste: Ermöglicht die Eingabe von Parameter des selektierten Elements.

DELETE-Taste: Löscht das selektierte Element aus der angezeigten Liste.

Lister-Feld-Optionen:

- INS=0: Einfügen nicht erlaubt
- DEL=0: Löschen nicht erlaubt
- Q=0: data.q:q wird nicht automatisch eingebaut
- H=Prozedurname: Handlerprozedur wird zugewiesen
- SS=0: Substruktur nicht erlaubt

3.7.9 Texteditor

Ein Texteditor erlaubt die Eingabe von mehrzeiligen Texten.

Funktionsweise der Tasten:

Enter: Setzt den Cursor auf eine neue Zeile.

Backspace: Fügt 2 Textzeilen zusammen. (Der Cursor muss am Zeilenanfang stehen)

Ctrl-Y: Löscht die Zeile, in der der Cursor steht.

Shift-F6: Markiert/unmarkiert die Zeile, in der der Cursor steht.

Shift-F7: Fügt oberhalb des Cursors eine leere Zeile ein.

Shift-F8: Löscht die Zeile, in der der Cursor steht. (Alternative: Ctrl-Y)

Shift-F9: Kopiert die markierte(n) Zeile(n) dorthin, wo der Cursor steht.

Shift-F10: Verschiebt die markierte(n) Zeile(n) dorthin, wo der Cursor steht.

Editor-Feld-Optionen:

- L=Anzahl Zeichen: maximale Textlänge
- H=Anzahl Zeilen: maximale Texthöhe
- T=1: Tekla-Syntax wird hervorgehoben

3.7.10 Strukturedítor

Der Strukturedítor ist ein Dialogfenster, in dem Strukturen beschrieben werden.

Eine Struktur besteht aus einer Hierarchie von Elementen.
Ein Element wird aus einer Masters-Tabelle ausgewählt, parametrisiert und in die Struktur eingefügt.
Jedes Element kann Unterelementen enthalten.

Funktionsweise:

- INSERT-Taste oder Einfügen-Knopf: fügt ein Element aus einen assoziierten Menu in die Struktur ein.

- ENTER-Taste oder Doppelclick: öffnet bzw. schliesst die Hierarchiestufe des selektierten Elements.

- Plus Taste (+): öffnet die Hierarchiestufe des selektierten Elements.

- Minus Taste (-): schliest die Hierarchiestufe des selektierten Elements.

- Stern Taste (*): öffnet alle Hierarchiestufen unter den selektierten Element.

- P-Taste, -> oder Parameter-Knopf: ermöglicht die Eingabe von Parameter des selektierten Elements.

- DELETE-Taste oder Löschen-Knopf: Löscht das selektierte Element.

- Rechte Maustaste: eröffnet ein Kontextmenu, falls es eine zugewiesen wurde.

3.7.11 Gitter

Zweck
Ein Gitter-Feld erlaubt die Eingabe von tabellenförmigen Daten.

Feld-Optionen
- INS=0: Einfügen nicht möglich
- COL1=0: erste Spalte (NR-Spalte) wird nicht gezeigt
- SCROLL=0: scrollbars werden nicht gezeigt
- SCROLL=1: vertikale scrollbar werden immer gezeigt (Vorbelegung)
- SCROLL=2: scrollbars werden nur gezeigt wenn alle Gitterzeilen nicht in Feldfenster sichtbar sind
- O=1: zeigt Spaltentitel als Knopf. Mit Doppelklick werden Zeilen nach diese Spalte geordnet
- L=1: ermöglicht Ziehen der Spaltenbreiten mit Maus
- EXP=1: ermöglicht export nach Excel mit rechten Mausklick
- F=S: Kleinschrift wird verwendet (small font) => Gitterfeld wird kleiner
- LC=<Fabennummer): bestimmt Zeilenfarbe
- HS=1: Horizontal scrollbar wird gezeigt
- FL=Zeilennummern: Fixiert angegebenen Zeilen; Beispiel: FL=(1,2,3);
- CLIP=1: Cut&Paste-Funktion wird aktiviert
- X=1: Excel-Eingabe-Modus
- A=0: "Aktiv suchen"-Funktion wird ausgeschaltet
- DOC=1: erlaubt Zellen zu dokumentieren: Rechtsklicken auf Zelle und "Dokument" wählen
- E=1: Möglichkeit Formeln einzugeben: wenn eine Zelle selektiert ist,
  = eintippen, danach erscheint den Formel-Editor.
  Beispiele von Formeln:
  =L1C2+L5C3 /Summe der Werte von Zeile1/Spalte2 mit Zeile5/Spalte3
  =sum(1,2,10,2) /Summe der Werte von Zeile1/Spalte2 bis Zeile10/Spalte2

- TITH=n Setzt die Titelzeilenhöhe auf n Zeilen
- G=1 erlaubt das Zeichnen von Grafiken in der Titelzeile

Spaltenbeschreibung
Befehle in der Definition der Spalten:

T: Spaltentitel
W: Spaltenbreite
C: Textfarbe einer Spalte, sowohl für allen Zeilen wie auch für einzelne Zeilen
Beispiel A: C12 = rot für alle Zeilen
Beispiel B: Cp.farbe = verwendet Farbe die in Zeilenparameter FARBE angegeben ist
B: Hintergrundfarbe einer Spalte, sowohl für allen Zeilen wie auch für einzelne Zeilen (ähnlich wie C)
V: Variable die gezeigt wird
H: definiert welche Handler-Prozedur verwendet wird (z.B: was geschehen soll at RCLICK)
F: Format des angezeigten Wertes (Anzahl Kommastellen, Datumformat)
A: Ausrichtung des Textes: A0=linksbündig A1=zentriert A2=rechtsbündig)
J1:Zellen enthalten eine Checkbox. Die Zellenvariable muss vom Typ Integer sein !
Je nach Zustand der Checkbox, die mit Doppelklick bedient wird, enthält die Variable 0 oder 1.
K + Variable: Zelle erscheint als Knopf. Die Variable enthält die Name der Prozedur die aufgerufen wird wenn der Knopf mit Doppleklick gedruckt wird.
L: Menuoptionenliste für Combox-Input
D: Combox-Höhe in Zeilen (muss vor L-Befehl geschrieben werden)
Q + Anzahl Zeichen: 1) für Combox-Zellen: Combox-Breite (muss vor L-Befehl geschrieben werden)
2) für Frage-Zellen: max. Eingabelänge in Zeichen (kann länger als Zellenbreite sein)
G + Farbennummer: Zellen-Hintergrundfarbe falls ihr Inhalt mauell verändert wird

Beispiele:

```
cols.0:s='A1,W5,Vp.a,Bp.bca,HHgrid';
cols.0:s='TProzess,Vp.name:s,Ip.prozess:s,D12,Q30,Lprocessmenu,W15';
```

Feldsteuerung
Im Handler-Prozedur (eine pro Spalte) kann das Verhalten der Gitterzellen bestimmt werden:

Beispiele:

```
at POSENTER    /nach drucken der Enter-Taste
  jumpfok:=2;  /cursor springt: 2=nach rechts, 3=nach links, 4=nach unten, 5=nach oben

at KRIGHT:     /nach drucken der Pfeiltaste nach rechts
  openmsg('Kright','');

at KLEFT:      /nach drucken der Pfeiltaste nach links
  openmsg('Kleft','');

at KUP:        /nach drucken der Pfeiltaste nach oben
  openmsg('Kup','');

at KDOWN:      /nach drucken der Pfeiltaste nach unten
  openmsg('Kdown','');

at DRAWC:      /zeichnet Spaltentitel
  drawline(xlfe,ytfe,xrfe,ybfe); /zeichnet Diagonale von obenlinks nach untenrechts
```

Bemerkungen
Cut&Paste-Funktion für Blöcke funktioniert mit Ctrl-C Ctrl-V (Windows-Ablage).
Die Mirakon-Ablage gilt nur für ganze Zeilen.

3.7.12 Tabelle

Eine Tabellenfeld ist ein Dialogfeld, das die Eingabe einer Tabelle erlaubt.

Funktionsweise:

- Einfügen-Knopf oder INSERT-Taste:
  Wenn eine Spalte oder Zeile selektiert ist, fügt eine neue Spalte
  bzw. Zeile rechts bzw. oberhalb vom Cursor.

- Löschen-Knopf oder DELETE-Taste:
  Löscht die selektierte Zelle, Spalte oder Zeile.

3.7.13 Checkbox

Die Checkbox ist ein Dialogfeld, der ein Ja/Nein Eingabe erlaubt.

Die Umschaltung erfolgt durch Anklicken mit der linken Maustaste oder
durch drücken der Leerzeichen-Taste.

3.7.14 Combobox

Ein Combobox ist ein Menufeld der nur eine Zeile beansprucht und sich
erst öffnet wenn der Anwender den Pfeil-Knopf betätigt. Je nach Position
des Felds auf den Bildschirm öffnet sich das Menu nach unten oder nach oben.
In geöffneten Zustand erfolgt die Bedienung wie das Menufeld.

Combobox-Feld-Optionen:

- DEL=0: Delete-Taste löscht Inhalt

3.7.15 Radioknöpfe

Radioknöpfe sind kreisförmigen Dialogfelder die die Auswahl einer von
wenigen Optionen erlaubt. Wenn einer Option aktiviert wird, wird die
zuletzt aktivierte deaktiviert.

Die Auswahl einer Option erfolgt durch Anklicken mit der linken Maustaste
oder durch drücken der ENTER-Taste.

3.7.16 Elastische Dialogmasken

Nicht immer kann der Wissensbankautor davon ausgehen, dass alle Anwender die gleiche Bildschirmauflösung zur Verfügung haben. Oft, bei Präsentationen liefert der Beamer eine niedrigere Auflösung als der Bilschirm am Arbeitsplatz. Die Folgen sind Dialogmasken mit geschnittene Felder die deshalb leer erscheinen. Dadurch kann die Anwendung unbrauchbar werden. Wenn der Autor sich an die minimale Auflösung hält, erscheinen die Dialogmasken, in 90% der Fälle, viel zu klein. Die so wichtige Bildschirmfläche wird also vergeudet.

Mit der sogenannten "elastische" Dialogmasken, kann dieses Problem gelöst werden.

Solche Masken passen sich an der zur Verfügung stehenden Fläche an, und der Author kann das Verhalten jedes Dialogfeldes festlegen.

Vorgehen

1. Im Dialogmasken-Editor: -> Einstellen -> Layout: Checkbox "Elastisch" aktivieren
2. Für jeden elastischen Feld: Rechtsklicken auf Dialogfeld -> "Verhalten bei Grössenänderung" auswählen und Verhalten beschreiben.

Beispiel

Ein quadratischen Dislpayfeld mit 100mm Kantenlänge wird elastisch angelegt mit Verhalten = Vergrössern und Verkleinern, und Maximale Veränderung = 50% Breite und 25% Höhe.

- Wenn die Dialogmaske automatisch um 20% vergrössert wird, weil mehr Platz zur Verfügung steht, wird das Displayfeld auf 120x120mm vergrössert.
- Wenn die Dialogmaske um 100% vergrössert wird, wird das Displayfeld, gemäss maximal-Werte, auf 150x125mm vergrössert.
- Wenn die Dialogmaske um 40% verkleinert wird, wird das Displayfeld, gemäss maximal-Werte, auf 60x75mm verkleinert.

4 Die Sprache Tekla

4.1 Sprachelemente

Die TEKLA-Sprache besteht aus folgenden Sprachelementen bzw. Begriffen:

Variable: Ist ein logischer Speicherplatz und besitzt einen Namen, einen Typ und einen Wert.
Damit der Speicherplatz für eine Variable reserviert wird, muss diese deklariert werden.

Typ: Bei der Programmausführung werden Werte eingegeben, berechnet und in Variablen gespeichert. Diese Werte können verschiedener Natur (Typ) sein: Zahlen, Texte, usw.
Der Typ einer Variable legt fest, welche Werte dort gespeichert werden sollen und welche Operationen darauf angewendet werden können.

Ausdruck: Ein Wert-Ausdruck ist eine Formel bestehend aus Operanden (Konstanten und Variablen), Operatoren und Funktionsaufrufe die einen Wert ergeben.
Die Benutzung von mehrfach geschachtelten Klammern wie in der Algebra ist erlaubt.

Befehl: Ein Befehl ist eine Anweisung die dazu dient Variablen zu deklarieren oder Werte zu berechnen, zu verändern oder zu transportieren.

Funktion: Eine Funktion ist ein Wert-Generator. Sie ist kein Befehl sondern teil eines Ausdrucks.
Sie besteht aber intern aus Befehlen die den Rückgabe-Wert errechnen.

Objekt: Ein Objekt ist eine zusammenhängende Liste von identifizierten Werten die sich in einer Datei als ganzes (Datensatz) speichern lässt.

Die ersten drei Werte sind immer:
- Nummer (NR:string)
- Bezeichnung (NAME:string)
- Status (STAT:recstatus)

Ein Objekt wächst und schrumpft mit der Anwendung und muss keine vordefinierte Datenstruktur haben.

Je nach Zweck, unterscheiden wir zwischen:

- Arbeitsobjekten werden vom Anwender während einer Anwendung verändert und bearbeitet. Z.B.: Produkte, Aufträge, Kunden, usw.

- Wissensobjekte enthalten Wissen und verändern sich nicht während einer Anwendung, sondern sie werden dabei genutzt. Z.B.: Materialtabellen, Komponentenmasters, Prozeduren, usw.

Tabelle: Ein Objekt, das eine Menge von Ausdrücken, Befehle und Kommentare enthält, die in Matrixform angeordnet sind. Eine Tabellenzelle kann mehrere Textzeilen enthalten.

Dialogmaske: Ein Dialogfenster mit zusammenhängenden Dialogfeldern und Grafikelementen.
Wenn sie aufgerufen wird, findet ein Dialog mit dem Anwender statt.
Eine Dialogmaske ist ein Objekt.

Datenbank: Eine Datei die eine Menge von zusammenhängenden Objekten enthält.
Ein Datenbank ist in Module gegliedert und für den raschen Datenzugriff organisiert.
Je nach Zweck der darin enthaltenen Objekte unterscheiden wir zwischen:
- Wissensbanken: für Wissensobjekte (Tabellen, Prozeduren, ...)
- Arbeitsbanken: für Arbeitsobjekte (Adressen, Artikeln, Aufträge, ...)

Modul: Eine Menge von zusammenhängenden Objekten in eine Datenbank.
Wir unterscheiden zwischen:

- Wissensmodul: enthält hierarchisch gegliederten Wissensobjekten die meistens von einem Autor stammen;

- Applikation: beschreibt eine Anwendung und enthält: Initialisierung, Hauptmenu, Prozeduren, Funktionen und Dialogmasken;

- Sammlung: enthält gleichartigen Arbeitsobjekte die nach ihrem Nummer geordnet sind;

Konfiguration: Datei mit Informationen über das Projekt: Titel, Anwender, Wissensbank-Datei, Arbeitsbank-Datei, ...

4.2 Typen

Es gibt elementare Typen und zusammengesetzte Typen.

Die elementare Typen lassen sich nicht weiter unterteilen.

Die zusammengesetzte Typen lassen sich in Felder aus elementaren Typen unterteilen.

Elementare Typen

b	Byte = Ganze Zahlen zwischen 0 und 255.
i	Integer = Ganze Zahlen zwischen -32000 und 32000 .
a	Adresse = 32-bit Zahl.
c	Currency = Reelle Zahl mit genaue Rundung bis 4 Dezimalstellen. Wichtig für Finanzbuchhaltung, denn reelle Zahlen werden nicht immer genau gerundet.
r	Real = Reelle Zahlen zwischen -1E18 und +1E18 .
s	String = Zeichenketten bis maximal 255 Zeichen.
n	Name = Ununterbrochene Zeichenkette die ein bestimmtes Datenobjekt (Variable, Tabelle, Strukturelement) bezeichnet.
d	Datum = Zeitpunktangabe mit Jahr, Monat, Tag, Stunde, Minute und Sekunde
q	Quantity = Reelle Zahl (Wert) + Einheit (Name).
l	List = Liste mit Variablen verschiedenen Typs.
li	List of Integer = Liste mit ganzen Zahlen.
lr	List of Real = Liste mit reellen Werten.
ls	List of String = Liste mit Zeichenketten (Text).
ln	List of Name = Liste mit Namen.
^...	Zeiger auf ... (Beispiele: pos1:^e; pr2:^r; pl3:^l;)

Zusammengesetzte Typen

Der Typ ELEMENT ist der Baustein für die Aufbau von Datenstrukturen
und wird deshalb an dieser Stelle erwähnt.

Weitere zusammengesetzte Typen werden später bei der jeweiligen Thema erläutert.

e Element; besteht aus folgenden Felder:
- ID:s; (Modul\Tabelle.Zeilennummer).
- NAME:s; (Bezeichnung).

- PARS:I; (Parameter liste) .
- DATA:I; (Weitere Attribute: Menge, Ressourcen, ...).
- SS:I; (Substruktur).

4.3 Variablen

Es gibt Systemvariablen die von Mirakon-System deklariert werden
und es gibt solche die der Tekla-Autor frei definieren kann.

Der Tekla-Autor kann Variablen mit gleichen Name wie die Systemvariablen
deklarieren. Dies hat aber zur Folge dass die Systemvariablen ausser Kraft
gesetzt werden, und wird deshalb nicht empfohlen.

Bezüglich ihren Lebensdauer können die Variablen global oder lokal sein.

Globale Variablen werden in der Initialisierung der Module deklariert (INIT-Prozedur)
und leben so lange wie die Applikation angewendet wird.

Lokale Variablen werden in Prozeduren, Dialogmasken und Tabellen deklariert
und leben nur so lange wie die Prozedur, Dialog oder Tabelle aktiv ist.

4.4 Situationen

Situationen sind Namen mit festgelegter Bedeutung.
Diese geben den Tekla-Autor die Möglichkeit bestimmte Befehle nur dann auszuführen,
wenn eine bestimmte Situation (im Ablauf der Anwendung) zutrifft.

Beispiel: Einem Artikel-Master werden Aktionen zugeordnet (A-Spalte), die aber
nicht alle und immer ausgeführt werden sollen. Mit den AT-Befehl kann man
situationsbezogenen Befehle programmieren.

4.5 Operatoren

Arithmetische Operatoren (ergeben real oder integer Werte):

+ Addition

- Subtraktion

***** Multiplikation

/ Division

Booleansche Operatoren (ergeben 0 oder 1):

= Gleichheit

Ungleichheit

in Einschliessung (braucht Leerzeichen davor und danach !)
Beispiel: if id in ('A_','B_'):inf('found');

out Ausschliessung (braucht Leerzeichen davor und danach !)
Beispiel: if nr out (1,4,12):inf('ok');

< Kleiner

> Grösser

<= Kleiner oder gleich

>= Grösser oder gleich

not Verneinung

and UND-Verknüpfung

Beispiel: if (a>20) and (b<350): inf('allowed');

or ODER-Verknüpfung

String Operatoren (ergeben string Werte):

+ Addition
Beispiel: title:s='Chapter '+nr; /nr ist auch ein String

***** Multiplikation
Beispiel: line:s=75*'-';

~ Negation: für negative Filter:
Beispiel: loader(...,gf='~A_');
/Filter für alle Objekte die nicht mit A beginnen

Namen Operatoren (ergeben namen Werte):

[ ] Auswertungsoperators
Beispiel: path:n=pr.bs.[i].ss.[id1];
/i:integer und id1:string

Zeiger Operatoren (ergeben Werte vom Typ der gezeigte Variable):

^ Dereferenziert Zeiger und ergibt die gezeigte Variable
Beispiel: x1:r:=pr2^; (pr2 ist ein Zeiger vom Typ ^R)

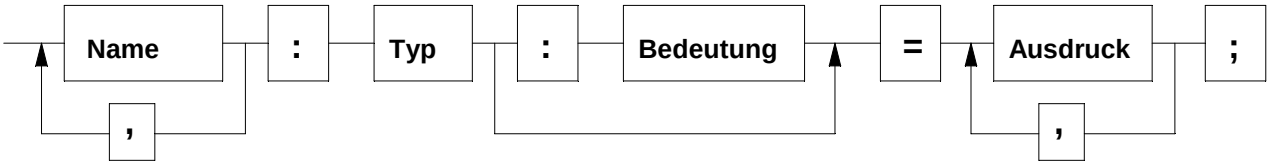
Entscheidungsoperator (für alle Typen):

{ } Ergibt den Wert der die erste Bedingung erfüllt.
Syntax: {Bedingung1:Wert1, Bedingung2:Wert2, sonstiger Wert};
Beispiel: x:r={length>10:2.8,width>200:3.5,5.7};

4.6 Befehle

4.6.1 Variablendeklaration

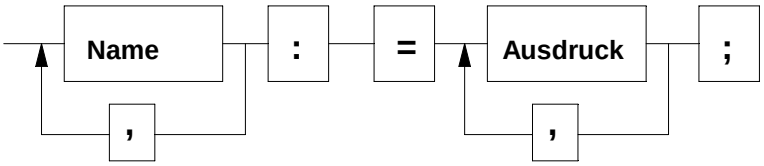
Die Variablendeklaration reserviert Speicherplatz für eine Variable.



Beispiel :
```
x:r:temperatur in grad;
x,y:r=2;    /ergibt x=2 y=2
x,y:r=2,5;  /ergibt x=2 y=5
l1:li=(1,2,3);
```

4.6.2 Wertzuweisung

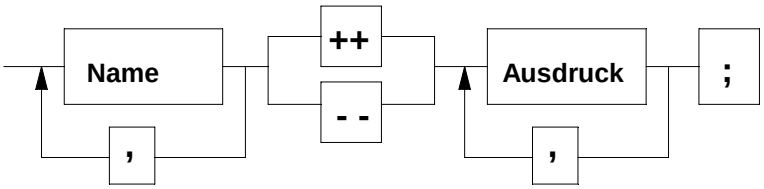
Die Wertzuweisung füllt den Speicherplatz einer Variable mit einem Wert.



Beispiel :
```
s1:='ABC';
x,y:=6,x/2; /ergibt x=6  y=3
```

4.6.3 Werterhöhung bzw. -verminderung

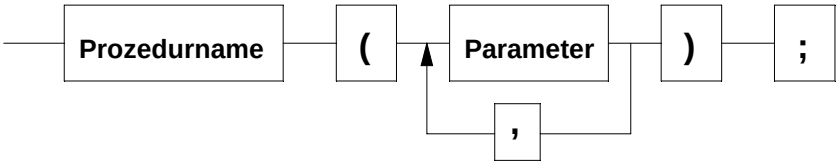
Addiert bzw. subtrahiert einen Wert (Zahl oder String) an einer Variable.



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

4.6.4 Prozeduraufruf

Die Prozeduraufruf überträgt die Parameter an die Prozedur und führt die dort enthaltenen Befehle aus.



Es gibt 2 Arten der Parameterübergabe:

1) Wertparameter: die Prozedur erwartet einen Wert für interne Berechnungen und Entscheidungen.

2) Variablenparameter: die Prozedur erwartet eine Variable in diese sie einen berechneten Wert ablegt. Dieser Wert wird somit exportiert. Diese Parameter müssen mit VAR als Präfix in dem Prozedurkopf deklariert werden.

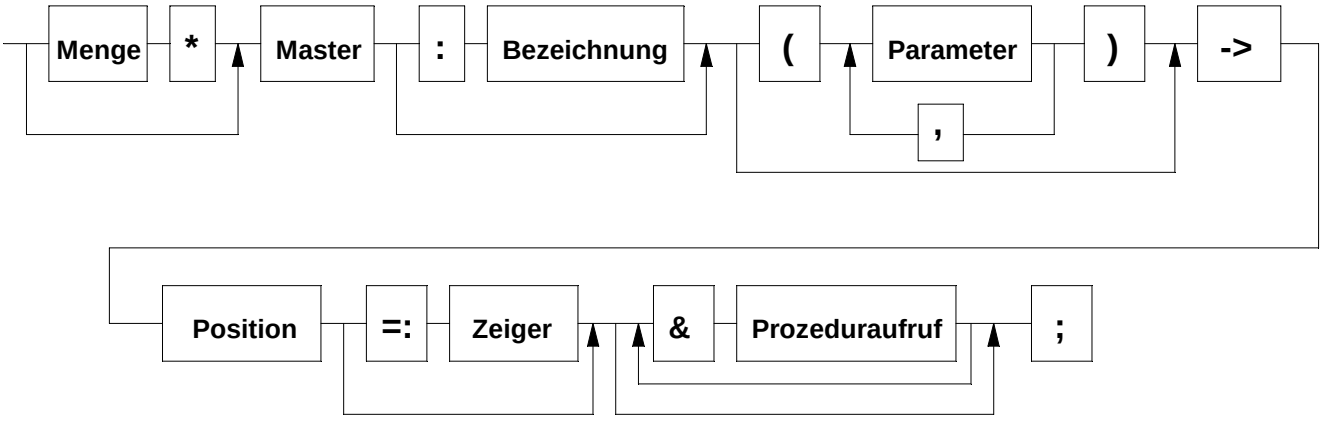
TEKLA bietet auch die Möglichkeit optionale Parameter zu deklarieren. Alle optionalen Parameter müssen mit einen / nach allen festen Parametern deklariert werden. Beim Prozeduraufruf, muss bei der Parameterübergabe von optionalen Parametern der Parametername mit = bezeichnet werden.

Beispiel :
```
/ Prozedurdeklarationen:

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

procedure calcd(d:d; /var jahr,mon,tag:i);
  /...
  end;

/ Prozeduraufrufe:

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

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

4.6.5 Einbauaktion

Die Einbauaktion fügt ein Element in eine Struktur ein.



Das eingebaute Element ist immer eine Instanz (Konkretisierung) eines Masters das in eine Wissensbank-Tabelle definiert wird.

Mit dem &-Operator, am Ende der Einbauaktion, können verwandte Prozeduren aufgerufen werden:

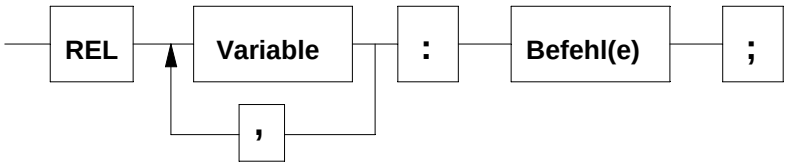
- &id=string : weist einen Identifikator dem eingebauten Element zu.
- &nss : eingebautes Element erhält keine Substruktur

Beispiel :

```
TAB1.A1->pr.bs.1;  /einfachste Einbaubefehl

p1:^e;
2*A2(10,'A')->ss.0=:p1;
/p1 zeigt nun auf das eingebaute Element

MOD5\TEILE.A3:'Welle links'(p.d,p.l+2)->p1^.ss.0;
/Welle wird unterhalb des von p1 gezeigten
/Elements eingebaut

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

4.6.6 Relationsbildung

Die Relationsbildung bildet eine Relation zwischen 2 oder mehreren Datenelementen/Variablen. Die Relation besteht aus Befehle die dann ausgeführt werden wenn irgendeine der relationierten Variablen sich ändert. Mit @1, @2, ..., @n werden die relationierten Variablen in den Befehlsteil erwähnt.



Beispiel :
```
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 Datenstruktur-Definition

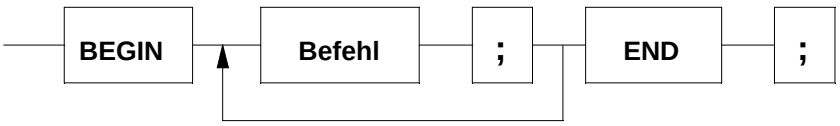
Eine Datenstruktur kann mit = definiert werden.

Beispiel :
```
buch=
  autor:s;
  jahr:i;
  verlag:s;
  end;

/Danach kann eine strukturierte Variable
/deklariert werden:

meinbuch:buch;

/ Wenn BUCH ein Master ist, kann in der PARS-Spalte
/ :buch  geschrieben werden (!)
```

4.7 Kontrollstrukturen

4.7.1 BEGIN-END-Befehlsblock

Mit BEGIN ... END können mehrere Befehle als Block,
einer Bedingung oder Wiederholungsschleife zuordnen.



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

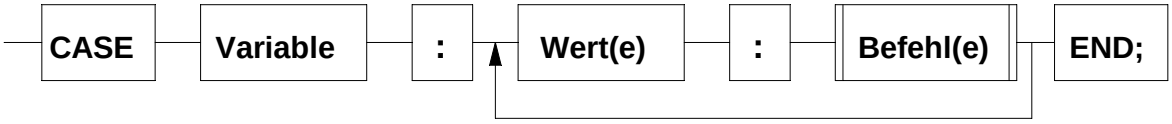
4.7.2 IF-Anweisung

Bedingte Ausführung von Befehle.

Beispiel :
```
if a>b:
  begin
  a++1;
  inf('ok');
  end;
else inf('error');
```

4.7.3 CASE-Anweisung

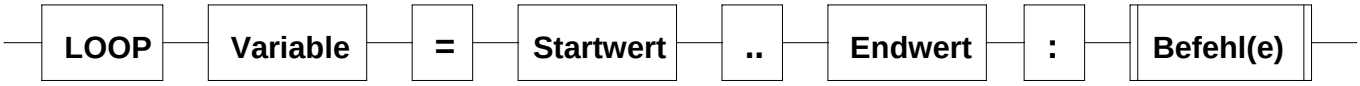
Bedingte Ausführung von Befehle je nach Wert
einer Variable. Diese Variable kann vom Typ
Integer, Currency, Real, Quantität, Datum, String oder Name sein.



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

4.7.4 LOOP-Anweisung

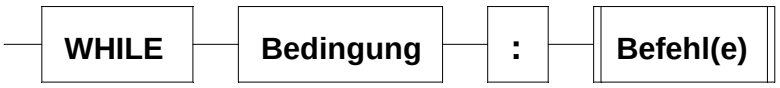
Wiederholt eine Befehlsblock n Mal.



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

4.7.5 WHILE-Anweisung

Wiederholt eine Befehlsblock bis eine Bedingung zutrifft.



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

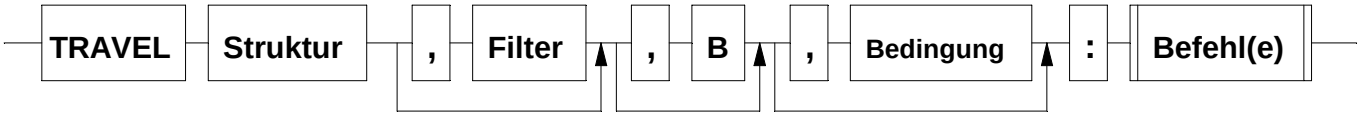
4.7.6 TRAVEL-Anweisung

Arbeitet sich durch eine Struktur (von oben nach unten) und führt die angegebenen Befehle für jedes Strukturelement aus. Damit die Daten der einzelnen Elemente greifbar sind, wird jedes Element zuerst aktiviert. Somit beziehen sich die Systemvariablen D, P, SS, ENR, EMOD, ENAME, EP, EH, PE auf das jeweilige Element.

Optionel kann ein Elementenfilter als zweiten Parameter (string) und Optionen (name) als dritte Parameter angegeben werden.

Optionen: B=backwards;

Als vierte Parameter können auch Bedingungen angegeben werden;



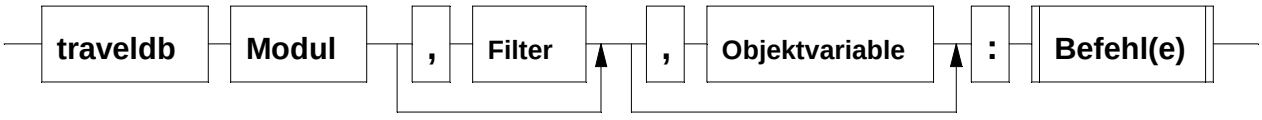
Beispiel :
```
travel pr.bs:
  begin
  writes(1,2,1,ep);
  writeq(0,14,1,d.q);
  writes(0,20,1,ename);
  end;

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-Anweisung

Arbeitet sich durch eine Arbeitsbank in alphabetischen Reihenfolge und führt die angegebenen Befehle für jedes Objektnummer aus. Die jeweilige Objektnummer ist in der Systemvariablen KEYI:string enthalten. Diese Variable KEYI ist nur gültig innerhalb der TRAVELDB-Befehle.

Optionel kann ein Objektnummerfilter als zweiten Parameter (string) und eine Objektvariable (list) als dritte Parameter angegeben werden.

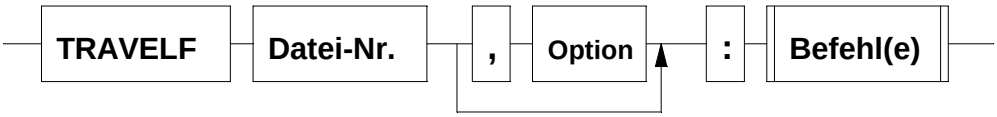
Wenn diese Objektvariable angegeben wird, wird jedes Objekt darin geladen.



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

4.7.8 TRAVELF-Anweisung

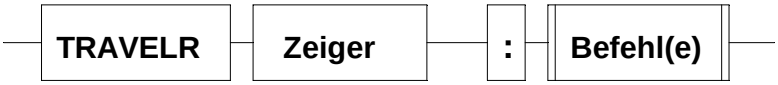
Geht durch eine Datei, byte für byte und führt für jeden Abschnitt die angegebenen Befehle aus. Die Abschnitte werden durch die Angabe eines Separators in der Variable TFS gebildet.
Optionen: L = Separator ist Carriage Return Line Feed.



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

4.7.9 TRAVELR-Anweisung

Arbeitet sich durch alle Relationen einer durch einen Zeiger gezeigten Variable und führt die angegebenen Befehle für jede Relation aus. Damit die einzelne Relationen greifbar sind, werden die Systemvariablen RTYP, RREF, PRE1, PRE2 und RDATA aktualisiert.



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

4.7.10 EXIT-Anweisung

Verlässt die aktive Prozedur.

4.7.11 CONTINUE-Anweisung

Bricht die momentane Runde in der aktive LOOP/WHILE/TRAVEL-Schleife ab und startet die nächste.

Beispiel :
```
travel pr.bs:
  begin
  if eh>2:continue;
  writes(1,20,1,ename);
  end;
```

4.7.12 BREAK-Anweisung

Verlässt die aktive LOOP/WHILE/TRAVEL-Schleife.

Beispiel :
```
travel pr.bs:
  begin
  if ei>10:break;
  writes(1,20,1,ename);
  end;
```

5 Die Mirakon-Bibliothek

Die Mirakon-Bibliothek enthält bereits programmierte Variablen, Typen, Prozeduren und Funktionen die aus der Applikationen aufgerufen werden können.

Darin enthalten sind auch Schemas für die Lösung komplexeren Problemen.

Ein Schema ist ein funktionell zusammenhängenden System von Sprachelementen.

Beispiel: Das Schema für die Aufbau von Strukturen besteht aus der Prozedur EDITST, die Variablen ENR, PE, ENAME, usw., die Situationen I1, I2,usw., die Händlerprozedur, die Master-Tabelle für die Strukturelementen, die entsprechenden Dialogmasken.

5.1 Allgemeines zur Programmsteuerung

Variablen und Konstanten

uid	n	Benutzer-Id
uname	s	Benutzer-Name
us	n	Benutzer-Status
up	l	Benutzer-Parameter-Liste
errormode	i	Wenn 1: Berechnungsfehler werden nicht angezeigt.
mirrev	s	Mirakon-Revision, z.B.: '10.128';
lang	i	Eingestellte Sprache (in der Konfiguration): 1=englisch 2=deutsch 3=französich 4=portuguiesich

Situationen für die Programmsteuerung

START	Nach dem Start einer Applikation

function EX (v:n):r;

Ergibt 1 falls die Variable V existiert bzw. 0 falls sie nicht existiert.

Beispiel : `if ex(pr.status)=0: inf('Status nicht vorhanden');`

function TYPOF (v:n):s;

Liefert den Typ eine Variable namens V.

Beispiel : `x:r; t:s=typof(x); /ergibt t='R'`

function SIZEOF (v:n):a;

Ergibt den Speicherverbrauch (in RAM) der mit V angegebenen Variable.

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

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

Setzt ein Bit in einer Variable.

bitnr : Bit-Position: 1 bis 16

bitval : Bit-Wert: 0 oder 1

v : Variable die geändert wird; muss von Typ Integer sein;

Beispiel : `a:i=0; setbit(3,1,a); /ergibt a=4`

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

Ergibt den Bitwert einer Variable.

bitnr : Bit-Position: 1 bis 16

v : Variable die geändert wird; muss von Typ Integer sein;

Beispiel :
```
a:i=4;  b:i=bit(3,a);  /ergibt b=1
```

procedure CLOSEPROG;

Beendet das Mirakon-Programm.

procedure CLOSEFRAME;

Beendet die aktuelle Applikation bzw. Frame.

procedure WAIT (n:i);

Wartet N hunderstel Sekunden.

Beispiel :
```
wait(200);   /wartet 2 Sekunden
```

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

Führt die im CODE enthaltenen Befehle aus.

code : Befehlsliste

o : Optionen: N: Fehler werden nicht angezeigt;

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

Führt den mit COM angegebenen Befehl aus.

com : Befehl

o : Optionen: N=Fehler werden nicht angezeigt;

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

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

Führt den angegebenen Windows-Programm aus (EXE-Datei).
Damit können externen Programme ausgeführt werden.
Wenn einen Dokumentdatei (z.B: DOC-Datei) angegeben wird, wird
das zugeordnete Windows-Programm aufgerufen.

cmd : Programm- oder Dokumentdatei.

o : Optionen:
- M = Programm erscheint in maximalen Fenster;
- W = Mirakon wartet mit der Ausführung der nächsten Tekla-Befehlen bis den ausgerufenen Programm geschlossen wird.

params : Aufrufparameter

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

procedure EXECSHELL (cmd:s);

Führt den angegebenen Windows-Befehl aus (Ausführungsbefehl).

cmd : Windows-Befehl

Beispiel : `execshell('net send XYZ nachricht von Meier');`

procedure DEBUGGER (op:i);

Startet den Debugger (op=1) und beendet ihm (op=2) und
zeigt die abgelaufene Anweisungen.

Beispiel :
```
debugger(1);
berechne1;
debugger(2);
```

function WINUID:s;

Ergibt den Windows-User-Id.

Beispiel : `if winuid='FM': inf('Guten Tag Herr Ferdinand Müller !');`

5.1.1 Fehler suchen und finden

5.1.1.1 Der Debugger

Mit dem integrierten Debugger können interne Abläufe detailliert durchleuchtet werden. Dies hilft enorm bei der Suche nach Fehler und zeigt wie programmierten Abläufe funktionieren. Der Debugger ist ein Werkzeug der wie ein Protokoll-führer funktioniert. Er zeigt alle ausgeführten Befehle sowie den Wert aller Variablen bei deren Ausführung.

Vorgehen:
1. Applikation starten und bis vor dem Fehlerauftritt ausführen;
2. F11-Taste drucken: Der Debugger wird gestartet;
3. Fehler verursachen;
4a. Bei der Fehlermeldung: Knopf Debugger drucken
oder (wenn keine Fehlermeldung erscheint)
4b. F11 erneut drucken: Debugger zeigt detaillierte Ablaufstruktur;

Bei Programmabstürze sollte der Debugger aus der Tools-Menü mit der Option "erzeugt DEBUG.TXT" schreiben eingeschaltet werden.
Nach der Absturz ist die Ablaufstruktur in der Datei DEBUG.TXT enthalten und kann mit einem beliebigen Texteditor eingesehen werden.

5.1.1.2 Datenstrukturen sehen

Im Strukturedítor können Sie mit F12 die Datenstruktur des selektierten Elements betrachten.

Mit der Menüleistenoption Infos/Systemvariablen können Sie den momentanen Wert aller Systemvariablen betrachten.

Mit der Menüleistenoption Infos/Applikationsvariablen können Sie den momentanen Wert aller, im laufender Applikation deklarierten Variablen betrachten.

Mit der Menüleistenoption Infos/Wissensbankvariablen können Sie den momentanen Wert aller in Basismodul deklarierten Variablen betrachten.

5.1.1.3 Befehle manuell ausführen

Mit der Menüleistenoption Tools/Befehlszeilen können Sie, während der Anwendung, Befehle schreiben und ausführen lassen. Dies ist nützlich sowohl um Daten abzufragen (mit INFV-Befehl) wie auch um Variablen zu verändern.

5.1.2 Zeitgesteuerte Abläufe

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

Die Prozedur P wird zur Zeitpunkt D0 aufgerufen.
Mit NEXTD kann der nächste Zeitpunkt ermittelt oder errechnet werden.

d0 :	Start-Datum
p :	Prozedur die aufgerufen werden soll
nextd :	Name der Funktion die den nächsten Starttermin ergibt

Beispiel :
```
startat(dstart,doit,nextd=mod1\getnextstart);
```

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

Ermöglicht eine Prozedur in Zeitintervalen aufzurufen (z.B. Datensicherungsoperationen). Fügt eine Pendenz in die Applikations-ToDo-Liste. Diese Pendenz wird dann aufgerufen wenn keine Tekla-Prozedur im Gang ist, also unmittelbar vor einen User-Input.

id :	Pendenz-Identifikator (irgendein Id, z.B: P1)
proc :	Prozedur die aufgerufen werden soll
hs :	Zeitinterval in Hunderstel-Sekunden

Beispiel :
```
/jeder 10.Minute wird die Prozedur SICHERUNG aufgerufen:
insatodo('P1',sicherung,60000);

/In Systemvariable ATODO (Liste) können Pendenzen gesehen
/oder mittels ID gelöscht werden:
del(atodo.p1);
```

5.2 Zahlen und Quantitäten

5.2.1 Algebra

Variablen und Konstanten

E	r	2.71828 (konstanter Wert)
PI	r	3.14159 (konstanter Wert)

function ABS (x:r):r;

Liefert den absoluten Wert von X.

Beispiel :
```
x:r=-4.5;  a:r=abs(x);  /ergibt a=4.5
```

function ARCTAN (x:r):r;

Liefert den Arcustangens von X in Grad.

Beispiel : `x:r=1; a:r=arctan(x); /ergibt a=45 (Grad)`

function COS (x:r):r;

Liefert den Cosinus des in Grad angegebenen Winkels x.

Beispiel : `x:r=60; a:r=cos(x); /ergibt a=0.5`

function EXP (x:r):r;

Liefert e hoch x.

Beispiel : `x:r=3; y:r=exp(x); /ergibt y=20.09`

function FRAC (x:r):r;

Liefert den nicht-ganzzahligen Teil von x.

Beispiel : `x:r=-24.75; a:r=frac(x); /ergibt a=-0.75`

function INTHI (x:r):r;

Liefert den aufgerundeten ganzzahligen Anteil von x.

Beispiel : `x:r=24.25; a:r=inthi(x); /ergibt a=25`

function INTLO (x:r):r;

Liefert den abgerundeten ganzzahligen Anteil von x.

Beispiel : `x:r=24.75; a:r=intlo(x); /ergibt a=24`

function LG (x:r):r;

Liefert den dezimalen Logarithmus von x (Basis 10).

Beispiel : `x:r=100; y:r=lg(x); /ergibt y=2`

function LN (x:r)r;

Liefert den natürlichen Logarithmus von x (Basis e).

Beispiel : `x:r=100; y:r=ln(x); /ergibt y=4.61`

function RND (x:r):r;

Ergibt den gerundeten Wert von X.

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

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

Erweiterte Rundungsfunktion.

val :	Ausgangwert
dec :	Anzahl Dezimal stellen die gerundet werden;
step :	Rundungsschritt;

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

function SIN (x:r):r;

Liefert den Sinus von x (in Grad).

Beispiel : `x:r=30; y:r=sin(x); /ergibt y=0.5`

function SQRT (x:r):r;

Liefert die Quadratwurzel von x.

Beispiel : `x:r=9; y:r=sqrt(x); /ergibt y=3`

function TAN (x:r):r;

Liefert den Tangens von x (in Grad).

Beispiel : `x:r=45; y:r=tan(x); /ergibt y=1`

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

Liefert a hoch b. (Power-Funktion)

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

5.2.2 Zahlenreihen

function RMAX (l:lr):r;

Liefert die grösste Zahl aus der Zahlenliste L.

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

function RMIN (l:lr):r;

Liefert die kleinste Zahl aus der Zahlenliste L.

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

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

Ergibt die grösste Zahl aus der Liste L, die kleiner-gleich als R0 ist.

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

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

Ergibt die kleinste Zahl aus der Liste L die grösser-gleich als R0 ist.

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

5.2.3 Quantitäten

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

Liefert die Quantität Q umgerechnet in der Einheit U.
U muss genauso geschrieben sein wie in der Einheiten-Tabelle in der Konfiguration.

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

function RQ (q:q):r;

Liefert den Wert (ohne Einheit) der Quantität Q.

Beispiel : `q1:q=1.4 kg; w:r=rq(q1); /ergibt w=1.4`

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

Liefert eine Quantität mit Wert V und Einheit U.
U muss genauso geschrieben sein wie in der Einheiten-Tabelle in der Konfiguration.

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

function UQ (q:q):i;

Liefert die Einheit der Quantität Q gemäss Einheiten-Tabelle.

Beispiel : `x:q=7.8 kg; u:i=uq(x); /ergibt u=401`

function QABREV (q:q):s;

Liefert die Einheitsabkürzung der Quantität Q gemäss Einheiten-Tabelle.

Beispiel : `x:q=7.8 kg; a:s=qabrev(x); /ergibt a='kg'`

5.2.4 Abmessungen und Toleranzen

function DIM (x:s):r;

Liefert das Nennmass von X.

x : Mass mit oder ohne Toleranzen

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

function ISO (d:s):r;

Liefert die Masstoleranz in ISO-Qualitätszahl der Abmessung x.

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

5.3 Strings

5.3.1 Handhabung von Strings

Variablen und Konstanten

non n Leerer Name (=NO Name)

function SLEN (s:s):i;

Ergibt die Länge der String S.

Beispiel : `i1:i=slen('15.7'); /ergibt i1=4`

function CHAR (n:i):s;

Ergibt das Zeichen dessen ANSI-Wert N ist.

Beispiel : `s1:s=char(65); /ergibt s1='A'`

function UPC (s:s):s;

Ergibt den upcased string von S.

Beispiel : `s1:s=upc('abc123');  /ergibt  s1='ABC123'`

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

Löscht Zeichen aus einen String.

s : Stringvariable die verändert wird
pos : Position ab welche Zeichen gelöscht werden
nchars : Anzahl Zeichen die gelöscht werden

Beispiel : `s:s='ABCDEF'; dels(s,2,4);  /ergibt s='AF'`

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

Ergibt die Position von N-ten Auftritt von OBJ in SOURCE.

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

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

Schneidet aus dem String S N-Zeichen beginnend bei Position P aus.

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

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

Fügt die String OBJ in der String S an der Position POS.

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

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

Baut die String OBJ in der String S an die Position P.
ADJ bestimmt die Ausrichtung von OBJ: 1=linksbündig 2=rechtsbündig.

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

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

Entfernt alle Leerzeichen von S.
SIDE: 0=alle; 1=links; 2=rechts; 3=links und rechts;

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

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

Ergibt die Anzahl Strings-Segmente in S die mit SEP getrennt sind.

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

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

Ergibt den N-ten Substring von S der mit SEP getrennt ist.

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

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

Ersetzt in String S alle Vorkommnisse von OLD durch NEW.

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

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

Ersetzt in Text L alle Vorkommnisse von OLD durch NEW.

Beispiel :
```
frsl(txt,'AB','AB/');
/verwendet FRS auf alle Zeilen von TXT
```

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

Ergibt den Inhalt einer Listenelement als String.

form : wenn 0 wird nur den Wert ausgegeben;
wenn 1 wird Identifikator + Wert ausgegeben;

l : Liste wo sich die Variable befindet;

nr : Position der Variable in Liste L;

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

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

Ergibt den String S plus ein Leerzeichen-Sufix so dass die Gesamtlänge
L ergibt. ADJ=Ausrichtung: 1=links 2=rechts.

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

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

Ergibt die Position von OBJ in string SOURCE mit variablen Treffgenauigkeit PREC in %.
Falls der prozentualer Anteil von OBJ nicht gefunden wurde, ergibt MATCHS den Wert 0.

Beispiel :
```
m1:i=matchs('B7A2','A1B7A1C3',50);  /ergibt m1=3
```

5.3.2 Formatierung von Strings

- Zeilenumbruch mit #13:
  Beispiel: t:s='Diese Zeile wird hier#13umgebrochen';

- Bitmaps und Icons einfügen mit <BMP=...>:
  Beispiel: t:s='Dies ■ ist ein Stop-Signal';

function SF (/):s;

Ergibt ein formatierte String gemäss den eingegebenen Parameter.
Jedes Parameter besteht aus einem Befehl (1.Zeichen) und einem
Operand (restlichen Zeichen).

Folgende Befehle sind möglich:

L + Zahl N Begrenzt die nächste Ausgabe auf N Zeichen;

V + Name N	Ausgabe der Inhalt der Variable N
X + Zahl N	Bewegt Cursor auf Position N, angegeben in Zeichen
A + Zahl N	Setzt Ausrichtung=N (1=links 2=rechts)
F + Zahl N	Setzt Format = N
S + String	Ausgabe der Stringkonstante
' + Text + '	Schreibt den in Hochkommas geschriebenen Text
M + Zahl N	Bewegt Cursor auf Position M (angegeben in mm); Diese Positionierung funktioniert nur mit linksbündigen Text.
CF + Zahl N	Setzt Textfarbe = N (Vorbelegung = 0)
CB + Zahl N	Setzt Hintergrundfarbe = N (Vorbelegung = -1). Wenn diese Farbe gesetzt wird, werden die nachfolgenden Buchstaben auf einen Hintergrundsrechteck gezeichnet. Mit CB-1 wird diese Funktion desaktiviert.
CR + Zahl N	Setzt Rahmenfarbe = N; Vorbelegung ist -1; Wenn diese Farbe gesetzt wird, werden die nachfolgenden Buchstaben umgerahmt. Mit CR-1 wird die Umrahmung desaktiviert.

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

5.3.3 Typenkonvertierung

function IS (s:s):i;

Wandelt den Stringwert S in einen Integerwert um.

Beispiel :
```
s:s='15'; i:i=is(s);  /ergibt i=15
```

function SI (i:i):s;

Wandelt den Integerwert I in einen String um.

Beispiel :
```
i:i=15;  s:s=si(15);  /ergibt s='15'
```

function RS (s:s):r;

Wandelt den String S in einen Realwert um.

Beispiel :
```
s:s='15.7';  x:r=rs(s);  /ergibt x=15.7
```

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

Wandelt den Realwert R mit K Kommastellen in einen String um.
Wenn K=0 bis 8: SR enthält K Kommastellen
Wenn K=9: SR enthählt soviele Kommastellen wie R hat, max.jedoch 9
Wenn K=10 bis 15: SR enthält K-10 Kommastellen und 1000-er Separatoren

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

function RN (n:n):r;

Wandelt den Namen N in einen Realwert um.

Beispiel :
```
n:n=15.7; x:r=rn(n);  /ergibt x=15.7
```

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

Wandelt den Realwert R mit K Kommastellen in einen Namen um.

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

function QS (s:s):q;

Wandelt den String S in einen Quantitätswert um.

Beispiel :
```
t:s='15.7 mm'; x:q=qs(t);  /ergibt x=15.7 mm
```

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

Wandelt den Quantitätswert Q mit K Kommastellen in einen String um.

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

5.3.4 Kodierung von Information in Strings

Manchmal ist es notwendig Information kompakt zu verpacken, sodass sie später leicht interpretiert werden kann.

Mirakon bietet 2 Syntax-Möglichkeiten:

- Charakteristiken-Strings (z.B: 'AB7C3'):
  Charakteristik = Buchstabe mit eine Bedeutung und ein zugehörigen Wert als Zahl;
  Vorteil: sehr kompakt;
  Nachteil: begrenzt (nur Zahlen als Werte, nur 26 Charakteristiken)
  Prozeduren: CHV, INSCH, DELCH;

- Merkmale-Strings, z.B: (z.B: 'A,B2=7,C7=aby'):
  Merkmal = Identifikator + Wert als Zahl oder String;
  Merkmale werden durch Kommas getrennt; Werte werden nach "=" angegeben;
  Vorteil: sehr flexibel (begrenzt nur durch Stringlänge von 255 Zeichen)
  Nachteil: weniger kompakt (braucht Separatoren)
  Prozeduren: INSC, DELC, SC, IC;

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

Fügt Charakteristik CH in string S.
Eine Charakteristik besteht aus einem Buchstabe die
von einer Zahl gefolgt werden kann (aber nicht muss).

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

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

Löscht Charakteristik CH aus string S. (Siehe auch INSCH)

Beispiel :
```
s1:s='AB7C';
delch('B',s1);   /danach ist s1='AC'
```

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

Ergibt Charakteristikwert von CH aus string S. (Siehe auch INSCH)

Beispiel :
```
s1:s='AB7C';
i1:i=chv('A',s1);  /ergibt i1=0
```

```
i2:i=chv('B',s1);  /ergibt i2=7
i3:i=chv('F',s1);  /ergibt i3=-1
```

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

Fügt oder ändert Merkmal ID mit dem Wert VAL in string S.

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

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

Löscht das Merkmal ID aus der String S.

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

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

Ergibt den String-Wert vom Merkmal ID aus der string S.

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

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

Ergibt den Integer-Wert von Merkmal ID aus der string S.

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

5.4 Datum und Zeit

function NOW :d;

Ergibt den im Moment gültigen Zeitpunkt (Datum und Zeit).

Beispiel : `d1:d=now;  /d1 enthält das heutige Datum und Zeitpunkt`

function TODAY :s;

Ergibt das heutige Datum als string.

Beispiel : `s1:s=today;  /s1 enthält das heutige Datum als String`

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

Untersucht das Datum D und extrahiert die ausgewählten Informationen
in den angegebenen Variablen.

D :	Datum das untersucht wird
YEAR :	Variable die das Jahr von D empfängt
MONTH :	Variable die den Monat von D empfängt
DAY :	Variable die den Tag von D empfängt
WEEK :	Variable die die Woche von D empfängt
WDAY :	Variable die den Wochentag von D empfängt
HOUR :	Variable die die Stunde von D empfängt
MIN :	Variable die die Minute von D empfängt
SEC :	Variable die die Sekunden von D empfängt

Beispiel : `ja,mo,tag,h,w:i;`

```
getd(now,year=ja,month=mo,day=tag,hour=h);
if h<12:inf('Guten Morgen');
getd(now,wday=w);
if w=7:inf('Guten Sonntag');
```

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

Bildet eine Datum-Variable (D) mit der Angabe von Jahr, Monat, Tag,
Stunde, Minute und Sekunde.

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

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

Bildet eine Datum-Variable (D) mit der Angabe von Jahr, Wochennummer,
Wochentag (1=Montag 2=Dienstag ...), Stunde, Minute und Sekunde.

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

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

Erhöht bzw. verkleinert die Variable D um die angegebene Anzahl von
Jahren, Monaten, Tagen, Stunden oder Minuten.

Beispiel : `d1:d=now; inc(d1,days=7);  / d1=heute in 7 Tagen`

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

Ergibt die Anzahl Tage zwischen Startdatum D1 und Enddatum D2.

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

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

Ergibt die Anzahl Stunden zwischen Startdatum D1 und Enddatum D2.

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

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

Ergibt die Anzahl Minuten zwischen Startdatum D1 und Enddatum D2.

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

function RD (d:d):r;

Wandelt Datum D in einen Realwert um.
Damit lassen sich z.B. Zeitdiagramme leichter berechnen.

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

function DR (r:r):d;

Wandelt einen Realwert R in einen Datumswert um.

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

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

Wandelt einen Datumswert in einen Sring mit dem Format F um.

Beispiel : `d:d=3.7.01/08:05;`

```
s1:s=sd(d,1);  /ergibt s1='03.07.01'
s2:s=sd(d,2);  /ergibt s2='3.7.2001'
s3:s=sd(d,3);  /ergibt s3='3.7.01'
s4:s=sd(d,4);  /ergibt s4='03.07.2001/08:05'
s5:s=sd(d,5);  /ergibt s5='08:05'
s6:s=sd(d,6);  /ergibt s6='200107030805'
s7:s=sd(d,7);  /ergibt s7='010703'
s8:s=sd(d,8);  /ergibt s8='200127' (Jahr+Wochennummer)
s9:s=sd(d,9);  /ergibt s9='20010703'
```

function DS (s:s):d;

Wandelt einen String in ein Datum um.

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

5.5 Listen

procedure EMPTY (var l:l);

Leert den Inhalt der Liste L.

Beispiel : `empty(pr.ks);`

procedure DEL (id:n);

Löscht das Listenelement ID;

Beispiel : `del(pr.x);`

function LLEN (l:l);

Ergibt die Anzahl der Listenkomponenten von l.

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

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

Fügt eine Variable in einer Liste ein.

id :	Identifikator des neuen Listelements
v :	Name der Variabel dessen Wert in der Liste eingefügt wird
l :	Ziel-Liste
nr :	Zielposition; wenn NR=0 wird V an der letzte Position eingefügt.
o :	Optionen (Buchstabenkette): - S = Elemente werden alphabetisch nach ID sortiert - N = Wenn ID in L bereits existiert, wird nicht eingefügt.

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

function IDOF (path:n);

Ergibt den Identifikator des Listenelements das durch PATH gezeigt wird.

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

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

Setzt den Identifikator ID in den NR-ten Element von L.

Beispiel : `l:li=(2,4,7);  setid(l,2,'B');`

```
/jetzt kann auch mit L.B auf das
/2.Listenelement zugegiffen werden
```

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

Ersetzt den Identifikator ID1 durch ID2 in allen Datenelementen von L
die ID1 als Identifikator haben.

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

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

Kopiert den Inhalt der Liste L1 auf der Liste L2.

l1 : Quelle

l2 : Ziel-Liste

nr : Einfügeposition in L2; wenn NR=0, wird L1 zuletzt angehängt;

O : Optionen (Zeichenkette):
- M: nur markierte Elemente werden kopiert

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

5.6 Tabellen

Variablen für den Tabellenzugriff

Nach POST, OPENT, PUTT werden folgende Variablen gesetzt bzw. aktualisiert:

trowid	s	Aktive Tabellenzeilen-NR
tcolid	s	Aktive Tabellenspalte-ID
tc	l	Aktive Tabellenzelle. Enthält Textzeilen.
tcels	s	Erste Zeile von aktiven Tabellenzelle
trows	l	Liste der Tabellenzeilen
trowp	i	Aktive Tabellenzeilenposition
tcols	i	Liste der Tabellenspalten
tcolp	i	Aktive Tabellenspaltenposition

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

Ergibt den Inhalt einer Tabellenzelle so wie er dort geschrieben ist.
Wenn ein oder zwei Parameter nicht angegeben sind bzw. leer sind, wird der jeweils zuletzt angegebene übernommen. Dies kann, bei wiederholten Gebrauch, Zeit und Code sparen und gilt für alle Tabellenzugriffe mit CT, IT, RT, ST, NT, DT und QT.

tabid : Tabellen Identifikator

rowid : Zeilennummer

colid : Spaltenname

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

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

Ergibt den Inhalt der Zelle der aktiven Tabelle so wie er dort geschrieben ist.

row : Zeilenposition

col : Spaltenposition

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

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

Holt den Inhalt einer Tabellenzelle und wertet ihn als INTEGER.
TABID, ROWID und COLID wie bei der Funkction CT.

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

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

Ergibt den Inhalt der Zelle der aktiven Tabelle ausgewertet als INTEGER.
ROW und COL wie bei der Funkction CTI.

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

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

Holt den Inhalt einer Tabellenzelle und wertet ihn als REAL.
TABID, ROWID und COLID wie bei der Funkction CT.

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

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

Ergibt den Inhalt der Zelle der aktiven Tabelle ausgewertet als REAL.
ROW und COL wie bei der Funkction CTI.

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

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

Holt den Inhalt einer Tabellenzelle und wertet ihn als STRING.
TABID, ROWID und COLID wie bei der Funkction CT.

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

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

Holt den Inhalt einer Tabellenzelle und wertet ihn als NAME.
TABID, ROWID und COLID wie bei der Funkction CT.

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

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

Holt den Inhalt einer Tabellenzelle und wertet ihn als LIST.
TABID, ROWID und COLID wie bei der Funkction CT.

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

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

Ergibt den Inhalt der Zelle der aktiven Tabelle ausgewertet als LIST.
ROW und COL wie bei der Funkction CTI.

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

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

Holt den Inhalt einer Tabellenzelle und wertet ihn als DATE.
TABID, ROWID und COLID wie bei der Funkction CT.

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

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

Holt den Inhalt einer Tabellenzelle und wertet ihn als QUANTITY.
TABID, ROWID und COLID wie bei der Funkction CT.

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

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

Ergibt eine Liste von Strings aus der Spalte COLID der Tabelle TABID.
Es werden nur die Zeilen genommen, deren Nummer (NR-Spalte) in ROWIDS enthalten ist. Falls BEDID angegeben wird, wird noch die Bedingung in der BEDID-Spalte überprüft.

tabid : Tabellen-ID

rowids : Zeilenfilter (Liste von Zeilen-Nummern)

colid : Spaltenname; um mehrere Spalten aufzulisten, können die gewünschte Spaltennamen, mit | getrennt, angegeben werden.

bedid : Name der Bedingungsspalte

o : Optionen (Zeichenkette):
- T = Reihenfolge von LTAB entspricht Tabellenreihenfolge statt Reihenfolge von ROWIDS

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

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

Sucht die Zeile aus der Tabelle TABID, deren Nummer in ROWIDS enthalten ist und für die, die Bedingung in der Spalte COLID stimmt.
Die Suchreihenfolge richtet sich nach der ROWIDS-Liste.
Die Nummer der zuerst gefundenen Zeile wird an ROWID zurückgegeben.

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

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

Sucht die Zeilen aus der Tabelle TABID, die in der Spalte COLID den Wert VAL enthalten, und speichert deren Nummern in der Liste ROWIDS.

Beispiel : `zeilen:l;  findt(TAB1,PREIS,'120',zeilen);`

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

Aktiviert eine Tabelle, Zeile oder Spalte je nach angegebene Parameter.
Wurde einmal eine Tabelle, Zeile bzw. Spalte aktiviert, bleiben sie so für die nachfolgenden POST-Befehle erhalten.
Nach POST werden die Variablen TC, TROWID, TROWP, TCOLID, TCOLP gesetzt.

tab : Tabellen-Id.

rid : Zeilen-Nr

cid : Spaltenname

r : Position der Zeile (startet mit 1)

c : Spaltenposition; Die erste Spalte ist nicht NR sondern die nachfolgender (meistens die NAME-Spalte).

Beispiel :
```
post(tab=M1\NT);  /aktiviert Tabelle NT in Modul M1
post(cid=PR,r=3); /aktiviert Spalte PR und die 3.Zeile
infv(TROWID);     /zeigt die Zeilen-NR
tc.1:s='5';  /fügt 1.Zeile in der aktiven Zelle ein
             /mit dem Inhalt = '5'
```

```
post(r=2);   /aktiviert die 2.Zeile (Spalte bleibt PR)
tc.1:='7';   /ändert den Inhalt der 1.Zeile der aktiven
             /Zelle auf 7
```

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

Fügt den Inhalt der Variable V in der aktiven Tabelle.
Wurde einmal eine Tabelle, Zeile bzw. Spalte aktiviert, bleiben sie
so für die nachfolgenden PUTT-Befehle erhalten.
Nach PUTT werden die Variablen TC, TROWID, TROWP, TCOLID, TCOLP gesetzt.

v :	Name der Variable dessen Wert in der Tabelle eingefügt wird
tab :	Tabellen-Id.
rid :	Zeilen-Nr
cid :	Spalten-ID
o :	Optionen (Zeichenketten): - I: Wenn Zeile RID nicht gefunden wurde, wird sie eingefügt - S: Eingefügte Zeilen werden alphabetisch sortiert - A: Werte werden dazu addiert
r :	Position der Zeile (startet mit 1)
c :	Spaltenposition; Die erste Spalte ist nicht NR sondern die nachfolgender (meistens die NAME-Spalte).
f :	Format; je nach Typ: Dezimalstellen oder Datumformat

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

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

Öffnet eine Tabelle. Nach OPENT ist T die aktive Tabelle.

t :	Name der Variable (Liste) die die Tabelle enthält
nr :	Tabellen-Identifikator
name :	Tabellen-Bezeichnung
cols :	String mit der Spaltennamen und -breiten jeweils getrennt durch : und ;.

Beispiel :
```
t1:l;
opent(t1,'T','Test','NAME:20;MENGE:8;PREIS:12');
```

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

Fügt eine Spalte in der aktiven Tabelle ein.

p :	Position der neuen Spalte.
w :	Breite (in Zeichen) der neuen Spalte.
id :	Name der neuen Spalte.

Beispiel :
```
inscol(3,20,'PREIS');
```

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

Ändert eine Spalte in der aktiven Tabelle.

p :	Spaltenposition
id :	Spalten-ID
w :	Neue Breite (in Zeichen) der Spalte.
id2 :	Neue Spalten-Id.

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

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

Löscht eine Spalte aus der aktiven Tabelle.

p : Spaltenposition

id : Spalten-ID

Beispiel :
```
delcol(p=2);
delcol(id=PREIS);
```

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

Fügt eine Zeile in der aktiven Tabelle ein.

p : Position der neuen Zeile.

h : Höhe (in Zeilen) der neuen Zeile.

lev : Hierarchiestufe der neuen Zeile (1 bis 20).

id : Nummer der neuen Zeile.

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

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

Ändert eine Zeile in der aktiven Tabelle.

p : Zeilenposition

id : Zeilen-NR

h : Neue Höhe (in Textzeilen) der Zeile.

lev : Neue Hierarchiestufe der Zeile (1 bis 20).

id2 : Neue Zeilen-NR.

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

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

Löscht eine Zeile aus der aktiven Tabelle.

p : Zeilenposition

id : Zeilen-Nr

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

function EXROW (nr:s):i;

Ergibt 1 falls die Zeile NR in der aktiven Tabelle existiert bzw. 0 falls sie nicht existiert.

Beispiel :
```
if exrow('A01')=0: inf('Zeile nicht vorhanden');
```

function EXCOL (nr:s):i;

Ergibt die Position der Spalte NR in der aktiven Tabelle falls sie existiert, bzw. 0 falls sie nicht existiert.

Beispiel :
```
if excol('NAME')=0: inf('Spalte nicht vorhanden');
```

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

Öffnet den Tabelleneditor um die Tabelle T zu editieren.
Wenn die Tabelle T leer ist, wird sie mit der Initialtabelle T0 gefüllt.

T :	Tabelle die editiert wird
T0 :	Initialtabelle
O :	Optionen = Zeichenreihenfolge mit folgender Bedeutung: L = Alle Zellen sind einzeilig; Enter-Taste springt auf nächste Zelle A = Automatische Zeilennummerierung (1,2,3,...)
B1 :	Optionaler Knopf: Syntax= 'Knopfname:Prozedur'

Beispiel :
```
edittab(tab1,tab0,o=LA);
edittab(t1,t0,b1='_Kontrole:kontroltab');
```

5.7 Strukturen

Strukturen sind Listen dessen Listelementen nur von Typ ELEMENT sind.
Diese Elemente können wiederum Substrukturen enthalten, sowie
Parameter und andere zugeordneten Daten.

5.7.1 Strukturen handhaben

Variablen für den Zugriff auf Strukturelemente

Nach POSE, oder beim TRAVEL-Befehl werden folgende Variablen
gesetzt bzw. aktualisiert:

pe	^e	Zeiger zum aktiven Element.
pe0	^e	Zeiger zum Vater des aktiven Elements.
enr	s	Nummer des Masters des aktiven Elements.
enr0	s	Nummer des Masters des Vaters vom aktiven Elements.
etabid	s	Tabelle des Masters des aktiven Elements.
emod	s	Wissensmodul des Master des aktiven Elements.
eid	s	Vollstaendiger Identifikator des aktiven Elements. Format: emod\etabid.enr
ename	s	Name des aktiven Elements.
p	l	Parameter des aktiven Elements.
d	l	DATA-Feld des aktiven Elements.
ss	l	Substruktur des aktiven Elements.
eh	i	Hierarchiestufe (1..40) des aktiven Elements.
ei	i	Position des aktiven Elements in seiner Hierarchiestufe.
ech	s	Characteristik des Masters des aktiven Elements (CH-Spalte).
etyp	n	Typ des aktiven Elements, z.B: R,L,E,GLINE, usw.
ep	s	Pfad des aktiven Elements.
efound	i	Resultat von Prozedur FINDE.
pefound	^e	Zeiger auf das mit FINDE gefundene Element.

procedure POSE (var e:e);

Aktiviert das mit E angegebene Element und setzt die zugehörigen
Systemvariablen D, P, SS, ENR, EMOD, ENAME, EP, EH, PE.

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

procedure PUSHST;

Speichert den aktuellen Strukturkontext (Variablen D, P, SS, ENR,...)

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

procedure POPST;

Stellt den zuvor mit PUSHST verlassenen Strukturkontext wieder her.

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

Ersetzt den Identifikator ID1 durch ID2 in allen Elementen von ST
die ID1 als Elementmaster-Nr haben.

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

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

Ersetzt den Modul ID1 durch ID2 in allen Elementen von ST
die ID1 als Modul haben.

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

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

Ersetzt die Tabellenreferenz TAB1 durch TAB2 in allen Elementen von ST
die TAB1 als Mastertabelle haben.

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

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

Sucht bestimmte Elemente in eine Struktur und ergibt die Anzahl
der somit gefundene Elemente.

st :	Struktur die durchsucht wird
nr :	Filter für Master-Nummer
h :	Anzahl Hierarchiestufen die durchsucht werden sollen; wenn nicht angegeben ist H=1;
c :	Bedingungsfunktion (optional) bestimmt, ob die gefundene Elemente gezählt werden oder nicht.
path :	Pfadbegrenzung (optional)

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

procedure FINDE (var st:l; /);

Sucht in der Struktur ST nach einen Element der mehreren Bedingungen
erfüllt. Diese Bedingungen können frei eingegeben werden.
Speziel die Bedingung PP bewirkt dass ein PUSHST vor bzw. POPST nach der
Positionierung ausgeführt werden.
Wenn das Element gefunden worden ist, bekommt die Systemvariable EFOUND
den Wert 1 und das System bleibt dort positioniert, d.h. die System-
variablen PE, ENR, ENAME, usw. gelten für diesen Element. Insbesondere
die Variable PEFOUND wird gesetzt und verwendet wenn die PP-Bedingung
eingegeben wurde.

Beispiel :
```
finde(pr.bs,pp,ex(p.laenge),p.laenge>120);
if efound: inf(pefound^.name+' gefunden');
```

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

Löscht in der Struktur ST alle Elemente dessen Nummer in IDS enthalten sind.
Mit C kann eine zusätzliche Bedingung für das Löschen formuliert werden.

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

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

Aktiviert alle Elemente in den aktuellen Strukturpfad;

h : maximale Hierarchie-Stufe;

p : Prozedur die für jede Stufe ausgeführt wird;

o : Optionen (Zeichen):
- P: ruft PUSHST vor der Positionierung auf und POPST danach;
  somit kann der Strukturzustand vor POSST wiedererstellt werden.

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

/ in Prozedur PROC1 könnte z.B. stehen:
  if etabid='MAT': material:=enr;
/ somit wird ein Rohmaterial-Position erkannt
/ ohne dass es an einer feste Position stehen muss
```

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

Sortiert die Struktur ST nach ein bestimmten Parameter.

st : Struktur die sortiert wird;

id : Name der Parameter der als Sortierkriterium dient;
Dieser Parameter kann aus folgenden Typen sein: i,r,s,n,d.

inv : wenn INV=1 wird ST in absteigenden Reihenfolge sortiert;

Beispiel :
```
sortst(pr.bs,p.breite);
```

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

Modifiziert die Pfadvariable PATH durch ändern der Hierarchiestufe um H
und der letzten Position um I.

Beispiel :
```
p1:n=1.2.7;
modp(p1,0,1);   /ergibt  p1=1.2.8
modp(p1,0,-1);  /ergibt  p1=1.2.6
modp(p1,1,1);   /ergibt  p1=1.2.7.1
modp(p1,-1,0);  /ergibt  p1=1.2
```

5.7.2 Strukturen generieren

Variablen für das Generieren von Strukturen

Folgende Variablen unterstützen das Generieren von Strukturen.
PEB, LEB und IEB werden nach jeden Einbaubefehl (->) gesetzt.

peb	^e	Zeiger auf der zuletzt eingebauten Element
leb	l	Liste die der zuletzt eingebauten Element enthält
ieb	i	Position der zuletzt eingebauten Element in LEB
enew	i	1 wenn aktives Element neu eingefügt wurde, ansonsten=0

as	s	Aktueller Ablaufzustand

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

Führt die, im Master vom Element (oder Elementenliste) E0, definierte Aktionenenkette aus.

AS : Situation für die auszuführenden Aktionen;

E0 : Name der Liste oder Element wo die Aktionen gestartet werden

T : Titel für die Protokolfenster

D : Displaymodus: 0=nichts zeigen; 1=message; 2=report (vorbelegt);

P1 : Prozedur die, bei jede Position, vor der Aktionen, ausgeführt wird;

PA1 : Id von Erzeugerpfad in DATA des neu erzeugten Elements;

TFP : Positive Filter: gibt an welche Positionen durchgesucht werden;

TFN : Negative Filter: gibt an welche Positionen nicht durchgesucht werden;

Beispiel :
```
act(PAI1,pr.m,'Generiert Prozess');
act(GENAP,pr.bs,'Generiert Prozess',pa1=BK);
act(K2,pr.ap,'Kosten',d=0,p1=posop,tfp='1._,2._');
```

procedure SETAS(s:s);

Setzt den Ablaufszustand.

Beispiel :
```
setas('PAI1');
```

5.7.3 Dialog mit Strukturen

Variablen für den Dialog mit Strukturen

Folgende Variablen unterstützen die Formatierung und Programmierung des Verhaltens eines Struktureditors:

eol	b	1 wenn der Cursor sich am Ende der Liste befindet.
xlfe	r	X-left-Koordinate der aktuellen Zeile
xrfe	r	X-right-Koordinate der aktuellen Zeile
ytfe	r	Y-top-Koordinate der aktuellen Zeile
ybfe	r	Y-bottom-Koordinate der aktuellen Zeile
copyok	i	Erlaubnis zum Kopieren in die Ablage in EDITST; Vorbelegt mit 1 (erlaubt); kann in Struktureditor-Handler auf 0 gesetzt werden;
cutok	i	Erlaubnis zum Ausschneiden in die Ablage in EDITST; Vorbelegt mit 1 (erlaubt); wird in Struktureditor-Handler gesetzt;
pasteok	i	Erlaubnis zum Einfügen von der Ablage in der selektierte Position; wird in Struktureditor-Handler gesetzt;
delok	i	Erlaubnis zum Löschen in EDITST; vorbelegt mit 1 (erlaubt); wird in Struktureditor Handler gesetzt;
insok	i	Erlaubnis zum Eingügen in EDITST (vorbelegt mit 0); wird in Struktureditor-Handler gesetzt;
parsok	i	Erlaubnis zum den Dialog öffnen, um die Elementparameter zu editieren in EDITST (vorbelegt mit 0); wird in Struktureditor-Handler gesetzt;

sttabid	s	Tabelle die als Quelle für den Struktureditor dienen soll. Wird in der Handler-Procedure von EDITST gebraucht.
stinsnr	s	Nummer des einzufügendes Element im Struktureditor. Somit erscheint das Einfügemenu nicht mehr. Wird in Handler-Prozedur von EDITST (at PREINS) gebraucht.
stinst	s	Titel für Einfügen-Menu im Struktureditor. Wird in der Handler-Procedure von EDITST gebraucht.
stinsf	s	Einfügen-Filter für Elementen in Struktureditor. Wird in der Handler-Procedure von EDITST gesetzt.
stinsfm	s	Einfügen-Filter für Module in Struktureditor. Wird in der Handler-Procedure von EDITST gesetzt.

Situationen für den EDITST-Händler

Folgende Situationen können in die Händlerprozedur verwendet werden:

PREINS	nach Drücken des INSERT-Knopfes in den Struktureditor
POSTINS	nach dem Einfügen eines Elements in einer Struktur
PREDEL	nach Drücken des DELETE-Knopfes in den Struktureditor
POSTDEL	nach dem Löschen eines Elements in einer Struktur
PREPARS	nach Drücken des PARAMETER-Knopfes in den Struktureditor
POSTPARS	nach dem parameter editierung eines Elements in einer Struktur
DRAG	bevor dem Beginnen zu schleppen
DROP	bevor die geschleppten Elemente fallengelassen werden
RCLICK	nach Right-Click mit Maus
COPYCLIP	nach Drücken von Ctrl-Ins (kopieren in die Ablage)
CUTCLIP	nach Drücken von Shift-Del (ausschneiden in die Ablage)
PASTECLIP	nach Drücken von Shift-Ins (einfügen aus der Ablage)

Situationen für die Mastertabellen

Folgende Situationen können in die A-Spalte der Mastertabellen verwendet werden:

I2	nach dem manuellen Einfügen eines Elements in einer Struktur (vor Dialogmaske)
I3	nach dem Einfügen eines Elements in einer Struktur (nach Dialogmaske)

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

Öffnet den Struktur editor für die Struktur ST.

TYP :	Editorlayout: 1 = mit einem Fenster: normala Ausführung 3 = mit 2 Fenster: links die Struktur, rechts die Dialogmaske des selektierten Elements
ST :	Struktur die editiert wird
ES :	Elementenquelle = Identifikator der Masterstabelle
AS :	Aktive Situation während des Editierens

OPT : Optionen = Zeichenreihenfolge mit folgender Bedeutung:
A : Editor mit Zoom alles-Knopf
C : Editor mit Kopieren-Knopf
D : Editor mit Dokumentation-Knopf
E : Editor ohne Löschen-Knopf
F : Editor ohne Einfügen-Knopf
G : Editor mit Gruppen-Knopf (fügt masterlosen Gruppen ein)
H : Hierarchische Position wird für jeden Element angegeben
K : F12-Funktion (Strukturenansicht) wird unterdrückt
L : Editor mit Relationen-Knopf
N : Editor mit Umnennen-Knopf
P : Editor mit Pars-Knopf
Q : Editor mit Menge-Knopf
R : Editor mit Ressourcen-Knopf
T : Editor mit Termine-Knopf
U : Editor mit Status-Knopf
W : Ressourcen-Editor mit Arbeitsplatz-Knopf. Voraussetzung:
Ressourcen-Datenbank muss als DW definiert sein !
Y : Transfer-Funktionen (Clipboard) werden unterdrückt
Z : Editor mit Zoom-Knopf um die Textzeilen zu vegrössern

T1 : Erste Titel für Frameregister

T2 : Zweite Titel für Frameregister

H : Optionale Steuerungsprozedur (handler): Damit können Strukturänderungen (Einfügen, Löschen, usw.) genauer gesteuert werden.

B : Optionale Knöpfe:
Syntax: 'Knopfname1:Prozedur1,Knopfname2:Prozedur2,...'

ID : Identifikation des Dialogfensters;

EF : Optionen für die Formatierung der Zeilen:
bit1->1: mit [doc] wenn ein Dokument zugewiesen ist;
bit2->1: mit [ress] wenn Ressourcen zugewiesen sind;

Beispiel :

```
editst(1,pr.bs,BK,B01,PQ,'Strukturanalyse',pr.nr);
editst(2,pr.lc,PROD,B07,PQ,'Stückliste',pr.nr);
editst(1,pr.bs,BK,,P,'','',b='Kunde:inpk,Ort:inpo');
editst(1,pr.ap,OP,,PRQ,'Ablauf','',h=APSTEUER);

Beispiel für Steuerungs-Prozedur APSTEUER:

at PREINS:  /nach dem Drücken des Einfügen-Knopfes
/auf Stufe 1 kann nichts eingefügt werden:
if eh=1:insok:=0&exit;
/Einfügemenu wird festgelegt:
if k1='MA':sttabid,stinst:='MA','Material';
if k1='AP':sttabid,stinst,stinsf:='AV','Vorgänge','D_';

at POSTINS: /nachdem Element eingefügt wurde
if etabid='AV':berechne_vorgangs_nr;

at PREDEL:  /nach dem Drücken des Löschen-Knopfes
/auf Stufe 1 darf nichts gelöscht werden:
if eh=1:delok:=0&exit;
/Elemente MA dürfen nicht gelöscht werden:
if enr='MA':delok:=0&exit;

at POSTDEL: / nachdem Element gelöscht wurde
```

```
if etabid='AV':berechne_vorgangs_nr;

at RCLICK:  / nach Maus-Click mit rechten Maustaste
del(d.menu);/ löscht voriges Menu
d.menu:l;   / bildet neues Menu
if enr='MA': d.menu.0:s='Materialblatt:printmatblatt';
d.menu.0:s='Kosten zeigen:showcost';
\
```

Formatierung der Strukturzeilen am Bildschirm
In der Spalte FORMAT kann das Aussehen eines Elements im Strukturedtor am flexibelsten gestaltet werden. Hier können Befehle eingegeben werden die die Systemvariablen EFICON oder EFSYMB, EFPREFIX, EFNAME, EFPARS, EFSUFIX setzen. Die ausgegebene Zeile besteht dann aus der Verkettung dieser Systemvariablen.
Mit der Variable EFCOLOR kann ausserdem die Zeilenfarbe bestimmt werden.

Es ist auch möglich in der ausgegebene Zeile Grafikelemente zu zeichnen. Dafür werden die üblichen Zeichnungsbefehle verwendet (draw, drawline, ...). Die Koordinaten des Zeilenrechtecks werden in der Systemvariablen XLFE, YTFE, XRFE, YBFE angegeben.
Beispiel :
```
efpars:=sr(p.d,0)+' * '+sr(p.l,2);
if p.zustand=2: efcolor:=40;
```

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

Erlaubt das Auswählen eines Elements aus einer Struktur.

ST :	Struktur woraus ein Element selektiert wird
P :	Name der Variable (vom Typ name) der den, vom Anwender, selektierten Pfad enthält
t :	Fenstertitel (vorbelegt mit 'Komponente')
w :	Breite des Dialogfensters in Zeichen; vorbelegt mit 80;
h :	Höhe des Dialogfensters in Zeichen; vorbelegt mit 24;

Beispiel :
```
p1:n; selp(pr.bs,p1,t='Bitte selektieren !');
```

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

Zeigt die Struktur ST aus der ein Element selektiert werden kann.

ST :	Struktur woraus ein Element selektiert wird
P :	Name der Zeiger-Variable (vom Typ ^e) der auf den, vom Anwender, selektierten Element zeigt
t :	Fenstertitel (vorbelegt mit 'Komponente')
w :	Breite des Dialogfensters in Zeichen; vorbelegt mit 80;
h :	Höhe des Dialogfensters in Zeichen; vorbelegt mit 24;

Beispiel :
```
p1:^e; sele(pr.bs,p1,t='Bitte selektieren !');
```

procedure DESELAE;

Deselektiert das Strukturfeld das mit EDITST aktiv ist.
Dies ist notwendig und wichtig unmittelbar bevor die gezeigte Struktur verändert wird. Mirakon versucht nähmlich immer zur vorher selektierten Element zurückzukehren. Wenn dieses nicht mehr existiert, folgt meistens ein Fehler.

Beispiel :
```
/ Knopf in Struktur-Editor ruft folgende Befehle:
```

```
deselae;
empty(pr.bs);
rsetm(MAIN);
```

function EMARK (p:^e):i

Informiert ob das mit P angezeigten Element markiert ist (emark=1) oder nicht (emark=0).

Beispiel : `if emark(pe): anzahl++1;`

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

Markiert (val=1) bzw. unmarkiert (val=0) das mit P angezeigten Element.

Beispiel : `marke(pe,1);`

5.7.3.1 Strukturediteren mit Mastertabellen

Mit EDITST werden Strukturen mit Elementen aus Master-Tabellen editiert.

Wenn ein Element neu eingefügt wird (INSERT-Taste), werden die in der PARS-Spalte deklarierten Parameter kreiert, und mit der in der MASK-Spalte angegebene Dialogmaske editiert.

Nach verlassen der Maske mit OK werden alle, für die Situation zutreffende Aktionen ausgeführt.

Wenn die Parameter eines bestehenden Elements mit dem PARS-Knopf verändert wird, kann das Systemverhalten mit der in der CH-Spalte angegebenen Charakteristiken beeinflusst werden:

- E = Die Unterstruktur des betreffenden Elements wird, vor dem Ausführen der jeweiligen Aktionen, geleert.
  Wenn diese Unterstruktur manuell verändert worden ist, frägt das Programm ob der Anwender diese überschreiben will, bzw. ob allfälligen manuell eingefügten Unterelemente behalten werden sollen.

5.8 Relationen

Variablen für die Handhabung von Relationen

rtyp	i	Typ der aktiven Relation. Wird mit TRAVELR gesetzt.
rref	i	Referenz der aktiven Relation. Wird mit TRAVELR gesetzt. 1=Element ist Referenz 2=Element ist referiert (Slave)
pre1	^e	Zeiger zum Referenz-Element der aktiven Relation. Wird mit TRAVELR gesetzt.
pre2	^e	Zeiger zum Dependent-Element der aktiven Relation. Wird mit TRAVELR gesetzt.
rdata	l	Daten der aktiven Relation. Wird mit TRAVELR gesetzt.

5.9 Prozesse

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.

5.9.1 Handhabung von Prozessen

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

Öffnet den Prozesseditor für den Prozess PROC.

proc : Prozess der editiert wird

tab : Identifikator der Masterstabelle (Elementenquelle)

relt : Identifikator der Masterstabelle für Relationen (option)

h : Optionale Steuerungsprozedur (handler): Damit können Prozessänderungen (Einfügen, Löschen, usw.) genauer gesteuert werden.

b : Optionale Knöpfe:
Syntax: 'Knopfname1:Prozedur1,Knopfname2:Prozedur2,...'

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

Tabelle TPE:

NR	NAME	PARS	MASK	G	A
T	Teile				

TB	Blechteil	tnr,tbez:s;
		mk:r;
		b:r:breite mm=40;
		h:r:höhe mm=20;
		bild:s;
			TPTB	draw(ab10,gb(0,0,p.b,-p.h));
				draw(tf3,th12,x2,y-5,tw1,vp.tb
				h2,b2:r=p.h-7,1.2*h2;
				bild:s='FPR1\'+p.bild;
				draw(gi(bild,2,-7,2+b2,-7-h2))	

V	Vorgänge				
VK	Kleben	nl:r; bez:s='Kleben'; b:r:breite mm=35;	TPVK	draw(ab14,gb(0,0,p.b,-p.h)); draw(tf3,th12,x2,y-5,tw1,'Kleb pars:s=si(p.nl)+' mm';	);
VP	Punkt-Schweiße	np:i; bez:s='Punktschwei b:r:breite mm=35;	TPVP	draw(ab3,gb(0,0,p.b,-p.h)); draw(tf3,th12,x2,y-5,tw1,'Punk pars:s=si(p.np)+' Punkte';	1,tc0); ssen',tw0)
Z	Zellen				ta1,tc0);
ZM	Montage-Zelle	znr:i; zbez:s; flaech:r; takt:r;	TPZM	draw(ab7,gb(0,0,p.b,-p.h)); titel:s='Zelle '+p.znr+' '+p.z draw(tf3,th12,x2,y-5,tw1,vtite	

Tabelle RELTAB:

NR	NAME	PARS	MASK	G

BKOP	Input Relation	funk:s;
		farbe:i=40;
		ap1:i=2;
		ap2:i=4;
			RELBKOP	p1,p2:point2d;
				geted(pie1,p2d=p1);
				geted(pie2,p2d=p2);
				x1,y1,x2,y2:r=p1.x,p1.y,p2.x,p2.y;
				b1,h1:r=pie1^.pars.b,pie1^.pars.h;
				b2,h2:r=pie2^.pars.b,pie2^.pars.h;
				draw(lcp.farbe,gl(x1,y1,x1b,y1b),
				gl(x1b,y1b,x2b,y2b),le2,
				gl(x2b,y2b,x2,y2),le1,lc0);
				xm,ym:r=(x1+x2)/2,(y1+y2)/2;
				draw(tf3,th12,tw1,ta3,tcp.farbe,

5.10 Dokumente

5.10.1 Listings

Textdokumente sind unformatierte Listen von Strings.
Seiten können nicht erstellt und nummeriert werden.
Der Text wird immer in Font Courier New geschrieben.

Wir empfehlen eher grafische Dokumente zu verwenden.

procedure OPENLST (var lst:l);

Eröffnet das Dokument LST.

Beispiel : `openlst(tdoc);`

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

Öffnet den Texteditor mit Titel T für das Dokument LST.
Mit W und H kann die Fensterbreite bzw. -höhe in Anzahl Zeichen angegeben werden.
Wenn W und H nicht angegeben sind, wird die ganze Arbeitsfläche genutzt.

Beispiel : `editlst(tdoc,'Stückliste');`

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

Schreibt ein Stringwert in das geöffnete Textdokument.

dr : Zeilensprung bezogen auf die zuletzt geschriebenen Zeile.
c : Spaltenposition.
adj : Ausrichtung: 1=linksbündig 2=rechtsbündig.
val : Stringwert der geschrieben wird

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

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

Schreibt ein Integer-Wert in das geöffnete Textdokument.
Parameter wie in WRITES.

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

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

Schreibt ein Real-Wert in das geöffnete Textdokument.
Parameter wie in WRITES.
F=Format=Anzahl Dezimalstellen

Beispiel : `writer(0,25,2,preis,2);`

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

Schreibt ein Datum-Wert in das geöffnete Textdokument.
Parameter wie in WRITES.
F = Datumformat (siehe Funktion SD)

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

5.10.2 Grafik-Dokumente

Typen für die Erstellung Grafikdokumenten

GIMAGE Bild (Bitmap) in BMP oder JPG-Format

Variablen für die Erstellung Grafikdokumenten

page i Aktuelle Seiten-Nummer des aktiven Dokuments

5.10.2.1 Grafikobjekte und Editoren

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

Eröffnet (formatiert) die Grafik G.

g : Grafik-Objekt
nr : Ident-Nr der Grafik
name : Bezeichnung der Grafik
w : Grafikbreite in mm
h : Grafikhöhe in mm
xm : Linke Rand in mm (optional)
ym : Obere Rand in mm (optional)
m : Seitenformat = Mask (optional)
np : Newpage-Prozedur = Prozedur die aufgerufen nachdem eine neue Seite eingefügt wird.
ep : EndofPage-Prozedur = Prozedur die aufgerufen wenn eine neue Seite eingefügt wird, wirkt aber noch auf die letzte Seite.
nd : 1=no destroy, d.h. Dokumentinhalt wird nicht geleert

Beispiel :
```
g1:l;
openg(g1,'A1','Skizze',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);

Öffnet den Grafikeditor für die Grafik G.

g : Grafik-Objekt
o : Optionen (Zeichenkette):
S = Grafikeditor wird als Tochterfenster des aktuellen Dialogfenster geöffnet (small window)
build : Name der Prozedur die die Grafik aufbaut; Wenn dieses Parameter angegeben wird, erscheint ein BILD-Knopf über den Grafikfenster;
zf : Initialer Zoomfaktor; Vorbelegung = 1.0;
w0 : Initiale Papierbreite (falls G leer ist)
h0 : Initiale Papierhöhe (falls G leer ist)
id : Identifikation des Dialogfensters;

Beispiel :
```
editg (pr.doc1,build=gendoc,zf=1.2);
```

procedure VIEWG (var g:l; /o,build:n; zf:r; id,nar:i);

Öffnet den Grafikviewer für die Grafik G.
Parameter: gleich wie in EDITG.

NAR : ohne aktive Regionen

5.10.2.2 Papierkoordinaten, Felder und Rahmen

procedure SETMARGIN (x,y:r);

Setzt den linken und oberen Rand, und verschiebt somit den absoluten Nullpunkt.

x : Linke Rand in mm (immer positiver Wert)
y : Obere Rand in mm (immer positiver Wert)

Beispiel : `setmargin(20,10);`

procedure SETZERO (x,y:r);

Setzt einen neuen Nullpunkt mit Koordinaten X,Y vom absoluten
Papier-Nullpunkt aus.

Beispiel : `setzero(20,-110);`

procedure DEFGF (id:n; xlp,ytp,xrp,ybp,xli,yti,xri,ybi:r);

Definiert ein rechteckiges grafisches Feld im aktiven Dokument.

id : Identifikator des Feldes;
xlp : X-left Koordinate auf Papier
ytp : Y-top Koordinate auf Papier
xrp : X-right Koordinate auf Papier

ybp : Y-bottom Koordinate auf Papier

xli : X-left Koordinate innerhalb des Feldes
yti : Y-top Koordinate innerhalb des Feldes
xri : X-right Koordinate innerhalb des Feldes

ybi : Y-bottom Koordinate innerhalb des Feldes

Beispiel : `defgf(BALKEN,20,-50,150,-130,10000,0,0,200);`

procedure SETGF (id:n);

Setzt grafisches Feld ID aktiv.

Beispiel : `setgf(BALKEN);`

5.10.2.3 Zeichnen von Texte

procedure SETFONT(font,h,sty,w,col:i);

Setzt die Schrift-Attribute für die nachfolgenden DRAW-Prozeduren.

font : Fontnummer in Konfiguration.
h : Fontgrösse in Punkte
sty : 0=normal 1=kursiv

w : 0=normal 1=fett

col : Textfarben-Nr.

Beispiel : `setfont(4,10,0,0,0);`

procedure DRAWI (x,y:r; adj:i; i:i);

Schreibt den Integer I an der Position X,Y (in mm).

adj : Ausrichtung: 1=linksbündig 2=rechtsbündig 3=zentriert

Beispiel : `drawi(150,-8,1,i1);`

procedure DRAWR (x,y:r; adj:i; r:r; f:i);

Schreibt den Real R an der Position X,Y (in mm).

adj : Ausrichtung: 1=linksbündig 2=rechtsbündig 3=zentriert

f : Format = Anzahl Dezimalstellen

Beispiel : `drawr(120,-40,2,preis,2);`

procedure DRAWD (x,y:r; adj:i; d:d; f:i);

Schreibt den Datum D an der Position X,Y (in mm).

adj : Ausrichtung: 1=linksbündig 2=rechtsbündig 3=zentriert

f : Datumformat (siehe Funktion SD)

Beispiel : `drawd(150,-8,1,now,1);`

procedure DRAWS (x,y:r; adj:i; t:s);

Schreibt die String T an der Position X,Y (in mm).

adj : Ausrichtung: 1=linksbündig 2=rechtsbündig 3=zentriert

Beispiel : `draws(10,-20,1,'Test');`

procedure DRAWQ (x,y:r; adj:i; q:q; f:i);

Schreibt die Quantität Q an der Position X,Y (in mm).

adj : Ausrichtung: 1=linksbündig 2=rechtsbündig 3=zentriert

f : Format = Anzahl Dezimalstellen

Beispiel : `drawq(150,-8,2,q1,2);`

procedure DRAWTEXT (x,y,dy:r; adj:i; var txt:l);

Schreibt alle Zeilen aus der Stringliste TXT an der Position X,Y (in mm).

dy : Zeilenabstand in mm

adj : Ausrichtung: 1=linksbündig 2=rechtsbündig 3=zentriert

Beispiel :
```
txt1:ls=('AAA','BBB','CCC');
drawtext(50,-25,4,1,txt1);
```

5.10.2.4 Zeichnen von Linien und Flächen

procedure SETLINE (sty,wid,end:b; col:i);

Setzt die Linien-Attribute für folgende DRAWLINE Prozeduren.

sty : Linienart: 1=voll 2=--- 3=... 4=-.-

wid : Liniendicke: 1=0.25mm 2=0.35mm 3=0.5mm 4=0.75mm 5=1mm

end : Linienende: 1=normal 2 = -> 3 = <-> 4 = -< 5 = -o

col : Linienfarben-Nr.

Beispiel : `setline(3,1,1,3);`

procedure DRAWLINE (x1,y1,x2,y2:r);

Zeichnet eine Linie von Punkt (x1,y1) zu Punkt (x2,y2).

Beispiel : `drawline(150,-8,200,-40);`

procedure DRAWARC (x0,y0,rx,ry,a1,a2:r);

Zeichnet einen Bogen.

x0 : X-Koordinate des Mittelpunktes

y0 : Y-Koordinate des Mittelpunktes

rx : Radius in X-Richtung

ry : Radius in Y-Richtung

a1 : Anfangswinkel in Grad

a2 : Endwinkel in Grad

Beispiel :
```
drawarc(40,-50,25,25,0,360); /Kreis mit Radius=25
drawarc(40,-50,5,20,0,180);  /obere Hälfte einer Elipse
```

procedure SETBOX (form,pat:b; bcol,fcol:i);

Setzt die grafischen Attribute für DRAWBOX.

form : Boxformat: 0=ohne Rand 1=normal 3=3-D-Box

pat : Füllmuster: 0=leer 1=voll 2=punkte 3=punkte 4=/// 5=\\\ 6=# # #

bcol : Hintergrundfarbe (Farben-Nr.)

fcol : Vordergrundfarbe (Farben-Nr.)

Beispiel : `setbox(1,2,14,2);`

procedure DRAWBOX (xl,yt,xr,yb:r);

Zeichnet eine Box.

xl : X-Koordinate des linken Boxenrandes

yt : Y-Koordinate des oberen Boxenrandes

xr : X-Koordinate des rechten Boxenrandes

yb : Y-Koordinate des unteren Boxenrandes

Beispiel : `drawbox(150,-8,200,-40);`

5.10.2.5 Pfade zeichnen und füllen

Pfade sind Linienzüge bestehend aus geraden Segmenten und/oder Kreisbogen.
Diese können durch Attributenzuweisung gefüllt werden.
Pfade werden als Datenstruktur definiert und danach in einer Grafik eingefügt.

Beispiel : `p1:path2d=(c=2, w=2, x=15, y=-20);`

```
p1.segs.0:segline2d=(x=15, y=5);  /vertikale gerade Segment
p1.segs.0:segline2d=(x=40, y=5);  /horizontale gerade Segment
p1.segs.0:segarcc2d=(xc=40, yc=0, x=45, y=0); / viertelkreis abwärts
insl('',p1,grafik,0);  /Pafd wird in bestehende Grafik eingefügt
```

Typen für die Pfadbeschreibung

Folgende Typen sind für die Beschreibung eines Pfads notwendig:

path2d Pfad-Kopfdaten, enthält:
- C:i; Farbennummer;
- S:b; Linienstyl; (siehe Prozedur SETLINE)
- W:b; Liniendicke; (siehe Prozedur SETLINE)
- E:b; Linienende; (siehe Prozedur SETLINE)
- FC:i; Füllfarbe;
- FS:b; Füllmuster; (siehe Prozedur SETBOX)
- X:r; X-Koordinate des Startpunktes
- Y:r; Y-Koordinate des Startpunktes
- SEGS:l; Liste der Pfadsegmente (segline2d oder segarcc2d)

segline2d Gerade Liniensegment:
- ATT1:i; freidefinierbares Attribut (z.B:Geschwindigkeit)
- ATT2:b; freidefinierbares Attribut (z.B:Genauigkeitsklasse)
- X:r; X-Koordinate des Endpunktes
- Y:r; Y-Koordinate des Endpunktes

segarcc2d Kreisbogen-Segment:
- RES:i; Zeichnungsauflösung in Grad (vorbelegt mit 10°)
- DIR:b; Richtung: 1=Uhrzeigersinn 2=Gegenuhrzeigersinn
- XC:r; X-Koordinate des Kreismittelpunktes
- YC:r; Y-Koordinate des Kreismittelpunktes
- X:r; X-Koordinate des Endpunktes
- Y:r; Y-Koordinate des Endpunktes

segarce2d Elipsebogen-Segment:
- RES:i; Zeichnungsauflösung in Grad (vorbelegt mit 10°)
- DIR:b; Richtung: 1=Uhrzeigersinn 2=Gegenuhrzeigersinn
- XC:r; X-Koordinate des Ellipse-Mittelpunktes
- YC:r; Y-Koordinate des Ellipse-Mittelpunktes
- X:r; X-Koordinate des Endpunktes
- Y:r; Y-Koordinate des Endpunktes
- RX:r; Radius der Ellipse in X-Richtung
- RY:r; Radius der Ellipse in y-Richtung

5.10.2.6 Übernehmen vorhandenen Grafiken

procedure DRAWIMAGE (id:s; xl,yt,xr,yb:r);

Zeichnet einen Bitmap aus der Bitmap-Ressourcen in einen Dokument.
Der Bitmap passt sich elastisch an der angegebenen rechteckigen Bereich.

id : Bitmap-ID so wie es in der Ressource angegeben ist.

xl : X-Koordinate des linken Fensterrandes

yt : Y-Koordinate des oberen Fensterrandes

xr : X-Koordinate des rechten Fensterrandes

yb : Y-Koordinate des unteren Fensterrandes

Beispiel : `drawimage('LOGO',150,-8,170,-20);`

procedure DRAWGRAFIC (var g:l; xl,yt,xr,yb:r; /dm:i);

Zeichnet eine Grafik G in einen Dokumentbereich

g : Grafik-Objekt das übernommen wird
xl : X-Koordinate des linken Fensterrandes
yt : Y-Koordinate des oberen Fensterrandes
xr : X-Koordinate des rechten Fensterrandes
yb : Y-Koordinate des unteren Fensterrandes

dm : Option: dm=1 bewirkt dass das Bild seine Proportionen beibehält

Beispiel : `drawgrafic(g1,150,-8,200,-40);`

procedure DRAWMASK (id:n);

Zeichnet eine bereits erstellte Maske in das aktive Dokument.

id : Masken-Identifikator

Beispiel : `drawmask(A000\M1);`

procedure IMPGIMAGE (file:s; var img:gimage; /fmt,compr:i);

Importiert eine externe Bild-Datei (bitmap) in einer Variable.
Das Dateiformat kann dabei konvertiert werden.

file : Dateipfad des zu importierenden Bitmaps
img : Bitmap-Variable wo das Bild kopiert wird
fmt : Bild-Format wenn es konvertiert werden soll: 1=BMP, 2=JPG
compr : Kompressionsfaktor in %; Vorbelegung=100;

Beispiel :
```
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);

Zeichnet einen Bitmap in den aktiven Dokument.
Der Bitmap passt sich elastisch an der angegebenen rechteckigen Bereich,
oder, wenn DM=1 angegeben wird, behält die ursprüngliche Proportionen.
(Siehe Beispiel in Prozedur IMPGIMAGE)

img : Bitmap-Variable
xl : X-Koordinate des linken Fensterrandes
yt : Y-Koordinate des oberen Fensterrandes
xr : X-Koordinate des rechten Fensterrandes
yb : Y-Koordinate des unteren Fensterrandes

dm : Option: dm=1 bewirkt dass das Bild seine Proportionen beibehält

procedure CONVGIMAGE (var img1,img2:gimage; typ2:i; /compr:i);

Konvertiert eine Bitmap-Variable in Format BMP nach JPG-Format.
Der Kompressionsfaktor kann dabei bestimmt werden.

img1 : Bitmap-Variable mit der Bildquelle

img2 : Bitmap-Variable wo das Bild konvertiert wird

typ2 : Erwünschtes Bild-Format in img2: 1=BMP, 2=JPG

compr : Kompressionsfaktor in %; Vorbelegung=100;

Beispiel :
```
logo1,logo2:gimage;
convgimage(logo1,logo2,2,compr=75);
```

5.10.2.7 Zeichnen von Wert- und Zeitskalen

procedure DRAWSCALE (x1,y1,x2,y2,v1,v2,s1,s2:r);

Zeichnet eine Wertskala von X1,Y1 bis X2,Y2 mit Werte von V1 bis V2.

s1 : Schrittgrösse zwischen geschriebene Werte

s2 : Schrittgrösse zwischen gezeichneten Striche

Beispiel :
```
drawscale(20,-100,20,-30,0,100,10,5);
```

procedure DRAWTSCALE (tu:i; xl,yt,xr,yb:r; t1,t2:d);

Zeichnet eine Zeitskala.
Die Schriftgrösse wird vorher mit SETFONT oder DRAW eingestellt.

tu : Skalaformat bestimmt was geschrieben wird:
1 = Monate, Wochen und Tage; 2 = Monate und Jahr;
3 = Wochen; 4 = Tage; 5 = Stunden;

xl,yt,xr,yb : Koordinaten des Vierecks auf dem Papier:
X-Left,Y-Top,X-Right,Y-Bottom

t1,t2 : Start- und Enddatum der Skala

Beispiel :
```
d1:d=21.3.94; d2:d=4.5.94;
drawtscale(3,0,0,200,-20,d1,d2);
```

5.10.2.8 Allgemeiner Zeichnungsbefehl DRAW

procedure DRAW (/)

Zeichnet gemäss den eingegebenen Parameter.
Jedes Parameter besteht aus einem Befehl (1.Zeichen und ev. auch 2.Zeichen) und einem Operand (restlichen Zeichen).

Befehle:

X + number : setzt aktuelle X-Koordinate

Y + number : setzt aktuelle Y-Koordinate

W + number : setzt aktuelle Breite

H + number : setzt aktuelle Höhe

AP + number : setzt Füllmuster (area pattern):
0=leer 1=voll 2=punkte 3=punkte 4=/// 5=\\\ 6=###

AC + number : setzt Vordergrundfarbe;

AB + number : setzt Hintergrundfarbe;

AE + number : setzt Flächenrand: 1=mit Umrisslinie;

BF + number : setzt Boxformat: 0=ohne Rand 1=normal 3=3-D-Box

LW + number : setzt Liniendicke

LS + number : setzt Liniestyl

LC + number : setzt Linienfarbe

LE + number :	setzt Linienendung: 1=normal 2=mit einem Pfeil 3=mit zwei Pfeile
TA + number :	setzt Textausrichtung: 1=links 2=rechts
TF + number :	setzt Textfont
TH + number :	setzt Textgrösse in Punkte
TW + number :	setzt Textdicke: 0=normal 1=fett
TS + number :	setzt Textstyl: 0=normal 1=italic
TC + number :	setzt Textfarbe
TL + number :	setzt maximale Textlänge in mm
TV + number :	setzt vertikale Textausrichtung: 1=Baseline 2=Top 3=Mitte
TR + number :	setzt Textwinkel (in Grad)
GH + Y-value :	zeichnet horizontale Linie aus der Position X-aktuel und Y-angegeben, mit aktuelen Breite.
GV + X-value :	zeichnet vertikale Linie aus der Position X-angegeben und Y-aktuel, mit aktuelen Höhe.
GL(x1,y1,...) :	zeichnet Polygon durch angegebenen Eckpunkten X1,Y1, usw.
GA(x1,y1,...) :	zeichnet Fläche mit angegebenen Eckpunkten X1,Y1, X2, Y2, usw.
GB(xl,yt,xr,yb) :	zeichnet gefüllte Box
GR(xl,yt,xr,yb) :	zeichnet Quader und setzt aktuelen X-, Y-, W- und H-Werte; ideal für Tabellen.
'Text' :	schreibt angegebene Text
V + name :	schreibt der Inhalt der angegebene Variable
F + number :	Setzt Format = Zahl
M0 :	setzt Spiegelfunktion aus
MX :	Spiegelt alles bzgl. X-Achse
MY :	Spiegelt alles bzgl. Y-Achse
NR :	nächste Zeile
NP :	nächste Seite
NX :	nächste X
NY :	nächste Y
ZX + number :	setzt X-Nullpunkt
ZY + number :	setzt Y-Nullpunkt
ON1 :	Automatisches Seitenwechsel einschalten. (Ausschalten mit ON0) Wenn, beim schreiben eines Textes, den maximalen Y-Wert (definiert mit RB-Parameter in OPENG-Befehl) überschritten wird, wird eine neue Seite eingefügt.
OP0 :	Option P0: Printer aussetzen: nach diesem Befehl werden die gezeichnete Elemente nicht gedruckt, sie erscheinen nur am Bildschirm (z.B:Briefköpfe);
OP1 :	Option P1: Printer wieder einsetzen; nach diesem Befehl werden die gezeichnete Elemente wieder gedruckt;
OZ0 :	Option Z0 : Zahlen mit Wert 0 werden nicht geschrieben
OZ1 :	Option Z1: Zahlen mit Wert 0 werden wieder geschrieben;

Beispiel :

```
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,'fertig !');
```

5.10.2.9 Seitenwechsel

Variablen für die Steuerung des Seitenwechsels

page	i	Aktuelle Seiten-Nr; wird mit INSPAGE/SETPAGE aktualisiert
ryt	r	Y-Top-Koordinate von aktuellen Zeile in Dokument
ryb	r	Y-Bottom-Koordinate von aktuellen Zeile in Dokument
rh	r	Zeilenhöhe in Dokument

procedure INSPAGE (nr:i);

Fügt eine neue Seite in die Position NR.
OPENG erzeugt bereits automatisch die erste Seite, deshalb braucht es
INSPAGE nur für die weiteren Seiten.
Wenn NR=0, wird eine Seite am Ende eingefügt.
Die Systemvariable PAGE wird aktualisiert.

Beispiel : `inspage(0);`

procedure SETPAGE (nr:i);

Aktiviert die Seite NR.
Wenn NR=0, wird die letzte Seite aktiviert.
Die Systemvariable PAGE wird aktualisiert.

Beispiel : `setpage(1);`

procedure NEXTPAGE;

Ruft die mit OPENG definierte Seitenwechsel-Prozedur auf;

5.10.2.10 Textumbruch

function SWMM (f,h,w,i:i; s:s):r;

Ergibt die Länge von String S in mm;

f : Font-Nummer
h : Texthöhe in Punkte
w : Textdicke: 0=normal 1=fett
i : Textneigung: 0=normal 1=italic
s : Text dessen Länge ermittelt wird

Beispiel : `if swmm(4,12,1,0,t)>20: nextrow;`

procedure ADJSTXT (f,h,w,i:i; wmax:r; s:s; var txt:l);

Teilt eine Textzeile in mehreren Zeilen so dass die einzelne
Zeilen von TXT die angegebene Breite (in mm) nicht überschreiten.

f : Font nummer
h : Font höhe
w : Fontdicke: 0=Normal; 1=Fett;
i : Fontneigung: 0=Normal; 1=Italic;

wmax : Maximale Breite in mm

s : Textzeile

txt : Zeilenliste als Ergebnis der Teilung

Beispiel : `adjstxt(4,11,0,0,50,s,tx1);`

5.10.2.11 Aktive Dokumente

Es ist möglich grafischen Elemente so zu gestalten dass diese auf einen rechten Mausklick reagieren. Dafür werden die Systemvariablen GACTIV, DRAWN, GEPARS und GEMENU verwendet.

Variablen für die Gestaltung aktiver Dokumenten

gactiv	i	wenn 1: kündigt aktives grafisches Element an; wenn 2: nächste Elemente sind nicht mehr aktiv;
gemenu	ls	Rightclick-Menu eines aktiven grafischen Elements; Jede String enthält: Optionstext:Prozedur; (siehe 6.7.8)
drawn	i	wird 1 wenn der letzte Zeichnungsbefehl nicht geclippt wurde, d.h. das grafische Element ist sichtbar.
gsel	l	Selektierte Grafik in Grafikeditor;

Beispiel :

```
1) In dem Dokumentaufbau:

/... openg(...); ...
gactiv:=1; /kündigt aktives Element an
drawbox(100,-50,200,-70);
if drawn: / wahr wenn box sichtbar ist
  begin
  gepars.nr:s=n1; /bildet Parameter NR in graf.Element
  gemenu.1:s='Prozess:showproc'; /1.Option in Rightclick-Menu
  gemenu.2:s='Termine:term1'; /2.Option in Rightclick-Menu
  end;
gactiv:=0; /nächste Elemente sind nicht mehr aktiv
/.....weitere Zeichnungsbefehle ... editg(...)

2) In der Prozedur SHOWPROC:

if exobj(ordf,gepars.nr)=0:inf('Auftrag fehlt');
else load(ordf,gepars.nr,of);
/....
```

procedure SETAREG (xl,yt,xr,yb:r; h:n; /bc:i; p1:s);

Definiert einen aktiven Bereich (active region) in einen Dokument.

xl : X-links-Koordinate des viereckigen Bereichs

yt : Y-top-Koordinate des viereckigen Bereichs

xr : X-rechts-Koordinate des viereckigen Bereichs

yb : Y-unten(bottom)-Koordinate des viereckigen Bereichs

h : Handler Prozedur

bc : Hintergrundfarbe

p1 : Stringparameter des viereckigen Bereichs

Beispiel : `setareg(12,-20,32,-35,goto1,p1=pfad);`

5.10.2.12 Grafische Komponenten

Wenn in die Ressourcen eines Moduls eine GCM-Tabelle (Grafic Component Masters) gefüllt ist, bietet der Grafikeditor die Möglichkeit parametrisierbaren grafischen Komponenten einzufügen.

Einem grafischen Komponenten können TEKLA-Befehle zugewiesen werden (rechte Mausklcick / Anweisungen), die dann ausgeführt werden, wenn der Anwender mit der linken Maustaste darauf klickt.

NR	NAME	PARS	MASK	G
FE	Formelemente			

FEAZ	Aussenzylinder	d,l:r;	GCFEAZ	x:r=p.l;
				y:r=p.d/2;
				draw(gl(0,0,0,y,x,y,x,0));

MB	Maschinenbau			
MBTO	Toleranzen	form:i=1; tol:s='0.01'; ref:s:referenz; h:r:höhe in mm=7;	GCMBTO	drawmbto;
MBWN	Werksnormen			
MBWNR	Rauheitstempel	nr:i=1;	GCMBWNR	drawmbwnr;
FILO	Firma-Logo		GCTWE1	drawlogo

5.10.2.13 Grafische Puffer

Mit dem Befehl INSGBUF können mehreren grafischen Symbolen an einer Grafik angehängt werden. Der Grafikeditor bietet dann die Möglichkeit diese Symbole eins nach dem andern in die Grafik G einzufügen.

Beispiel :
```
/pr.g ist Grafik die einem Symbolenpuffer zugewiesen wird

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);

Eröffnet die Definition eines Grafikpuffersymbols für die Grafik G.

g : Grafik

id : Identifikator des Symbols (nicht immer notwendig)

name : Bezeichnung des Symbols (erscheint in Grafikpuffermenu)

w : Breite in mm der Fläche die der Symbol enthält;

h : Höhe in mm der Fläche die der Symbol enthält;

Beispiel :
```
insgbuf(pr.g,'','Station A1',100,100);
drawline(2,-5,20,-5);
draws(2,-10,1,'STATION A1');
```

procedure OPENGROUP;

Eröffnet eine Grafikgruppe in der aktiven Grafik. Alle Grafikelemente die danach mit DRAWxxx-Befehle angelegt werden, werden in diese Gruppe gespeichert und erscheinen somit gruppiert.

Beispiel :
```
opengroup;
drawline(x1,y1,x2,y2);
drawline(x3,y3,x4,y4);
closegroup;
```

procedure CLOSEGROUP;

Deaktiviert die zuvor mit OPENGROUP eröffnete Grafikgruppe.

5.10.3 Fliesstext-Dokumente

5.10.3.1 Dokumentobjekte und Editoren

procedure OPENDOC (var doc:l; nr,name:s; w,h:r; /lm,tm,rm,bm:r);

Eröffnet (formatiert) das Dokument DOC.

doc :	Dokument-Objekt
nr :	Ident-Nr des Dokuments
name :	Bezeichnung des Dokuments
w :	Dokumentbreite in mm
h :	Dokumenthöhe in mm
lm :	Linke Rand in mm (optional)
tm :	Obere Rand in mm (optional)
rm :	Rechte Rand in mm (optional)
bm :	Untere Rand in mm (optional)

Beispiel :
```
d1:l;
opendoc(d1,'A1','Bericht',210,290,lm=20,tm=10);
editdoc(d1);
```

procedure EDITDOC (var doc:l; /mode:i);

Öffnet den Dokument-Editor für das Dokument DOC.

doc :	Dokument-Objekt
mode :	Arbeitsmodus: 1=Textrahmen 2=Gesamtfläche (Vorbelegung=1)

Beispiel : `editdoc(pr.doc1,mode=2);`

procedure VIEWDOC (var doc:l; /mode:i);

Öffnet den Dokumentviewer für den Dokument DOC.

doc :	Dokument-Objekt
mode :	Arbeitsmodus: 1=Textrahmen 2=Gesamtfläche (Vorbelegung=1)

Beispiel : `viewdoc(pr.doc1,mode=2);`

5.10.3.2 Allgemeiner Zeichnungsbefehl DOC

procedure DOC (/)

Schreibt gemäss den eingegebenen Parameter.
Jedes Parameter besteht aus einem Befehl (1.Zeichen und ev. auch 2.Zeichen) und einem Operand (restlichen Zeichen).

Befehle:

X + Number : setzt aktuelle Kolonne
Y + Number : setzt aktuelle Absatz
TF + Number : setzt Textfont
TH + Number : setzt Textgrösse in Punkte
TW + Number : setzt Textdicke: 0=normal 1=fett
TS + Number : setzt Textstyl: 0=normal 1=italic
TC + Number : setzt Textfarbe
TP + String : setzt Absatzformat
'Text' : schreibt angegebene Text
V + Name : schreibt der Inhalt der angegebene Variable
F + Number : Setzt Format = Zahl

Beispiel :
```
doc(X1,Y1,TF3,TH12,'Titel',Y5,TH9,'Datum:');
```

5.10.4 Buch-Dokumente

5.10.4.1 Buchobjekte und Editoren

procedure OPENBOOK (var doc:l; nr,name:s);

Eröffnet (formatiert) das Buch DOC.

doc : Buch-Objekt
nr : Buch-Ident-Nr
name : Buch-Bezeichnung

Beispiel :
```
b1:l;
openbook(b1,'A1','Bericht',210,290,lm=20,tm=10);
editbook(b1);
```

procedure EDITBOOK (var doc:l; /zf:r);

Öffnet den Bucheditor für das Buch DOC.

doc : Buch-Objekt
zf : Initialer Zoomfaktor; Vorbelegung = 1.0;

Beispiel :
```
editbook (bericht,zf=1.2);
```

procedure VIEWBOOK (var doc:l; /zf:r);

Öffnet den Buch-Viewer für das Buch DOC.
Parameter: gleich wie in EDITBOOK.

5.11 Dateien und Verzeichnisse

function EXFILE (f:s):i;

Informiert ob eine Datei vorhanden ist.
Ergenis = 1 wenn vorhanden bzw. 0 wenn nicht vorhanden.

F : Dateipfad

Beispiel : `if exfile('C:\temp\test.doc')=0: inf('Dokument fehlt');`

procedure COPYFILE (f1,f2:s);

Kopiert eine Datei.

f1 : Datei die kopiert wird

f2 : Zielpfad für Dateikopie

Beispiel : `copyfile('IMPORT.TXT','D:\TEMP\ABLAGE\IMPORT03.TXT');`

procedure RENFILE (f1,f2:s);

Nennt eine Datei um.

f1 : Datei die umgenannt wird

f2 : Neue Dateiname

Beispiel : `renfile('C:\IMPORT.TXT','C:\IMPORT2.TXT');`

procedure DELFILE (f:s);

Löscht die Datei F.

Beispiel : `delfile('C:\IMPORT.TXT');`

procedure CREATEDIR (dir:s);

Erstellt ein neues Verzeichnis.

dir : Verzeichnispfad

Beispiel : `createdir('C:\MIRAKON\ARCHIV');`

procedure REMOVEDIR (dir:s; /o:n);

Löscht ein vorhandenes Verzeichnis.

dir : Verzeichnispfad

o : Optionen (Zeichenkette):
E = der Verzeichnisinhalt wird gelöscht

Beispiel : `removedir('C:\MIRAKON\ARCHIV',O=E);`

function EXDIR (dir:s):i;

Informiert ob ein Verzeichnis existiert.

dir : Verzeichnispfad

Beispiel : `if exdir('C:\temp'): inf('Verzeichnis gefunden');`

5.12 Datenbanken

Um Objekte aus Datenbanken zu Laden bzw. darin zu speichern muss man die Datenbankenmodule benennen. Dies geschieht durch die Angabe des Datenbank-Ids, ein Punkt und den Modul-Id.

Beispiel: save(KNO.A000,obj);
/speichert Objekt OBJ in Modul A000 der Wissensbank KNO

Für den Zugriff auf Module der standard Arbeitsbank DAT (häufigster Fall) braucht es keine Angabe des Datenbank-Ids:
Beispiel: save(ART,obj);

5.12.1 Handhabung von Objekten

function EXOBJ (wm:n; nr:s; /db:i):i;

Informiert ob ein Objekt in einer Datenbank vorhanden ist.
Ergenis = 1 wenn vorhanden bzw. 0 wenn nicht vorhanden.

wm : Datenbankmodul

nr : Objekt-Nummer

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
if exobj(ART,'123')=0: inf('Artikel 123 fehlt');
if exobj(KNO.A000,'RS')=0: inf('Tabelle RS fehlt');
```

procedure LOAD (wm:n; key:s; var wobj:l; /o,st0:n; max,db:i);

Lädt ein Objekt aus einer Datenbank in einer Variable.

wm : Datenbankmodul

key : Objekt-Nummer

wobj : Variable die das geladene Objekt beinhalten wird

o : Optionen: Reihe von Buchstaben mit folgender Bedeutung:
- C: Kreiert ein neues Objekt falls NR nicht vorhanden;
- N: Formatiert das Objekt neu falls vorhanden;
- U: Lädt nur wenn das Objekt nicht gesperrt ist;
- L: Sperrt das Objekt nach dem Laden;

st0 : Prozedur mit Definition der Initialstruktur;

max : Anzahl Objektfelder die geladen werden sollen;

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
load(ART,'123',a);
load(ARC02.ART,nr1,a1,o=C);
```

procedure SAVE (wm:n; var obj:l; /i,db,compress:i);

Speichert ein Objekt in einer Datenbank.

wm : Datenbankmodul

obj : Objekt das gespeichert wird

i : wenn 1: informiert wenn Datenbank gesperrt ist.

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

compress : compress=1 bewirkt eine Komprimierung des Objekts vor dem Speichern.

Beispiel : `save(ART,a);`

procedure LOADE (wm:n; key,varname:s; varid:n; /db:i);

Lädt ein Teil eines Objekts aus einer Datenbank in einer Variable.
Dieser Teil muss sich auf der ersten Hierarchiestufe des Objekts befinden.

wm : Datenbankmodul

key : Objekt-Nummer

varname : Name der Variable die geladen werden soll.

varid : Variable die das geladene Objekt-Teil beinhalten wird

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
knr:s;
loade(ART,'123','KUNDENNR',knr);
```

procedure SAVEE (wm:n; key,varname:s; varid:n; /db:i);

Ändert die teil VARNAME von die Objeckt KEY mit dem Wert, der in VARID enthalten.
Dieser Teil muss sich auf der ersten Hierarchiestufe des Objekts befinden.

wm : Datenbankmodul

key : Objekt-Nummer

varname : Name der Objekt teil die geändert werden soll.

varid : Variable die das wert enthalten

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
knr:s='K025';
savee(ART,'123','KUNDENNR',knr);
```

procedure DELWOBJ (wm:n; nr:s; /db:i);

Löscht ein oder mehrere Objekte aus einer Datenbank.

wm : Datenbankmodul

nr : Nummer des zu löschenden Objekts. Dieser kann auch als Filter für das Löschen mehreren Objekte dienen (siehe Beispiel).

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
delwobj(ART,'123'); /löscht das Objekt '123' von ART
delwobj(ART,'M_');  /löscht alle Objekte die
                    /mit M beginnen
```

function OBJNAME (wm:n; nr:s; /db:i):s;

Ergibt den Namen eines Objekts aus einer Datenbank ohne ihm laden zu müssen. Vorteil: Schnelligkeit.

wm : Datenbankmodul

nr : Objekt-Nummer

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel : `t:s=objname(ART,a.nr);`

function OBJINF (wm:n; nr:s; /db:i):s;

Ergibt die Merkmalsleiste eines Objekts aus einer Datenbank ohne ihm laden zu müssen. Vorteil: Schnelligkeit.
Die Merkmalsleiste ist ein String die mit INF identifiziert ist und immer an 4.Stelle in der Objektstruktur steht.

wm : Datenbankmodul

nr : Objekt-Nummer

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
merk:s=objinf(ART,a.nr);
if poss('V',merk,1)>0: inf('validiert');
```

procedure PUTOBJINF (var obj:l; inf:s);

Fügt Merkmalsleiste INF in Objekt OBJ ein. Wenn das Objekt bereits eine Merkmalsleiste hat, wird diese ersetzt. Siehe auch Fuktion OBJINF.

Beispiel :
```
putobjinf(pr,'A4,V,X=20');
```

procedure FMTOBJ (var obj:l; nr,name:s; st0:n);

Formatiert ein Objekt mit eine Initialstruktur.
In früheren Versionen wurde die Prozedur FMTWOBJ verwendet.
Diese wird zwar weiterhin aus Kompatibilitätsgründen ausgeführt
jedoch nicht mehr empfohlen.

obj : Objekt das formatiert wird

nr : Objekt-Nummer

name : Objektname

st0 : Identifikator der Prozedur die die Initialstruktur enthält

Beispiel :
```
fmtobj(artikel,'123','Schraube',ARTST0);
```

procedure LOCKOBJ (wm:n; nr:s; /db:i);

Sperrt ein Objekt für alle andere Anwender. Falls es bereits
gesperrt ist, wird die Prozedur nicht ausgeführt und die Systemvariable
DBRES erhält den Wert 2.

wm : Datenbankmodul

nr : Objekt-Nummer

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
lockobj(ART,'A1'); /sperrt Objekt A1
if dbres>0:inf('Objekt ist bereits gesperrt');
```

procedure UNLOCKOBJ (wm:n; nr:s; /db:i);

Entsperrt ein Objekt für alle andere Anwender. Falls es bereits
für einen anderen Anwender gesperrt ist, wird die Prozedur nicht ausgeführt
und die Systemvariable DBRES erhält den Wert 2.

wm : Datenbankmodul

nr : Objekt-Nummer

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
unlockobj(ART,'A1'); /entsperrt Objekt A1
```

function OBJLOCKED (wm:n; nr:s; /db:i):i;

Informiert ob ein Objekt gesperrt ist (Ergebnis=1).

wm : Datenbankmodul

nr : Objekt-Nummer

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
if objlocked(ART,nr)=1: inf('Objekt gesperrt');
```

procedure GENKEY (var key:s; wm:n; prefix:s; ndigits:i; /db:i; startnr:a; wm2:n);

Generiert eine noch nicht existierende Objektnummer.
Dieser Nummer besteht aus einem Präfix und einen Laufnummer.
Funktionsweise: startend mit 0 werden die Laufnummern inkrementiert;
für jeden Nummer, wird untersucht ob ein solches Objekt bereit existiert;

key : Variable die den generierten Nummer enthalten soll

wm : Datenbankmodul

prefix : Präfix von Objektnummer

ndigits : Anzahl Ziffern des generierten Laufnummeranteils

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

startnr : Startnummer, wenn es nicht 0 sein sollte

wm2 : zweiten Datenbankmodul dass auch herangezogen werden soll;
Beispiel: ein Archiv

Beispiel :
```
/angenommen es existieren die Objekte A01 und A03
genkey(k,PROD,'A',2); / ergibt K = 'A02'
```

procedure SETSAVE (objid:n; dm:n);

Ordnet einem Objekt einen neuen Datenbankmodul zu.
Zweck: Wenn der Anwender zwischen verschiedenen Arbeitsmodule schaltet
muss das zu speichernde Objekt umgeleitet werden.

objid : Variable die das Objekt enthält

dm : neues Datenbankmodul

Beispiel :
```
pr:l&save(PROD)
modulid:s='PROD';
/nachdem modulid verändert wird ...
setsave(pr,[modulid]);
```

function WBL (wm:n; form:i; keys:s; /db:i):ls;

Liefert eine Liste von Nr und/oder Namen der Objekte einer Datenbank.

wm : Datenbankmodul

form : 1: nur Nummern 2: nur Namen 3: Nr+Namen

keys : Muster der gewünschten Nummern.

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
l1:l=wbl(PROD,3,'A_,B_');
```

function WBML (db:i):l;

Liefert die Liste aller Modulen eines Datenbanks in Form einer Struktur (Liste von Elementen).

db : Datenbanknummer

Beispiel :
```
l1:l=wbml(KNO); infv(l1);
l2:l=wbml(DAT); infv(l2);
```

procedure FILTERWB (wm:n; var kl:l; kpat:s; /f:n; h,db:i);

Filtriert alle Objektnummern aus der Arbeitsbank WB die in dem Muster
KPAT passen und speichert sie in der Liste KL.
Mit F kann eine Funktion angegeben werden die weitere Bedingungen
überprüft. Diese Funktion soll von Typ Real sein, wenn sie 0 ergibt
wird das jeweilige Objektnummer nicht in KL aufgenommen.

wm : Datenbankmodul

kl : Liste der filtrierten Objekten-Nummern

kpat : Objektnummerfilter

f : Filterfunktion

h : Anzahl Sortierstufen (maximal 2)

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel 1:

```
keys:l;
filterwb(PROD,keys,'A_',f=C7);  /ohne Sortierung

funktion C7:i;
  c7:=0;
  load(PROD,keyi,pr);
  if pr.breite>200:c7:=1;
  end;
```

Beispiel 2:

```
filterwb(PERSONAL,keys,'_',f=C8,h=1); /1-stufige Sortierung

function C8:i;
  c8:=1;
  load(PERSONAL,keyi,pers);
  sortid1:=pers.name;  /Sortierkriterium=Personenname
  end;

mögliches Resultat:
KEYS:list
|
+-- MEIER PETER:s='7803';
+-- MÜLLER HANS:s='3354';
+-- WILLIAMS JOHN:s='5072';
```

Beispiel 3:

```
filterwb(PROD,keys,'_',f=C9,h=2); /2-stufige Sortierung

Funktion C9:i;
  c9:=1;
  load(PROD,keyi,pr);
  sortid1:=pr.kunde; /1.Sortierkriterium=Kundenname
  sortid2:=pr.name;  /2.Sortierkriterium=Produktname
  end;

möglichesResultat:
KEYS:list
|
+- BMW:list;     1.Kunde
|    |
|    +- ANTRIEB:s='G870012';
|    +- WELLE:s='A201456.1';
|
+- MERCEDES:list; 2.Kunde
|    |
|    +- BOLZEN:s='A048765B';
|    +- WELLE:s='A338945.0';
```

procedure SECOBJ (var obj:l; level,cond:i);

Diese Prozedur sichert ein Objekt gegen bestimmten Operationen
durch bestimmten Benutzer.

obj : Objekt das gesichert werden soll

level : Sicherheitsstufe:
1 = erlaubt das Laden und Sehen/Verändern des Objektes
jedoch nicht das Speichern, Löschen und Umbennen;
2 = erlaubt nicht das Sehen, Editieren, Kopieren, Exportieren
und Printen des Objektes;
3 = erlaubt nicht das Laden (d.h benutzen) des Objektes.

cond : Schutzbedingung:
1 = Schutz gegen alle Benutzer mit ungleichen Passwort.
2 = Schutz gegen alle Benutzer mit ungleichen Status.
3 = Schutz gegen alle Benutzer mit ungleichen User-Id.

Beispiel :
```
secobj(pr,1,1);
```

procedure GETREFNR (wm:n; key:s; var nr:a; /db:i);

Ergibt die interne Referenznummer eines Objekts aus einer Datenbank-Modul.
Diese Referenznummer ändert sich nicht wenn das Objekt umbenannt wird.

WM : Datenbankmodul

KEY : Objekt-Nummer

NR : Variable die die interne Referenznummer als Ergebnis enthalten soll

DB : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
ref:a;
getrefnr(ART,'123',ref);
```

procedure GETKEYBYREF (db:n; nr:a; var key:s; /dbext:i);

Ergibt die Objektnummer ausgehend von seiner internen Referenznummer.

DB : Datenbank-Id gemäss Konfiguration

NR : Interne Referenznummer

KEY : Variable die die Objekt-Nummer als Ergebnis enthalten soll.

DBEXT : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
objnr:s;
getkeybyref(DAT,1257,objnr);
```

procedure GETMODBYREF (db:n; nr:a; var wm:s; /dbext:i);

Ergibt die Modul-Id eines Objekts ausgehend von seiner internen Referenznummer.

DB : Datenbank-Id gemäss Konfiguration

NR : Interne Referenznummer

WM : Variable die die Modul-Id als Ergebnis enthalten soll

DBEXT : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
objmod:s;
getmodbyref(DAT,1257,objmod);
```

procedure LOADBYREF (db:n; nr:a; var obj:l; /dbext:i);

Lädt ein Objekt aus einer Datenbank mit Angabe seiner Referenznummer.

DB : Datenbank-Id gemäss Konfiguration

NR : Interne Referenznummer

OBJ : Variable die das geladene Objekt enthalten soll

DBEXT : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
obj:l;
loadbyref(DAT,253,obj);
```

5.12.2 Handhabung von Datenbanken

procedure MAKEDB (file:s);

Erzeugt eine neue Mirakon-Datenbank (.DAT- oder .KNO-Datei).
Dies ist nur dann sinnvoll wenn die Datenbank nicht in der Konfiguration angemeldet ist.

file : Datei (Pfad + Name)

procedure OPENDB (var db:i; file:s; /accmode,compress:i);

Öffnet eine Mirakon-Datenbank (.DAT- oder .KNO-Datei) und schreibt die zugeteilten Datenbanknummer in der Variable DB. Dies ist nur dann sinnvoll wenn die entsprechenden Datenbanken nicht in der Konfiguration angemeldet sind.

db : Variable die die Datenbanknummer empfängt

file : Datei (Pfad + Name)

accmode : 1=Direkt; 2=Über Server (nur für Client\Server version)

compress : compress=1 bewirkt dass die Objekte die in die Datenbank gespeichert werden, davor komprimiert werden.

Beispiel :
```
archiv:i; opendb(archiv,'c:\arc\DATEN02.DAT');
```

procedure CLOSEDB (db:i);

Schliesst die mit DB angegebene Datenbank.

db : Datenbanknummer (gemäss OPENDB-Befehl)

Beispiel :
```
closedb(db1);
```

procedure LOCKWB (wb:n; /i:i);

Sperrt die Arbeitsbank WB für alle andere Anwender. Falls sie bereits gesperrt ist wird die Prozedur nicht ausgeführt und die Systemvariable DBRES erhält den Wert 1.

wb : Datenbank

i : mit I=0 kann eine Information über einen eventuellen Misserfolg unterdrückt werden. Der Ausgangszustand ist immer I=1.

Beispiel :
```
lockwb(ART,i=0);
if dbres>0:inf('Artikeldatei ist gesperrt');
```

procedure UNLOCKWB (wb:n; /i:i);

Entsperrt die Arbeitsbank WB für alle andere Anwender. Falls sie bereits gesperrt ist wird die Prozedur nicht ausgeführt und die Systemvariable DBRES erhält den Wert 1.

wb : Datenbank

i : mit I=0 kann eine Information über einen eventuellen Misserfolg unterdrückt werden. Der Ausgangszustand ist immer I=1.

Beispiel :
```
unlockwb(ART);
```

function WBLOCKED (wb:n):i;

Ergibt 1 wenn die Datenbank gesperrt bzw. 0 wenn sie entsperrt ist.

wb : Datenbank

Beispiel :
```
if wblocked(ART)=1: inf('Artikeldatei ist gesperrt');
```

function EXWMOD (id:s; /db:i):i;

Informiert ob ein Modul in einer Datenbank existiert.

id : Modul-ID

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
if exwmod('ART')=0: inf('Modul ART existiert nicht');
```

procedure INSWMOD (id,name:s; /p,db:i);

Fügt ein Modul in einer Datenbank, falls es nicht vorhanden ist.

id : Modul-ID

name : Modul-Bezeichnung

p : Position in Modulverzeichnis;

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
inswmod('ART','Artikel',p=3);
```

procedure DELWMOD (id:s; /db:i);

Löscht ein Modul aus einer Datenbank, falls es vorhanden ist.

id : Modul-ID

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
delwmod('ART');
```

procedure REBUILDDB (/file:s; id:n; refnr:i);

Rekonstruiert eine Datenbank.

file : Datenbank-Dateiname (als Alternative zur ID)

id : Datenbank-Id gemäss Konfiguration

refnr : wenn 1 werden die internen Referenznummern neu erzeugt (Vorbelegung = 0)

Beispiel :
```
rebuilddb(id=ART);
rebuilddb(file='C:\ARCHIV\ART.DAT',refnr=1);
```

function NKEYS (wm:n):a;

Ergibt die Anzahl Objekte (Datensätze) in der Sammlung WM

WM : Datenbankmodul

procedure COPYDB (id,dir:s; /fname:s);

Kopiert geöffnete Datenbank (Arbeitsbank oder Wissensbank) in einer Verzeichnis.
Zwecks Datensicherungen in Client/Server-Architektur.

id : Datenbank-id so wie es in der Konfiguration steht

dir : Zielverzeichnis

fname : neuer Dateiname (optional)

Beispiel : `copydb('DAT','c:\archiv\',fname='DAT'+today+'.DAT');`

5.12.3 Interactive Zugriff auf Datenbanken

Variablen für den Datenbankzugriff mit LOADER

dbok	i	Erlaubnis zum Laden/Löschen/Einfügen
dbkey	s	Nummer des selektierten Objekts
dbname	s	Name des selektierten Objekts
dbkey0	s	Objekt-Nr-Vorgabe für Prozedur LOADER mit Option A.
dbobjid	s	bestimmt die Objektzeile in LOADER
dbobjcolor	i	Farbe der Objektzeile in LOADER.
idxnr	s	Indexnummer des geladenen Objekts
idxname	s	Indexname des geladenen Objekts
idxtab	l	Indextabelle die zuletzt gebraucht wurde
lmask	i	Schalter für LOADER-Dialogue (Vorbelegung=1): wenn lmask=0, werden die Standard Dialogue für das Einfügen, Kopieren oder Umnennen unterbunden, und Sie können, mit dem MASK-Befehl, die Dialogue selbst gestalten. Nach den Dialogue müssen Sie, mittels DBKEY und DBNAME, Mirakon mitteilen wie das Objekt nummeriert bzw. benannt wird. Wenn DBOK=1 ist übernimmt Mirakon den LOADER-Vorgang (Einfügen, Kopieren oder Umnennen). Siehe Beispiel 1 in Prozedur LOADER: at L5:...

Situationen für den Zugriff auf Datenbanken

Bei den Zugriff auf Datenbanken mit dem LOADER-Befehl, können folgende Situationen genutzt werden:

L1	Vor dem Laden eines Objektes
L2	Nach dem Laden eines Objektes
L3	Vor dem Löschen eines Objektes
L4	Nach dem Löschen eines Objektes
L5	Vor dem Einfügen-Dialog
L6	Nach dem Schliessen des Laders
L7	Vor dem Umnennen eines Objektes
L8	Nach dem Umnennen eines Objektes
L9	Vor dem Kopieren eines Objektes
L10	Nach dem Kopieren eines Objektes
ARC	Beim Archivieren
RST	Beim Restaurieren

procedure LOADER (wm,hp:n;var obj:l; /o,sp,st0,idx,flt,st:n;gf,of,ti,b:s;db:i);

Öffnet den Lader (Dialog zum Laden) damit der Anwender ein Objekt aus einer Datenbank auswählen kann. Der Lader erlaubt auch das Löschen, Umbenennen und Kopieren von Objekten.

wm : Datenbankmodul

hp : Händlerprozedur: diese Prozedur bestimmt was geschehen soll

in den verschiedenen LOADER-Situationen L1,L2,L3,usw.
Durch setzen von DBOK:=0 kann das Laden oder das Löschen verhindert werden.

obj : Variable die das geladene Objekt enthalten soll

o : Optionen: Reihe von Buchstaben mit folgender Bedeutung:

A: Automatische Nummernvergabe.
Wenn der Anwender ein Produkt einfügen will, wird das Nummernfeld mit dem Inhalt der Variable DBKEY0 vorbelegt.

G + Zahl: Mit Knopf "Generieren" für automatische Nummernvergabe.
Wenn dieser Knopf gedrückt wird, sucht das Programm eine neue Nummer, bestehend aus der vorhandenen Präfix im Nummernfeld und der kleinsten Laufnummer mit sovielen Stellen wie der Zahl nach dem G. Diese neue Nummer erscheint dann in Nummerfeld.

J: Automatischer Sprung in der Einfüge-Maske ohne dass der Anwender die Insert-Taste drücken muss.

H + Zahl: Bestimmte Standard-Knöpfe erscheinen nicht:
H1 : INSERT-Knopf (d.h. der Anwender kann kein Objekt einfügen)
H2 : DELETE-Knopf (d.h. der Anwender kann kein Objekt löschen)
H3 : RENAME-Knopf (d.h. der Anwender kann kein Objekt umnennen)
H4 : COPY-Knopf (d.h. der Anwender kann kein Objekt kopieren)
H6 : FILTER-Knopf (d.h. der Anwender kann nicht filtern)
H7 : ENTER-Knopf (d.h. der Anwender kann kein Objekt laden)
H0 : Alle Standard-Knöpfe (H1 bis H7)

M: Automatischer Sprung zum OK-Knopf in der Einfüge-Maske ohne dass der Anwender die Enter-Taste zweimal drücken muss.

O: Mit Archivieren-Knopf.

P: Mit Restaurieren-Knopf.

R: Behält letzte Position in Gruppenindex.

S: Erlaubt Eingabe von Objektnummer.

T: Mit TRANSFER-Knopf. Damit können Objekte zwischen 2 Arbeitsbanken transferiert werden.

U: Geladene Objekte werden nicht gesperrt.

X: Der Index-Präfix erscheint nicht in der Nummereingabefeld sondern in einen separaten Displayfeld.

W + Zahl: Breite des Indexmenus (default -> 28).
Wenn W0: Der Index erscheint nicht

Z: Indexmenu ohne die Option "Alle"

Q: Ermöglicht den Export und Import von externen Objekten (OBJ-Dateien)

sp : Stringformatierungs-Prozedur: Damit kann die Objektliste formatiert werden. (siehe Beispiel 4)

st0 : Prozedur mit Definition der Initialstruktur

idx :	Tabelle mit Inhaltsverzeichnis
flt :	Tabelle mit Filterdefinitionen
st :	Sortiertabelle
gf :	Gruppenfilter: Damit kann der Gruppenindex begrenzt werden.
of :	Objektenfilter: damit kann die Objektenliste begrenzt werden
ti :	Titel für Laderfenster.
b :	Optionale Knöpfe: Syntax: 'Knopfname1:Prozedur1,Knopfname2:Prozedur2,...'
db :	Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel 1:

```
loader(PROD,hp1,pr);

procedure HP1;
at L2:
  prname:=pr.nr+' '+pr.name;
  dspsf(D1);
at L3:
  if pr.status=2:
    begin
    inf('Löschen nicht erlaubt');
    dbok:=0;
    end;
at L5:  /vor dem Einfügen neuen Objekten
  lmask:=0;
  unserenr,unsereprefix,unserename:s='';
  mask(NEUARTIKEL,o=A);
  if escaped: dbok:=0&exit;
  fehler:i=0;
  controlliere(unserenr,unsereprefix,fehler);
  if fehler: dbok:=0&exit;
  dbkey:=unsereprefix+'.'+unserenr;
  dbname:=unserename;
  dbok:=1;
```

Beispiel 2:

```
dbkey0:='A-'+today;
loader(PROD,hp1,pr,o=AJG3);
```

Beispiel 3:

```
loader(ORD,hp2,auftrag,o=W0,b1='_Termine');

procedure HP2;
at LB1:
load(ORD,dbkey,auftrag);
mask(TERM);
save(ORD,auftrag);
```

Beispiel 4:

```
loader(PROD,lp,pr,SP=fprod);

procedure FPROD;
  obj:l;
  load(PROD,dbkey,obj);
  dbobjid:=obj.nr+' '+obj.kunde;
  /oder:  dbobjid:=sf(Vdbkey,M15,Vdbname);
  end;
```

Beispiel 5:

```
loader(PROD,lp,pr,gf='A_,B_');
/nur die Gruppen A_ und B_ erscheinen zur Auswahl
```

function LASTOBJS (wm:n; / db:i);

Ergibt sie Liste der zuletzt geladene Objekte von Modul M;

wm : Datenbankmodul

db : Nummer einer externen Datenbank (mit OPENDB geöffnet)

Beispiel :
```
nr:s;
options:l=lastobjs(prod);
selid(nr,options,'Letzte Produkte',20,-5,20,6);
load(prod,nr,pr);
```

5.12.3.1 Inhaltverzeichnisse in LOADER

Datenbankzugriff mit Inhaltverzeichnisse
Der Zugriff auf Datenbanken kann mit einem Inhaltsverzeichnis wesentlich freundlicher gestaltet werden.

Vorgehen:
- Inhaltsverzeichnis-Tabelle (auch Gruppenindex genannt) angelegen.
- Loader mit optionalen Parameter IDX=Tabellen-Id aufrufen

Beispiel: Inhaltsverzeichnis für Produkte-Datenbankmodul:

NR	NAME
ET	Einzelteile
ETG	Gussteile
ETGG	Gussteile gross
ETGK	Gussteile klein
ETR	Rotationsteile
E	(alle Einzelteile)
BG	Baugruppen

Die hierarchische Gruppierung des Inhaltsverzeichnisses wird durch die Indentierung der NR-Einträge und soll dem Anwender helfen, möglichst schnell und ohne die Objektnummer zu kennen, das gesuchte Objekt zu finden.
Die Gruppennummer in der NR-Spalte muss sogleich der Präfix der Objektnummer sein, nur dann befindet sich das Objekt in der Gruppe.

Beispiel: Gruppe ETGG enthält die Objekte ETGG001, ETGG007A, ETGGTEST, usw.

5.12.3.2 Initialstrukturen in LOADER

Wenn ein Objekt neu in einer Arbeitsbank eingefügt wird, muss Mirakon seine Initialstruktur wissen. Dieses wird in Form einer Prozedur beschrieben. und dem LOADER mitgeteilt.

Diese Prozedur enthält Deklarationen von Variablen (Objektfelder) die beim Erstellen eines neuen Objekts in der Objektliste angehängt werden und somit die anfängliche Objektstruktur definieren.
Beispiel : `/Prozedur KUNST0, die Initialstruktur eines Kunden:`

```
adr:l;    / Adresse
tel:s;    / Telefon
plz:s;    / Postleitzahl
bem:l;    / Bemerkungen
ereig:l;  / Liste von Ereignisse
eig:l;    / Liste von Eigenschaften
ums:r;    / Umsatz
```

```
/Während der Anwendung können die Listenfelder EREIG und
/EIG mit neuen Listenelementen gefüllt werden sowie neue
/Felder an das Objekt angehängt werden.
/Die Initialstruktur ist ein Skelett das im Laufe der
/Anwendung mit Daten gefüllt wird.
```

5.12.3.3 Aufbau von Filtern in LOADER

Die Suche von Objekten in einer umfangreichen Arbeitsbank kann mit benutzerdefinierten Filtern unterstützt werden.
Vorgehen:

1 LOADER muss mit optionellen Parameter FLT (Filtertabelle) aufgerufen werden;
Beispiel: loader(PROD,lp,pr,flt=PRODFLT);

2. Filterkriterien-Variablen global anlegen (in INIT-Prozedur);
Z.B: fltnr:s:filter für produktnummer='_';
fltname:s:filter für produktname='_';
fltkun:filter für kunde='_';

3. Filtertabelle mit Spalten NAME, MASK und A anlegen:
Die Filtertabelle enthält ein Filter pro Zeile;
Jeder Filter kann ein Dialog aufrufen (Dialogmaske in MASK-Spalte);
Die A-Spalte enthält die Tekla-Befehle die entscheiden ob ein Objekt durch den angelegten Filter durchkommt oder nicht;

NR	NAME	MASK	A

F1	Allg.Filter	MF1	if o.nr out (fltnr):exit;
			if o.name out (fltname):exit;
			load(PROD,o.nr,o);
			ok:=o.ad.kunde in (fltkun);

F2	Techn.Daten	MF2	.....

4. Dialogmasken anlegen, am besten neben der Filtertabelle;
Hier werden Werte für die verschiedenen Filterkriterien-Variablen gefragt, z.B: Produktnummerfilter in FLTNR, Produktnamenfilter in FLTNAME und Kundenfilter in FLTKUN;

Funktionsweise:

1. Der Anwender drückt den Knopf FILTER (während LOADER-Ausführung);

2. Mirakon öffnet Filtermenu gemäss Filtertabelle;

3. Anwender wählt ein Filter;

4. Mirakon öffnet die entsprechende Dialogmaske;

5. Anwender antwortet auf die Fragen nach Filterkriterien und drückt OK;

6. Mirakon startet Filterprozess:
Jedes Objekt wird mit den ersten 4 Felder in die Variable O geladen,
Die Variable OK wird auf 0 gesetzt;
Die Befehle in der A-Spalte werden ausgeführt;
Wenn danach die Variable OK = 0, wird das Objekt ausgefiltert;
Wenn ein Filterkriterium weitere Objektfelder benötigt, muss das komplete Objekt mit LOAD(...) geladen werden;

7. Danach erscheinen in LOADER-Menu nur die Objekte die nicht ausgefiltert wurden;

5.12.4 Schneller Zugriff auf Objektdaten

Manchmal wünscht man die Auflistung einige Objektfelder von vielen grösseren Objekten (tausende). Dies bedeutet das Laden jedes Objekt und das Herauslesen der erwünschten Werte, was viel Zeit (Minuten) dauern kann.

In dieser Situation bietet Mirakon den Einsatz von FAVTABs (Fast Access Variable Tables), Tabellen für den schnellen Zugriff auf Objektfelder. Die Idee dahinter ist die doppelte Speicherung der gewünschte Felder aller Objekten in einer einzigen Tabelle. Diese Tabelle ist das Objekt mit dem Nummer $FAVTAB. Somit beschleunigen sich das Suchen, Listen, Filtern und Sortieren um ein vielfaches.

Vorgehen um eine FAVTAB anzulegen:

1) in Datenbankeditor, gewünschte Sammlung selektieren, Status-Knopf drücken, und die Option "Variablen mit schnellen Zugriff" aktivieren.

2) rechtsklicken auf der selektierte Sammlung, Option "Variablen mit schnellen Zugriff" wählen. Eine leere Tabelle steht dann zum editieren bereit.

3) Tabellenspalten anlegen. Jede Spalte entspricht einen Objektfeld.
Ein Punkt bedeutet ein Hierarchiesprung. Ein Doppelpunkt deutet ein Feld innerhalb eines zusammengesetzen Typs.
Beispiele:
KUNDE bedeutet: Objekt.Kunde
BS:SS.1:NAME bedeutet: Name des ersten Elements von Objekt.bs

procedure FINDINFT(wm:n; var colids,exprs,rowids:l; /mode,sort,msg,db:i; rows,sep:s)

Ergibt in ROWIDS die Zeilen, die den Ausdruck in EXPRS für jede Spalte in COLIDS erfüllen.

wm :	Datenbankmodul
colids :	Spalten die gefiltert werden. Z.B. colids:ls=('VD.PREIS','LIEFERANT');
exprs :	Bedingungen die für jede Spalte auszuwertet werden. Die Reihenfolge entspricht die Spaltenreihenfolge in COLIDS. Z.B. : exprs:ls=('rs(tc.1)>100;','tc.1='L001';');
rowids :	Ergebnis: Liste der Zeilennummern (Objektnummern), die die Bedingungen in EXPRS erfüllt haben.
mode :	1=AND (alle Spalten müssen die Bedingungen erfüllen), 0=OR (nur eine Spalte muß ihre Bedingung erfüllen)
sort :	1=sortiert ROWIDS alphabetisch, 0=keine Sortierung
msg :	1=zeigt Anzeige für jede Zeile (unten rechts)
db :	Optional: externe Datenbank (mit OPENDB geöffnet)
rows :	Spalten das in ROWIDS geschrieben werden sollen. Wenn nicht definiert, nur die Spalte NR wird geschrieben. Beispiel: rows='NR|PREIS|LIEFERANT';
sep :	Separator der verwendet werden soll, wenn mehrere Spalten geschrieben werden. Vorbelegung ist "#".
asle :	1=Ergebnis wird als Struktur (Liste von Elementen) ausgegeben
asls :	1=Ergebnis wird als Liste von Longstrings ausgegeben
elid :	Wert für Element-Id wenn Ergebnis eine Struktur sein sollte

leid : Id der Listelemente inder Ergebnisstruktur (wenn asle=1)

acid : Spalten-Id für die Gruppierung von Elementen

Beispiel :
```
colids:ls=('PREIS');
exprs:ls=('rs(tc.1)>100');
rows:s='LIEFERANT|PREIS|NR';
result:l;
findinft(PRODS,colids,exprs,result,sort=1,rows=rows);
/ Ergibt Lieferant, Preis und Objektnummer für allen
/ Objekten (z.B. Aufträgen) dessen Preis grösser 100
/ sortiert nach Lieferant + Preis + ObjektNr
```

Beispiel einer FAVTAB:

NR	PREIS	LIEFERANT	QUANTITAT
A01	100	L001	90
A02	120	L001	32
A03	50	L002	4
A04	200	L003	66

5.12.5 Query Tables

Query Tables erlauben eine relationalen Datenaufbau für bestimmte Felder einer Objektstruktur innerhalb einer Datenbank.

Diese Datenfelder werden somit redundant gespeichert aber immer in Arbeitspeicher gehalten. Dies garantiert einen sehr schnellen Zugriff auf diese Daten ohne die einzelnen Objekten ins Hauptspeicher laden zu müssen.

procedure OPENQUERYTABLE (var qth:i; modid:n; tabid:s);

Öffnet die Query Table TABID von Modul MODID:
- Tabelle wird in Hauptspeicher geladen (falls es noch nicht geschehen ist)
- Interner Tabellen-Handle QTH wird ermittelt

qth : Query-Table nummer (handle)

modid : Datenbankmodul

tabid : Query-Table-Id

Beispiel 1:
```
qt1:i;
openquerytable(qt1,PROD,'MAIN');
```

procedure NEWQUERYTABLE (var qth:i; tabid:s);

Erzeugt eine neue provisorische Query Table im Hauptspeicher des Client-PCs.

qth : Query-Table nummer (handle)

tabid : Query-Table-Id

Beispiel 1:
```
qt2:i;
newquerytable(qt2,'LOCAL');
```

procedure CLOSEQUERYTABLE (qth:i);

Schliesst eine geöffnete Query Table im Hauptspeicher des Client-PCs.

qth : Query-Table nummer (handle)

Beispiel 1:
```
closequerytable(qt2);
```

function NQTROWS (qth:i):a;

Ergibt die Anzahl Zeilen in der Query-Table QTH

qth : Nummer einer Query Table (handle)

function NQTCOLS (qth:i):i;

Ergibt die Anzahl Spalten einer Query-Table

qth : Nummer einer Query Table (handle)

procedure GETQUERYTABLE (qth1,qth2:i; colids,filter:ls; keys:s);

Füllt eine Querytable (qth2) mit einem Auszug aus einer anderen Querytable (qth1)

qth1 : Nummer der Quellen-Query-Table

qth2 : Nummer der neu zu füllenden Query-Table

colids : Liste der zu übertragenden Spalten

filter : Filter für die zu übertragenden Zeilen aufgrund der Werten in bestimmten Spalten

keys : Nummerfilter für die zu übertragenden Zeilen

Beispiel 1:

```
qt1,qt2:i;
openquerytable(qt1,PROD,'MAIN');
newquerytable(qt2,'LOKAL');
cols:ls=('NAME','BESTAND','ORT','ZUSTAND');
filt:ls=('BESTAND>0','ZUSTAND="AKTIV"');
getquerytable(qt1,qt2,cols,filt,'_');
viewqt(qt2);
inventar1,inventar2:l;
expqt2txt(qt2,'NR,NAME,BESTAND,ORT',inventar1,',');
expqt2st(qt2,'NR,NAME,BESTAND,ORT',inventar2);
```

procedure VIEWQT (qth:i);

Visualisiert eine Querytable.
(Siehe Beispiel in getquerytable)

qth : Nummer der Query-Table

procedure EXPQT2TXT (qth:i; var colids:ls; var txt:l; sep:s);

Exportiert den Inhalt einer Querytable in einen Text (list of strings).
(siehe Beipiel in getquerytable)

qth : Nummer der Query-Table (handle)

colids : Liste der zu übertragenden Spalten

txt : Liste die gefüllt wird

sep : Separator zwischen den einzelnen Felder in einer Zeile

procedure EXPQT2ST (qth:i; var colids:ls; var st:l; / grpid,grpenr:s; nss:i);

Exportiert den Inhalt einer Querytable in einer Struktur (list of element).
Eine Zeile aus der Tabelle entspricht ein Element in die Zielstruktur. Die Spaltenwerte werden dann als Parameter des Elements gespeichert.
(siehe Beipiel in getquerytable)
Optional kann die resultierende Struktur mit GRPID groupiert werden.

qth : Nummer der Query-Table (handle)

colids : Liste der zu übertragenden Spalten

st : Struktur die gefüllt wird

grpid : Id der Spalte die die Gruppierungskriterien enthält.

grpenr : ID der Elemente die eingefügt werden

nss : in Fall einer Gruppierung, wenn nss=1, (no sub structure) werden

nur die Gruppen gebildet, ohne Elementen als Substruktur.
Wenn nss=2 werden die Elemente ohne Substruktur eingebaut.

procedure EXPQT2XLS (qth:i; var colids:ls; xlc,xlr:i);

Exportiert den Inhalt einer Querytable in einer geöffnete Excel-Blatt.

qth : Nummer der Query-Table (handle)

colids : Liste der zu übertragenden Spalten

xlc : Spaltennummer in Excel ab welche die Daten gefüüllt werden

xlr : Zeilennummer in Excel ab welche die Daten gefüüllt werden

Beispiel 1:
```
colids:ls=('NR','NAME','BESTAND');
expqt2xls(qt2,colids,2,9);
```

procedure EXPLORER (wm:n; var obj:l; /colst,st0,idx,o,hp:n; ti,b:s; us:i);

Öffnet den Explorer (Dialog Fenster) damit der Anwender ein Objekt aus einer Datenbank auswählen kann. Der Explorer erlaubt auch das Löschen, Umbenennen und Kopieren von Objekten.

Bis auf den Parameter COLST sind alle andere Parameter zu handhaben wie in LOADER

wm : Datenbankmodul

obj : Variable die das geladene Objekt enthalten soll

hp : Händlerprozedur (siehe LOADER)

o : Optionen (siehe LOADER)

st0 : Prozedur mit Definition der Initialstruktur

idx : Tabelle mit Inhaltsverzeichnis

colst : Tabelle mit Spalten-Merkmale der Explorer-Gitter

ti : Titel für Laderfenster.

b : Optionale Knöpfe:
Syntax: 'Knopfname1:Prozedur1,Knopfname2:Prozedur2,...'

us : wenn us=1 (Vorbelegung), kann der Anwender selber Spalten formatieren. Die anwenderspezifische Formatierung wird danach gespeichert und immer wieder verwendet. Sollte dies nicht erwünscht sein, muss us=0 gesetzt werden.

Beispiel 1:
```
explorer(PROD,pr,colst=PRX,us=0);

Aufbau der der Tabelle PRX:
- Spalte NAME: Spaltenbezeichnungen in der Gitterfeld
- Spalte QTCOLID: Spalten-Id in ersten Query-Table
- Spalte I: Bedingung für das erscheinen der Spalte
- Spalte TYPE: Typ der Spalteninhalt
```

NR	NAME	QTCOLID	I	TYPE
1	Nummer	NR	1	S
2	Bezeichnung	NAME	1	S
3	Bestand	BEST	appid='LAGER'	R

5.13 Dialog-Steuerung

Variablen für Dialogsteuerung

fid	s	Identifikator des selektierten Feldes
stageid	i	Identifikation der aktuellen Dialogfenster (vergeben von Autor in EDITG, EDITST oder EDITPLAN);
fchanged	i	diese Variable informiert ob das deselektierte Dialogfeld verändert wurde (fchanged=1) oder nicht (fchanged=0)
jumpfok	i	wenn diese Variable = 0 gesetzt wird, in den Dialogfeldbefehle, bleibt der cursor in gleichen Feld.
okayed	i	wird 1 nachdem der Anwender eine Dialogmaske mit OK verlassen will, sonst ist okayed = 0
escaped	i	wird 1 nachdem der Anwender eine Dialogmaske mit Escape verlassen will (oder hat), sonst ist escaped = 0
exitok	i	Wenn 0 gesetzt in der Dialogfinalisierung, verhindert das Verlassen der Dialogmaske, sonst ist exitok = 1
xlfa	r	X-left-Koordinate der zuletzt selektierten Feld
xrfa	r	X-right-Koordinate der zuletzt selektierten Feld
ytfa	r	Y-top-Koordinate der zuletzt selektierten Feld
ybfa	r	Y-bottom-Koordinate der zuletzt selektierten Feld
infmode	i	1 = Informationen erscheinen am Bildschirm und Applikation wird unterbrochen. 2 = Informationen werden in INFL geschrieben und Applikation wird nicht unterbrochen
infl	l	Informationen behälter wenn INFMODE=2

5.13.1 Cursor, Tastatur, Lautsprecher

procedure SETCURSOR (mode:i);

Ändert Cursorform.

mode : Cursorform: 1=normal 2=Sanduhr

Beispiel : `setcursor(2); tuetwas; setcursor(1);`

procedure BEEP;

Lässt einen Beep über den PC-Lautsprecher ertönen.

function INPUTKEY:i;

Ergibt den Wert der zuletzt gedrückten Taste (ANSI-Code).

Tastenwerte:

Back = 8 Home = 371 PgUp = 373
Tab = 9 End = 379 PgDn = 381
Enter = 13 Left = 375 Del = 383
Esc = 27 Right = 377 Ins = 382
BTab = 315 Up = 372 Down = 380
F1..F10 = 359..368
Shift-F1..Shift-F10 = 384..393

Beispiel : `if inputkey=27:exit;`

5.13.2 Informationen zeigen

procedure OPENMSG (s1,s2:s; /cm,wait:i);

Eröffnet links unten eine Messagebox mit zwei Textzeilen s1 und s2.

s1 : erste Zeile

s2 : zeite Zeile

cm : Ändert Cursorform: 1:normal 2:Sanduhr

wait : Anzahl 1/100-Sekunden die gewartet werden

Beispiel :
```
openmsg('Dokument wird erstellt',cm=2,wait=50);
```

procedure CLOSEMSG;

Schliesst eine Messagebox

procedure INF (s:s; /z,pos:i);

Zeichnet eine Infobox mit dem Text s und einem OK-Knopf.

s : Informationstext

z : Mit Z kann einen Standard-Sufix anghängt werden:
1 = ' nicht vorhanden'
2 = ' existiert bereits !'
3 = ' nicht gültig !'
4 = ' nicht kompatibel !'
5 = ' nicht erlaubt !'

pos : Infobox position: 1=oberes links, 2=Unteres Recht (Vorbelegung) 3=Mitte

Beispiel :
```
x:r=a+b;  anr:s='123456';
inf('X='+sr(x,2));
if x>1200:inf('Gewicht zu hoch !');
inf('Artikel '+anr,z=1);
```

procedure INFV (v:n; /bars,pos:i; w,h:r);

Informiert über den Wert der mit V angegebenen Variable.
Wenn bars=1 angegeben wird, wird den Speicherbedarf (in Bytes),
als grafischen Balken, von allen Unterpositionen angezeigt.

v : Variablenname

bars : bars=1 bewirkt eine grafische Darstellung
des Speicherverbrauchs einzelner Positionen

pos : Infv position: 1=oberes links, 2=Unteres Recht (Vorbelegung) 3=Mitte

w : Fensterbreite in Textzeichen

h : Fensterhöhe in Textzeilen

Beispiel :
```
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);

Eröffnet ein Reporter-Fenster. Danach können Texte mit der
Prozedur REPORT darin geschrieben werden.

t : Fenstertitel

w : Fensterbreite in Texteinheiten

h : Fensterhöhe in Texteinheiten

Beispiel : `openreport('Protokol');`

procedure REPORT (dr,c:i; s:s; /w:i);

Schreibt eine Textzeile in das geöffnete Reporter-Fenster.

dr : Zeilensprung, bezogen auf die zuletzt geschriebene Zeile.

c : Spaltenposition.

s : Textzeile

w : Wartezeit in 0.01 Sekunden

Beispiel : `report(1,3,'Artikel '+a.nr,w=50);`

procedure CLOSEREPORT;

Schliesst das geöffnete Reporter-Fenster.

5.13.3 Handhabung von Dialogfelder

procedure DSPF (/);

Die angegebenen Dialogfelder werden in die aktuelle Dialogmaske neu gezeichnet.

Beispiel : `dspf(A1,A2,B1,C1);`

procedure DSPSF (ID:N);

Das angegebene Dialogfeld wird in die Statusfläche neu gezeichnet.

ID : Feld-Identifikator

Beispiel : `dspf(D1);`

procedure SELF (ID:N);

Das angegebene Dialogfeld wird selektiert, d.h. der Cursor springt dort hin.
Die Standard-Felder OK und Abbrechen können mit ZOK bzw. ZESC selektiert werden.

ID : Feld-Identifikator

Beispiel : `self(ZOK); /selektiert OK-Knopf`

procedure JUMPF (J:I);

Springt zu einem anderen Feld.

J : Anzahl Felder die gesprungen werden

Beispiel :
```
jumpf(1);    springt zum nächsten Feld
jumpf(-1);   springt zum vorigen Feld
```

procedure DELF (id:n);

Das angegebene Dialogfeld oder Grafikelement wird gelöscht.
ID kann auch ein Muster sein.

id : Dialogfeld-ID oder Muster

Beispiel : `delf(A1); delf(B_);`

procedure SETF (id:n; stat:i; /bc,fc:i);

Ändert den Status von Dialogfelder und Grafikelementen in dem aktuellen Dialog.
ID kann auch ein Muster sein.

id : Feld-ID oder Muster

stat : Statuswert:
1=normal;
2=gesperrt: erscheint grau und kann nicht selektiert werden;
4=unsichtbar: erscheint nicht;

bc : Hintergrundfarbe

fc : Vordergrundfarbe (Textfarbe)

Beispiel :
```
setf(A_,2);
/alle Felder die mit A starten, werden gesperrt
```

procedure SETFCOL (id:n; bcol,fcol:i; /pbcol,pfcol:i);

Ändert die Farben eines Dialogfeldes.

id : Identifikator des Feldes (kann auch ein Muster sein).

bcol : Hintergrundfarbe

fcol : Textfarbe

pbcol : Hintergrundfarbe in gesperrten Zustand;

pfcol : Textfarbe in gesperrten Zustand;

Beispiel :
```
setfcol(A_,3,0,pbcol=5);
```

procedure RSETF (id:n; /path:s; var lo:l);

Baut ein Dialogfeld neu auf. Dies ist immer dann nötig wenn die Struktur
eines Felds (Lister, Menu oder Strukturediteors) sich verändert hat.
Wenn ID nicht angegeben wird, wird das selektierte Feld gewählt.

id : Feld-ID

path : neue Strukturposition (bei EDITST)

lo : Optionenliste; muss dann angegeben werden,
wenn ein Menu-Feld eine neue Optionenliste erhalten soll

Beispiel :
```
menu1:ls=('Option1','Option2 neu')
rsetf(M1,lo=menu1);
```

procedure SETTIP (id:s; tips:ls);

Ordnet einen Windows-Tip einem Feld ID zu.
Zweck: Wenn der Anwender mit dem Maus, eine Sekunde über einem Feld
verweilt, erscheint ein Fenster mit Zusatzinformation.

id : Feld-ID

tips : Informationstext

Beispiel :
```
info:ls=('Dieses Feld','bedeutet ...');
settip(A1,info);
```

5.13.4 Handhabung von Dialogmasken

procedure MASK (id:n; /ok,o,vl:n; mode,nr:i);

Öffnet eine Dialogmaske und startet den Dialog mit dem Anwender.
Eine Dialogmaske kann auch mehrere Seiten enthalten. Diese ercheinen
als Register oben rechts.

id : Dialogmaske-ID

ok : Prozedur die aufgerufen wird, wenn die OK-Taste gedruckt wird

o : Optionen: N=ohne OK-Knopf; A=mit Abbrechen-Knopf

vl : Sichtbare Layers

mode : Arbeitsmodus: 1=Felder andere Masken werden unselektiertbar

nr : Seiten-Nr (falls die Dialogmaske mehreren Seiten enthält)

Beispiel : `mask(DATEN,ok=kontroldaten,mode=1,o=A);`

procedure OPENSTAT (id:n);

Öffnet eine Statusmaske oben links in der Anwendungsfläche.

id : Dialogmaske-ID

Beispiel : `openstat(STAT);`

procedure OPENMAIN (xl,yt,w,h:r; id:n);

Öffnet Hauptmenu in der Anwendungsfläche.

xl : X-left-Koordinate des Menüs (Texteinheiten) in der Arbeitsfläche

yt : Y-top-Koordinate des Menüs (negativer Wert !) in Texteinheiten;
Das Programm sorgt dafür dass die Statusfläche nicht gedeckt wird;

w : Breite des Menufelds (in Texteinheiten);

h : Höhe des Menufelds (in Texteinheiten);

id : Hauptmenu-Tabelle

Beispiel : `openmain(1,-3,30,40,MAIN);`

procedure OPENTABS (p:i; x,y:r; t:s; id:n);

Öffnet ein Registermenü in der Anwendungsfläche.

p : Positionierung:
0: füllt gesamte Arbeitsfläche (empfohlene Wert);
In diesem Fall haben die Parameter X und Y keinen Einfluss;
1: Zentriert in der Arbeitsfläche:
X : linke Rand = rechte Rand (in Texteinheiten)
Y : obere Rand = untere Rand (in Texteinheiten)
2: Zentriert in der Arbeitsfläche:
X : Breite (in Texteinheiten)
Y : Höhe (in Texteinheiten)
7: Unten rechts an Arbeitsflächenkante gebunden;
X : X-left Koordinate (in Texteinheiten);
Y : Y-top Koordinate; negativer Wert ! (in Texteinheiten) ;

x : Angaben zur Positionierung: Rand oder Breite

y : Angaben zur Positionierung: Rand oder Höhe

t : Titel für Registermenu; erscheint oben links;

id : Registermenu-Tabelle

Beispiel :
```
opentabs(0,0,0,'',MAIN);
opentabs(7,0,-3,'Hauptmenu',MAIN);
```

procedure RSETM (id:n);

Baut die Dialogfenster ID neu auf. Dies ist immer dann nötig wenn die
Adressen oder Werte der gezeigten Variablen sich geändert haben.
Wenn ID nicht angegeben wird, wird das selektierte Fenster gewählt.

id : Dialogmasken-ID

Beispiel :
```
rsetm(MAIN);
```

procedure SELM (id:n; /fnr:i);

Selektiert eine Dialogfenster.

id : Dialogmaske-ID

fnr : Feld das selektiert werden soll (optional);

Beispiel :
```
selm(MAIN,fnr=2);
```

procedure DELM (id:n);

Löscht die angegebene Dialogfenster.

Beispiel :
```
delm(MAIN);
```

procedure SETML (layer,mode:i; /fnr,rset:i);

Setzt in der aktiven Dialogmaske einen bestimmten Layer sichtbar/unsichtbar.

layer : Layernummer

mode : 0:unsichtbar 1:sichtbar

fnr : bestimmt die Seite wo der Layer geändert wird

rset : mit RSET=1 wird die Seite neu aufgebaut (aber unsichtbar)

Beispiel :
```
setml(2,1);
```

procedure SECDATA (/);

Ermöglicht Daten zu sichern bei der Eröffnung einer Maske.
Dieser Befehl wird in der Initialisierung der Maske aufgerufen
und bewirkt die Erscheinung eines RESET-Knopfs (neben den OK-Knopf).
Damit kann der Anwender, nachdem er Daten bereits verändert hat,
die Ursprungssituation restaurieren.

/ : Variablen die zu sichern sind (mit Komma getrennt)

Beispiel :
```
secdata(p,v1);
/sichert alle aktuellen Parameter und die Variable V1
```

5.13.5 Standard Dialogue

procedure ASK (t:s; var txt:ls; but:s; var a:i);

Öffnet eine Dialogmaske mit einer Frage und die dazu
passenden Antworten als Knöpfe, lässt der Anwender
einen Knopf drücken und gibt in A den Nummer des gedrückten
Knopfes. Die Dialogmaske ersheint in Zentrum des Bildschirms.

t : Titel der Dialogmaske

txt : Text in Dialogmaske (list of strings)

but : Knopftexte (mit Komma getrennt)

a : Variable die die Antwort (Knopfnummer) empfängt;

Beispiel :
```
antwort:i;
txt:ls=('Wollen Sie','löschen ?');
ask('Frage',txt,'Ja,Nein',antwort);
if antwort=2: exit; /Nein wurde gewählt
```

procedure SELID (var id:s; var opts:l; t:s; xl,yt,w,h:r);

Öffnet eine Dialogmaske mit einem Menu, lässt der Anwender eine Option wählen und gibt seine Antwort zurück in ID.

id : Variable die die Antwort empfängt

opts : Liste der Optionen (list of strings)

t : Titel der Dialogmaske

xl : X-Left-Position der Dialogmaske in Texteinheiten

yt : Y-Top-Position der Dialogmaske in Texteinheiten

w : Breite der Dialogmaske in Texteinheiten

h : Höhe der Dialogmaske in Texteinheiten

Beispiel :
```
antwort:s;
options:ls=(G='Gross',M='Mittel',K='Klein');
selid(antwort,options,'Grösse',20,-5,20,6);
if antwort='G': inf('Gross wurde gewählt !');
```

procedure CHOOSEFILE (var fp:s; sp:s);

Startet Windows-Dialog um eine Datei auszuwählen;

fp : Variable die den gewählten Dateipfad enthalten soll

sp : Startpfad

Beispiel :
```
datei:s;  choosefile(datei,'C:\MIRAKON');
```

procedure CHOOSEFOLDER (var dir:s);

Startet Windows-Dialog um eine Verzeichnis auszuwählen;

dir : Variable die den Startpfad angibt und den gewählten Pfad zurückgibt

Beispiel :
```
Pfad:s='C:\MIRAKON';
choosefolder(pfad);
```

procedure SAVEFILE (var fn:s; /ltxt:ls);

Startet Windows-Dialog um eine Datei neu anzulegen;

fn : Variable die den Dateipfad angibt und den gewählten Dateipfad zurückgibt

ltxt : Text der in die gewählte Datei gespeichert werden soll (optional)

Beispiel :
```
datei:s='c:\mirakon\neu.txt';
savefile(datei);
```

procedure CHOOSEKEY(wm,fid:n; var key:s; /o,sp,idx,flt,st:n;gf,of,b,ti:s;w,h:r;db:i)

Öffnet den Lader (Dialog zum Laden) damit der Anwender ein Objekt aus einer Datenbank auswählen kann. Im Unterschied zum LOADER wird nicht ein Objekt geladen, sondern nur sein Nummer gewählt.

Parameter O, SP, IDX, FLT, ST, GF, OF, TI, B, DB gleich wie bei LOADER.

wm : Datenbankmodul

fid : Wenn FID einen Namen enthält, wird das damit bezeichnete Dialogfeld neu gezeichnet.

key : Variable die den gewählten Objektnummer enthalten soll

w : Fenster-Breite in Texteinheiten (Vorbelegung=65)

h : Fenster-Höhe in Texteinheiten (Vorbelegung=22)

Beispiel :
```
anr:s;  choosekey(ART,,anr,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)

Funktioniert wie CHOOSEKEY, ergibt aber mehrere Objektnummern.
Der Anwender markiert die Objekte die in KEYS gespeichert werden;

wm : Datenbankmodul

keys : Liste der markierten Objekte-Nummer

Beispiel :
```
lnr:ls; choosekeyl(ART,lnr);
```

procedure ASKME (var m:e; tab,as:n; t1,q1:s);

Öffnet die Dialogmaske für die Master-Auswahl und Parameter-Eingabe.

m : Das ausgewählte Masterelement.

tab : Tabelle mit allen vorhandenen Masters.

as : Aktive Situation bei der Masterwahl.

t1 : Titel für die Dialogmaske.

q1 : Text vor den Mastermenu.

Beispiel :
```
askme(pr.pm,PR,PRE,'Technische Daten','Produkt-Master');
```

5.13.6 Programmierbare Dialogue

procedure OPENDIALOG (x,y,w,h:r; t:s; cb:i);

Öffnet ein Dialogfenster (ohne OK-Knopf).

x : X-Koordinate des linken Fensterrands in Texteinheiten

y : Y-Koordinate des oberen Fensterrands in Texteinheiten (immer ein negativen Wert !)

w : Fensterbreite in Texteinheiten

h : Fensterhöhe in Texteinheiten

t : Titel des Fensters

cb : Nummer der Hintergrundfarbe

Beispiel :
```
opendialog(10,-8,30,10,'Eingabe',3);
/...openstring, openquestion, usw.
rundialog(ok=closedialog); /Dialogfenster schliesst mi OK
```

procedure RUNDIALOG (/ok:n);

Startet den, mit OPENDIALOG, geöffneten Dialog.

ok : Prozedur die beim Drucken des OK-Knopfes aufgerufen wird

procedure CLOSEDIALOG;

Schliesst das aktuelle Dialogfenster;

procedure OPENSTRING (x,y:r; s:s; /color,font,size,bold,ital,adj:i);

Zeichnet eine Textzeile in dem, mit OPENDIALOG, geöffneten Dialogfenster.

x : X-Koordinate der Textzeile in mm
y : Y-Koordinate der Textzeile in mm
s : Textzeile
color : Farbnummer (Voreinstellung 0)
font : Fontnummer (Voreinstellung 2)
size : Schriftgrösse in Punkte (Voreinstellung 12)
bold : Schriftdicke: 0:normal 1:fett (Voreinstellung 1)
ital : Schriftneigung: 0:normal 1:kursiv (Voreinstellung 0)
adj : Ausrichtung: 1:links, 2:rechts, 3:mitte (Voreinstellung 1)

Beispiel : `openstring(2,-1,'Input',color=3,font=5,size=20);`

procedure OPENGRAFIC (var g:l; x,y:r; /id:n);

Zeichnet eine vorhanden Grafik in das geöffnete Dialogfenster.

g : Grafik
x : X-Koordinate der oberen linken Ecke in mm
y : Y-Koordinate der oberen linken Ecke in mm
id : Identifikator der Grafik in der Dialogmaske (optional);

procedure OPENDISPLAY (id:n; x,y,w,h:r; v:n; f:i);

Öffnet ein Displayfeld in das geöffnete Dialogfenster.

id : Feld-Identifikator
x : X-Koordinate des linken Feldrands in mm
y : Y-Koordinate des oberen Feldrands in mm
w : Feldbreite in Texteinheiten
h : Feldhöhe in Texteinheiten
v : Name der Variable die gezeigt werden soll
f : Zeigeformat: je nach Typ: Dezimalstellen oder Datumformat

Beispiel : `opendisplay(A1,15,-2,8,1,wert1,3);`

procedure OPENCHECKBOX (id:n; x,y:r; t:s; v:n; /p:n);

Öffnet ein Checkbox-Feld in dem, mit OPENDIALOG, geöffneten Dialogfenster.

id : Feld-Identifikator
x : X-Koordinate des linken Feldrands in mm
y : Y-Koordinate des oberen Feldrands in mm
t : dazugehöriger Textzeile; erscheint rechts vom Feld;
v : Name der Variable die gefragt werden soll
p : Prozedur die, beim Verlassen des Felds, aufgerufen wird

Beispiel : `opencheckbox(A1,15,-2,'Mit Rand',o1);`

procedure OPENQUESTION (id:n; x,y,w:r; v:n; f:i; /p:n);

Öffnet ein Fragefeld in dem, mit OPENDIALOG, geöffneten Dialogfenster.

id : Feld-Identifikator
x : X-Koordinate des linken Feldrands in mm
y : Y-Koordinate des oberen Feldrands in mm
w : Breite des Felds in Text-Einheiten
v : Name der Variable die gefragt werden soll
f : Zeigeformat: je nach Typ: Dezimalstellen oder Datumformat
p : Prozedur die, beim Verlassen des Felds, aufgerufen wird

Beispiel :
```
openquestion(A2,15,-2,8,wert1,3,p=controlwert1);
```

procedure OPENBUTTON (id:n; x,y,w,h:r; t:s; p:n);

Öffnet ein Knopf in dem, mit OPENDIALOG, geöffneten Dialogfenster.

id : Feld-Identifikator
x : X-Koordinate des linken Feldrands in mm
y : Y-Koordinate des oberen Feldrands in mm
w : Feldbreite in mm
h : Feldhöhe in mm
t : Text der in der Knopfmitte erscheinen soll
p : Prozedur die, beim Drucken des Knopfs, aufgerufen wird

Beispiel :
```
openbutton(B1,10,-6,20,2,'Berechnen',calcwert1);
```

procedure OPENSELECTOR (id:n; x,y,w:r; v:n; var opts:l; p:n);

Öffnet ein Selektor-Feld in dem, mit OPENDIALOG, geöffneten Dialogfenster.

id : Feld-Identifikator
x : X-Koordinate des linken Feldrands in mm
y : Y-Koordinate des oberen Feldrands in mm
w : Feldbreite in Texteinheiten
v : Name der Variable die gefragt werden soll
opts : Liste mit den Menuoptionen
p : Prozedur die, beim Verlassen des Felds, aufgerufen wird

Beispiel :
```
l1:ls=('Leicht','Normal','Schwer');
openselector(M1,10,-4,20,answer,l1);
```

procedure OPENMENU (id:n; x,y,w,h:r; v:n; var opts:l; p:n);

Öffnet ein Menufeld in dem, mit OPENDIALOG, geöffneten Dialogfenster.

id : Feld-Identifikator
x : X-Koordinate des linken Feldrands in mm
y : Y-Koordinate des oberen Feldrands in mm
w : Feldbreite in Texteinheiten
h : Feldhöhe in Texteinheiten
v : Name der Variable die gefragt werden soll
opts : Liste mit den Menuoptionen
p : Prozedur die, beim Verlassen des Felds, aufgerufen wird

Beispiel :
```
l1:ls=('Leicht','Normal','Schwer');
```

```
openmenu(M1,10,-4,40,12,answer,l1);
```

procedure OPENCOMBO (id:n; x,y,w,h:r; v:n; var opts:l; /p:n);

Öffnet ein Combobox-Feld in dem, mit OPENDIALOG, geöffneten Dialogfenster.

id : Feld-Identifikator
x : X-Koordinate des linken Feldrands in mm
y : Y-Koordinate des oberen Feldrands in mm
w : Feldbreite in Texteinheiten
h : Feldhöhe in Texteinheiten
v : Name der Variable die gefragt werden soll
opts : Liste der Menuoptionen
p : Prozedur die, beim Verlassen des Felds, aufgerufen wird

Beispiel :
```
l1:ls=('Leicht','Normal','Schwer');
opencombo(M1,10,-4,40,12,answer,l1,p=control01);
```

procedure OPENEDITOR (id:n; x,y,w,h:r; var txt:l; /l:i);

Öffnet ein Texteditor-Feld in das geöffnete Dialogfenster.

id : Feld-Identifikator
x : X-Koordinate des linken Feldrands in mm
y : Y-Koordinate des oberen Feldrands in mm
w : Feldbreite in Texteinheiten
h : Feldhöhe in Texteinheiten
txt : Text der editiert werden soll
l : Maximale länge einer Textzeile

Beispiel :
```
openeditor(E1,15,-2,8,1,wert1,l=40);
```

5.13.7 Selbstablaufende Demos

procedure INPUT (/);

Fügt Tastatureingaben oder Befehle in den Inputbuffer (für selbstablaufende Demos).

Befehle:

K + Number : Tastatureingabe: Zahl=Tastencode
B + Name : Maske wird aufgerufen
P + Number : setzt allgemeine Pause zwischen Tastendrücken; Zahl in 0.01-Sekunden
W + Number : fügt einmalige Pause ein; Zahl in 0.01-Sekunden
'text' : Texteingabe aus der Tastatur
M(x,y) : Mausbewegung nach X,Y in Pixel
L : Linke Maustaste wird gedruckt
R : Rechte Maustaste wird gedruckt

Beispiel :
```
input(K13,'ABC',W100,BM02);
```

NR	NAME	WERT
0	Back	8
1	Tab	9
2	Enter	13
3	Escape	27
4	Btab	315

5.14 Konfiguration

procedure GETFILEPARS (id:n; /var file:s);

Ergibt Informationen (Dateipfad) über konfigurierten Datenbanken.

id : Datenbank-ID sowie es in der Konfiguration angegeben ist

file : Dateipfad

Beispiel :
```
datei:s;
getfilepars(DAT,file=datei);
if exfile(datei)=0: inf(datei+' nicht vorhanden');
```

procedure SETFILEPARS (id:n; /file:s; accmode:i);

Ändert Dateipfade und/oder Zugriffsart der konfigurierten Datenbanken.
Dieser Befehl wird in der Konfiguration bei der Anwender-Login, at LOGIN, aufgerufen, um z.B. die Wissensbank zu wechseln.

id : Datenbank-ID sowie es in der Konfiguration angegeben ist

file : Neuen Dateipfad

accmode : Zugriffsmodus (access mode):
1 = direkt (via Client)
2 = via Server (nur für Client/Server-Version)

Beispiel :
```
/ in Anwender-CODE
at LOGIN:
localdat:s='C:\MIRAKON\MEINEDATEN.DAT';
setfilepars(DAT,file=localdat,accmode=1);
```

procedure GETDBASES (var dbs:l);

Ergibt in DBS (Liste von Elementen) die konfigurierten Datenbank-Parameter:
Datenbank-ID in: files.[i].id
Datenbank-Pfad in: files.[i].pars.fname
Zugriffsmodus in: files.[i].pars.accmode
Datenbanknummer in: files.[i].pars.dbnr

Beispiel :
```
dbs:l;
getdbases(dbs);
travel dbs:
  begin
  if exfile(p.fname)=0:
  inf('Datenbank'+pe^.id+' fehlt');
  end;
```

function LISTUSERS(/filt:s; ulevel,active:i):l

Ergibt die liste von Benutzer.

filt : Benutzerid filter

ulevel : Benutzer level
1:Anwender
2:Autor
3:Verwalter

active : Wenn 1 nur die aktiven Benutzer zurückbringt

Beispiel :
```
l1:l=listusers(ulevel=3); /ergibt die Benutzer das Verwalter sind
```

procedure GETUPERMS(uid:s; var uperms:l)

Ergibt in UPERMS die Liste der Erlaubnissen des Benutzers UID.

procedure GETUSERS (var users:l);

Liest die Konfigurationsdaten in eine Struktur-Variable.
Dieser Befehl kann nur von Administratoren ausgeführt werden.

users : Liste die die Konfigurationsdaten enthalten soll

Beispiel :
```
users:l;
getusers(users);
infv(users);
```

procedure SAVEUSERS (var users:l);

Speichert die Konfigurationsdaten aus einer Struktur-Variable.
Dieser Befehl kann nur von Administratoren ausgeführt werden.

users : Liste die die Konfigurationsdaten enthält

Beispiel : `saveusers(users);`

procedure SAVEUSER;

Speichert die aktive Benutzerliste.

procedure GETLFC (var lfcs:ls; /form:i)

Ergibt die Kalenderliste aus der Konfiguration.

lfcs : Variable, die die Kalenderliste empfängt

form : Ergebnisformat:
0 = Kalender ID (Vorbelegung)
1 = Kalender ID + '|' + Kalenderbeschreibung
2 = Kalenderbeschreibung

Beispiel : `lfc:ls; getlfc(lfc,form=2);`

procedure GETFC(id:s; var fc:l)

Ergibt in die liste FC der Kalender ID.

Beispiel : `fc:l; getfc('ALLG',fc);`

procedure GETSTARTMENU(var sm:l)

Ergibt in SM die Startsmenüliste aus der Konfiguration.

procedure SAVEUNITSTAB

Speichert die, in der Systemvariable UNITSTAB enthaltene, Einheiten Tabelle,
in der Konfiguration.

procedure GETFONTS(var fnts:l)

Ergibt die Liste der Schriftarten aus der Konfiguration.

5.15 Printer

procedure PRINT (mode:i; var doc:l);

Printet das Dokument DOC.

mode : 1:Text 2:Grafik

doc : Dokument

Beispiel :
```
print(2,gdoc);
```

procedure SETPRINTER (/lm,rm,tm,bm:r; sc,size,scale,copies,orient,psource:i; pname,doc:s);

Setzt Printerparameter.

lm : Linke Rand in mm

rm : Rechte Rand in mm

tm : Obere Rand in mm

bm : Untere Rand in mm

sc : Automatische Skalierung: 1:Ein 0:Aus

size : Papiergrösse: 1=A4; 2=A3;

scale : Skalierung in % (Vorbelegung->100)

copies : Anzahl Kopien

orient : Papierausrichtung: 1=Hoch(portrait); 2=Quer(landscape);

psource : Papierfach-Nummer:
1=Auto 2=Cassette 3=Envelope 4=Envmanual 5=Formsource
6=Largecapacity 7=LargeFmt 8=Lower 9=Manual 10=Middle
11=onlyone 12=tractor 13=smallFmt

pname : Printerbezeichnung, so wie es in Windows vorkommt

doc : Dokumentname

Beispiel :
```
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);

Ergibt Printerparameter.
Parameter gleich wie bei SETPRINTER.

Beispiel :
```
pn:s;
getprinter(pname=pn);
inf('Der aktiv drücker ist'+pn);
```

5.16 Schnittstellen

5.16.1 Bitmaps importieren

Im Grafikeditor können Bitmaps in Windows-BMP- und JPEG-Format importiert werden.

Vorgehen:

- Option Import/von Datei aus der Menuleiste auswählen.

- Datei mit BMP oder JPG-Endung auswählen.

- Viereckige Bildrahmen mit der Maus an der gewünschte Position bringen und links clicken.

Sollte das Bild zu gross für die Grafik sein, sollten Sie vorher die Grafik zurückzoomen.

5.16.2 Dateien lesen und schreiben

Variablen für das Lesen von externen Dateien

TFS	s	Dateiabschnitt in TRAVELF
TFP	a	Dateiposition in TRAVELF

procedure READTEXT (file:s; var text:ls);

Öffnet eine Textdatei und schreibt deren Inhalt in einer Liste.

file : Textdatei (Pfad + Name)

text : Variable die den Text enthalten soll

Beispiel : `daten:ls; readtext('IMPORT.TXT',daten);`

procedure WRITETEXT (file:s; var text:ls);

Schreibt ein Text in einer Textdatei.

file : Textdatei (Pfad + Dateiname)

text : Liste mit Textzeilen

Beispiel : `writetext('EXPORT.TXT',daten);`

procedure MAKEF (file:s; size:a);

Erstellt eine Datei und füllt sie mit Leerstellen.

file : Datei (Pfad + Name)

size : Anzahl Leerstellen

Beispiel : `makef('DATEN.DAT',512);`

procedure OPENF (var f:i; file:s);

Öffnet eine Datei (beliebigen Art) und schreibt die zugeteilten Filenummer (Handle) in der Variable F.

f : Variable die die Dateinummer empfängt

file : Datei (Pfad + Name)

Beispiel : `f1:i; openf(f1,'c:\temp\DATEN.DAT');`

procedure CLOSEF (f:i);

Schliesst die mit F angegebene Datei.

f : Dateinummer (gemäss OPENF-Befehl)

Beispiel : `closef(f1);`

procedure READFS (f:i; pos:a; n:i; var s:s; /c:i);

Liest ein Anzahl Bytes ab einer bestimmten Position aus einer Datei.

f : Dateinummer (gemäss OPENF-Befehl)

pos : Position, in der Datei, ab welcher gelesen wird;
Wichtig: Die erste Position ist 1, nicht 0.

n : Anzahl Bytes die gelesen werden

s : Variable die die gelesene Zeichen empfängt

c : Stringbereinigung:
0: Entfernt alle Leerzeichen von S;
1: Entfernt alle Leerzeichen die links stehen;
2: Entfernt alle Leerzeichen die rechts stehen;
3: Entfernt alle Leerzeichen die links und rechts stehen;

Beispiel : `name1:s; readfs(f1,1250,20,name1,c=3);`

procedure READFI (f:i; pos:a; n:i; var i:i);

Liest ein Anzahl Bytes ab einer bestimmten Position aus einer Datei
und interpretiert sie als Integer.

f : Dateinummer (gemäss OPENF-Befehl)

pos : Position, in der Datei, ab welcher gelesen wird;
Wichtig: Die erste Position ist 1, nicht 0.

n : Anzahl Bytes die gelesen werden

i : Variable die das gelesene Zahl empfängt

Beispiel : `v1:i; readfi(f1,1250,20,v1);`

procedure READFR (f:i; pos:a; n:i; var r:r);

Liest ein Anzahl Bytes ab einer bestimmten Position aus einer Datei
und interpretiert sie als reelen Zahl.

f : Dateinummer (gemäss OPENF-Befehl)

pos : Position, in der Datei, ab welcher gelesen wird;
Wichtig: Die erste Position ist 1, nicht 0.

n : Anzahl Bytes die gelesen werden

r : Variable die das gelesene Zahl empfängt

Beispiel : `v1:r; readfr(f1,1250,20,v1);`

procedure READFD (f:i; pos:a; n:i; var d:d);

Liest ein Anzahl Bytes ab einer bestimmten Position aus einer Datei
und interpretiert sie als Datum.

f : Dateinummer (gemäss OPENF-Befehl)

pos : Position, in der Datei, ab welcher gelesen wird;
Wichtig: Die erste Position ist 1, nicht 0.

n : Anzahl Bytes die gelesen werden

d : Variable die das gelesene Datum empfängt

Beispiel : `d1:d; readfd(f1,1250,20,d1);`

procedure WRITEFS (f:i; pos:a; n:i; s:s);

Schreibt einen Anzahl Bytes ab einer Position in eine Datei.

f : Dateinummer (gemäss OPENF-Befehl)

pos : Position, in der Datei, ab welcher gelesen wird;
Wichtig: Die erste Position ist 1, nicht 0.

n : Anzahl Bytes die geschrieben werden

s : Variable die die zu schreibenden Zeichen enthält

Beispiel :
```
writefs(f1,1250,3,'ABC');
```

procedure FINDFSEQ (f:i; headl,sepl,recl,keyp:i; key:s; var recp:a);

Sucht ein Datensatz in einer sequentiellen Datei.

f : Dateinummer (gemäss OPENF-Befehl)

headl : Datei-Kopflänge in bytes

sepl : Separatorlänge in Bytes

recl : Datensatzlänge in Bytes

keyp : Schlüsselposition

key : gesuchter Primärschlüssel

recp : Variable die die gefundene Position empfängt;
Wenn die Schlüssel nicht gefunden wird ist RECP=0;

Beispiel :
```
recpos:a; name1:s;  f:i;
openf(f,'TEST.DAT');
findfseq(f,128,2,730,1,'A007X',recpos);
readfs(f,recpos+12,20,name1);
```

5.16.3 ODBC-Schnittstelle

Mit folgenden Prozeduren können Daten aus ODBC-tauglichen Datenbanken (Access, Excel, Oracle, u.s.w) direkt, d.h. ohne Ausstieg aus dem Programm, gelesen bzw. geschrieben werden.
Bedingungen für das Funktionieren der Schnittstelle:

1. ODBC-Treiber-Manager (Microsoft-Programm) muss vorhanden sein.

2. ODBC-Treiber für den jeweiligen Datenbank muss vorhanden und installiert sein. Dieser wird meistens von Datenbankersteller geliefert und in Ihren Rechner mitinstalliert.

3. Die Datenbank worauf Sie zugreifen möchten muss als Datenquelle angemeldet sein. Dies können Sie mit dem ODBC-Admnistrator (Microsoft-Programm) erledigen.
Der von Ihnen vergebene Datenquellenname wird später verwendet.

1) ODBC-Datenquellen-Administrator von Microsoft starten:
(Start / Einstellungen / Systemsteuerung / Verwaltung / Datenquellen-ODBC)

2) In Register Treiber: vergewissern dass der nötigen Treiber vorhanden ist

3) In Register Benutzer-DSN: Datenquelle einfügen:
Einfügen, Datenquelle-Id eingeben (z.B: ADRESSEN), Kurzbeschreibung eingeben, Datei auswählen.

Anwendungsbeispiele:

Beispiel 1:
```
/ Abfragen welchen Treiber + Datenquellen
/ installiert sind

info:l;
openodbc;
infodbc(info);
editlst(info,'INFO');
closeodbc;
```

Beispiel 2:
```
/ Abfragen welche Tabellen in Datenquelle
/ D1 installiert sind

tabs:l;
openodbc;
connectodbc('DSN=D1');
tabsodbc(tabs);
editlst(tabs,'Tabellen');
disconnectodbc;
closeodbc;
```

Beispiel 3:
```
/ SQL-Abfrage ausführen:
/ Namen und Alter allen Mitarbeiter aus der
/ Tabelle PERSONAL aus der Datenquelle FIRMADATEN.

sqlabfrage:s='SELECT NAME, ALTER FROM PERSONEN';
result:l;
openodbc;
connectodbc('DSN=FIRMADATEN');
sqlodbc(sqlabfrage,result);
editlst(result,'SQL-Result');
disconnectodbc;
closeodbc;
```

Beispiel 4:
```
/ Werte erfahren ohne dafür eine SQL-Abfrage
/ schreiben zu müssen: Alter von Lisa Simpson.

alter:r;
openodbc;
connectodbc('DSN=FIRMADATEN');
getvalodbc(PERSONNEN,ALTER,NAME,'Lisa Simpson',alter);
infv(alter);
disconnectodbc;
closeodbc;
```

procedure OPENODBC;

Initialisiert das ODBC-Schnittstellensystem.

procedure CLOSEODBC;

Beendet das ODBC-Schnittstellensystem.

procedure INFODBC (var info:l);

Schreibt in der Liste INFO alle installierten ODBC-Treiber und Datenquellen.

procedure CONNECTODBC (code:s);

Richtet eine verbindung zu einer Datenquelle ein. In CODE muss die Datenquellenname angeführt mit 'DSN=' angegeben werden.
Optional können Anwender-ID (UID) und Passwort (PWD) angegeben werden.

Beispiel :
```
connectodbc('DSN=ADRESSEN')
connectodbc('DSN=QUELLE1;UID=PF;PWD=XY')
```

procedure DISCONNECTODBC;

Schliesst die aktive Verbindung zu einer Datenquelle.

procedure SQLODBC (code:s; var res:l);

Sendet eine SQL-Abfrage-string (CODE) und erhält das Resultat in RES.

Beispiel :
```
resultat:l;
sqlodbc('SELECT NAME, ALTER FROM PERSONEN',resultat);
```

procedure GETVALODBC (tabid,colid,keyid:n; key:s; varid:n);

Liest ein Wert aus einer Tabelle.

tabid : Tabelle-ID

colid : Spalte wo sich die gesuchte Information befindet

keyid : Spalte wo sich den Suchbegriff befindet

key : Wert des Suchbegriffs

varid : Variable die den gelesenen Wert enthalten soll

Beispiel :
```
getvalodbc(PERSONNEN,ALTER,NAME,'Lisa Simpson',alter);
infv(alter);
```

procedure TABSODBC (var tabs:l);

Schreibt in der Liste TABS alle Tabellen in der verbundene Datenquellen.

5.16.4 ADO-DB-Schnittstelle

Mit folgenden Prozeduren können Daten aus ADO-tauglichen Datenbanken (Access, Excel, Oracle, u.s.w) direkt, d.h. ohne Ausstieg aus dem Programm, gelesen bzw. geschrieben werden. Im Gegensatz zur ODBC-Schnittstelle, müssen keine Datenbanken in Betriebssystem angemeldet werden.

Anwendungsbeispiele:
/ schneller Zugriff auf Excel-Daten mit ADO-Technologie von Microsoft

```
res1:ls;
coms:l;
coms.1:s='SELECT * FROM [Preise$]';  / SQL-Befehl
connectadodb('d:\test\tab.xls');
runadodbsql(coms,res1,fmt=1);
disconnectadodb;
infv(res1);
```

procedure CONNECTADODB (dbpath:s; /passw,server,user:s; xlfirstl:i);

Richtet eine ADO-Verbindung zu einer Datenquelle ein.

dbpath : Datenbank-Datei
passw : Passwort (optional)
server : Server (optional)
user : User-Id (optional)
xlfirst : nur für Excel-Dateien: 1=erste Zeile wird als Datenzeile eingelesen (normalerweise erwartet ADO das die erste Zeile die Feldnamen enthält)

procedure DISCONNECTADODB;

Schliesst die aktive Verbindung zu einer Datenquelle.

procedure RUNADODBSQL (var sql:l; var res:l; /fmt:i);

Sendet eine SQL-Abfrage (mehrzeilig) und erhält das Resultat in RES.

sql : SQL-Abfrage
res : Ergebnis der SQL-Abfrage
fmt : Format: vorläufig immer = 1

procedure GETADODBTABLES (var tabs:l);

Schreibt in der Liste TABS alle Tabellen in der verbundene Datenquelle.

procedure READXLFILE (fname:s; var res:l; /line1:i);

Liest eine gesamten Excel-Datei in einer Liste mit ADO-Technologie.

fname : Excel-Datei
res : Exceldaten in Listenformat
line1 : wenn line1=1 wird die erste Excel-Zeile als Datenzeile gelesen (optional)

Beispiel :
```
xldaten:ls;
readxlfile('C:\TEMP\TAB.XLS,xldaten,line1=1);
```

5.16.5 Excel-Schnittstelle

Beispiel :
```
/ Anlegen eine Excel-Datei

openexcel('c:\temp\test.xls');
excelvisible(1);
setworksheet(wsnr=1,wsname='STUNDEN');
setworksheet(wsnr=2,wsname='PREISE');
setworksheet(wsnr=3,wsname='TOTALE');
werte:lr=(1,4,12,16);
loop i=1..4: cells(Ri,C1,Vwerte.[i]);
cells(R1,C1,I1,TH12);
s1:s='blabla';
cells(R2,C2,VS1);
cells(R3,C3,E'=sum(A1:A10)');
cells(R3,C3,E(=sum(A1:A10)));
cells(R3,C3,TC12,TF5,TW1);
cells(R1,C1,TF1,TS1,W15);
```

```
cells(R1,H25);
setworksheet(wsnr=2);
cells(R1,C1,AV1,AH1);
cells(R1,C1,F1);
cells(R2,C1,F11);
cells(S'C1:F1',M,BS1,BC1);
cells(S'A1',BS1,BS0);
cells(S'A2',BS2,BC2);
newworksheet(0,'NOVA');
closeexcel(1);
```

procedure OPENEXCEL (file:s; /wbtmp:s);

Öffnet bzw. erstellt einen Excel-Arbeitsmappe. Wenn die angegebene Datei nicht existiert, wird ein neues Arbeitsmappe erstellt.
CLOSEEXCEL muss unbedingt zuletzt aufgerufen werden.

file :	Dateipfad (inkl.Dateiname) des Dokuments
wbtmp :	Arbeitsmappe-Vorlage (workbook template)

procedure CLOSEEXCEL (save:i);

Schliesst einen geöffnete Excel-Arbeitsmappe.

save :	wenn -1 werden alle Änderungen nicht gespeichert; wenn 0 wird die Arbeitsmappe nicht automatisch gespeichert; wenn 1 wird die Arbeitsmappe gespeichert;

procedure EXCELVISIBLE (vis:i);

Zeigt bzw. versteckt die geöffnete Arbeitsmappe.

vis :	wenn 1 zeigt sie an; wenn 0 versteckt sie;

procedure CELLS (/);

Schreibt und formatiert Zellen in der aktuellen Tabelle.
Jedes Parameter besteht aus einem Befehl (1. und ev. 2.Zeichen) und einem Operand (restlichen Zeichen).

Befehle:

R + Zahl :	setzt aktuelle Zeile (row); wenn 0 wird die ganze aktuelle Spalte aktiviert
C + Zahl :	setzt aktuelle Spalte (column); wenn 0 wird die ganze aktuelle Zeile aktiviert;
S + String :	aktiviert eine Menge von Zellen; Z.B: S'C1:C20';
M :	Verbindet die aktivierten Zellen;
N1 :	Fügt neue Spalte ein und schiebt die aktivierte Spalte nach rechts.
N2 :	Fügt neue Zeile ein und schiebt die aktivierte Zeile nach unten.
GR :	Gruppiert Zeilen der aktiven Zellen, und erzeugt eine vertikale Hierarchiestufe;
GC :	Gruppiert Spalten der aktiven Zellen, und erzeugt eine horizontale Hierarchiestufe;

L + Name :	Liest der Inhalt der aktiven Zelle in die angegebene Variable
V + Name :	schreibt der Inhalt der angegebene Variable in der aktiven Zellen;
E + String :	schreibt eine Formel in der aktiven Zellen; Z.B: E'=sum(C1:C20)';
W + Zahl :	setzt Breite der Spalte(n) der aktive(n) Zellen;
WA :	setzt die optimale Spaltenbreite der aktive(n) Zellen;
H + Zahl :	setzt Höhe der Zeile(n) der aktive(n) Zellen;
AH + Zahl :	setzt horizontale Ausrichtung in der aktiven Zelle(n): 1=links 2=rechts 3=mittel 4=blocksatz
AV + Zahl :	setzt vertikale Ausrichtung in der aktiven Zelle(n): 1=oben 2=mittel 3=unten
IC + Zahl :	setzt Hintergrundfarbe der aktiven Zellen; (Mirakon-Farbennummer)
IR(r,g,b) :	setzt Hintergrundfarbe der aktiven Zellen in RGB-Format; Beispiel: IR(128,128,128) ergibt grau;
BI + Zahl :	setzt Rahmenindex der aktiven Zellen(block): 0=ganze Box 1=diagonal sinkend 2=diagonal steigend 3=untere Seite 4=linke Seite 5=rechte Seite 6=obere Seite 7=horizontal in der Mitte 8=vertikal in der Mitte Die Optionen 0,1,2,7 und 8 gelten für jede Zelle im Block. Die Optionen 3,4,5 und 6 gelten für den markierten Block.
BC + Zahl :	setzt Rahmenfarbe der aktiven Zellen; (Mirakon-Farbennummer)
BR(r,g,b) :	setzt Rahmenfarbe der aktiven Zellen in RGB-Format; Beispiel: BR(255,0,0) ergibt rot
BS + Zahl :	Zeichnet und setzt Rahmen-Linienart der aktiven Zellen: 0=ohne 1=voll 2=--- 3=... 4=-.-.- 5=-..-.. 6=-//- 9=doppelt;
TF + Zahl :	setzt Textfont der aktiven Zellen;
TH + Zahl :	setzt Textgrösse (in Punkte) der aktiven Zellen;
TW + Zahl :	setzt Textdicke: 0=normal 1=fett
TS + Zahl :	setzt Textstyl: 0=normal 1=italic
TC + Zahl :	setzt Textfarbe der aktiven Zellen;
TR(r,g,b) :	setzt Textfarbe der aktiven Zellen in RGB-Format; Beispiel: TR(0,0,64) ergibt dunkel blau;
TO + Zahl :	setzt Textorientierung in Grad: Z.B: TO-90;
TI + Zahl :	erhöht/verkleinert den Texteinzug der aktiven Zellen; Beispiel: cells(S'A1:A10',TI2,S'B1:B10',TI-1);
Fxyz :	setzt Format der aktive(n) Zellen: X: 1= Nummer; 2=Text Y: Tausenderseparator (optional): 0=ohne 1=mit; Z: Dezimalstellen (optional): 0 bis 9;
F + String :	setzt Format der aktive(n) Zellen gemäss Excelformatsprache; Beispiel: F'TT.MM.JJJJ';
X + String :	Führt eine Makro aus; z.B.: X'MAKRO1(par1,par2,...)';
K0 :	Schützt die aktiven Zellen vor der nachfolgenden Sicherung (siehe Befehl PROTECTWORKSHEET) Beispiel: cells(S'A1:G2',K0); /entsichert die Zellen A1:G2
DR :	aktive Zeilen löschen. Beispiel: cells(S'D5:D10',DR);
DC :	aktive Spalten löschen. Beispiel: cells(S'B1:C5',DC);
UC :	aktive Zellen kopieren. Beispiel: cells(S'A1:A10',UC); um eine Zeile zu kopieren, C0 setzen, z.B: cells(R5,C0,UC);

um eine Spalte zu kopieren, R0 setzen, z.B: cells(C5,R0,UC);
um eine Tabelle zu kopieren: Zellenblock kopieren, neue Tabelle einfügen und aktivieren, Zelle A1 aktivieren und mit UP einfügen.

UX : aktive Zellen auschneiden. Beispiel: cells(S'A1:A10',UX);

UP : kopierten bzw. ausgeschnittene Zellen auf aktive Zelle einfügen.
Beispiel: cells(S'D1',UP);

OA : Autofilter für die aktiven Zellen ein/ausschalten.
Beispiel: cells(S'A1:D1',OA);

OF + 0/1 : Fenster fixieren (OF1) oder unfixieren (OF0)
Beispiel: cells(S'C10',OF1);

OOR + 0/1 : Setzt Checkbox "Hauptzeilen unter Detaildaten" in Daten/Gliederung/Einstellungen. Beispiel: cells(OOR0);

OOX + 0/1 : Setzt Lesemodus:
- ooX0: L-Befehl liest Zellen-Werte
- ooX1: L-Befehl liest Zellen-Formel

procedure ACTIVATECELL (r,c:i);

selektiert eine Zelle in Excel (z.B.: damit folgender Makro funktioniert)

r : Zeilen-Nummer

c : Spalten-Nummer

Beispiel :
```
activatecell(5,4);
```

procedure SETWORKSHEET (/wsnr:i; wsname:s; var retnr:i; var retname:s; var vis:i);

Aktiviert eine Tabelle in der geöffneten Arbeitsmappe;
Wenn beide Parameter angegeben werden, wird die Tabelle umgenannt;

wsnr : Nummer (Position) der Tabelle

wsname : Name der Tabelle

retnr : Nummer (Position) der jetzt aktive Tabelle

retname : Name der jetzt aktive Tabelle

vis : VIS setzt Tabellenzustand: 1=sichtbar, 0=versteckt
wenn VIS=-1, wird nur Tabellenzustand in VIS zurückgegeben:

Beispiel :
```
setworksheet(wsnr=2);
setworksheet(wsname='STUNDEN');
setworksheet(wsnr=3, wsname='PREISE');

tvis:i=-1;
setworksheet(wsnr=2,vis=tvis); /Blattzustand wird abgefragt
if tvis=0:
  begin
  tvis:=1;
  setworksheet(wsnr=2,vis=TVis);
  end;
```

procedure NEWWORKSHEET (pos:i; wsname:s);

Fügt eine neue Tabelle in der geöffnete Arbeitsmappe ein;

pos : Position der neue Tabelle

wsname : Name der neue Tabelle

Beispiel :
```
newworksheet(4,'TOTALE');
```

procedure DELWORKSHEET (/wsnr:i; wsname:s;);

Löscht eine Tabelle in der geöffneten Arbeitsmappe;

wsnr : Nummer (Position) der Tabelle

wsname : Name der Tabelle

Beispiel :
```
delworksheet(wsnr=2);
delworksheet(wsname='STUNDEN');
```

procedure IMPXLIMAGE (file:s; /il,it,io:r);

Importiert und positioniert eine Grafikdatei in der geöffnete Arbeitsmappe;

file : Dateipfad (inkl.Dateiname) der Grafik

il : Abstand der Grafik zur linken Rand der aktiven Tabelle (in pixels)

it : Abstand der Grafik zur oberen Rand der aktiven Tabelle (in pixels);

io : Orientierung der Grafik in Grad (vorbelegt mit 0)

Beispiel :
```
impxlimage('\bilder\B01.BMP',il=50,it=20);
```

procedure XLSAVEAS (file:s);

Speichert die geöffnete Excel-Arbeitsmappe unter eine neue Datei.

file : Dateipfad (inkl.Dateiname) des Dokuments

procedure COPYWORKSHEET (pos:i) ;

Kopiert die aktive Tabelle in die Position POS.

pos : Zielposition (0 kopiert in die letzte Position)

Beispiel :
```
copyworksheet(0);
```

procedure MOVEWORKSHEET (pos:i) ;

Bewegt die aktive Tabelle in die Position POS.

pos : Zielposition (0 kopiert in die letzte Position)

Beispiel :
```
moveworksheet(4);
```

procedure PROTECTWORKSHEET (pwd:s);

Sichert die gesamte aktuelle Tabelle mit dem Passwort PWD.
Mit Prozedur CELLS, Befehl K0, können bestimmten Zellen aus der Sicherung ausgenommen werden. Dies muss aber vor Protectworksheet geschehen.

Beispiel :
```
protectworksheet('123456');
```

procedure READCLIPTAB (var tab:l);

Liest einen Tabellenauschnitt aus der Windows-Ablage in die aktuellen Tabelle.

tab : Tabelle

Beispiel :
```
tab:l;
opent(tab,'TAB','Tabelle','')
post(tab=tab);
readcliptab(tab);
edittab(tab,);
```

function function COUNTWS:i;

Ergibt die Anzahl Tabellen (Blätter) im geöffneten Excel-Dokument.
Die versteckten Tabellen werden auch mitgezählt.

5.16.6 Project-Schnittstelle

Beispiel :

```
/ Anlegen eine Project-Datei

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)

Öffnet bzw. erstellt einen MS. Project projekt. Wenn die angegebene Datei nicht existiert, wird ein neues Projekt erstellt.
CLOSEMSPROJ muss unbedingt zuletzt aufgerufen werden.

file : Dateipfad (inkl.Dateiname) des Projekt

procedure CLOSEMSPROJ(save:i)

Schliesst einen geöffnete MS. Project projekt.

save : wenn -1 werden alle Änderungen nicht gespeichert;
wenn 0 wird die Projekt nicht automatisch gespeichert;
wenn 1 wird die Projekt gespeichert;

procedure PROJVISIBLE(vis:i)

Zeigt bzw. versteckt der geöffnete Projekt.

vis : wenn 1 zeigt sie an; wenn 0 versteckt sie;

function EXPROJCAL(id:s)

Informiert ob ein Kalender in der aktiv Projekt vorhanden ist.
Ergenis = 1 wenn vorhanden bzw. 0 wenn nicht vorhanden.

id : Kalender name

procedure NEWPROJCAL(id:s; /basecal:s)

Fügt eine neu Kalender in der geöffnete Projekt ein;

id : Kalender Id

basecal :

Beispiel :
```
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)

Definiert die Kalendereigenschaften.

id : Kalender Id. (oder Resource Id. wenn RESC=1)

rset : Zurückstellen den Kalender.

wd : Woche Tag (1 = Montag bis 7 = Sonntag)

d : Tag

m : Monat

y : Jahr

iswork : 1: wenn Periode arbeit Periode ist
0: wenn Periode nicht arbeit Periode ist

shiftn : Produktionsverlagerung Nr. (1 bis 5)

resc : Wenn 1, wir arbeiten eine Resourcekalender und ID ist das Resource ID.

pstart : Periode Anfangsdatum

pend : Periode Ende Datum

ds : Produktionsverlagerung Anfang

de : Produktionsverlagerung Ende

Beispiel :
```
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)

Stellt die Parameter der aktiv Projekt ein

stg : Strategie:
1 = vorwärts
2 = rückwärts

prior : Priorität, 1 bis 10 (1 höchste Priorität, niedrigste Priorität 10)

defdu : Stellt die vorgelebt Dauermaßeinheit ein.
1=Minuten
2=Stunden
3=Tage
4=Wochen

defwu : Stellt die vorgelebt Arbeit Maßeinheit ein.
1=Minuten
2=Stunden
3=Tage
4=Wochen

hpd : Stellt die Stunden pro Tag für Aufgaben in einem Projekt ein

hpw : Stellt die Stunden pro Woche für Aufgaben in einem Projekt ein.

dpm : Stellt die Tagen pro Monat für Aufgaben in einem Projekt ein.

start : Stellt das Anfangsdatum für ein Projekt ein.
Die Einstellung des Anfangsdatums veranläßt auch das Projekt, von seinem Anfangsdatum festgelegt zu werden.

end : Stellt das Endedatum für ein Projekt ein.
Die Einstellung des Endedatums veranläßt auch das Projekt, von seinem Endedatum festgelegt zu werden.

Beispiel :
```
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)

Informiert ob das Resource ID im aktiv Projekt existiert.

id : Resource ID.

ruid : Ergibt den einzigartig nummer der Resource.

index : Ergibt die Resourceposition im Resourcen-Blatt.

procedure NEWPROJRES(id:s; /initials,basecal,email:s; type,pos:i; maxunits:r; var ruid:a)

Fügt ein neues Resource im aktiven Projekt ein.

id : Resource Id.

initials : Resource initialen

basecal : Resource-Kalender-Kennzeichnung.

email : Resource-Email

type : Resource art: (1=Arbeit (vorgelebt), 2=Material)

pos : Resourceposition im der Resourcen-Blatt.

maxunits : Maximale Resourcemaßeinheiten (in %)

ruid : Wenn angezeigt, ergibt der einzigartig nummer der Resource

Beispiel :
```
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)

Informiert ob die Aufgabe ID, existiert im aktiven Projekt.

id : Aufgabe ID.

tuid : Ergibt den einzigartig nummer der Aufgabe.

index : Ergibt die Aufgabeposition im Aufgaben liste.

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)

Fügt eine neue Aufgabe im aktiven Projekt ein.

id : Aufgabe Id.

dur : Aufgabe Dauer (z.B.: '10 Minuten)

pos : Aufgabe Position in der Aufgaben Liste

prior : Priorität, 1 bis 10 (1 höchste Priorität, niedrigste Priorität 10)

constrt : Begrenzung Art:
1=As spät als mögliches
2=As bald wie mögliches
3=Ende nicht früh als Begrenzung Datum
4=Ende nicht später als Begrenzung Datum
5=Ende des Begrenzung Datums auf
6=Anfang des Begrenzung Datums auf
7=Anfang nicht früh als Begrenzung Datum
8=Anfang nicht später als Begrenzung Datum

type : Aufgabe Art:
1 = Maßeinheiten
2 = Dauer
3 = Arbeit

percw : Stellt die Prozente komplett ein.

indent : Aufgabe Einrückung (X = unten X Niveaus, - X = herauf X Niveaus)

effdr : 1 wenn die Aufgabe Bemühung-gefahren wird

ignorerc : 1 wenn der Resourcekalender ignoriert wird, wenn man die Aufgabe festlegt

miles : 1 wenn die Aufgabe ein Meilenstein ist

est : 1 wenn die Dauer einer Aufgabe eine Schätzung ist

deadl : Stellt einen Stichtag für die Aufgabe ein

constrd : Stellt das Begrenzung Datum ein.

ds : Stellt das Anfangsdatum ein.

de : Stellt das Endedatum ein.

cal : Stellt den Kalender, zu verwendet zu werden wenn, die Aufgabe festlegend ist

tuid : Wenn eingeben, ergibt der Aufgabe einzigartig nummer

Beispiel :
```
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)

Weisen NUNITS eines Resource (RID oder RUID) einer Aufgabe zu (TID oder TUID).

nunits : Zahl der Resourcemaßeinheiten (in %)

tid : Aufgabe Id.

rid : Resource Id.

tuid : Einzigartig nummer der Aufgabe.

ruid : Einzigartig nummer der Resource.

Beispiel :
```
assignrestotask(90,tid='TASKG1',rid='DEFRES');
assignrestotask(100,tid='TASKG1',rid='DEFRES2');
```

procedure RELTASKS(type:i; lag:s; /tids,tidm:s; tuids,tuidm:a)

Verursacht eine Relation zwischen zwei Aufgaben.

type : Relation Art:
1 = Ende zu Anfang
2 = Anfang zu Angang
3 = Ende zu Ende
4 = Anfang zu Ende

lag : Relation Sträfling

tids : Sklavenaufgabe Id.

tidm : Vorlagenaufgabe Id.

tuids : Sklavenaufgabe einzigartig nummer

tuidm : Vorlagenaufgabe einzigartig nummer

Beispiel :
```
reltasks(1,'10 mins',tids='TASKG1',tidm='TASKG2');
```

5.16.7 DXF-Schnittstelle

Im Grafikeditor können Zeichnungen in DXF-Format importiert werden.

Einschränkungen:
- DXF-binär-Format werden nicht gelesen, nur die üblichen Textdateien;
- Firmenspezifischen Erweiterungen des DXF-Formats werden nicht unterstützt;

Vorgehen:

- Option Import/von Datei aus der Menuleiste auswählen.

- Datei mit DXF-Endung auswählen.

- Viereckige Bildrahmen mit der Maus an der gewünschte Position bringen und links clicken.

Sollte das Bild zu gross für die Grafik sein, sollten Sie vorher die Grafik zurückzoomen.

5.16.8 Serielle-Schnittstelle

procedure OPENSERIAL(var c:i;portnr:i;/rate:a;bit,stopbit,parity,flowc:i);

Öffnet die Serielle Schnittstelle (COM-Port);

c :	Handler für COM-Port; wird von Windows zurückgegeben;
portnr :	COM-Port-Nummer: 1 bis 3
rate :	Baudrate; Vorbelegung = 9600; Mögliche Werte: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000, 256000
bit :	Anzahl Bits; Vorbelegung = 8; Mögliche Werte: 5,6,7,8
stopbit :	Anzahl Stopbits: Vorbelegung = 0; Mögliche Werte: 0,1,2
parity :	Parität: 0=keine, 1=odd, 2=even; 3=mark; 4=space;
flowc :	Flowcontrol: 0=keine 1=software 2=hardware

procedure CLOSESERIAL (c:i);

Schliesst die Serielle Schnittstelle;

c :	Händler für COM-Port; erhalten von OPENSERIAL;

procedure WRITESERIALS (c:i; data:s; /eol:s);

Sendet ein String über die geöffnete Serielle Schnittstelle (COM-Port);

c :	Handler für COM-Port; wurde von OPENSERIAL vorher ermittelt;
data :	String die gesendet wird;
eol :	End Of Line String (optional); Vorbelegung = '';

procedure READSERIALS (c:i; var data:s; t:r; /eol:s);

Liest ein String über die geöffnete Serielle Schnittstelle (COM-Port)
bis ein EOL-Zeichen gelesen wurde oder die Timeout-Zeit T erreicht wird.

c :	Handler für COM-Port; wurde von OPENSERIAL vorher ermittelt;
data :	Stringvariable die gefüllt wird;
t :	Time Out in 0.01 Sekunden;
eol :	End Of Line String (optional); Vorbelegung = '';

procedure WRITESERIALL (c:i; var data:l; /eol,eof:s);

Sendet eine Liste (of strings) über die geöffnete Serielle Schnittstelle;

c :	Handler für COM-Port; wurde von OPENSERIAL vorher ermittelt;
data :	Liste die gesendet wird;
eol :	End Of Line String (optional); Vorbelegung = '';
eof :	End Of File String (optional); Vorbelegung = '';

procedure READSERIALL (c:i; var data:l; t:r; /eol,eot,sot:s)

Liest eine Liste (of strings) aus der geöffneten Seriellen Schnittstelle.

c :	Handler für COM-Port; wurde von OPENSERIAL vorher ermittelt;
data :	Variable die die Liste empfängt
t :	Timeout in 1/100 seconds
eol :	End of line
eot :	End of Text

sot : Start of Text

5.16.8.1 Implementationsbeispiele

Beispiel 1: /String via COM1 senden

```
h:i;
crlf:s=char(13)+char(10);
openserial(h,1,rate=2400);
writeserials(h,'1234567890ABCDEFG',eol=crlf);
closeserial(h);
```

Beispiel 2: /String aus COM1 lesen

```
h:i;
crlf:s=char(13)+char(10);
v:s;
openserial(h,1,rate=2400);
readserials(h,v,700,eol=crlf); /timeout=7sekunden
closeserial(h);
```

Beispiel 1: /Text via COM1 senden

```
/Annahme: text befindet sich in obj.prog
hser:i;  / windowshandler für serial port
openmsg('Daten werden','übertragen.....');
openserial(hser,1,rate=2400,bit=7,stopbit=2,parity=2);
if hser=0: inf('Schnittstelle zu !')&exit;
writeseriall(hser,obj.prog,eol=crlf);
closeserial(hser);
closemsg;
```

Beispiel 1: /Text via COM1 lesen

```
text:l;
hser:i;  / windowshandler für serial port
lf2:s=char(10)+char(13)+char(13);
openmsg('Daten werden','übertragen.....');
openserial(hser,1,rate=2400,bit=7,stopbit=2,parity=2,flowc=1);
if hser=0: inf('Schnittstelle zu !')&exit;
readseriall(hser,text,1500,eol=lf2,eot='%',sot='%');
closeserial(hser);
closemsg;
```

5.16.9 Dateien finden und durchsuchen

procedure GETFILES (dir:s; var lst:ls);

Ergibt die Liste der Dateien innerhalb einen Verzeichnis.

dir : Verzeichnispfad

lst : Liste der Dateinamen (strings)

Beispiel :

```
dateien:ls;
getfiles('C:\MIRAKON\',dateien);
```

procedure GETDIRST (dir:s; var st:l; /maxl,netw,head:i);

Ergibt die Unterstruktur eines Verzeichnisses als hierarchische Liste von Elementen.

Der Element-Typ (pe^.typ) informiert über die Art der Verzeichnisposition:
- 0=Laufwerk;
- 1=Datei;
- 2=Verzeichnis

dir : Verzeichnispfad

st : Verzeichnisstruktur (hierarchische Liste von Elementen).
Je nach Element-Typ (pe^.typ) befinden sich in ST:
- Laufwerke (Typ=0)
- Verzeichnisse (Typ=1)
- Dateien (Typ=2)
- Verzeichnisse in Netzwerk (Typ=10)

maxl : Maximale Anzahl Hierarchiestufen die untersucht werden soll.
Diese optionallen Parameter erlaubt unnötige zeitaufwendigen Untersuchungen zu vermeiden. Wenn nicht angegeben werden alle vorhandenen Unterverzeichnisse untersucht und aufgelistet.

netw : 0=Netzwerke werden nicht untersucht (Vorbelegung)
1=Netzwerke werden untersucht (kann sehr lange dauern)

head : 0 = ST enthählt nur die Unterstruktur von DIR
1 = ST enthählt die Startposition als Header-Element, und darunter, die Unterstruktur von DIR

Beispiel :
```
dir:l;
getdirst('C:\MIRAKON\',dir);
travel dir:
  begin
  if pe^.typ=2: inf('Verzeichnis='+ename);
  if pe^.typ=1: inf('Datei='+ename);
  end;
```

procedure CHOOSEPATH (dir:s; var path:s; /netw:i);

Untersucht die Unterstruktur eines Verzeichnisses und stellt sie als hierarchische Menu dem Anwender zur Verfügung. Er kann dann eine Position auswählen.

dir : Verzeichnispfad

path : Pfad der gewählten Position

netw : 0=Netzwerke werden nicht untersucht (Vorbelegung)
1=Netzwerke werden untersucht (kann sehr lange dauern)

Beispiel :
```
docfile:s;
choosepath('C:\DOCUMENTS\',docfile);
```

5.16.10 Wörter in Dateien suchen

procedure SEARCHINFILES (var res:ls; w,dir,flt:s; /ow,case,maxl,inf:i);

Sucht Dateien die ein Wort enthalten.
Für Office-Dateien (.DOC und .XLS) werden MS-Office-Funktionen verwendet.
Für PDF-Dateien werden Acrobat-Funktionen verwendet; die Dateien werden entpackt und danach durchsucht.
Ansonsten werden die Dateien, unabhängig deren Typ, binär durchsucht.
Als Ergebnis wird in RES die Liste der Dateien die das gesuchte Wort enthalten.

res : Liste der Dateinamen (strings) die das gesuchte Wort enthalten. Diese Liste wird von SEARCHINFILES nicht geleert.

w : Wort das gesucht wird

dir : Verzeichnis wo gesucht wird. Unterverzeichnisse werden auch, je nach MAXL durchsucht.

flt : Dateifilter in Microsoft-Syntax. Beispiel: '*.DOC'

ow : 1=nur ganze wörter (Vorbelegung=0)

case : 1=berücksichtigt Gross-/Kleinschreibung (Vorbelegung=0)

maxl : Maximaler Anzahl Verzeichnisstufen die durchsucht werden

inf : Information:
1=Dateien werden in Message-Fenster (unten rechts) angezeigt
2=Dateien werden in Report-Fenster angezeigt

Beispiel :
```
files:ls;
searchinfiles(files,'Mirakon','C:\TEMP\','*.TXT');
searchinfiles(files,'Mirakon','C:\TEMP\','*.DOC');
/ durchsucht alle TXT und DOC Dateien in c:\temp
/ die das Wort "Mirakon" enthalten, und speichert
/ die zutreffenden in die Liste FILES
```

procedure VIEWINFILES (var files:ls; word:s; /ow,case:i);

Zeigt die Dateien, bzw. die Mirakon-Objekte die ein Wort enthalten.
Input (Parameter FILES) ist das Ergebnis von SEARCHINFILES.
Für Office-Dateien (.DOC und .XLS) werden MS-Office-Programme verwendet.
Für PDF-Dateien wird den Acrobat-Reader verwendet.
Für TXT-Dateien wird Notepad verwendet.
Für Mirakon-Objekte wird ein allgemeiner Mirakon-Viewer verwendet dass die Objektstruktur zeigt.

files : Liste der Dateinamen (strings) die das gesuchte
Wort enthalten. Diese Liste wird von SEARCHINFILES erzeugt.

word : Wort das gesucht wird

ow : 1=nur ganze wörter (Vorbelegung=0)

case : 1=berücksichtigt Gross-/Kleinschreibung (Vorbelegung=0)

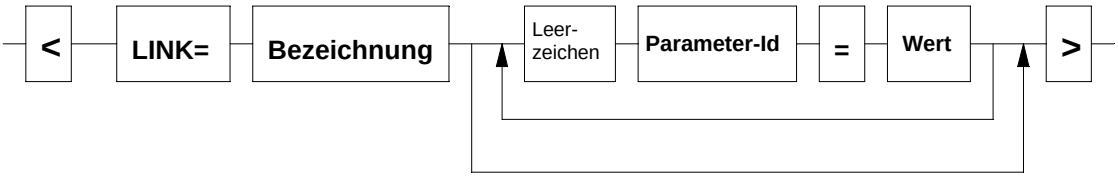
Beispiel :
```
files:ls;  wort:s='Vorrichtung';
searchinfiles(files,wort,'C:\TEMP\','*.*');
viewinfiles(files,wort);
```

5.16.11 Links

Ein Link ist eine Verbindung zwischen einen Satz, Bild oder Element und einen Mirakon-Objekt, externen Datei oder Website.

Ein Link wird als String definiert, mit einer HTML-ähnlichen Syntax, und kann in einem Text eingebettet, einem grafischen Element zugeordnet oder in einem Strukturelement gespeichert werden.

Syntax:



Link-Parameter:

- **NAME** Link-Bezeichnung sowie es dem Anwender erscheint
- **FILE** Pfad zur externen Mirakon-Datenbank falls diese nicht in der Konfiguration deklariert wurde.
- **OPEN** Pfad zur Viewer-Programm für externen Dateien, falls einen anderen, als das in der Windows-Registry angegebenen, erwünscht wird.
- **ACT** Link-Handler: Prozedur die aufgerufen wird wenn Link angeklickt wird (noch nicht implementiert)

```
Beispiel 1:  Hier ist die (link=KNO.A000\KA name=KA-Tabelle> als ...
Beispiel 2:  Klicken Sie (link=www.mirakon.com name=hier> um ...
Beispiel 3:  In diesem (link=g:\ber1.doc name=Bericht> steht ...
```

5.17 Kostenberechnung

Typen für die Kostenberechnung

COSTELEM Kostenelement. Enthält folgende Felder:
- VC:q; Berechneter Wert
- QR:q; Bezugsmenge (Anzahl Operationen in Prozess)
- CPU:q; Kosten pro Werteinheit (z.B: DM pro Std)
- REF:s; Referenzen = String mit Infos getrennt mit ";":
  ka;fepath;bkpath;oppath;rsnrs;wpnrs;lfnr;land;atts

Variablen für die Kostenberechnung

bkm	q	Baukomponentenmenge kumuliert über alle Hierarchiestufen Nur in Zusammenhang mit EDITBS, GENAP, ...
kla	l	Liste der Kostenelemente die zum selektierten Element im Kosteneditor gehören (nur gültig in der KA-Tabelle)
ks	l	Liste aller Kostenelemente während CALCCOST;
cpucolid	s	Aktive Kostensatzspalte in RS-Tabelle (für CALCCOST)

Situationen für die Kostenberechnung

Bei der Kostenberechnung und Strukturgeneratoren, Prozeduren CALCCOST, GENBS, GENAP, können folgende Situationen genutzt werden:

K0	Kostenberechnung Phase 1: Masteraktionen
PAA0	Kostenberechnung Phase 2: Arbeitsprozessanalyse
K1	Kostenberechnung Phase 3: Kostenelemente einbauen
PBA0	Arbeitsprozess generieren Phase 1: Baustrukturanalyse
PAI0	Arbeitsprozess generieren Phase 2: Prozess formatieren aus PR
PAI1	Arbeitsprozess generieren Phase 3: Baukomponente bauen Operationen im Arbeitsprozess ein
PAI2	Arbeitsprozess editieren
PFI0	Funktionsstruktur generieren (ausgehend von Produktklasse)
PFI1	Funktionsstruktur editieren
PFA0	Baustruktur generieren Phase 1: Funktionsstrukturanalyse
PBI0	Baustruktur generieren Phase 2: Baustruktur formatieren aus PR
PBI1	Baustruktur generieren Phase 3: Funktionen bauen Baukomponente in der Baustruktur ein
PBI2	Baustruktur editieren
PBA1 bis PBAn	Baustruktur komplettieren Phase 1, Lauf 1 bis n
PBC1 bis PBCn	Baustruktur komplettieren Phase 2, Lauf 1 bis n

procedure INSR (rs:n; /nr:s; need:q; o:n);

Ordnet einen Ressourcemaster der aktiven Operation zu.

rs :	Nummer der Ressourcemaster aus der RS-Tabelle;
nr :	Ressourcen Nummern die für die aktive Operation in Frage kommen. Mit NR kann einen Arbeitsplatz (vor der Terminierung) zugewiesen werden.
need :	Arbeitsbedarf (für Planung)
o :	Optionen: Zeichen mit folgender Bedeutung: E : Ressourcenstruktur wird geleert vor der neuen Zuweisung;

Beispiel : `insr(D01,o=E);`

procedure INSRS (rss:ln; /nr:s; need:q);

Ordnet eine Ressource der aktiven Operation zu. RSS enthält alle Ressourcen Nummern die für die aktive Operation in Frage kommen. Das Programm überprüft dann die Einsatzbedingung jeder Ressource in der B-Spalte der RS-Tabellen. Die erste somit gefundene Ressource wird zugeordnet und die Suche abgebrochen.

rss :	Liste der möglichen Ressourcen-Masters
nr :	Mit NR kann einen Arbeitsplatz (vor der Terminierung) zugewiesen werden.
need :	Arbeitsbedarf (für Planung)

Beispiel :
```
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);

Fügt ein Kostenelement in die Kostenstruktur des Produktes ein.

ka :	Kostenart-Nummer gemäss KA-Tabelle
val :	Kostenwert in der Einheit vorgesehen in KA-Tabelle
t :	Beschreibungstext
rs :	Ressource-Nr aus der RS-Tabelle
rsi :	Ressource-Position in D.RS-Liste
u :	Kosteneinheit falls nicht gleich KA-Tabelle
supl :	Lieferanten-Nr. (supplier)
country :	Land-Nr.

Beispiel :
```
insce(TR,50);
insce(TE,p.te,t='Drehzeit');
insce(MK,200,t='Guss',supl='G05',land='CH');
```

procedure CALCCOST (var obj,opl:l; /o,cpu:n; d,n:i; var ks:l);

Berechnet und generiert die Kostenstruktur eines Objekts aus einer Operationenliste. Die Kostenstruktur wird in obj.KS gespeichert.

obj :	Kalkulationsobjekt (Produkt, Auftrag oder Arbeitsplan)
opl :	Operationenliste
o :	Optionen: H: HKB - Kompatibel;
cpu :	Prozedur die Kostensatz berechnet
d :	Displaymode: 1:no display
n :	Anzahl Durchläufe (at K1,K2,...) bei der Kostenberechnung; Vorbelegung=1;
ks :	Kostenstruktur (optional) falls nicht obj.ks;

Beispiel :
```
calccost(pr,pr.ap);
calccost(pr,pr.ap,d=1,n=2);
```

Die Voraussetzungen für CALCCOST sind:

- Ressourcenliste jeder Operation in operation.DATA.RS.
- Kostenarten in KA-Tabelle in Basismodul. Diese Tabelle muss die Kostenarteneinheit in die U- oder UN-Spalte enthalten.
- Ressourcenmasters in RS-Tabellen.
- Arbeitseinheit jeder Ressource in Spalte WU der RS-Tabellen. Wenn diese Spalte nicht existiert wird die Einheit Stunde (202) gewählt.
- Kosten pro Arbeitseinheit jeder Ressource in Spalte CPU oder KS der RS-Tabellen. Wenn die Kosteneinheit nicht angegeben wird, wird die Einheit sFr (301) gewählt (!).

Die Kostenberechnung besteht aus der Ausführung von Aktionen aus der Wissensbank mit dem Zweck, alle notwendigen Kostenelemente in die Kostenstruktur einzubauen.

Dies geschieht in 3 Phasen:

1.Phase / Situation=K0 / Objektmaster-Aktionen:

Die Aktionen der zugeordneten Produktmaster werden ausgeführt.

Meistens werden zu diesem Zeitpunkt globale Variablen gesetzt:

Beispiel: Aktion von Produktmaster "Teil":

at K0: A000\menge:=p.lg; "Menge global setzen"

2.Phase / Situation=PAA0 / Arbeitsprozessanalyse:

Der Arbeitsprozess wird von oben bis unten untersucht. Für jede Operation werden die Aktionen des entsprechenden Masters ausgeführt. Zweck dieser Aktionen ist das Erkennen von Produktmerkmalen, die für dessen Herstellung wichtig sind. Dieses Erkennen besteht in der Zuweisung von Werten an Variablen der Wissensbank, die als Indikatoren in die Produktstruktur gespeichert werden.

3.Phase / Situation=K1 / Kostenelemente einbauen:

Der Arbeitsprozess wird nochmals von oben bis unten untersucht. Für jede Operation werden die Aktionen des entsprechenden Masters ausgeführt. Zweck dieser Aktionen ist das Einfügen und Auswerten von Kostenelementen in die Kostenstruktur des Produktes. Dies geschieht mit dem INSCE-Befehl.

Beispiel: at K1: insce(TH,120);

Folgende Tabellen sind für die Kostenberechnung notwendig und sollten in den Basismodul angelegt sein:

- KA-Tabelle: enthält die Definitionen der Kostenarten.

- RS-Tabelle: für Ressourcen-Masters.

procedure EDITCOST (fmt:n; var obj:l; /o:n; n:i);

Zeigt die Kostenstruktur eines Objektes.

fmt : Ausgabenformat: FMT = Zeilennummer der KOF-Tabelle

obj : Kalkulationsobjekt (Produkt, Auftrag oder Arbeitsplan)

o : Optionen:
C = mit Berechnungsknopf;
H = HKB - Kompatibel;

n : Anzahl Durchläufe (at K1,K2,...) bei der Kostenberechnung;
Vorbelegung=1;

Beispiel : `editcost(AP,pr,o=C,n=2);`

NR	NAME	ST	FW	KAS	CODE
BS	Nach Baugruppen	2	90	('MK/5/0','TG/5/0','HK/8/2')	
AP	Nach Operationen	3	90	('MK/5/0','TG/5/0','HK/8/2')	
FS	Nach Funktionen	1	90	('MK/5/0','TG/5/0','HK/8/2')	
LF	Nach Lieferanten	5	90	('MK/5/0','TG/5/0','HK/8/2')	

ST = Struktur: 1=Funktionsstruktur 2=Baustruktur 3=Arbeitsprozess
4=Ressourcenliste 5=Lieferantenliste 6=Landliste
FW = Formularbreite in Zeichen
KAS = Kostenartenspalten (list of strings).
Ein String -> EINE Kostenartspalte.
Stringaufbau: Kostenart/Spaltenbreite/Kommastellen
CODE = Befehle für die Mitgestaltung des Formulars. Bei der Situation
LST.4 kann der Applikationsautor Kopfzeilen hinzufügen. Dafür
wird die Systemvariable TXI:ls benutzt.

Beispiel: at LST.4:
txi.1:=pr.nr+' '+pr.name+' '+pr.kunde;
txi.2:='Losgrösse='+sr(pr.lg,0);

function SUMKE (calc:i; var kel:l; kas:s; /u:n):q;

Ergibt die Summe (quantity) aller Kostenelemente aus der Liste KEL die dem Kostenartenmuster KAS angehören.

calc : Kalkulationsmethode:
1 = addiert alle Kostenelementwerte
2 = addiert alle Kostenelementwerte multipliziert mit der Anzahl Operationen
3 = addiert alle Kostenelementwerte multipliziert mit dem entsprechenden Kostensatz
4 = addiert alle Kostenelementwerte multipliziert mit dem entsprechenden Kostensatz und Anzahl Operationen

kel : Liste mit Kostenelementen

kas : Kostenartenmuster

u : Gewünschte Einheit

Beispiel :
```
tgsum:=sumke(1,pr.ks,'TH,TN');
matsum:=sumke(2,pr.ks,'M_',u=DM);
```

procedure FILTERKE (crit:i; filt:s; var l1,l2:l; /o:n; nh:i);

Schreibt alle Kostenelemente die ein(en) bestimmtes(n) Kriterium/Filter erfüllen von der Liste l1 in die Liste l2.

crit : Filter-Kriterium:
1: Kostenelemente die einer Funktionseinheit angehören.
FILT = Pfad dieser Funktionseinheit.
2: Kostenelemente die einer Baukomponente angehören.
FILT = Pfad dieser Baukomponente.
3: Kostenelemente die einer Operation angehören.
FILT = Pfad dieser Operation.
4: Kostenelemente die einer Kostenart angehören.
FILT = Muster von Kostenarten.
5: Kostenelemente die einem Funktionseinheit Master angehören.
FILT = Muster von Funktionseinheiten-Masternummern.
6: Kostenelemente die einem Baukomponent Master angehören.
FILT = Muster von Baukomponenten-Masternummern.
7: Kostenelemente die eine Operations Master angehören.
FILT = Muster von Operationen-Masternummern.
8: Kostenelemente die eine Kostenelement Master angehören.
FILT = Muster von Kostenelementen-Masternummern.
9: Kostenelemente die eine Ressource Master angehören.
FILT = Muster von Ressourcen-Masternummern.

10:Kostenlemente die in einem Lieferant erzeugt worden sind
FILT = Lieferantennummer.
11:Kostenlemente die in einem Land erzeugt worden sind
FILT = Landnummer.

filt : Filterwert (siehe oben) abhängig von Filterkriterium

l1 : Ausgangsliste mit Kostenelementen

l2 : Zielliste die filtrierte Kostenelementen empfängt

o : Optionen:
- E : leert L2 vor dem Filterprozess
- R : (reset) initialisiert Kostenberechnungsvariablen; nötig wenn z.B. das Objekt seit den letzten FILTERKE ersetzt wurde.
- N : Filter mit negativen Vorzeichen: alle Kostenlementen die die Kriterien erfüllen werden weggefiltert.

nh : gibt die Anzahl Hierarchiestufen die durchsucht werden ausgehend von angegebenen Pfad.

Beispiel :
```
l1,l2:l;
filterke(3,'2.7',pr.ks,l1);
filterke(7,'D_',l1,l2,o=E);
filterke(10,'L_',l1,l2);  /gefiltert nach Lieferanten L...
filterke(11,'CH',l1,l2);  /gefiltert nach Land CH
```

procedure GENAP (/o:n; d:i);

Generiert ein Arbeitsprozess ausgehend aus einer Baustruktur.

o : Reihe von Buchstaben mit folgender Bedeutung:
L = Die vorhandene Struktur wird ohne Rückfrage gelöscht (wichtig für 'batch-artige' Prozesse).

d : Displaymode: 1:no display

Wie funktioniert GENAP:
Das Generieren des Arbeitsprozesses besteht aus der Ausführung von Aktionen aus der Wissensbank mit dem Zweck, alle notwendigen Operationen in der Produktstruktur einzubauen.

Dies geschieht in drei Phasen:

1.Phase / Situation=PBA0 / Baustrukturanalyse:

Die Baustruktur wird von oben bis unten untersucht.
Für jede Baukomponente werden die Aktionen des entsprechenden Masters ausgeführt. Zweck dieser Aktionen ist das Erkennen von Produktmerkmalen, die für dessen Herstellung wichtig sind. Dieses Erkennen besteht in der Zuweisung von Werten an Wissensbankvariablen, die als Indikatoren in die Produktstruktur gespeichert werden.

Beispiel: Aktion von Planfläche:

at PBA0: bknd:=bknd+1; /Anzahl gedrehte Elemente erhöhen

2.Phase / Situation=PAI0 / Arbeitsprozess formatieren:

Die Aktionen der zugeordneten Produktmaster (-klasse) werden

ausgeführt. Meistens werden zu diesem Zeitpunkt Gruppen von
Operationen, als Skelett des Arbeitprozesses, eingebaut:

Beispiel: Aktion von Produktmaster "Lokomotive":

```
at PAI0: K01->ap.0;  /Operation Lauftest einfügen
```

3.Phase / Situation=PAI1 / Operationen einbauen:

Die Baustruktur wird nochmals von oben bis unten untersucht.
Für jede Baukomponente werden die Aktionen des entsprechenden
Masters ausgeführt. Zweck dieser Aktionen ist das Einfügen von
Arbeits- und Teilvorgängen und die Zuordnung von Ressourcen.
Wenn eine Operation eingefügt wird, prüft Mirakon ob die eingebaute
Operation ihrerseits auch Folgeaktionen unter PAI1 hat,
und führt diese aus (Kettenreaktion).

Beispiele:

- Aktion von Baukomponentmaster Planfläche:
```
at PAI_: D7(p.l,p.d)->popd^.ss.0; /Längsschruppen unter Drehen einfügen
```

- Aktion von Operation Bohren:
```
at PAI_: AR01->ss.0;      /Teilvorgang Rüsten einfügen
at PAI_: insrs((B_,F_));  /Ressource B_ oder F_ zuordnen
```

procedure GENBS (/o:n; nfa:i; d:i);

Generiert eine Baustruktur ausgehend aus einer Funktionsstruktur.

o : Optionen: Reihe von Buchstaben mit folgender Bedeutung:
L = Die vorhandene Struktur wird ohne Rückfrage gelöscht
(wichtig für 'batch-artige' Prozesse).

nfa : Anzahl Funtionstruktur-Analyse-Durchläufe.
Wenn nicht angegeben, gibt es nur 1 Durchlauf mit Zustand PFA0.
Wenn nfa->2, gibts es 2 Durchläufe mit Zuständen PFA0 und PFA1.
Bei nfa->3, gibts es 3 Durchläufe mit PFA0, PFA1 und PFA2. U.s.w.

d : Displaymode: 1:no display

Beispiel : `genbs(o=L);`

procedure COMPLBS (n:i; /o:n);

Komplettiert (ergänzt) die Baustruktur über N Durchläufe.
Ein Durchlauf enthält 2 Phasen:
- Baustrukturanalyse: situation PBA1 bis PBAn
- Baustrukturaufbau: situation PBC1 bis PBCn

n : Anzahl Durchläufe

o : Optionen: Reihe von Buchstaben mit folgender Bedeutung:
L = Die vorhandene Struktur wird ohne Rückfrage gelöscht
(wichtig für 'batch-artige' Prozesse).

Beispiel : `complbs(2);`

procedure GENFS (/o:n; d:i);

Generiert eine Funktionsstruktur ausgehend vom Produktmaster.

o : Optionen: Reihe von Buchstaben mit folgender Bedeutung:
L = Die vorhandene Struktur wird ohne Rückfrage gelöscht
(wichtig für 'batch-artige' Prozesse).

d : Displaymode: 1:no display

5.18 Termin- und Kapazitätsplanung

Typen für die Produktionsplanung

Folgende Typen sind für die termin und Kapazitätsplanung notwendig:

opterm Terminierungsbedingungen für eine Operation, enthält:
- STG:b; Strategie: 1=vorwärts 2=rückwärts 3=frühstens rückwärts;
- STAT:b; Status: bit1=fixiert bit2=validiert
- CAP:b; Belastungsrechnung: 1=mit begrenzte Kapazität 2=mit unbegr.Kap. 3=mit Engpassanalyse 4=gemäss Planeinstellung
- NINTR:i; Maximale Anzahl Unterbrechungen
- CMAX:r; Maximale Kapazitätnutzung in % von Gesamtkapazität
- WMIN:r; Minimale Belastungsblock in Stunden pro Tag
- TMS1:d; Erwünschter Starttermin
- TMS2:d; Frühster Starttermin (Grenze für Terminierung)
- TME1:d; Erwünschter Endtermin
- TME2:d; Spätester Endtermin (Grenze für Terminierung)

opstatus Termin-Ergebnisse für eine Operation/Auftrag, enthält:
- OK:b; 1=ok 0=unmöglich zu terminieren
- WC:q; Geplante Arbeit: Stunden, Stück, Gewicht, usw.
- WR:q; Realisierte Arbeit: Stunden, Stück, Gewicht, usw.
- CC:q; Geplante Kosten
- CR:q; Realisierte Kosten
- TPS:d; Geplante Starttermin
- TPE:d; Geplante Endtermin
- TRS:d; Realisierte Starttermin
- TRE:d; Realisierte Endtermin

loaditem Belastungseinheit an einen Arbeitsplatz, bestehend aus:
- TS:d; /Starttermin
- TE:d; /Endtermin
- W:q; /Last (Arbeit)

Variablen für die Produktionsplanung

wp1	s	Nummer des primären Arbeitsplatzobjekts

5.18.1 Terminierung

procedure TREL (typ,cat:i; dt:q; /p1,p2:^e; ncy,shift:r);

Baut eine Anordnungsbeziehung (Zeitrelation) zwischen 2 Operationen.
Die bezogene Operation ist die aktive Operation oder, falls P2 angegeben ist, die die mit P2 gezeigt wird.
Die Bezugsoperation ist die grössere Schwester der bezogene Operation (d.h. die auf gleiche Stufe vorhergehende) oder, falls P1 angegeben ist, die die mit P1 gezeigt wird.

typ : 1:Ende-Start-Beziehung (Normalfolge)
2:Start-Start-Beziehung (Anfangsfolge)
3:Ende-Ende-Beziehung (Endfolge)
4:Start-Ende-Beziehung
5:Start-Start + Ende-Ende (Flussbeziehung)

cat : 1:minimale Zeitinterval: Frühestens-Beziehung

dt : Zeitinterval zwischen beide Operationen

p1 : Zeiger auf Bezugsoperation

p2 : Zeiger auf bezogene Operation

ncy : Anzahl Zyklen (Vorgabe->1)

shift : Phasenverschiebung

Beispiel 1:
```
trel(1,1,20 min);
```
Beispiel 2:
```
pref:^e=^pr.ap.1;
pslave:^e=^pr.ap.2;
trel(1,1,2 h,p1=pref,p2=pslave);
```

procedure OT (stg,cap:i; tms1,tme1,tme2,tms2:d; /ni:i);

Fügt Terminierungsbedingungen T:opterm in DATA des aktiven elements;

stg : Strategie:
0 = gemäss Planeistellung
1 = vorwärts
2 = rückwärts
3 = frühstens rückwärts

cap : Belastungsrechnung:
1 = mit begrenzte Kapazität
2 = mit unbegrenzte Kapazität
3 = mit Engpassanalyse
4 = gemäss Planeinstellung

tms1 : Starttermin

tme1 : Soll-Endtermin

tme2 : End-Termingrenze für Planung

tms2 : Start-Termingrenze für Planung

ni : Anzahl erlaubte Unterbrechungen

Beispiel :
```
OP.A1(tr,te)->ss.0&ot(1,3,ts1,te1,temax);
```

procedure TERMOF (var of:l; /o:n; d,stg,cap,qual:i; tms1,tme1:d);

Berechnet die Termine alle Operationen in dem Auftrag OF.
Die berechneten Termine werden in Operation.DATA.S:opstatus gespeichert.
Die berechnete Auftragstermine werden in OF.S:opstatus gespeichert.

of : Auftragsobjekt

o : Optionen mit folgenden Bedeutung:
I : aufgerufen während laufenden produktionsplanung;
N : Ressourcen werden nicht geladen und Auftrag nicht gespeichert;

d : Displaymodus: 0:ohne reporting

stg : Planungsstrategie: 1:vorwärts 2:rückwärts 3:frühstens rückwärts

cap : Kapazitätsmodell: 1:begrenzt 2:unbegrenzt 3:Engpassanalyse
4:gemäss Plan

qual : Planungsqualität: 1:fein 2:grob

tms1 : erwünschter Starttermin

tme1 : erwünschter Endtermin

Beispiel :
```
termof(auftrag,stg=1,cap=4,qual=2,d=0,o=I);
```

procedure IMPOSETERMS (pop:^e; ts,te:d);

Erzwingt Start- und Endtermine einer Operation, ohne dass diese
Belastungen verursachen muss.

pop : Zeiger auf die Operation

ts : Starttermin

te : Endtermin

Beispiel :
```
imposeterms(pdrehen,d1,d2);
```

5.18.2 Ressourcen, Kapazitäten und Belastungen

procedure INSWP (irs:i; nr:s; /im:i);

Weist ein Arbeitsplatz einer Ressource zu.

irs : Reihenfolge der Ressource in der aktiven Operation.

nr : Arbeitsplatz-Nr.
Wenn NR leer ist wird eine vorhandene Zuweisung anuliert.

im : 1 = Arbeitsplatz wurde vom Anwender gewählt
2 = Arbeitsplatz wurde vom System gewählt;

Beispiel :
```
inswp(2,'1298',im=1);
inswp(1,'');
```

procedure SETRSQ (q:q; /rs:n; rsi:i)

Setzt den Bedarf einer Ressource der aktiven Operation.

q : Bedarf

rs : Ressourcenmaster-ID aus der RS-Tabelle.

rsi : Reihenfolge der Ressource in der aktiven Operation.

Beispiel :
```
setrsq(2 std);
setrsq(1,rs=X);
setrsq(2.5 min,rsi=2);
```

procedure EDITCAP (var cap:l; / ucap:i; mtab,o:n);

Öffnet den Kapazitätseditor für die Kapazitätsliste CAP.

cap : Kapazitätsstruktur (Liste von Regeln)

ucap :

mtab : Kapazitäten-Masters-Tabelle mit Kapazitätsprofile die mit
INSCAP-Befehle in der A-Spalte definiert werden;

o :

Beispiel :
```
in der MAIN-Tabelle:   editcap(rs.cap,mtab=A000\MCAP);
in der MCAP-Tabelle:   loop i=1..7:empty(rs.cap.[i].ss);
inscap(rs.cap,1,4,5 std,5 std,7,0,12,00);
inscap(rs.cap,5,5,4 std,4 std,7,0,11,0);
```

procedure INSCAP (var cap:l; day1,day2:i; ctot,cop:q; h1,m1,h2,m2:i);

Fügt mehrere gleiche Kapazitätselemente in die Kapazitätsliste CAP in den
Tagen DAY1 bis DAY2 ein.

cap : Kapazitätsstruktur

day1, day2 : 1=Montag 2=Dienstag 3=Mittwoch ... 7=Sonntag;

ctot : Gesamtkapazität in den Kapazitätselement;

cop : Kapazität pro Operation in den Kapazitätselement;

h1 : Startzeitpunkt: Stunde

m1 : Startzeitpunkt: Minute

h2 : Endzeitpunkt: Stunde

m2 : Endzeitpunkt: Minute

Beispiel :
```
inscap(rs.cap,1,4,5 std,5 std,7,30,12,30);
/ Fügt ein 5-stundige Kapazitätselement zwischen 7 Uhr 30 und
/ 12 Uhr 30 am Montag, Dienstag, Mittwoch und Donnerstag ein
```

procedure INSCAPEX (var c:l; w:s; dct,dco:q; h1,m1,h2,m2,cy:i; d1,d2:d);

Fügt eine Kapazitätsausnahme in die Kapazität einer Ressource.
Parameter H1,M1,H2,M2 wie INSCAP.

c : Kapazitätsstruktur

w : Grund für die Ausnahme;

dct : Abweichung der Gesamtkapazität;

dco : Abweichung der Kapazität pro Operation;

cy : Wiederholzyklus in Tage

d1 : Startdatum der Abweichung

d2 : Enddatum der Abweichung

Beispiel :
```
inscapex(rs.cap,'Überzeit',8 std,4 std,19,30,23,30,1,8.4.02,12.4.02);
/ Fügt ein 4-stundige Kapazitätserhöhung zwischen 19 Uhr 30 und
/ 23 Uhr 30 jedentag zwischen 8. und 12.4.02
```

procedure CALCRSCAP (var rs:l; ts,te:d; var cto,cop:q);

Berechnet die Kapazität einer Ressource zwischen zwei Datums.

rs : Ressource-Objekt

ts : Startdatum

te : Enddatum

cto : Gesamtkapazität (Ergebnis)

cop : Kapazität pro Operation(Ergebnis)

procedure UNLOADORD (var ord:l);

Entlastet alle Ressourcen vom Auftrag ORD.

procedure RSVRS (rs:s; ts,te:d);

Reserviert die Ressource-Nr RS von TS bis TE füe den aktiven Auftrag.

procedure BUILDLOADS (var ll,wp:l; d1,d2:d; ordpat,oppat:s; /var ovl:l);

Baut die Liste LOADS mit den Belastungen von Resource RS zwischen D1
und D2 geordnet nach Datum.

ll :

wp :

d1 :

d2 :

ordpat : Auftragsfilter = Muster von Auftragsnummern

oppat : Operationsfilter = Operationspfad

ovl : Liste mit Überlastungen

procedure CALCRSLOAD (var rs:l; ts,te:d; var load:r; /ordpat:s);

Berechnet Belastung von Ressource RS zwischen TS und TE.
Das Ergebnis wird in LOAD in Stunden angegeben.

Wenn ORDPAT angegeben wird, werden nur die Belastungen die zur Auftragsfilter ORDPAT passen.

procedure CALCRSOVLOAD (var rs:l; ts,te:d; var ovl,ovlmax:r);

Berechnet die Überlastungssumme (in OVL) und die maximale Überlastung (in OVLMAX) von Ressource RS zwischen TS und TE.

procedure DISTRRSW (var rs,distr:l; d1,d2:d);

Baut die Liste DISTR mit den Belastungen von Resource RS geordnet nach Datum und gegliedert nach Operationen.
Es werden nur die Operationen herangezogen die Belastungen zwischen D1 und D2 enthalten.

Format von DISTR:

```
|
+- #1:list = Daten für die 1.Operation
|    |
|    +- OPP:s  = Operationspfad, z.B: '2.4.3'
|    +- ONR:s  = Auftrags-Nr, z.B: 'A0127'
|    +- WOP:q  = Gesamt Arbeitsbedarf für diese Operation, z.B: 12 Std
|    +- W:q    = Geleistete Arbeit zwishen D1 und D2, z.B: 7 Std
|    +- VCUM:r = Kumulierte Arbeit zwische Operationsstart und D1
|    +- IRS:i  = Ressource-Reihenfolge in Operation, z.B: 1
|    +- L:list = Liste der Belastungen zwischen D1 und D2
|         |
|         +- #1:loaditem = 1.Belastung
|        ...
|
+- #2:list = Daten für die 2.Operation
|    |
...  ...
```

procedure UNLOADRS (var rs:l);

Löscht alle Belastungen in Ressource RS;

5.18.3 Produktionsplanung

procedure EDITPLAN (var plan:l; /o,h:n; im,id,nc:i; b:s);

Öffnet integrierte Editor + Planer eines Produktionsplans.

plan : Produktionsplan

o : Optionen: Reihe von Buchstaben mit folgenden Bedeutung:
T = Systemdatum und erfasste Ist-Termine werden berücksichtigt.
Dies hat zur Folge:
1) Beendete Operationen werden nicht mehr terminiert.
2) Gestarteten aber nicht beendeten Operationen werden ab Systemdatum weiter geplant.
3) Nicht gestarteten Operationen können erst ab Systemdatum

geplant werden.
M = Ignoriert aktuellen Lagerbestand;
S = Plansimulation -> Ressourcen werden nur in Plan belastet;
L = Planeditor mit Relationen-Knopf;
N = Planeditor ohne Planen-Knopf;
Q = Operationszeiten werden für jede Ressource, während der Planung, neu evaluiert

h : Handler-Prozedur um Editor zu steuern.

im :

id : Identifikationsnummer des Editorfensters. Mit STAGEID kann in einer Prozedur abgefragt werden welche Fenster aktiv ist.

b : Optionale Knöpfe:
Syntax:'Knopfname1:Prozedur1,Knopfname2:Prozedur2,...'

nc : Anzahl Planungsrunden (calculations); Vorbelegung=1;

Datenstruktur von PLAN:

- ST:l = Planstruktur bzw. Auftragsliste (notwendig);

- CAP:i = Belastungsrechnung (opt.):
1 = planen mit begrenzte Kapazität;
2 = planen mit unbegrenzte Kapazität;
3 = planen mit Engpassanalyse;

- QUAL:i = Planungsqualität (opt.):
1 = Feinplanung (Vorbelegung): Minuten-Auflösung
2 = Grobplanung: Nur Tage werden berechnet;

- PR:l = Liste der Lagerbestände aller vorkommenden Produkte;

- PROOPT:s = Protokolloptionen:
A = ein Protokoll pro Operation wird in operation.data.pro:list geschrieben;
Darin beschreibt Mirakon alle ausgeführte Planungsschritte.

- PROFIL:s = Auftragsfilter für Protokoll

- ORDFIL:s = Auftragsfilter für Planung

- NOFMAX:a = Maximaler Anzahl zu planende Aufträge

Voraussetzungen für EDITPLAN sind:

1. Definitionen:
   - DW==(Ressourcen-Modul)
   - DO==(Aufträge-Modul)
   - MPP==(Produktionsplan-Modul)
   - MPR==(Produkte-Modul)
   - APID==(Arbeitsprozess-Id in Auftrag)
   - EXID==(Lagerbestände-Id in Produkt; Vorbelegung=EX)

2. Tabelle PP mit Planposition-Masters (mit CH-Spalte):

   - Gruppen: Beispiel: Bestellungen (kein Eintrag in CH-Spalte)

   - Lieferungen: Beispiel Bestellpositionen, Fabrikationsprogramm;

Eine Lieferungsposition muss die Charakteristik L aufweisen.
=> Eintrag L in die CH-Spalte;

- Aufträge: Fertigungsauftrag, Einkaufsliste;
Eine Auftragsposition muss die Charakteristik O aufweisen und den Auftragsnummer in den String-Parameter OFNR enthalten.
=> Eintrag O in die CH-Spalte;
Nur diese Positionen werden terminiert.
Sollte ein Auftrag vor dem Planen gelöscht werden, muss die Charakteristik ein D aufweisen.

3. Coaching-Prozeduren:

- PLANST: Verarbeitet Planstruktur bevor einzelne Positionen behandelt werden;

- PLANLV: Initialisiert Lieferungsposition vor Terminierung;
Folgende Systemvariablen werden zur Verfügung gestellt:
  - PLV:^e:zeiger zur Planungsposition;
  - PR:l:produkt der geplant werden soll (falls vorhanden);
  - STO:l:Liste der Lagerdestände von PR in dem Plan;

- PLANOF: Initialisert Auftrag vor Terminierung;
Bedingung: Parameter OFNR:s mit Auftragsnummer in Planposition;
Folgende Systemvariablen werden zur Verfügung gestellt:
  - OF:l:Auftrag der geplant werden soll (falls vorhanden);

- procedure FINOF; Finalisiert Auftrag nach Terminierung;

- procedure FINLV; Finalisiert Lieferungsposition nach Terminierung;

- procedure FINST; Finalisiert Plan nach Terminierung aller Aufträgen;

Funktionsweise / Planungsablauf:

Wenn der "Planen"-Knopf gedruckt wird, geschieht folgendes:

1. Prozedur PLANST wird ausgeführt (falls vorhanden);

2. Planstruktur wird, von oben bis unten, bearbeitet:

Wenn eine Lieferung angetroffen wird:
- die Prozedur PLANLV wird ausgeführt (falls vorhanden);
- die untere Positionen werden bearbeitet;
- die Prozedur FINLV wird ausgeführt (falls vorhanden);

Wenn ein Auftrag angetroffen wird:
- Auftragsobjekt wird in Systemvariable OF geladen;
- Systemvariable PLANOK wird = 1 gesetzt;
- die Prozedur PLANOF wird ausgeführt (falls vorhanden);
  wurde PLANOK = 0 gesetzt, wird der Auftrag nicht geplant;
- der Auftrag wird terminiert (siehe unten);
- die Prozedur FINOF wird ausgeführt (falls vorhanden);
- Auftrag wird gespeichert;

3. Prozedur FINST wird ausgeführt (falls vorhanden);

4. Produktionsplan und Ressourcenbelastungen werden gespeichert;

Ablauf der Terminierung eines Auftrags:

1. Status von Auftrag und alle darin enthaltene Operationen werden initialisiert.

2. Jede Operation wird geplant:

2.1 Für jede Operation werden die Aktionen für den Zustand CTP0 ausgeführt. Diese Aktionen sind in die Tabelle OP geschrieben.
Beispiel: automatisches einfügen von Relationen mit TREL für aufeinanderfolgenden Operationen.

2.2 Aufbau der Liste alle möglichen Ressourcenkombinationen.

2.3 Auswertung der vorhandene Relationen mit anderen Operationen.

2.4 Überlagerung der in 2.3 berechneten Terminbedingungen mit der operationsspezifischen absoluten Terminbedingungen.

2.5 Für jede Operation werden die Aktionen für den Zustand CTP1 ausgeführt (auch in Tabelle OP).
Beispiel: EVALRST um einen Ressourcekombination auf allen folgenden Suboperationen zu erzwingen und somit die Kontinuität einer Ressource zu gewährleisten (Fa.Hader).

2.6 Überlagerung der in 2.3 und 2.4 berechneten Terminbedingungen mit der Auftragsterminbedingungen und mit ev.vorhandene Terminbedingungen aus Eltern-Operationen (siehe 2.5).

2.7 Falls Option T (Realplanung) wird das Systemdatum als Terminbedingung berücksichtigt.

2.8 Wenn alle Terminbedingungen möglich sind, werden die benötigten Aufwände geplant, d.h. freie Kapazitäten werden gesucht und mit Belastungen belegt.
Wenn mehreren Ressourcenkombinationen existieren, wiederholt sich diesen Schritt für jede Kombination, und diejenige die die besten Terminresultate liefert, wird ausgewählt.

2.9 Die somit berechneten Start- und Endtermine werden auf alle Eltern-Operationen-Stati übertragen.

2.10 Für jede Operation werden die Aktionen für den Zustand CTP2 ausgeführt.
Beispiel: erkennen und behalten der gewählten Ressourcen

3. Für den ganzen Auftragsprozess werden die Aktionen für den Zustand CTPP ausgeführt. Diese Aktionen sind in die Tabelle OP geschrieben.
Beispiel: nachträgliche Reservierung eine Ressource mit RESERVRS für einer Operationsgruppe damit Zwischenkapazitäten nicht von anderen Aufträge genutzt werden.

4. Auftragsstatus setzen und Auftrag speichern.

Zustand von Auftragsprozess:

- Jedem Arbeitsvorgang (nicht aber Teilvorgänge) muss eine Ressource-

master aus der Wissensbank zugewiesen sein.

- Jedem Ressourcemaster sollte eine Ressourcenummer aus der Arbeitsbank zugewiesen sein. Wenn dies nicht zutrifft, sucht IOS eine passende Ressource (Arbeitsplatz) anhand der F-Spalte (Auswahlfilter) in der RS-Tabelle in der Wissensbank.

- Auftragskosten müssen vorher mit CALCCOST berechnet worden sein.

- Auftragsobjekt enthält ein Feld T vom Typ opterm an der 1.Hierarchiestufe. Hier müssen die Randbedingungen für die Terminierung gespeichert sein.

- Terminierungsbedingungen für einzelne Operationen, falls vorhanden (selten), müssen in operation.DATA.T:opterm gespeichert sein.

- Die erfasste Ist-Termine müssen unter operation.S:opstatus gespeichert sein.

Struktur eines Ressourcenobjekts (rs):

- Kapazitätslisten alle vorkommenden Ressourcen müssen mit EDITCAP unter rs.CAP erstellt worden sein, oder durch Angabe einen Musterressource in rs.CAP0:s.

- Belastungsliste muss als rs.LOADS vorhanden sein.

- Wenn Feiertage-Kalender berücksichtigt werden soll, muss den Feiertag-Kalender-Nr in rs.FC:s angegeben sein.

Notwendige Tabellen in der Wissensbank:

- Operationenmasters befinden sich in OP-Tabellen

- Ressourcenmasters befinden sich in RS-Tabellen

- Feiertage-Kalender (wenn gewünscht) befindet sich in der Konfiguration

5.19 Konstruktionssystem

Typen für die Konstruktion

P3D 3D-Punkt; Enthält folgende Felder:
- X:r; X-Koordinate
- Y:r; Y-Koordinate
- Z:r; Z-Koordinate

VOPTS Ansichtsoptionen (viewoptions) mit folgenden Felder:
- status:i; / bit1=visible , bit2=in background, bit3=with grips
- symb:b; / Symbol-Nr wenn>0
- c:i; / Farben-Nr wenn>0
- lw:i; / Liniendicke wenn >0
- ls:i; / Linienstyl wenn >0
- le:i; / Linienende wenn >0
- sym:b; / 0=view all, 1=shows upper half 2=shows lower half
- dim:b; / 0=without dimensioning 1=with dimensioning
- lab:b; / 0=without labels 1=with labels
- r3d:i; / 1=wire frame 2=hidden line 3=shaded
- mat:i; / material 1=stil, 2=whod, 3=rubber
- prec:i; / precision -> for mesh, etc (later use)
- uo:i; / user options
- patt:i; / pattern id

Variablen für die Konstruktion

VO Viewoptions des aktiven Elements;

GRIPNR Nummer des selektierten Grifpunktes

Situationen für die Konstruktion

GINS Nach dem Einfügen eines Konstruktionelements in der Baustrutur.

GMOVE Wenn der Anwender ein Element bewegen möchte.

procedure OPENCS (var cs:l; nr,name:s; /o:n);

Eröffnet (formatiert) die Konstruktion CS.

cs : Konstruktions-Objekt

nr : Ident-Nr der Konstruktion

name : Bezeichnung der Konstruktion

o : Optionen:
- I = Fügt ein leeres Dokument mit einem 3D-Frame

Beispiel :
```
k1:l;
opencs(k1,'A1','Haus',o=I);
```

procedure EMPTYCS (var cs:l);

Leert die Konstruktion CS.

procedure EDITCS (var cs:I; /h,matb,dtab,build:n);

Öffnet den Konstruktionseditor für die Konstruktion CS.

cs :	Konstruktions-Objekt
h :	Optionale Steuerungsprozedur (handler): Damit können Strukturänderungen (Einfügen, Löschen, usw.) genauer gesteuert werden.
mtab :	Masterstabelle für Strukturelementen
dtab :	Tabelle für Dokumenten-Masters
build :	Name der Prozedur die die Struktur generiert; Wenn dieses Parameter angegeben wird, erscheint ein BILDEN-Knopf über den Struktfenster;

Beispiel : `editcs(pr.bs,h=HBS,mtab=WI\KON,dtab=WI\DOCS,build=GENBS);`

procedure FORM (/);

Definiert eine 3D-Geometrie gemäss den eingegebenen Parameter.
Jedes Parameter besteht aus einem Befehl (1.Zeichen und ev. auch 2.Zeichen) und einem Operand (restlichen Zeichen).

Befehle:

PA(n,x,y,z) :	definiert Punkt Nummer N mit absoluten Raumkoordinaten X,Y,Z
PR(n,r,dx,dy,dz)	definiert Punkt N relativ zur Punkt R (Referenzpunkt) mit Abstände DX,DY,DZ
PL(n0,lp) :	definiert Punkte ab Nummer N0 (Startnummer) aus einer Liste von 3D-Punkte
LW(w) :	setzt Liniedicke = W
LS(nr) :	setzt Linienstyl = NR 1 = vollgezogen 2 = gestrichelt 3 = punktiert 4 = strichpunktiert
LC(nr) :	setzt Linienfarbe = NR
LP(n,p1,p2,...) :	definiert eine Polygonlinie N die über die Punkten P1, P2,... verläuft
LA(n,p0,p1,pp,w)	definiert einen Kreisbogen N, mit Zentrum P0, der in P1 startet, ein Winkel W (in Grad) beschreibt, und in der Ebene P0,P1,PP liegt (PP ist ein beliebiger Punkt in der gewünschte Ebene).
DG(p1,p2,...) :	definiert die Punkte P1, P2, ... als Greifpunkte (grips)
SQ(n,p1,p2,p3,p4	definiert ein Quader (Nummer=N) P1 = Punkt vorne, unten, links P2 = Punkt vorne, oben, links P3 = Punkt vorne, oben, rechts P4 = Punkt hinten, oben, rechts

procedure NEWGC3D (pce:^e);

Erstellt die grafische Darstellung eines Konstruktionselements
in den aktiven Dokument.
Danach bleiben Konstruktionselement und die entsprechende Grafik
intern gebunden.

pce : Zeiger auf den Konstruktionselement

procedure BUILDGC (var gcg:l; id:s; /x,y:r; var pgc:^e; pos,mode:i; var dest:l);

Baut eine grafische Komponente in die activen Grafik ein.

gcg : Grafik der grafische Komponente

id : Identifikator der grafischen Komponente

x : X-Position der grafischen Komponente in Zielgrafik

y : Y-Position der grafischen Komponente in Zielgrafik

dest : Zielgrafik wo grafische Komponente eingebaut wird,
nur für den Fall es handelt sich nicht um die aktiven Grafik

Beispiel :
```
pkontur:^e;  graf:l;
getpointtogc('KONTUR',pkontur,g=graf);
```

procedure GETPOINTTOGC (id:s; var pgc:^e; /var g:l);

Ergibt einen Zeiger auf einen grafischen Komponenten die mit ID identifiziert ist.
Optionel wird die Grafik selbst als Liste in G kopiert;
Wichtig: Grafik die die gesuchte Komponente enthält, muss aktiv sein;

id : Identifikator des grafischen Komponents

pgc : Zeiger auf gefundenen grafischen Komponent

g : Grafik der grafischen Komponente

Beispiel :
```
pkontur:^e;  graf:l;
getpointtogc('KONTUR',pkontur,g=graf);
```

procedure GETGCPOS (var x,y:r; /id:s; pgc:^e; g:l);

Ergibt die Position einer grafischen Komponente in X und Y.
Die Bestimmung der Komponente kann über ihre ID oder über den Zeiger PGC.
Optional kann die zu durchsuchende Grafik angegeben werden.

x : Ergebnis: X-Koordinate der grafische Komponente

y : Ergebnis: Y-Koordinate der grafische Komponente

id : Identifikator des grafischen Komponents

pgc : Zeiger auf gemeinten grafischen Komponent

g : Grafik wo die grafische Komponente zu suchen ist

Beispiel :
```
x,y:r;
getgcpos(x,y,id='NP',g=p.pos.graf);
```

procedure SETGCPOS (x,y:r; /id:s; pgc:^e; g:l);

Positioniert einer grafischen Komponente in Position X,Y.
Die Bestimmung der Komponente kann über ihre ID oder über den Zeiger PGC.
Optional kann die zu durchsuchende Grafik angegeben werden.

x : X-Koordinate der neuen Posiotion

y : Y-Koordinate der neuen Posiotion

id : Identifikator des grafischen Komponents

pgc : Zeiger auf gemeinten grafischen Komponent

g : Grafik wo die grafische Komponente zu suchen ist

Beispiel :
```
getgcpos(0,-20,id='ARC',g=p.pos.graf);
```

procedure DELGC (id:s; /g:l);

Löscht eine grafischen Komponente die mit ID identifiziert ist.
Optional kann die zu durchsuchende Grafik angegeben werden.

id : Identifikator der grafischen Komponente

g : Grafik wo die grafische Komponente zu suchen ist

Beispiel :
```
x,y:r;
getgcpos(x,y,id='NP',g=p.pos.graf);
```

5.19.1 Dokumente

procedure INS3DFRAME (id,name:s; xlp,ytp,xrp,ybp:r; /sc1,sc2:i; xvn,yvn,zvn,dx,dy:r);

Fügt ein 3D-Rahmen in den aktiven Dokument.

ID : Identifikator des Rahmens

NAME : bezeichnung des Rahmens

XLP,YTP,XRP,YBP : Position des Rahmens auf dem Papier in mm:
XLP = X-left
YTP = Y-top
XRP = X-right
YBP = Y-bottom

DX,DY : Verschiebung des Raum-Nullpunkt von Rahmenzentrum

SC1, SC2 : Skalierung sc1:sc2
Vorbelegung ist SC1=1, SC2=1;

XVN,YVN,ZVN : Sichtrichtungsvektor (Normalvektor zur Projektionsebene).
Vorbelegung ist: XVN=0, YVN=0, ZVN=-1;

procedure POSDOCE (pdoc:^e; / fr:s);

Aktiviert ein Konstruktionsdokument und, falls FR angegeben wird, ein 3D-Rahmen.
Notwendig bevor Konstruktionselemente gezeichnet werden, etwa mit NEWGC3D.

PDOC : Zeiger auf Dokument (Element in Dokumentenstruktur)

FR : Rahmen-Identifikator

5.19.2 IGES-Schnittstelle

procedure EXPIGES (file:s; var g:l);

Exportiert den Grafikobjekt G in einen IGES-Datei (mit IGS-Endung).

Beispiel : `expiges('C:\TEST\MYGRAFIC.IGS',pr.g);`

5.20 Mathematik

5.20.1 Regressionsrechnung

procedure CALCREGRESSION (tab:n; var lexp,lcoef,ldev,lsqr:l; /var nrbest:i);

Berechnet 6 mögliche Regressionsformeln für eine Versuchsreihe mit maximal 5 unabhängigen Variablen.

tab :	Versuchstabelle
lexp :	Liste mit allen Formeln
lcoef :	Liste mit 5 Listen mit Koeffizienten für die 6 Formeln
ldev :	Liste der mittlere Abweichungen für die 6 Formeln
lsqr :	Liste der Summen der quadratischen Abweichungen für die 6 Formeln
nrbest :	Nummer der beste Formel

Aufbau der verschiedenen Formeln:

1.Linear	Y = c0 + c1*x1 + c2*x2 + ...
2.Exponentiell	Y = c0 * c1^x1 * c2^x2 + ...
3.Potenz	Y = c0 * x1^c1 * x2^c2 + ...
4.Bruch	Y = c0 + c1/x1 + c2/x2 + ...
5.Quadratisch	Y = c0 + c1/x1 + c2/x1^2 + c3*x2 + c4*x2^2 ...
6.Kubisch	Y = c0 + c1/x1 + c2/x1^2 + c3*x1^3 + c4*x2 ...

Beispiel :
```
vtab:l; /versuchstabelle
lf:l;   /Formelliste
lc,ld,lsq:l;
b:i;
calcregression(vtab,lf,lc,ld,lsq,nrbest=b);
inf('Die beste Formel ist '+lf.[b]);
```
Beispiel einer Versuchstabelle:

NR	X1	X2	X3	X4	Y
V01	1.6	300	20	2	45
V02	1.8	320	20	2	53
V03	1.9	350	40	2	60
V04	2.3	380	40	2	67
V05	2.7	400	20	5	79
V06	4.2	480	20	5	120
V07	4.3	500	40	5	125
V08	5.8	510	40	5	150

5.21 Internet-Dienste

5.21.1 HTML-Schnittstelle

Vorgehen um Mirakon-Tabellen in HTML-Format zu exportieren:

- In Wissensbankeditor: Tabelle selektieren aber nicht öffnen;

- Option Export/in Datei aus der Menuleiste auswählen.

- Datei-Typ (unten) HTML auswählen.

- Dateiname und Zielverzeichnis wählen und Knopf Speichern drucken.

Vorgehen um Mirakon-Tabellen-Ausschnitte in HTML-Format zu exportieren:

- In Tabelleneditor: Zellen-Block mit der Maus markieren (mit gedruckten linke Taste);

- Option Export/in Datei aus der Menuleiste auswählen.

- Datei-Typ (unten) HTML auswählen.

- Dateiname und Zielverzeichnis wählen und Knopf Speichern drucken.

Vorgehen um Mirakon-Prozeduren und -Funktionen in HTML-Format zu exportieren:

- In Wissensbankeditor: Prozedur selektieren aber nicht öffnen;

- Option Export/in Datei aus der Menuleiste auswählen.

- Datei-Typ (unten) HTML auswählen.

- Dateiname und Zielverzeichnis wählen und Knopf Speichern drucken.

Vorgehen um Mirakon-Grafiken und -Dokumente in HTML-Format zu exportieren:

- In Grafik-Editor: Option Export/in Datei aus der Menuleiste auswählen.

- Datei-Typ (unten) HTML auswählen.

- Dateiname und Zielverzeichnis wählen und Knopf Speichern drucken.

Es wird ein SVG-Datei in einen Unterverzeichnis angelegt der vom HTM-Datei aufgerufen wird.

5.21.2 E-Mail

procedure SENDMAIL (to,subj:s; var cc,msg,att:l; /server,from,fromid:s);

Sendet ein Email-Nachricht mit oder ohne Anhang.

server :	SMTP-Server (Ausgangsserver)
from :	Email-Adresse des Absenders
fromid :	Name des Absenders
to :	Email-Adresse des Empfängers

subj : Betreff

cc : Liste weiterer Empfänger (Email-Adressen)

msg : Nachricht (List of strings)

att : Liste der Anhang-Dateien

Beispiel :
```
s1:s='mail.mysever.com';
s2:s='mail@mirakon.com';
s3:s='Hans Müller';
dest:s='meier@myclient.com';
t:s='Happy new year !';
cc:ls;
msg:ls=('Hello Mr.Meier','...','...');
att:ls=('c:\mail\myfile.doc','c:\picture.bmp');
sendemail(dest,t,cc,msg,att,server=s1,from=s2,fromid=s3);
```

5.21.3 FTP

procedure CONNECTFTP (/server,user,pwd,acc,account:s; var res:i);

Erstellt eine Verbindung mit einem FTP-Server

server : FTP-Server

user : User-ID

pwd : Password

acc : Konto-Nr in FTP-Server

account : Konto-Nr in Mirakon-Konfiguration

res : Ergebnis: 0=nicht erfolgt 1=erfolgt

Beispiel :
```
s1:s='ftp.microsoft.com';
s2:s='Hans Müller';
s3:s='hans';
s4:s='myaccount123';
connectftp(server=s1,user=s2,pwd=s3,acc=s4);
```

procedure DISCONNECTFTP;

Schliesst die aktuelle Verbindung mit dem FTP-Server.

procedure PUTFTPFILE (localfile,remotefile:s; /showprog:i; var res:i);

Sendet und speichert eine Datei in dem FTP-Server.

localfile : Datei die gesendet wird

remotefile : Dateipfad in FTP-Server

showprog : Zeigt Ablauffortschritt in Nachrichtenfeld

res : Ergebnis: 0=nicht erfolgt 1=erfolgt

procedure GETFTPFILE (localfile,remotefile:s; /showprog:i; var res:i);

Lädt eine Datei aus dem FTP-Server.

localfile : Dateipfad in lokalen Rechner

remotefile : Dateipfad in FTP-Server

showprog : Zeigt Ablauffortschritt in Nachrichtenfeld

res : Ergebnis: 0=nicht erfolgt 1=erfolgt

procedure DELETEFTPFILE (remotefile:s; /var res:i);

Löscht eine Datei aus dem FTP-Server.

remotefile : Dateipfad in FTP-Server
res : Ergebnis: 0=nicht erfolgt 1=erfolgt

procedure RENAMEFTPFILE (file1,file2:s; /var res:i);

Nennt eine Datei um in dem FTP-Server.

file1 : Datei die umgenannt wird
file2 : Neue Dateiname
res : Ergebnis: 0=nicht erfolgt 1=erfolgt