ptc

creo®

Creo Elements/Direct Drafting Makroprogrammierung

Creo Elements/Direct Drafting 20.6.0.0

PTC Inc., 121 Seaport Blvd, Boston, MA 02210 USA

Inhalt

Vorwort

Creo Elements/Direct Drafting ist ein vielseitiges 2D-System für Konstruktionen und Zeichnungen, mit dem sich jede Phase des Konstruktionsprozesses optimieren lässt. Mithilfe von Creo Elements/Direct Drafting können Sie im Handumdrehen 2D-Zeichnungen erstellen und ändern.

Dieses Handbuch erklärt das Schreiben von Makros zur Verwendung mit Creo Elements/Direct Drafting. Makros können Ihre tägliche Arbeit beschleunigen und automatisieren.

An wen richtet sich dieses Handbuch?

Dieses Handbuch richtet sich an:

- Systemadministratoren

  Sie sind ein Systemadministrator mit Erfahrungen mit UNIX- und Windows-basierten Betriebssystemen. Sie sind mit Startdateien und Anpassungsdateien vertraut. Sie nutzen Makros zur Vereinfachung von Prozessen innerhalb Ihrer Gruppe.

- Designer oder Zeichner

  Sie sind ein erfahrener Creo Elements/Direct Drafting Nutzer. Sie führen viele sich wiederholende Aufgaben manuell durch und möchten wissen, wie man mit Makros diese Aufgaben automatisieren kann. Sie benötigen keine Programmier-Vorkenntnisse.

Ziel dieses Handbuchs

Dieses Handbuch enthält Informationen zu den folgenden Themen:

- Geometrie erstellen
- Spezifikationen aus Datendateien extrahieren
- Mit Programmiersprachen wie Pascal oder C kommunizieren, um komplexe Berechnungen zu erstellen (nur UNIX-basierte Systeme)
- Zeichnungen beschriften
- Teilelisten erstellen
- Startprozesse anpassen
- Menüanzeigen und -funktionen steuern
- Tastatur anpassen
- Schriftarten anpassen

So verwenden Sie dieses Handbuch

Systemadministratoren

Nachstehend finden Sie einige Empfehlungen zur Verwendung dieses Handbuchs:

1. In den Kapiteln 1 bis 4 finden Sie allgemeine Informationen zur Makroprogrammierung.
2. In Kapitel 12 finden Sie Informationen dazu, wie Sie einen Makrobefehl oder eine Makrofunktion mit einer Aufgabe verknüpfen, die Sie manuell über die Benutzeroberfläche durchführen.
3. In Kapitel 13 erhalten Sie Informationen zur Anpassung.
4. In Kapitel 14 erhalten Sie eine kurze Beschreibung von Befehlen und Funktionen.
5. Anhang A enthält Informationen zu logischen Tabellen und Anzeigetabellen.

Designer oder Zeichner

Nachstehend finden Sie einige Empfehlungen zur Verwendung dieses Handbuchs:

1. In den Kapiteln 1 bis 4 finden Sie allgemeine Informationen zur Makroprogrammierung.
2. In Kapitel 5 können Sie Ihr Wissen über Vektoren auffrischen.
3. In den Kapiteln 6 bis 11 finden Sie Detailinformationen zu Ihren täglichen Aufgaben.
4. In Kapitel 12 finden Sie Informationen dazu, wie Sie einen Makrobefehl oder eine Makrofunktion mit einer Aufgabe verknüpfen, die Sie manuell über die Benutzeroberfläche durchführen.
5. Kapitel 13 hilft Ihnen bei der Anpassung Ihrer eigenen Umgebung.
6. In Kapitel 14 erhalten Sie eine kurze Beschreibung von Befehlen und Funktionen.

Gliederung dieses Handbuchs

Kapitel 1	What is a Macro? enthält eine Einführung in Makros sowie das Speichern, Ausführen und Debuggen von Makros.
Kapitel 2	Using the Editor to Write a Macro beschreibt die Verwendung des integrierten Editors zum Schreiben Ihrer Makros.
Kapitel 3	Macro Basics erklärt die Nutzung von Syntaxdiagrammen, lokalen und globalen Variablen, Kontrollstrukturen, Klammern und defensiver Programmierung.
Kapitel 4	Inquiring about the Environment and Elements erklärt, wie Sie auf Umgebungsinformationen wie Maßeinheiten und Linienarten zugreifen und wie Sie Umgebungsinformationen wiederherstellen, die Sie während der Ausführung Ihres Makros ändern.
Kapitel 5	Quick Review of Points and Vectors wappnet Sie für die Verwendung von Makros.
Kapitel 6	Writing Geometry Macros erklärt, wie Sie die erste Vektoranalyse durchführen und diese Analyse in Ihr Makro integrieren.
Kapitel 7	File Input/Output and Text Strings zeigt, wie Sie Zeichenfolgendaten verarbeiten und Zeichenfolgen aus ASCII-Dateien extrahieren.
Kapitel 8	Using Dimensions Stored in a Data File zeigt, wie Sie Geometrie mithilfe von Tabellendaten aus den Dateien erzeugen.
Kapitel 9	Useful Macros enthält eine Auswahl von Makros, die für Ihre täglichen Aufgaben hilfreich sind.
Kapitel 10	Recording the System Operation erklärt Ihnen, wie Sie Creo Elements/Direct Drafting Operationen aufzeichnen und wie Sie diese Aufzeichnungen für Makros nutzen können.
Kapitel 11	Using the Interface to Find a Command zeigt, wie Sie einen Makrobefehl mit einer Aufgabe verknüpfen, die Sie auf der Benutzeroberfläche ausführen.
Kapitel 12	Customizing erklärt Ihnen, wie Sie den Startvorgang, die Umgebung, die Tastatur und Schriftarten anpassen können.
Kapitel 13	Brief List of Commands and Functions bietet ein- oder zweizeilige Beschreibungen für Befehle und Funktionen.
Anhang A	Logical and Display Tables befasst sich mit logischen Tabellen und Anzeigetabellen.

Online-Hilfe

Eine vollständige Beschreibung und die Syntax alle Befehle und Funktionen finden Sie in der Online-Hilfe.

Wenn Sie beispielsweise Hilfe hinsichtlich des Befehls `MODIFY` benötigen, geben Sie Folgendes in die Befehlszeile ein:

```
help modify
```

Daraufhin werden die gewünschten Informationen zum Bearbeiten Ihrer Zeichnung auf dem Creo Elements/Direct Drafting Bildschirm angezeigt.

Schriftarten und ihre Bedeutung

Dieses Handbuch verwendet unterschiedliche Schriftarten, um zwischen gewöhnlichem Text, Dateinamen, Menübefehlen usw. zu unterscheiden.

Tabelle 1. Konventionen in diesem Handbuch

Konvention	Erläuterung
Fett	Menüpfade, Dialogfeldoptionen, Schaltflächen und andere auswählbare Elemente aus der Benutzeroberfläche. Beispiel: **LINIENGITT**, **ZEIGEN > SCHICHT**, **ZOOM** usw.
`Courier`	Benutzereingabe, Systemnachrichten, Verzeichnisse und Dateinamen.

1

Makros - Kurze Einführung

In diesem Kapitel wird erklärt, was ein Makro ist und wie ein Makro erstellt, geladen, gespeichert, ausgeführt und beendet wird.

Arbeit durch Makros erleichtern

Ein Makro bietet eine schnelle und einfache Methode, eine Abfolge von Befehlen automatisch auszuführen. Der Benutzer muss kaum oder gar nicht eingreifen. Es ist zweckmäßig, dass Sie sich für alle Befehlsfolgen, die Sie häufiger verwenden wollen, Makros erstellen.

Beispiel: Sie melden sich mindestens einmal am Tag bei Ihrer Workstation ab. Bevor Sie sich abmelden, speichern Sie Ihre Zeichnung, und geben Sie dann `exit confirm` in der Befehlszeile ein.

Dieser Vorgang kann mit dem Makro `Quit` automatisiert werden:

```
DEFINE Quit
{##############################################}
{##  This macro stores your current           ##}
{##  drawing in 'filename', then ends         ##}
{##  your ME-CAD session.                     ##}
{##############################################}
  STORE ALL DEL_OLD 'filename'
  EXIT CONFIRM
END_DEFINE
```

Das Makro enthält die gleichen Befehle, die Sie üblicherweise zum Speichern Ihrer Zeichnung und zum Abmelden von Ihrem Arbeitsplatzrechner eingeben. Die Kommentare am Anfang des Makros sollen dem Benutzer helfen, die Funktion des Makros besser zu verstehen.

Die Befehle sind in einer Datei gespeichert und werden beim Abarbeiten des Makros ausgeführt. Ein Makro wird durch Eingabe des Makronamens in die Befehlszeile oder durch Wahl aus einem Bildschirmmenü ausgeführt.

Makros werden in erster Linie verwendet zum:

- Erstellen von geometrischen Figuren
- Durchführen von Berechnungen
- Beschriften von Zeichnungen
- Erstellen von Teilelisten
- Steuern von Menüanzeigen und -funktionen

Zunächst wird beschrieben, wie mit dem im ME-System integrierten Editor Makros erstellen und editieren und welche Schritte zum Speichern eines Makros erforderlich sind.

Sie können Makros mit verschiedenen (Text-)Editoren erstellen:

- Mit einem beliebigen Editor.
- Mit dem zum Lieferumfang der ME-CAD-Systemsoftware enthaltenen Bildschirmeditor.

Der Bildschirmeditor arbeitet ähnlich wie PC-Editoren. Alle Beispiele in diesem Kapitel können mit diesem Editor eingegeben werden.

Makrodatei erstellen

Bevor Sie ein Makro schreiben, muss eine Datei erstellt werden, in der das Makro gespeichert wird. Sie möchten beispielsweise eine Datei namens `cad_mac.m` erstellen. Geben Sie in der ME-CAD-Umgebung Folgendes in der Befehlszeile ein:

```
EDIT_FILE 'cad_mac.m'
```

Existiert eine Datei mit diesem Namen bereits, wird sie vom System angezeigt und Sie können das Makro in die Datei eingeben. Wenn es noch keine Datei mit diesem Namen gibt, wird vom System eine leere Datei erstellt und Sie können das Makro eingeben. Sie geben Ihr Makro wie später unter Makros erstellen auf Seite 19 beschrieben ein.

Makros speichern

Um Ihr Makro zu speichern und zum ME-CAD-Bildschirm zurückzukehren, drücken Sie `[Ctrl] [D]`. Hierdurch wird die Datei auf der Festplatte Ihres Systems gespeichert. Gleichzeitig wird eine entsprechende Meldung angezeigt.

```
writing 'cad_mac.m'
```

Wenn Sie das aktuelle Verzeichnis angeben, wird `'cad_mac.m'` in der Liste angezeigt.

Wenn Sie die Datei im Editor ändern, möchten Sie die Änderungen evtl. nicht speichern. Drücken Sie `[ESC]` oder `[Break]`, um den Editor zu verlassen. Die Änderungen werden dann nicht auf die Festplatte geschrieben.

Es können beliebig viele Makros in eine Datei gestellt werden. Dabei spielt die Reihenfolge der Makros keine Rolle. Wenn Sie die Makros jedoch alphabetisch nach Namen sortieren, erleichtert dies das spätere Auffinden.

Wenn Sie ein neues Makro erstellen und Makrofehler beheben, möchten Sie dieses neue Makro evtl. an den Anfang der Datei stellen. Die erste Seite einer Datei wird angezeigt, wenn Sie den Editor und den Befehl `EDIT_FILE` verwenden, damit Sie die Datei nicht durchblättern müssen, um das Makro zu finden. Wenn das Makro dann fehlerfrei ist, kann es an die vorgesehene Stelle in der Datei gestellt werden.

Eine weitere Methode, um ein neues Makro zu erstellen und Makrofehler zu beheben, ist, eine separate Datei für das neue Makro zu erzeugen. Wenn die Makrofehler vollständig behoben sind, können Sie das Makro an die normale Makrodatei anhängen. Diese Methode hat zwei Vorteile:

- Wenn Sie den `EDIT_FILE`-Befehl verwenden, wird Ihr Text schneller angezeigt.
- Der `INPUT`-Befehl wird schneller ausgeführt. (Der Befehl `INPUT` kompiliert und lädt das Makro und wird im nächsten Schnitt beschrieben.)

Es ist üblich, jedes Makro in einer eigenen Datei zu speichern. Dabei kann die Datei den gleichen Namen wie das Makro haben. Wenn mit dem Makro andere Makros aufgerufen werden, müssen diese Makros gegebenenfalls gesondert geladen werden.

Makros, die Sie seltener benutzen, können auf andere Art gespeichert werden. Stellen sie ähnliche Makros in eine gemeinsame Datei. Beispielsweise können alle Bolzenmakros in der Datei `'bolt.m'` und alle Flanschmakros in `'flange.m'` gespeichert werden. Dabei sollten die Makronamen innerhalb der Dateien gleich sein. Beispielsweise heißt das erste Bolzenmakro und das erste Flanschmakro `macro1`, das zweite Bolzenmakro und das zweite Flanschmakro `macro2` usw. Jede Datei kann nach Bedarf geladen werden. Bei jedem Laden einer Datei werden gleichnamige Makros im Arbeitsspeicher (RAM) überschrieben. Beispielsweise überschreibt das Bolzenmakro `macro1` das Flanschmakro `macro1` usw. Hierdurch wird Speicherplatz nicht unnötig von Makros belegt, die nur selten verwendet werden.

Wenn Sie den `INPUT`-Befehl in einem Makro verwenden, muss er mit dem Kennzeichner `IMMEDIATE` verwendet werden (siehe auch INPUT auf Seite 33).

Makros löschen

Sie können ein einzelnes Makro aus dem RAM mit dem Befehl `DELETE_MACRO` löschen. Dieser Befehl kann z.B. verwendet werden, wenn mehr Speicherplatz zum Laden einer großen Zeichnung benötigt wird. Wenn Sie beispielsweise das Makro `Quit` löschen möchten, verwenden Sie den folgenden Befehl:

```
DELETE_MACRO Quit
```

Makros ausführen

Zum Ausführen eines Makros muss es kompiliert und in den Arbeitsspeicher (RAM) geladen werden. Dazu geben Sie den Namen der das Makro beherbergenden Datei folgendermaßen in die Befehlszeile ein:

```
INPUT 'cad_mac.m'
```

Nach dem Laden der Datei in den Arbeitsspeicher haben Sie zwei Möglichkeiten ein Makro aus dieser Datei zu starten. Beide Methoden werden nachfolgend beschrieben.

Makros über die Befehlszeile starten

Sie können nun durch Eingabe des entsprechenden Makronamens in die Befehlszeile jedes beliebige Makro in dieser Datei ausführen. Wenn die Datei `cad_mac.m` das Makro `Quit` enthält, können Sie das Makro ausführen, indem Sie Folgendes in die Befehlszeile eingeben:

```
 Quit
```

Das Makro wird Zeile für Zeile abgearbeitet. Währenddessen werden seine Befehle und Anweisungen auf Fehler überprüft. Dies ist mit "interpretierenden" Versionen der Programmiersprache BASIC vergleichbar. Tritt dabei ein Fehler auf, wird die Ausführung des Makros abgebrochen und eine entsprechende Meldung angezeigt.

Fehler in Makros beheben (debugging)

Kann das Makro nicht ausgeführt werden, müssen Sie es für die Fehlerbehebung zunächst editieren. Die Bearbeitung eines Makros entspricht der Erzeugung eines Makros: Verwenden Sie `EDIT_FILE`, um die Datei zu öffnen, und nehmen Sie dann Änderungen vor.

Nach Aufrufen der Datei wird die erste Seite der Datei angezeigt. Mit den Cursortasten können Sie nun den Cursor auf das zu ändernde Makro setzen. Überprüfen Sie das Makro Zeile für Zeile, bis Sie den Fehler gefunden haben. Bei den meisten Fehlern handelt es sich um Syntaxfehler, wie z.B. eine fehlende Klammer oder ein fehlendes Unterstreichungszeichen.

Falls ein Fehler nicht schnell genug gefunden werden kann, können Sie mit der Funktion `BREAKPOINT` den Fehlersuchmodus aktivieren. Fügen Sie `BREAKPOINT` an der gewünschten Position im Makrocode ein. Dies kann sowohl durch direkte Bearbeitung der Datei als auch mit der Funktion `EDIT_MACRO` erreicht werden. Wenn der Code einen `BREAKPOINT` erreicht, wechselt das System in den Fehlerbehebungsmodus.

Die Funktion für Unterbrechungspunkte ist standardmäßig aktiviert. Wenn Sie feststellen, dass dies nicht funktioniert, prüfen Sie, ob `ENABLE_BREAKPOINT` auf `ON` eingestellt ist.

So aktivieren Sie die `BREAKPOINT`-Funktion in Creo Elements/Direct Drafting:

- Klicken Sie auf **Verschiedenes**, und aktivieren Sie anschließend in der Gruppe **System** das Kontrollkästchen **Unterbrechungspunkte**.

Bei der Verwendung von aktiven Unterbrechungspunkten stellt das System eine Parametertabelle dar.

Falls der Fehlerbehebungsmodus bereits aktiv ist und das Fehlerbehebungsprogramm bereits mindestens ein Token vorwärts gegangen ist, wird mit dem Schlüsselwort `BREAKPOINT` ein Unterbrechungspunkt an der aktuellen Stelle eingefügt.

Falls im Makro-Code ein `BREAKPOINT` zwischen den Schlüsselwörtern `PARAMETER` und `LOCAL` gesetzt wird, wird vor der ersten der beiden Anweisungen im Makro das Fehlerbehebungsprogramm unterbrochen.

Hinweis:Bitte beachten Sie, dass der Unterbrechungspunkt nur während der aktuellen CoCreate Drafting-Sitzung gültig ist.Er wird nicht in der entsprechenden Datei gespeichert.

Sobald Sie im Fehlersuchmodus sind, können Sie mit folgenden Befehlen durch den Code gehen und die Fehlerstelle isolieren:

- `STEP_NEXT`: Verwenden Sie diese Funktion, um schrittweise ein Token im Makrocode weiterzugehen.
- `STEP_OUT`: Verlassen Sie die aktuelle Makrofunktion.
- `STEP_OVER`: Überspringen Sie das nächste Token, auch wenn dieses eine Makrofunktion ist.
- `STEP n`: Überspringen Sie mehrere Token in einem Schritt.

- `SKIP n`: Überspringen Sie das aktuelle Token mehrmals.
- `CONTINUE`: Fahren Sie bis zum nächsten Unterbrechungspunkt mit der Ausführung des Makros fort.
- `GO`: Führen Sie den restlichen Code aus, ohne die Unterbrechungspunkte zu beachten.
- `REMOVE_BREAKPOINT`: Entfernen Sie den aktuellen Unterbrechungspunkt.
- `ENABLE_BREAKPOINTS`: Aktivieren Sie alle Unterbrechungspunkte.
- `LIST_BREAKPOINTS`: Listen Sie alle Unterbrechungspunkte in einem Editor auf.

Wenn Sie mit `BREAKPOINT` Fehler beheben, bedenken Sie Folgendes:

- Verwenden Sie den Befehl `DISPLAY`, um variable Werte für Ihr Makro anzuzeigen.
- Während der Unterbrechung kann jede Creo Elements/Direct Drafting Funktion ausgeführt werden.
- Creo Elements/Direct Drafting Befehle unterbrechen das Makro.
- Wenn ein Makro aufgrund der Befehle `INPUT, EDIT_MACRO, DELETE_MACRO, or DEFINE` in der Aufrufliste geändert wird, wird auch das Fehlerbehebungsprogramm unterbrochen.
- Das Fehlerbehebungsprogramm verfügt über keine grafische Benutzeroberfläche. Zusätzliche Informationen wie den aktuellen Befehl, den letzten Token, den nächsten Token und den nächsten Befehl finden Sie im Fenster der Befehlszeile.
- Ein Betrachtungsprogramm für den Code ist im Fehlerbehebungsprogramm ebenfalls nicht enthalten. Um den Code während der Fehlersuche anzeigen zu können, benötigen Sie ein externes Betrachtungsprogramm.

Wenn Sie den Fehler gefunden und ihn behoben haben, drücken Sie `[Ctrl] D`, um zum ME-CAD-Bildschirm zurückzukehren. Die alte Version der Datei wird von der neuen überschrieben.

Die kompilierte Version der Datei im RAM bleibt unverändert. Diese kann nur durch Laden der neuen Version in den RAM geändert werden. Dazu ist der Befehl `INPUT` zu verwenden. Ein häufig begangener Fehler besteht darin, nach Editieren der Datei das Makro aufzurufen, ohne es zuvor über den Befehl `INPUT` zu aktivieren. Hierdurch wird die alte Version des Makros ausgeführt.

Zum erneuten Aufrufen der Datei geben Sie folgendes ein:

```
INPUT 'cad_mac.m'
```

Hinweis

Wenn Sie Befehle wiederholen möchten, drücken Sie die `[PgUp]`-Taste. Wenn Sie einen Befehl ausführen möchten, der einem vorherigen Befehl ähnelt, drücken Sie die `[PgUp]`-Taste, und editieren Sie anschließend den Befehl.

Der Inhalt von `cad_mac.m` wird von der Festplatte neu geladen und überschreibt die alte Kopie im RAM. Nun kann das Makro erneut ausgeführt werden. Nach dem Laden der Datei kann ein Makro beliebig oft ausgeführt werden.

Wenn Sie das Makro `Quit` in der Datei `cad_mac.m` erstellen und Fehler beheben, gehen Sie wie folgt vor:

```
EDIT_FILE 'cad_mac.m '
INPUT 'cad_mac.m '
Quit
```

Es ist eine `trace`-Funktion vorhanden, die für die Behebung von Fehlern in Makros nützlich ist. Dies wird unter Ablaufverfolgung zum Beheben von Fehlern auf Seite 46 beschrieben.

Ausführung von Makros abbrechen

Wenn Sie das Makro aus einem bestimmten Grund (wie eine Endlosschleife) anhalten möchten, verwenden Sie die `[Ctrl] [Break]`-Taste. Die Auswirkungen von `[Break]` hängen von Ihrer Betriebsumgebung ab.

Hinweis

Wenn Sie beim Starten von Creo Elements/Direct Drafting im Arbeitsbereichmenü oder Grafikfenster die `[Break]`-Taste oder die `[Again]`-Taste drücken, kommt es erst zu einer Unterbrechung, wenn das System Eingaben verarbeiten kann.

Näheres dazu kann in der Dokumentation zu Windows nachgelesen werden.

2

Makros erstellen

In diesem Kapitel wird erklärt, wie mit dem integrierten ME-Editor Makros erstellt werden können. Sie lernen hier, wie Sie:

- Zeilenblöcke innerhalb einer Datei bewegen.
- Text aus anderen Dateien in die aktuelle Datei laden.
- eine bestimmte Zeichenfolge in der Datei finden.
- eine bestimmte Zeichenfolge durch eine andere ersetzen.
- Text rechtsbündig ausrichten.

Mit den Editiertasten arbeiten

Im Editor kann mit folgenden Editiertasten gearbeitet werden: Diese Tasten umfassen Folgendes:

- `[Insert char]` oder `[Ins]`
- `[Delete char]` oder `[Del]`
- `[Clear line]` oder `[Alt] [Del]`
- `[Clear display]` oder `[Control] [Del]`
- Pfeiltasten: `[<] [>] [^] [v]`

Den Editor aufrufen und verlassen

Der Editor kann nur in der ME-CAD-Umgebung aufgerufen werden. Hierzu geben Sie über die Tastatur `EDIT_FILE` und den Namen der zu editierenden Datei in einfachen Anführungszeichen ein. Wenn Sie beispielsweise eine Datei mit dem Namen `'macros'` editieren möchten, geben Sie in der Befehlszeile Folgendes ein:

```
EDIT_FILE 'macros'
```

Existiert die Datei bereits, wird sie auf dem Bildschirm angezeigt. (Wenn nicht, bleibt der Bildschirm leer.)

Wenn Sie den Editor verlassen und die Änderungen in der Datei speichern möchten, drücken Sie `[Ctrl] [D]`.

Wenn Sie den Editor verlassen, die Änderungen jedoch nicht auf der Festplatte speichern möchten, drücken Sie `[ESC]` oder `[Ctrl] [Break]`

Mit `EDIT_PORT` den Editor schnell öffnen

Der `EDIT_PORT`-Befehl öffnet ein kleines Darstellungsfenster auf dem ME-CAD-Bildschirm. Dieses Fenster erscheint nur, wenn der Editor aufgerufen wird. Je kleiner das Fenster ist, desto schneller kann der Editor aufgerufen und verlassen werden, da hierdurch die Zeit zum Neuzeichnen der Zeichnung auf dem Bildschirm verkürzt wird.

Um den Befehl zu verwenden, geben Sie `EDIT_PORT` in der Befehlszeile ein. Daraufhin erscheint folgende Eingabeaufforderung:

```
Enter background color or border width or corner of new port
```

Legen Sie erst die linke obere Ecke des ME-Bildschirms und anschließend die rechte untere Ecke des gewünschten Fensters durch Antippen mit dem Tablettgriffel fest.

Überprüfen Sie das Ergebnis, indem Sie `EDIT_FILE` und einen beliebigen Dateinamen in einfachen Anführungszeichen in die Befehlszeile eingeben. Wiederholen Sie den Vorgang, wenn das Fenster nicht die richtige Größe hat. Die Größe der Darstellungsfenster bleibt unverändert, bis Sie sie mit einem anderen

EDIT_PORT-Befehl ändern oder Ihre ME-CAD-Sitzung mit EXIT CONFIRM beenden. Wenn Sie das nächste Mal Ihr ME-CAD-System verwendet, müssen Sie EDIT_PORT erneut verwenden.

EDIT_PORT ist besonders dann von Nutzen, wenn Sie Text zu Zeichnungen hinzufügen. Da solcher Text normalerweise nicht länger als einige Wörter ist, reicht in den meisten Fällen ein Fenster von 8 cm Breite und 2 cm Höhe (in der linken oberen Ecke des ME-CAD-Bildschirms) aus. Hierdurch verkürzt sich außerdem die Zeit zum Neuzeichnen der Zeichnung erheblich.

Beachten Sie, dass HELP_PORT auf die gleiche Weise funktioniert. HELP_PORT beschleunigt den Zugriff auf die help-Datei, indem ein kleines Darstellungsfenster verwendet wird, um Bildschirmneuzeichnungen zu minimieren.

Mit Markierungen arbeiten

Viele der später beschriebenen Editierbefehle können nur mit Markierungen ausgeführt werden. Wenn Sie z.B. einen Textblock innerhalb einer Datei bewegen wollen, müssen Sie entweder den Anfang oder das Ende eines Blocks markieren.

Eine Markierung ist ein Zeichen, mit dem eine Zeile gekennzeichnet wird, das selbst jedoch in der Datei nicht sichtbar ist. Einige Markierungen werden vom System automatisch gesetzt.

Die Tabelle unten enthält die gültigen Markierungszeichen.

Markierungen	Erklärung
^	Wird vom System zum Markieren der ersten Zeile einer Datei gesetzt.
!	Wird vom System zum Markieren der letzten Zeile einer Datei gesetzt.
0 . . . 9	Wird vom Benutzer zum Markieren einer beliebigen Zeile in der Datei gesetzt.

Im folgenden soll die Verwendung von Markierungen anhand einiger Beispiele näher erläutert werden. Wenn Sie sich im ME-CAD-Bildschirm befinden, ist zum Erstellen einer neuen Datei folgendes in die Befehlszeile einzugeben:

```
EDIT_FILE 'markers'
```

Geben sie nun einige Zeilen Text ein, z.B.:

```
AAAAAAAAA
BBBBBBBBBBB
CCCCCCCCCCC
DDDDDDDDDDD
EEEEEEEEE
FFFFFFFFF
```

Die Zeilen mit Bs, Cs und Ds sind länger als die anderen Zeilen, damit sie später einfach identifiziert werden können.

Wenn mit den durch das System definierten Markierungen z.B. Anfang und Ende der Datei gekennzeichnet werden sollen, ist wie folgt vorzugehen.

Geben Sie $ ein. (Drücken Sie nicht [Enter].) Daraufhin wird der Cursor (mit dem $-Zeichen davor) am unteren Bildschirmrand angezeigt. Das $-Zeichen ist das standardmäßig festgelegte ESC-Zeichen und muss vor allen Editierbefehlen eingegeben werden.

Geben Sie nun ^ ein, und drücken Sie [Enter]. Der Cursor erscheint in der ersten Zeile der Datei.

Geben Sie $! ein, und drücken Sie [Enter]. Der Cursor springt in die letzte Zeile der Datei.

Mit den Markierungen kann jede beliebige Zeile in dieser Datei gekennzeichnet werden. Dazu ist der Cursor in die gewünschte Zeile zu bewegen und folgendes einzugeben:

```
$sm n
```

Das Zeichen $ ist das standardmäßig festgelegte ESC-Zeichen. sm ist der festgelegte Markierungsbefehl. n ist eine Zahl zwischen 0 und 9. Das Leerzeichen zwischen sm und n ist optional. Die Zeile wird mit der angegebenen Zahl gekennzeichnet, diese Markierung ist auf dem Bildschirm jedoch nicht sichtbar.

Beispiel: Wir verwenden die Markierung 1 in der Zeile mit Ds. Verschieben Sie den Cursor in der Zeile mit Ds, und geben Sie dann Folgendes ein:

```
$sm1
```

Setzen Sie nun den Cursor in eine andere Zeile und geben Sie folgendes ein:

```
$1
```

Der Cursor springt zur Zeile mit Ds.

Text kopieren

Im folgenden soll gezeigt werden, wie ein Textblock kopiert wird. Es sollen die drei Zeilen mit Bs, Cs und Ds kopiert und diese hinter der Zeile mit Es platziert werden. Gehen Sie dazu folgendermaßen vor:

- Um den Block mit drei Zeilen zu definieren, können Sie die Zeile mit Bs kennzeichnen und den Cursor dann in der Zeile mit Ds platzieren. Oder Sie können die Zeile mit Ds kennzeichnen und den Cursor dann in der Zeile mit Bs platzieren. Markierung 1 befindet sich bereits in der Zeile mit Ds.
- Um den Text nach der Zeile mit Es einzufügen, verwenden Sie in der Zeile mit Es eine Markierung. Verschieben Sie den Cursor in der Zeile mit Es, und geben Sie dann Folgendes ein:
  ```
  $sm2
  ```
- Platzieren Sie den Cursor an einer beliebigen Stelle in der Zeile mit Bs.

Denken Sie daran, dass die Markierungen nicht sichtbar sind. Wenn die Markierungen jedoch sichtbar wären, würde die Datei wie folgt aussehen:

```
AAAAAAAAA
BBBBBBBBBBBB    * (cursor line)
CCCCCCCCCCCC
DDDDDDDDDDD     1 (marker)
EEEEEEEEE       2 (marker)
FFFFFFFFF
```

Zur Erinnerung: Es sollen die Zeilen mit Bs, Cs und Ds kopiert und diese hinter der Zeile mit Es platziert werden. Geben Sie dazu Folgendes ein:

```
$c12
```

Der Textblock von der aktuellen Cursorposition bis zur Markierung 1 wird hinter die Markierung 2 kopiert. Die Datei hat nun folgendes Aussehen:

```
AAAAAAAAA
BBBBBBBBBBB
CCCCCCCCCCC
DDDDDDDDDDD
EEEEEEEEE
BBBBBBBBBBB
CCCCCCCCCCC
DDDDDDDDDDD
FFFFFFFFF
```

Mit den Editierbefehlen arbeiten

In diesem Abschnitt werden die Editierbefehle erläutert. Den Beschreibungen der Befehle liegt folgende Terminologie zugrunde:

- Ein Element in eckigen Klammern ([. . .]) ist optional.
- Ein Element in einfachen Anführungszeichen ('. . .') bedeutet, dass die Zeichenfolge, die Sie eingeben, in einfachen Anführungszeichen stehen muss.
- Ein kursiv gedrucktes Element steht als Platzhalter für das einzugebende Element. Der Befehl

  ```
  $marker
  ```

  bedeutet, dass Sie einen numerischen Wert für die Markierung eingeben müssen, wie 2.

  Ein anderes Beispiel:Der Befehl zum Laden einer Datei lautet:

  ```
  L 'filename'
  ```

  Das heißt, dass der Name der Datei (zwischen den einfachen Anführungszeichen) hinter dem L eingegeben werden muss. Wenn die zu ladende Datei cad_mac.m ist, müssen Sie Folgendes eingeben:

  ```
  L 'cad_mac.m'
  ```

- Die aktuelle Zeile ist die Zeile, in der sich der Cursor gerade befindet.

In der folgenden Tabelle sind sämtliche Editierbefehle aufgeführt. Bevor Sie einen Editorbefehl ausführen, müssen Sie $ eingeben. Daraufhin wird der Cursor (mit dem $-Zeichen davor) am unteren Bildschirmrand angezeigt. Das Zeichen $ ist das standardmäßig festgelegte ESC-Zeichen.

Befehl	Funktion
marker	Setzt den Cursor in die Zeile, in der sich die angegebene Markierung befindet. Wenn Sie $ 4 eingeben, wechselt der Cursor in die Zeile mit Markierungsnummer 4.
'string'	Setzt den Cursor an die Stelle, an der die angegebene Zeichenfolge als nächstes auftritt. Wenn Sie $ 'SolidDesigner' eingeben, wechselt der Cursor an die Stelle, an der die Zeichenfolge SolidDesigner als Nächstes auftritt.

Befehl	Funktion
? ['keyword']	Die Systemdatei help wird im Abschnitt mit der Beschreibung des Schlüsselworts angezeigt. Wenn Sie ? ohne ein Schlüsselwort eingeben, wird der Abschnitt zum Bildschirmeditor angezeigt. Um zu Ihrer Datei zurückzukehren, drücken Sie [Ctrl] [D].
AC	Mitte anpassen. Für jede Zeile eines Absatzes zentralisiert dieser Befehl den Text zwischen den Rändern.
AF	Füllung anpassen. Dieser Befehl passt einen Abschnitt zwischen den Rändern an. Der Text ist links bündig und rechts nicht bündig.
AJ	Ausrichtung anpassen. Dieser Befehl passt einen Abschnitt zwischen den Rändern an. Der Text ist links und rechts bündig. Der Abstand zwischen den einzelnen Wörtern ist hierbei unregelmäßig.

Hinweis

Verwenden Sie nicht AF oder AJ, um Makros zu editieren, da jede Zeile mit dem Ende der vorherigen Zeile verbunden wird. Diese Befehle sollten nur für die Bearbeitung von Text verwendet werden.

Befehl	Funktion
C marker1 marker2	Kopieren. Kopiert alle Textzeilen zwischen der aktuellen Zeile und marker1 (inklusive) und fügt sie nach marker2 ein. Wenn Sie $C34 eingeben, wird der gesamte Text zwischen der aktuellen Zeile und marker 3 kopiert und nach marker 4 eingefügt.
D marker	Löschen. Löscht alle Textzeilen von der aktuellen Zeile bis zur Markierung (inklusive).
H ['keyword']	Hilfe. Entspricht dem Befehl ?.
L 'filename'	Laden. Kopiert alle Textzeilen der angegebenen Datei und fügt sie nach der aktuellen Zeile ein. Wenn Sie $L 'cad_macros' eingeben, wird die Datei "cad_macros" nach der aktuellen Zeile eingefügt.
M marker1 marker2	Verschieben. Verschiebt alle Textzeilen zwischen der aktuellen Zeile und

Befehl	Funktion
	marker1 (inklusive) und fügt sie nach marker2 ein. Quelle und Ziel dürfen sich nicht überlappen. Wenn Sie $M67 eingeben, wird der gesamte Text zwischen der aktuellen Zeile und marker 6 verschoben und nach marker 7 platziert.
N	Weiter. Wiederholt die letzte Zeichenfolgensuche.
O 'filename' [marker]	Überschreiben. Kopiert alle Textzeilen zwischen der aktuellen Zeile und der Markierung (inklusive) in eine Datei mit dem angegebenen Namen. Wenn keine Markierung angegeben wird, wird die Cursorposition ignoriert und die gesamte Datei kopiert. Eine bereits vorhandene Datei mit diesem Namen wird hierbei überschrieben. Wenn Sie $O 'cad_macros' ! eingeben, wird der Text zwischen der aktuellen und der letzten Zeile (inklusive) in die Datei cad_macros kopiert. Wenn Sie das ! auslassen, wird die gesamte Datei kopiert.

Befehl	Funktion
`R [V] ['string1'] ['string2'] [marker]`	Ersetzen. Ersetzt alle Vorkommen von `string1` durch `string2` zwischen der aktuellen Zeile und der Markierung (inklusive). Mit dem V können Sie jede Änderung von `string` prüfen. Wenn Sie keine Markierung eingeben, wird die Zeichenfolge nur einmal ersetzt. Falls eine der Zeichenfolgen fehlt, wird standardmäßig die zuletzt verwendete entsprechende Zeichenfolge (`string`) angenommen. Wenn Sie beispielsweise `$RV 'black' 'white' !` eingeben, wechselt der Cursor zum nächsten Vorkommen von `black`. Sie haben dann die Möglichkeit, es durch `white` zu ersetzen oder zum nächsten Vorkommen von `black` zu wechseln: R = ersetzen, '' = nicht ersetzen, S = abbrechen. Durch Drücken von `R` wird `black` durch `white` ersetzt. Durch Drücken der Leertaste wechseln Sie zum nächsten Vorkommen von `black`. Durch Drücken von `S` wird der Vorgang abgebrochen.
`SE 'character'`	ESC-Zeichen festlegen. Definiert das ESC-Zeichen neu. Mit dem Umschaltzeichen wird der Datei-Editor aufgerufen, so dass Editor-Befehle eingegeben werden können. Das Zeichen `$` ist das standardmäßig festgelegte ESC-Zeichen. Sie können jedes Zeichen außer `0 ... 9, A ... Z, a ... z, ... !` und `?` verwenden. Das Umschaltzeichen bleibt solange gültig, bis ein neues Umschaltzeichen definiert wird. Wenn Sie das ESC-Zeichen als # definieren möchten, geben Sie Folgendes ein: `$ SE '#'`

Befehl	Funktion
SL	Linken Rand festlegen. Legt den linken Rand auf die aktuelle Cursorposition fest.
SM n	Markierung festlegen. Legt die Markierung fest, die in der aktuellen Zeile angegeben ist. n kann jeden Wert von 0 bis 9 aufweisen. Das Leerzeichen zwischen SM und n ist optional. Wenn Sie $ SM7 eingeben, wird die Markierung 7 für die aktuelle Zeile verwendet.

Befehl	Funktion
SR	Rechten Rand festlegen. Legt den rechten Rand auf die aktuelle Cursorposition fest.
W 'filename' [marker]	Schreiben. Kopiert alle Textzeilen zwischen der aktuellen Zeile und der Markierung (inklusive) in eine Datei mit dem angegebenen Namen. Wird keine Markierung eingegeben, so kopiert das System die gesamte Datei. Ist bereits eine Datei mit dem angegebenen Namen vorhanden, so wird der Befehl nicht ausgeführt. Um den Text zwischen der aktuellen Zeile und Markierung 6 in die Datei cad_macros zu kopieren, geben Sie Folgendes ein: `$ W 'cad_macros' 6`

Hinweis

Der Text wird mit SL, SM und SR erst angepasst, wenn Sie AF oder AJ verwenden. Denken Sie daran, dass AF und AJ normalerweise nicht mit Makros verwendet werden.

EDIT_MACRO verwenden

In diesem Kapitel erfolgte die komplette Bearbeitung von Makros mit EDIT_FILE. Der EDIT_FILE-Befehl ist von Vorteil, da Sie jede Datei mit diesem Befehl überprüfen können; die Datei muss kein Makro enthalten.

Der EDIT_MACRO-Befehl kann nur für Makros verwendet werden.

Wir möchten Ihnen den EDIT_MACRO-Befehl vorstellen und ihn dann mit EDIT_FILE vergleichen. Danach können Sie selber entscheiden, welchen Befehl Sie für die Bearbeitung Ihrer Makros verwenden wollen.

Bevor Sie den EDIT_MACRO-Befehl verwenden können, muss das Makro bereits im RAM vorhanden sein. Es gibt zwei Möglichkeiten ein Makro zu laden. Sie werden nachfolgend beschrieben.

INPUT-Befehl für eine vorhandene Datei verwenden

Diese Methode wurde bisher in Kapitel 1 und 2 verwendet und ist in Kapitel 1 "Makros ausführen" beschrieben und dürfte Ihnen bereits bekannt sein.

Wenn Sie den INPUT-Befehl in einem Makro verwenden, muss er mit dem Kennzeichner IMMEDIATE verwendet werden (siehe auch INPUT auf Seite 33).

Makro in die Befehlszeile eingeben

Mit diesem Verfahren geben Sie DEFINE ein, gefolgt vom Namen des Makros. Wenn Ihr Makro Slot_mac heißt, geben Sie Folgendes ein:

```
DEFINE Slot_mac
```

Das System reagiert mit der Aufforderung:

```
Enter macro definition
```

Geben Sie jetzt die verbleibenden Zeilen Ihres Makro ein; eine nach der anderen. Immer wenn Sie [Enter] drücken, reagiert das System mit folgender Eingabeaufforderung:

```
Enter macro definition
```

Sie können das Makro jetzt starten, indem Sie den Makronamen in die Befehlszeile eingeben. Beachten Sie, dass Sie mit dieser Methode nicht INPUT verwenden müssen.

Sobald sich ein Makro im RAM befindet, können Sie das Makro mit dem Befehl EDIT_MACRO gefolgt vom Namen des Makros editieren. Beispielsweise geben Sie ein:

```
EDIT_MACRO Slot_mac
```

Das Makro wird vom Editor aufgerufen, und es kann genauso editiert werden, als hätten Sie es mit EDIT_FILE aufgerufen. Um den Editor zu verlassen, drücken Sie [Ctrl] [D]. Sie starten das Makro, indem Sie den Makronamen in die Befehlszeile eingeben. Sie müssen den INPUT-Befehl nicht verwenden.

Wenn Sie den Editor mit EDIT_MACRO aufrufen, wird das Makro nicht durch Drücken von [Ctrl] [D] gespeichert. Um das Makro zu speichern, geben Sie SAVE_MACRO in der Befehlszeile ein, gefolgt vom Namen des Makros.

Das System reagiert mit der Aufforderung:

```
Enter SCREEN, DEL_OLD, APPEND or 'file name'
```

Geben Sie SCREEN ein, falls Sie das Makro mit dem Editor einsehen möchten. Geben Sie DEL_OLD ein, um eine vorhandene Datei gleichen Namens zu überschreiben. Geben Sie APPEND ein, um das Makro am Ende einer vorhandenen Datei hinzuzufügen. Um eine neue Datei zu erhalten, geben Sie den Dateinamen ein.

Vergleich von `EDIT_FILE` und `EDIT_MACRO`

`EDIT_MACRO` ist besser geeignet, wenn Sie mehrere Versuche benötigen, um Fehler in einem Makro zu beheben, da Sie nicht jedes Mal `INPUT` verwenden müssen, nachdem Sie Änderungen vorgenommen haben.

Der Nachteil von `EDIT_MACRO` ist, dass das Macro nicht automatisch gespeichert wird, wenn Sie `[Ctrl] [D]` verwenden, um den Editor zu beenden. Wenn Sie die Creo Elements/Direct Drafting Umgebung mit `EXIT CONFIRM` verlassen, geht Ihr Makro verloren.

3

Grundlagen

Dieses Kapitel beschreibt den Aufbau eines Makros, zeigt die Unterschiede zwischen lokalen und globalen Variablen auf, erklärt die Verwendung von Klammern und erläutert, wie Fehler in Makros mit Hilfe der `trace`-Funktion behoben werden.

Aufbau eines Makros

Ein Makro kann aus folgenden sechs Bestandteilen bestehen.

- DEFINE Makroname
- Parameter
- Lokale Variablen
- Benutzereingabe
- Hauptteil eines Makros
- END_DEFINE

Ein Makro muss nicht alle diese Bestandteile umfassen. Das folgende Beispiel veranschaulicht sechs Bestandteile eines Makros:

```
DEFINE Circle_mac
PARAMETER P2
LOCAL P1
  READ 'Indicate center of circle' P1
  CIRCLE CENTER P1 P2 END
END_DEFINE
```

Sie rufen dieses Makro auf, indem Sie `Circle_mac` und den Wert für den Radius eines Kreises in die Befehlszeile eingeben. Dies ist offensichtlich kein sehr nützliches Makro, aber dieses Beispiel zeigt in wenigen Zeilen alle Teile eines Makros.

DEFINE

Diese Zeile beginnt immer mit dem Wort `DEFINE`, gefolgt vom Namen des Makros. In unserem Beispiel ist der Name des Makros `Circle_mac`:

```
DEFINE Circle_mac
```

Innerhalb des Makros ist die Groß-/Kleinschreibung zu beachten. Schlüsselwörter (für Befehle und Funktionen) sollten grundsätzlich durch Großschreibung hervorgehoben werden.

Schlüsselwörter wie Befehle und Funktionen bestehen komplett aus Großbuchstaben. Beispielsweise sind `DEFINE`, `LOCAL` und `END_DEFINE` Schlüsselwörter.

Das System akzeptiert ein Schlüsselwort in Groß- oder Kleinbuchstaben, jedoch keine gemischte Schreibweise. Beispiel: Das System akzeptiert `DEFINE` oder `define`, jedoch nicht `Define`, `dEFINE`, `DeFiNE` oder eine andere gemischte Version.

Der erste Buchstabe eines Makronamens ist normalerweise ein Großbuchstabe. Wenn ein Begriff wie `Circle_mac` in einer Makroliste enthalten ist, wissen Sie, dass es sich bei `Circle_mac` um ein Makro und kein Schlüsselwort handelt.

In diesem Kapitel wird die Groß-/Kleinschreibung vorbildlich verwendet.Sie haben jedoch als Benutzer grundsätzlich die Freiheit, eigene von diesem Vorbild abweichende Gewohnheiten (im Rahmen der festgelegten Regeln) zu entwickeln. Wenn Sie also aufgefordert werden,

```
EDIT_FILE 'cad_macros'
```

in die Befehlszeile einzugeben, dann können Sie beispielsweise auch eingeben:

```
edit_file 'cad_macros'
```

Hinweis

Jede in einfachen Anführungszeichen stehende Zeichenfolge muss genauso eingegeben werden, wie sie hier dargestellt ist.

Parameter

Dieser Teil des Makros definiert Variablen, die als Argument an das Makro weitergegeben werden. Sollen mehr als ein Parameter verwendet werden, müssen sie in einer bestimmten Reihenfolge aufgeführt werden. Weitere Informationen zu Parametern finden Sie unter Dateiein-/Ausgabe und Zeichenfolgen auf Seite 71. Das hier verwendete Beispiel enthält nur eine Parameter-Anweisung (statement).

```
PARAMETER P2
```

Lokale Variablen

Variablen, die nur innerhalb eines Makros existieren, werden auch innerhalb des Makros als lokale Variablen definiert. Dazu gehören normalerweise alle Variablen im Hauptteil oder im Teil für die Benutzereingabe. Später wird noch ausführlicher auf die Verwendung lokaler Variablen eingegangen. Das obige Beispielmakro enthält folgende lokale Variablen:

```
LOCAL P1
```

Benutzereingabe

Variablen, die dem System beim Kompilieren noch nicht bekannt sind, müssen ihm beim Ausführen des Makros durch eine Eingabe des Benutzers mitgeteilt werden. Im folgenden Beispiel wird der Wert für `P1` vom Benutzer definiert:

```
  READ 'Indicate center of circle' P1
```

Eingabeaufforderung vom System:

```
Indicate center of circle
```

Der daraufhin vom Benutzer eingegebene Wert wird der Variablen `P1` zugeordnet. Der Benutzer kann die `x,y`-Koordinaten des Punkts eingeben oder den Punkt auswählen. In beiden Fällen werden die Koordinaten der Variable `P1` zugewiesen.

Lokale Variablen müssen am Anfang des Makros definiert werden. Dies kann jedoch nach Fertigstellung des restlichen Makros erfolgen.

Hauptteil eines Makros

Der Hauptteil eines Makros enthält den ausführbaren Code. In diesem Teil des Makros wird also die eigentliche Arbeit verrichtet. Jede Zeile beginnt hier mit einem Befehl oder einer Funktion gefolgt von verschiedenen Parametern, Ausdrücken oder Operatoren. Diese Fachbegriffe werden später noch genauer behandelt. Der Hauptteil des Beispielmakros sieht folgendermaßen aus:

```
CIRCLE CENTER P1 P2 END
```

END_DEFINE

Mit dieser Anweisung wird ein Makro beendet.

INPUT

Wenn Sie den `INPUT`-Befehl in einem Makro verwenden, muss er mit dem Kennzeichner `IMMEDIATE` verwendet werden, beispielsweise:

```
DEFINE xyz
  LINE 0,0 1,1 END
  INPUT IMMEDIATE 'filename'
  ....
  ....
END_DEFINE
```

Minimalmakros

Ein Makro kann aus sechs Bestandteilen bestehen. Es gibt auch Makros mit weniger als sechs Bestandteilen. Wenn das Makro keine Argumente akzeptiert, ist `PARAMETER` nicht vorhanden. Wenn ein Makro beispielsweise keine Variablen verwendet und auch keine Benutzereingaben benötigt, entfallen die betreffenden Komponenten. Unter Umständen gibt es keine Benutzereingaben.

Ein Makro muss jedoch mindestens folgende drei Bestandteile aufweisen:

- DEFINE
- Hauptteil
- END_DEFINE

Das Makro am Anfang dieses Kapitels ist ein sogenanntes Minimalmakro (ein Makro mit nur drei Komponenten). Hier ist es erneut dargestellt:

```
DEFINE Quit
{##############################################}
{##  This macro stores your current           ##}
{##  drawing in 'filename', then ends         ##}
{##  your ME-CAD session.                     ##}
{##############################################}
  STORE ALL DEL_OLD 'filename'
  EXIT CONFIRM
END_DEFINE
```

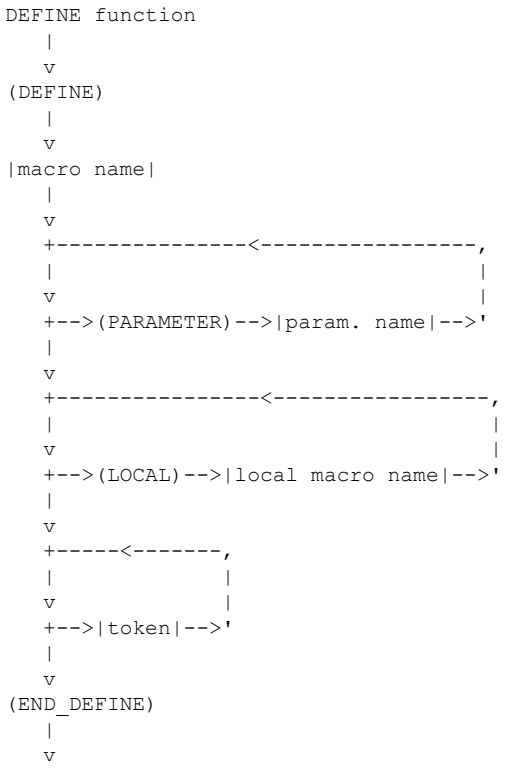
Anweisungen sollten wegen der Übersichtlichkeit immer in separate Zeilen geschrieben werden. Das System könnte das Makro auch abarbeiten, wenn es in folgender Form vorliegen würde:

```
DEFINE Quit STORE ALL DEL_OLD 'filename' EXIT CONFIRM END_DEFINE
```

Syntaxdiagramme

In diesem Zusammenhang soll auch kurz auf die Syntaxdiagramme eingegangen werden. Anhand dieser Diagramme wird deutlich, warum es dem Rechner egal ist, ob die Anweisungen in separaten Zeilen oder fortlaufend in einer Zeile angeordnet sind.

Wenn Sie sich das Syntaxdiagramm für DEFINE ansehen, wird klar, warum der Computer das Makro versteht. Die Syntaxdiagramme für alle Befehle und Funktionen sind in der Online-Hilfe enthalten. Das Diagramm für DEFINE finden Sie unten:



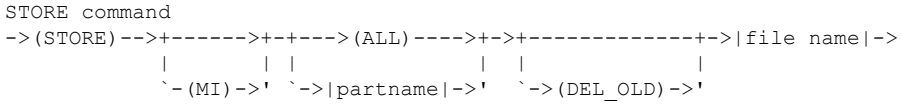
Nach der DEFINE-Anweisung erwartet der Computer Folgendes:

- Schlüsselwort PARAMETER,
- Schlüsselwort LOCAL oder
- Eine Zeichenfolge (Token)

Wenn eines dieser obengenannten Merkmale vorhanden ist, geht das System zur nächsten Anweisung über. Andernfalls wird die Verarbeitung des Makros beendet.

Unter TOKEN in der Online-Hilfe wird dargelegt, dass ein Befehl ein zulässiges Token ist.

Sehen Sie sich nun das Syntaxdiagramm für STORE an:



STORE ist ein Befehl, d.h. STORE kann als Token verwendet werden.

Nach STORE muss entweder das Schlüsselwort MI, das Schlüsselwort ALL oder der Name eines Teils angegeben werden. Wir verwenden ALL.

Nach ALL muss DEL_OLD oder ein Dateiname folgen. Wir verwenden DEL_OLD und dann einen Dateinamen in einfachen Anführungszeichen.

Sie können überprüfen, ob CONFIRM hinter EXIT steht.

Versuchen Sie, das Makro ohne ALL zu verwenden:

```
DEFINE Quit STORE DEL_OLD 'workfile' EXIT CONFIRM END_DEFINE
```

Daraufhin erscheint eine Aufforderung zur Eingabe eines Parameters oder Teils.

```
Enter option or 'part_name'
```

Der Computer ist verwirrt, da er `MI`, `ALL` oder einen Teilenamen erwartet.

Der übersichtliche Aufbau von Makros (jede Anweisung in einer separaten Zeile) hilft also in erster Linie dem Benutzer selbst. Das System interpretiert die Anweisungen nach festen Algorithmen wie beispielsweise Syntaxdiagrammen.

Erläuterungen zu lokalen Variablen

Lokale Variablen sind nur im aktuellen Makro oder in Makros, die vom aktuellen Makro aus aufgerufen werden, einsetzbar. Variablen, die nicht als lokal definiert sind, sind automatisch global. Globale Variablen sind in allen Makros sichtbar. Grundsätzlich sollten alle Variablen als lokale Variablen definiert werden. Globale Variablen können unerwünschte Auswirkungen haben und sollten deshalb vermieden werden.

Zum besseren Verständnis des Unterschieds zwischen lokalen und globalen Variablen sollen nun die folgenden drei Makros in eine Datei eingegeben werden. Diese Makros sind sich sehr ähnlich.Deswegen ist es sinnvoll, zunächst das erste Makro einzutippen, dieses danach zweimal als Block zu kopieren und anschließend die erforderlichen Änderungen in den beiden kopierten Blöcken vorzunehmen. Sie können Kommentare weglassen.

```
DEFINE Outer_macro
{##############################################}
{##   This macro shows the use of            ##}
{##   global variables                       ##}
{##############################################}
  LET X 5                   {initialize the variable "X"}
  DISPLAY_NO_WAIT ('outer X = '+STR(X))
                            {display a message on the
                             command line }
  WAIT 3                    {allow time to read the message}
  Middle_macro              {call another macro }
  DISPLAY_NO_WAIT ('outer X = '+STR(X))
  WAIT 3
END_DEFINE
DEFINE Middle_macro
  DISPLAY_NO_WAIT ('middle X = '+STR(X))
  WAIT 3
  Inner_macro
  DISPLAY_NO_WAIT ('middle X = '+STR(X))
  WAIT 3
END_DEFINE
DEFINE Inner_macro
  DISPLAY_NO_WAIT ('inner X = '+STR(X))
  WAIT 3
  LET X 20
  DISPLAY_NO_WAIT ('inner X = '+STR(X))
  WAIT 3
END_DEFINE
```

Kommentare sind durch geschweifte Klammern gekennzeichnet. Dies bedeutet, dass beim Kompilieren alles, was zwischen den geschweiften Klammern steht, nicht berücksichtigt wird. Kommentare dienen dazu, sich selbst und anderen

Benutzern die Funktionen des Makros zu erklären. Wenn das Makro sehr komplex ist, können Kommentare Ihnen helfen, es besser zu verstehen – besonders sechs Monate später!

Ihre Kommentare sollten den Zweck des Befehls erläutern. Der Benutzer kann dann anhand des zugehörigen Syntaxdiagramms näheres zu diesem Befehl nachlesen. Ein Kommentar wie beispielsweise:

```
{allow time to read the message}
```

ist aussagekräftiger als:

```
{wait approximately 3 seconds}
```

Hinweis

Kommentare können nicht verschachtelt werden. Aus diesem Grund ist es nicht möglich, einen Code-Abschnitt, der bereits Kommentare enthält, bei der Fehlersuche durch weitere Kommentare vom Code auszuschließen.

Sie sehen, dass `outer_macro` `middle_macro` aufruft, was wiederum `inner_macro` aufruft. Dieser Vorgang wird Verschachtelung genannt.

Keines der Makros verwendet lokale Variablen. Die Variable `X` ist also eine globale Variable. Dies bedeutet, dass `X` in allen drei Makros verfügbar ist.

Zunächst muss die Datei mit den drei Makros geladen und anschließend Folgendes in die Befehlszeile eingegeben werden:

```
Outer_macro
```

In der Befehlszeile sollte nun folgendes (in der genannten Reihenfolge) erscheinen:

```
'outer X = 5'
'middle X = 5'
'inner X = 5'
'inner X = 20'
'middle X = 20'
'outer X = 20'
Enter command
```

Bevor näher auf die Verwendung der globalen Variablen eingegangen wird, sollen zunächst einige Makro-Anweisungen kurz erläutert werden.

`DISPLAY_NO_WAIT` wird verwendet, um eine Meldung in der Befehlszeile anzuzeigen. Nachdem die Meldung angezeigt wird, ist keine Benutzeraktion erforderlich. Diese Funktion unterscheidet sich von `DISPLAY`, wo der Benutzer eine Taste drücken muss, damit das Makro fortgesetzt wird.

`DISPLAY_NO_WAIT` druckt zunächst:

```
outer X =
```

Was bewirkt `STR(X)`? `X` ist eine `NUMBER` mit einer internen Rechnerdarstellung, die "set" oder "cleared" verwendet. Nur ASCII-Zeichen können mit der `DISPLAY_NO_WAIT`-Anweisung angezeigt werden. `STR(Age)` konvertiert die interne Darstellung von `X` in eine äquivalente ASCII-Zeichenfolge, sodass eine Anzeige möglich ist.

Bei Verwendung mit Zeichenfolgen verkettet + die Zeichenfolgen. Beispiel: `String1` ist `'to'`, `String2` ist `'get'` und `String3` ist `'her'`. In diesem Fall ist `String1+String2+String3` `together`. Leerzeichen innerhalb der Hochkommata werden ebenfalls angezeigt.

Bei `WAIT 3` wartet das System ca. drei Sekunden nach der Anzeige der Meldung. So bleibt dem Benutzer genügend Zeit, um sie zu lesen.

Die Anweisung `Middle_macro` ruft ein anderes Makro auf. Dieses Makro muss dem System beim Aufruf bereits bekannt sein. Da sich die drei Beispielmakros alle in derselben Datei befinden, ist auch `Middle_macro` jederzeit verfügbar.

Jetzt soll näher auf die globalen Variablen eingegangen werden. Der Wert von `X`, der `5` entspricht (definiert in `Outer_macro`), war den beiden anderen Makros bekannt. In `Inner_macro` wurde der Wert von `X` in `20` geändert. Dieser geänderte Wert ist ebenfalls in den beiden anderen Makros bekannt.

Jetzt soll dem mittleren Makro nach der Anweisung `DEFINE` folgende Zeile hinzugefügt werden:

```
DEFINE Middle_macro
  LOCAL X                                  {added}
  DISPLAY_NO_WAIT ('middle X = '+STR(X))
  WAIT 3
  Inner_macro
  DISPLAY_NO_WAIT ('middle X = '+STR(X))
  WAIT 3
END_DEFINE
```

Rufen Sie die Datei durch Eingabe von `Outer_macro` in der Befehlszeile auf. Daraufhin erscheint folgende Ausgabe:

```
'outer X = 5'
***The macro X is not defined
```

Das System gibt an, dass es den Wert von `X` nicht kennt. Dies liegt daran, dass `X` jetzt eine lokale Variable in `Middle_macro` ist. Andere Variablen mit demselben Namen in einem äußeren Makro sind `Middle_macro` nicht bekannt.

Um das Makro funktionsfähig zu machen, muss folgende Zeile zum mittleren Makro hinzugefügt werden:

```
DEFINE Middle_macro
  LOCAL X
  LET X 10                                 {added}
  DISPLAY_NO_WAIT ('middle X = '+STR(X))
  WAIT 3
  Inner_macro
  DISPLAY_NO_WAIT ('middle X = '+STR(X))
  WAIT 3
END_DEFINE
```

Rufen Sie die Datei durch Eingabe von `Outer_macro` in der Befehlszeile auf.

Daraufhin erscheint folgende Ausgabe:

```
'outer X = 5'
'middle X = 10'
'inner X = 10'
'inner X = 20'
'middle X = 20'
'outer X = 5'
Enter command
```

Es geschieht Folgendes:

- Die Anweisung `LOCAL X` in `Middle_macro` macht `Middle_macro` sicher. Werte von `X` können dies nicht durchdringen. Dies bedeutet, dass Werte von `X` nicht nach innen von `Outer_macro` zu `Middle_macro` übergeben werden können, oder nach außen von `Middle_macro` zu `Outer_macro`. `X` kann jedoch nach innen zu jedem Makro übergeben werden, das von `Middle_macro` aufgerufen wird, es sei denn diese inneren Makros sind auch vor `X` geschützt.
- Wenn `X` gleich `10` in `Middle_macro` ist, kann dieser Wert an `Inner_macro` übergeben werden.
- Wenn `X` gleich `20` in `inner_macro` ist, kann dieser Wert nach außen an `Middle_macro` übergeben werden.
- Da `X` eine lokale Variable in `Middle_macro` ist, kennt `Outer_macro` den Wert von `X` von `Middle_macro` nicht. Der Wert von `X` ist der Wert, der ursprünglich in `Outer_macro` definiert wurde (`5`).

Schließlich soll `X` eine lokale Variable in allen drei Makros sein:

```
DEFINE Outer_macro
  LOCAL X                                {added}
  LET X 5
  DISPLAY_NO_WAIT ('outer X = '+STR(X))
  WAIT 3
  Middle_macro
  DISPLAY_NO_WAIT ('outer X = '+STR(X))
  WAIT 3
END_DEFINE
DEFINE Middle_macro
  LOCAL X
  LET X 10
  DISPLAY_NO_WAIT ('middle X = '+STR(X))
  WAIT 3
  Inner_macro
  DISPLAY_NO_WAIT ('middle X = '+STR(X))
  WAIT 3
END_DEFINE
DEFINE Inner_macro
  LOCAL X                                        {added}
  LET X 15                                       {added}
  DISPLAY_NO_WAIT ('inner X = '+STR(X))
  WAIT 3
  LET X 20
  DISPLAY_NO_WAIT ('inner X = '+STR(X))
  WAIT 3
END_DEFINE
```

Rufen Sie die Datei durch Eingabe von `Outer_macro` in der Befehlszeile auf. Daraufhin erscheint folgende Ausgabe:

```
'outer X = 5'
'middle X = 10'
'inner X = 15'
'inner X = 20'
'middle X = 10'
'outer X = 5'
Enter command
```

Verstehen Sie die Ausgabe?

Aufgaben lokaler Variablen

Ein wichtiger Grund für die Verwendung von lokalen Variablen ist, unsere Makros so sicher wie möglich zu machen, um unerwünschte Nebeneffekte zu vermeiden. Diese Nebeneffekte treten auf, wenn eine Variable versehentlich durch eine Variable mit demselben Namen in einem inneren Makro beeinflusst wird.

Angenommen, in einer Makrobibliothek ist bereits ein nützliches Makro vorhanden, das Ihnen einen großen Code-Block in Ihrem eigenen Makro erspart. Aus diesem Grund wollen Sie das vorhandene Makro von Ihrem eigenen Makro aus aufrufen, ohne dabei prüfen zu müssen, ob das aufgerufene Makro die gleichen Variablennamen verwendet, wie Ihr eigenes Makro. Bei dem aufgerufenen Makro kann es sich bereits um ein verschachteltes Makro handeln, was bedeutet, dass dieses Makro seinerseits ein oder mehrere andere Makros aufruft. Auch bei diesen Makros sollten Sie sicher sein, dass keine Konflikte mit den Variablennamen auftreten. Wenn der Schreiber dieser Makros lokale Variablen verwendet hat, können Sie sich darauf verlassen, dass Sie in dieser Hinsicht keine Probleme bekommen werden.

Das folgende Beispiel zeigt ein Makro, dass unter Umständen nicht immer so arbeitet, wie es der Programmierer vorgesehen hat.

```
DEFINE Outer_loop
  LOCAL X
  LET X 1            {initialize the loop counter}
  WHILE (X < 3)      {while X is less than 3,
                      execute the code up to the next
                      END_WHILE statement}
    DISPLAY_NO_WAIT ('outer X = '+STR(X))
    WAIT 3
    Inner_loop
    LET X (X+1)      {increment the loop counter}
  END_WHILE
END_DEFINE
DEFINE Inner_loop
  LET X 1
  WHILE (X < 4)
    DISPLAY_NO_WAIT ('inner X = '+STR(X))
    WAIT 3
    LET X (X+1)
  END_WHILE
END_DEFINE
```

Erzeugt das Makro folgende Ausgabe? Wenn nicht, warum nicht?

```
'outer X = 1'
'inner X = 1'
'inner X = 2'
'inner X = 3'
'outer X = 2'
'inner X = 1'
'inner X = 2'
'inner X = 3'
Enter command
```

Nun, da Sie Experte für lokale Variablen sind, kennen Sie die Ausgabe.

Da Variablen wie `X`,`I` und `N` so gängig sind wie Zähler in Schleifen, ist es wichtig, dass der Programmierer diese Werte vor anderen Makros schützt.

Variablen deklarieren

In den meisten Programmiersprachen müssen Variablen deklariert werden. Beispiel: In Pascal gibt es Variablen wie integer, real und char. In C gibt es int, float, char usw.

Deklarieren bedeutet in diesem Zusammenhang, dass im Speicher eine bestimmte Adresse und Größe für eine Variable reserviert ist.

In Makros werden Variablen normalerweise nicht deklariert.

Ein dem Deklarieren von Variablen in etwa entsprechender Vorgang ist das Lesen einer Variablen mit der Anweisung READ. Diese Anweisung gibt eine Eingabeaufforderung in die Befehlszeile aus. Dieser Abschnitt enthält einige Beispiele.

```
DEFINE Read_test
  READ STRING 'Enter file name' File
                       {The string 'Enter file name'
                        appears on the command line. You must
                        enter a string is single quotes. The
                        string is assigned to the variable,
                        "File"}
  READ NUMBER 'Enter your age' Age
  READ PNT 'Enter a point' P1
  READ PNT 'Digitize a point' RUBBER_LINE P1 P2
  DISPLAY_NO_WAIT ('File = '+File)
  WAIT 3
  DISPLAY_NO_WAIT ('Age = '+STR(Age))
                       {"Age" is converted to an ASCII
                        string, then printed after 'Age = ' }
  WAIT 3
  DISPLAY_NO_WAIT ('P1 = '+STR(P1))
  WAIT 3
  DISPLAY_NO_WAIT ('P2 = '+STR(P2))
  WAIT 3
END_DEFINE
```

Hier wurde auf lokale Variablen bewusst verzichtet, um den Schreibaufwand zu minimieren.

Beachten Sie die erste READ-Anweisung:

```
READ STRING 'Enter file name' File
```

Beim Ausführen des Makros (probieren Sie es aus) erscheint folgende Eingabeaufforderung in der Befehlszeile:

```
Enter file name
```

Daraufhin muss der Benutzer eine Zeichenfolge in einfachen Anführungszeichen eingeben. Falls die Hochkommata vergessen werden, erscheint die Eingabeaufforderung erneut. Die Angabe des Variablentyps unmittelbar nach der READ-Anweisung bietet folgenden Vorteil: Bei falschem Eingabetyp führt das System diese Makrozeile erneut aus, bis die Eingabe dem angegebenen Typ entspricht.

Bei der zweiten READ-Anweisung wird die Eingabeaufforderung erneut angezeigt, wenn eine Zahl ohne einfache Anführungszeichen bzw. ein Punkt anstelle einer Zahl eingegeben wird. Bei der Eingabe sind also nur Werte gültig, die dem Typ NUMBER entsprechen.

Es ist möglich, READ-Anweisungen zu verwenden, ohne den Typ anzugeben, wie STRING oder NUMBER. In diesem Fall akzeptiert das System allerdings alle für die READ-Anweisung eingegebenen Daten (also auch ungültige Eingaben). Das System reagiert erst, wenn das Makro die Daten verwendet. In unserem Makro entspricht dies dem Zeitpunkt der DISPLAY_NO_WAIT-Anweisung. Versuchen Sie, eine der Typanweisungen nach einer READ-Anweisung zu entfernen. Führen Sie das Makro aus, und stellen Sie dann absichtlich einige falsche Daten bereit.

Ist Ihnen aufgefallen, dass wir STR(Age) und STR(Point) und nicht STR(File) verwendet haben? STR(Age) konvertiert die interne Darstellung von Age in ein ASCII-Äquivalent, sodass eine Anzeige möglich ist. Mit STR(Point) wird dasselbe erreicht. Bei File liegt der Wert jedoch durch die Benutzereingabe bereits als ASCII-Zeichenfolge vor, sodass keine Umwandlung mehr erforderlich ist.

Das folgende Beispiel zeigt, wie die Ausgabe eines vorhandenen bzw. berechneten Werts am Bildschirm erfolgt. Dabei wird der Parameter der Option DEFAULT ausgewertet und das Ergebnis in der Eingabezeile angezeigt. Wenn Sie die [Return]-Taste drücken, wird der (von Ihnen eventuell editierte) Wert als Antwort auf die READ-Anforderung eingegeben.

```
DEFINE Test
LOCAL Value
LET Value (100/2)
READ NUMBER 'Current value is:' DEFAULT Value Value
DISPLAY (STR Value)
END_DEFINE
```

Verwendung von Steueranweisungen

Makroanweisungen sollen nicht immer in der Reihenfolge ausgeführt werden, in der sie im Makro angezeigt werden. Einige Anweisungen sollen eventuell mehrmals ausgeführt werden. Andere Anweisungen sollen evtl. nur ausgeführt werden, wenn eine bestimmte Bedingung erfüllt wird. Mit Steueranweisungen können Sie die Reihenfolge, wie ein Makro ausgeführt wird, beeinflussen.

WHILE ... END_WHILE

Wir haben eine WHILE-Anweisung im Abschnitt "Aufgaben lokaler Variablen?" verwendet. Ein Teil des Makros aus diesem Abschnitt wird hier noch einmal wiederholt:

```
DEFINE Outer_loop
  LOCAL X
  LET X 1            {initialize the loop counter}
  WHILE (X < 3)      {while X is less than 3,
                      execute the code up to the next
                      END_WHILE statement}
    DISPLAY_NO_WAIT ('outer X = '+STR(X))
    WAIT 3
    Inner_loop
    LET X (X+1)      {increment the loop counter}
  END_WHILE
END_DEFINE
```

So lange X kleiner als 3 ist, wird der Code zwischen WHILE und END_WHILE ausgeführt. Vor der WHILE-Anweisung wird X auf 1 eingestellt und X um 1 erhöht, wenn die Schleife ausgeführt wird. Diese Schleife müsste also zweimal (nicht dreimal!) durchlaufen werden. Aufgrund von unerwünschten Einflüssen durch globale Variablen im aufgerufenen Makro wurde diese Schleife jedoch nur einmal abgearbeitet.

Es wird ein WHILE . . . END_WHILE-Konstrukt verwendet, um die Anzahl der Schleifenausführungen zu steuern. Ein wichtiges Merkmal dieser Konstruktion ist, dass die Schleife unter Umständen auch gar nicht abgearbeitet wird. So führt eine nicht erfüllte Bedingung nach der WHILE-Anweisung dazu, dass die Schleife nicht durchlaufen wird. Später wird das REPEAT . . . UNTIL-Konstrukt behandelt, das eine Schleife generiert, die mindestens einmal ausgeführt werden muss.

LOOP ... EXIT_IF ... END_LOOP

Das WHILE . . . END_WHILE-Konstrukt fragt die Bedingung am Anfang der Schleife ab. Im LOOP . . . EXIT_IF . . . END_LOOP-Konstrukt kann die Bedingungsabfrage zu jedem Punkt in der Schleife erfolgen.

Im folgenden Codefragment wird der Benutzer dazu aufgefordert, einen Wert einzugeben. Gibt er einen unwahren Wert ein, wird die Eingabeaufforderung erneut angezeigt.Dieser Vorgang wiederholt sich so lange, bis der richtige Wert eingegeben wird.

```
LOOP
  READ NUMBER 'Enter a fractional value for the split' Fract
EXIT_IF ((Fract>0)) AND (Fract< 1))
END_LOOP
```

Im Hauptteil der Schleife können beliebig viele EXIT_IF-Anweisungen verwendet werden. Siehe dazu das folgende Beispiel:

```
LOOP
  ...
  ...
EXIT_IF
  ...
  ...
EXIT_IF
  ...
  ...
END_LOOP
```

Wenn keine EXIT_IF-Anweisung verwendet wird, wird das Makro nur angehalten, wenn der Benutzer END eingibt oder einen Befehl ausführt. Beispiel:

```
DEFINE Circ_rad
{Create several circles with the same }
{radius, or several circles passing through the same peripheral}
{point                                                          }
{ local variables here }
READ 'Indicate peripheral point or enter radius or END' P2
LOOP
  READ 'Indicate center of circle' P1
  CIRCLE CENTER P1 P2
  END
END_LOOP
END_DEFINE
```

Was geschieht, wenn wir die erste READ-Anweisung im LOOP . . . END_LOOP-Konstrukt platzieren? Das geänderte Makro sieht dann wie folgt aus:

```
DEFINE Circ_rad
LOOP
  READ 'Indicate peripheral point or enter radius or END' P2
  READ 'Indicate center of circle' P1
  CIRCLE CENTER P1 P2
  END
END_LOOP
END_DEFINE
```

Das LOOP . . . EXIT_IF . . . END_LOOP-Konstrukt ist sehr nützlich, da das Durchlaufen der Schleife an einem beliebigen Punkt (nicht nur am Anfang oder Ende der Schleife) beendet werden kann. Darüber hinaus kann der Benutzer das Makro durch die Eingabe von END beenden.

Durch die Eingabe von END wird nicht nur die Schleife, sondern das gesamte Makro beendet. Weitere Anweisungen, die nach END_LOOP folgen, werden also nicht mehr ausgeführt.

REPEAT ... UNTIL

Dieses Konstrukt ähnelt dem LOOP . . . EXIT_IF . . . END_LOOP-Konstrukt, wobei die EXIT_IF-Anweisung direkt vor END_LOOP platziert ist. Da die Bedingung bei UNTIL erst am Ende der Schleife abgefragt wird, muss diese Schleife immer mindestens einmal durchlaufen werden.

IF ... ELSE_IF ... ELSE ... END_IF

Die IF-Anweisungen werden verwendet, um Entscheidungen zu treffen. Je nach Entscheidung werden bestimmte Teile des Makros ausgeführt und andere nicht.

Im folgenden Codefragment wird der Benutzer gefragt, ob eine Hilfslinie waagerecht, senkrecht oder normal zu einer vorhandenen Linie gezeichnet werden soll.

```
READ "ENTER 'H' FOR HORIZ, OR 'V' FOR VERT, OR 'P' FOR PERP" Q
IF (Q='H')
  C_LINE HORIZONTAL P2      {if Q='H', only this statement}
                                          {is executed}
ELSE_IF (Q='V')
    C_LINE VERTICAL P2   {if Q<>'H', but Q='V', only these}
    LET X 3                   {two statements are executed}
ELSE
    C_LINE PERPENDICULAR P1 P2     {if Q<>'H', and Q<>'V',}
                         {but Q='P', only this statement is}
                                             {executed}
END_IF
```

Die Bedingungen werden von oben nach unten (in der Form einer "Leiter") abgefragt. Sobald eine wahre Bedingung vorliegt, werden die Anweisungen, die mit der Bedingung verknüpft sind, ausgeführt und der Rest der "Leiter" nicht berücksichtigt.

Sie können beliebig viele ELSE_IF-Anweisungen verwenden. Sie können nur eine ELSE-Anweisung verwenden. Diese muss am Ende stehen. Die letzte ELSE-Anweisung ist die Standardbedingung. Mit anderen Worten: Wenn alle anderen Bedingungsabfragen fehlschlagen, wird die letzte ELSE-Anweisung ausgeführt.

Wenn die letzte ELSE-Anweisung nicht vorhanden ist und die anderen Bedingungen nicht erfüllt werden, geschieht nichts. Gelegentlich wird die letzte ELSE-Anweisung bei der defensiven Programmierung verwendet, um Fehler zu beheben.

Die absolute minimale IF-Anweisung weist weder ELSE_IF noch ELSE auf. Beispiel:

```
IF (X = 4)
  LET P2 P5
END_IF
```

IF-Anweisungen können verschachtelt sein. Jede IF-Anweisung muss jedoch ein eigenes END_IF-Konstrukt aufweisen. Zur Veranschaulichung wird das obige Beispiel erweitert:

```
IF (X < 6)
  IF (X = 4)
    LET P2 P5
  END_IF
  LINE TWO_PTS P1 P2 END
END_IF
```

Verwendung von Klammern

Der Programmierer muss sich bei jeder Programmiersprache mit der Verwendung von Klammern vertraut machen. Die folgenden Richtlinien sollen ihm dabei helfen, die Klammern in Makros richtig zu setzen.

Boolesche Ausdrücke

Boolesche Ausdrücke sind immer mit Klammern zu verwenden.

Ein Boolescher Ausdruck (auch Wahrheitsfunktion genannt) kann den Wert "wahr" oder "falsch" haben.

Boolesche Ausdrücke werden im Testteil der folgenden vier Hauptkonstruktionen verwendet:

```
IF (boolean expression) ... END_IF
LOOP...EXIT_IF (boolean expression) ... END_LOOP
REPEAT ... UNTIL (boolean expression)
WHILE (boolean expression) ... END_WHILE
```

Arithmetische, algebraische and trigonometrische Ausdrücke

Ausdrücke, die als Token eingesetzt werden, sind immer mit Klammern zu verwenden.

Beispiele:

- X + Y
- X - 5
- X - 5.147
- X * 312
- X/312

- X DIV Y
- POINT1 + 4.29, 3.42
- SIN 30
- SIN (Angle3 + 30)
- SQRT 16.238
- SQRT (X + 16.238)
- LEN 'Guten Tag!'

In der Online-Hilfe befindet sich eine vollständige Auflistung aller zulässigen Zeichenfolgen. Wenn für eine im Syntaxdiagramm erforderliche Zeichenfolge ein obengenannter Ausdruck verwendet wird, muss dieser in Klammern eingegeben werden. Beispiel:

```
LET Radius1 (X + Y)
LET Radius1 ((X + Y) * COS 60 )
LET Point2 Point1
LET Point2 (Point1 + 4.29,3.42)
DISPLAY_NO_WAIT Point2
DISPLAY_NO_WAIT (Point1 + 4.29,3.42)
```

Die obigen Beispiele verdeutlichen einige interessante Punkte. Sehen wir uns diese Beispiele einzeln an.

Fangen wir mit dem Syntaxdiagramm für `LET` im Syntaxkapitel an. Sie sehen, dass nach `LET` der Name einer Variable und ein Token folgt. Im ersten Beispiel ist `X + Y` ein Ausdruck, der als Token verwendet wird. Das heißt, er muss in Klammern eingeschlossen werden.

Sehen wir uns nun das zweite Beispiel an. Hier ist der Ausdruck

```
(X + Y) * COS 60
```

In diesem Ausdruck sind die Klammern erforderlich, um die richtige Reihenfolge bei der Berechnung zu gewährleisten. Wenn dieser Ausdruck als Zeichenfolge verwendet wird, ist ein zweites paar Klammern erforderlich.

Das dritte Beispiel zeigt eine einfache Zeichenfolge, bei der keine Klammern erforderlich sind.

```
LET Point2 Point1
```

Wenn diese einfache Zeichenfolge jedoch so verändert wird, dass sich daraus ein Ausdruck ergibt, müssen Klammern gesetzt werden.

```
LET Point2 (Point1 + 4.29,3.42)
```

Auch die folgende Anweisung ist gültig:

```
LET Point2 (Point1)
```

Im Syntaxdiagramm für `DISPLAY_NO_WAIT` im Syntaxkapitel wird deutlich, dass nach `DISPLAY_NO_WAIT` ein Token folgen muss. Die Beispiele für `DISPLAY_NO_WAIT` sind mit den letzten beiden Beispielen für `LET` vergleichbar.

Aus diesen Beispielen wird deutlich, dass für die Anweisung `LET` in einigen Fällen Klammern für das Token erforderlich sind, in anderen Fällen dagegen nicht. Wenn Sie noch nicht viel Erfahrung im Schreiben von Makros haben, sollten Sie in `LET`-Anweisungen grundsätzlich immer Klammern verwenden.

Ablaufverfolgung zum Beheben von Fehlern

Verfolgen ist ein nützliches Tool für die Fehlerbehebung. **Verfolgen** wird im folgenden Beispiel jedoch dazu benutzt, darzustellen, wie Boolesche und andere Ausdrücke vom System berechnet werden.

Die Ablaufverfolgung soll für das folgende Makro ausgeführt werden:

```
DEFINE Parenth
  LET P1 (0,0)                {parentheses not necessary}
  LET P2 (10,0)               {parentheses not necessary}
  LET X (1)                   {parentheses not necessary}
  WHILE ( X < 5 )             {parentheses necessary}
    LET P1 (P1 + 0,10)        {parentheses necessary}
    LET P2 (P2 + 0,10)        {parentheses necessary}
    LINE TWO_PTS P1 P2 END
    LET X ( X + 1)            {parentheses necessary}
  END_WHILE
END_DEFINE
```

Dieses Makro zeichnet vier Linien. Wenn `X 1` entspricht, wird eine Linie von `0,0` bis `10,0` gezogen. Wenn `X 2` entspricht, wird eine Linie von `0,10` bis `10,10` gezogen usw.

Im Kommentar ist vermerkt, wo Klammern erforderlich sind und wo nicht.

Dieses Makro zeigt ein wichtiges Prinzip. Sehen Sie sich das Syntaxdiagramm für `LINE TWO_PTS` im *Creo Elements/Direct Drafting Programmierungsreferenz-Handbuch* an. In einem Makro kann eine Endlosschleife nur über die Anweisung `END` beendet werden. Dies ist eine allgemeine Regel: Wenn ein Syntaxdiagramm mit einer rekursiven Schleife endet, verwenden Sie eine `END`-Anweisung, um eine Schleife zu beenden.

Sehen Sie sich das Diagramm für `LINETYPE` an. Dieses Diagramm endet nicht mit einer rekursiven Schleife. In einem Makro ist keine `END`-Anweisung erforderlich.

Nun verwenden wir **Verfolgen** in unserem Makro und speichern die Ergebnisse in einer Datei namens `trace`.

Ändern Sie die zweite Linie in `parenth` vor der Verwendung von **Verfolgen** von

```
LET P1 (0,0)
```

in

```
LET P1 0,0
```

Geben Sie folgende Befehle in die Eingabezeile ein, und drücken Sie nach jedem Befehl die EINGABETASTE:

```
input 'parenth'
```

```
trace
```

```
parenth
```

Wenn das Macro endet, deaktivieren Sie **Verfolgen**:

1. Klicken Sie auf **Verschiedenes**, und klicken Sie anschließend in der Gruppe **System** auf **Verfolgen**.
2. Klicken sie auf **Aus**.

   in der Statusleiste gibt an, dass **Verfolgen** nicht aktiviert ist.

Hinweis

Wenn Ihre `trace`-Datei leer oder unvollständig ist, haben Sie evtl. vergessen, **Verfolgen** zu deaktivieren.

Wenn Sie die Ergebnisse einer **Verfolgen**-Operation an eine vorhandene Datei anhängen möchten, gehen Sie wie folgt vor:

1. Klicken Sie auf **Verschiedenes**, und klicken Sie anschließend in der Gruppe **System** auf **Verfolgen**.
2. Klicken Sie auf **Ein**.
3. Klicken Sie auf **Anfügen**.

   in der Statusleiste gibt an, dass **Verfolgen** aktiviert ist.

Wenn Sie eine bereinigte Datei für jede Ausführung benötigen:

1. Klicken Sie auf **Verschiedenes**, und klicken Sie anschließend in der Gruppe **System** auf **Verfolgen**.
2. Klicken Sie auf **Ein**.
3. Klicken Sie auf **Neu**.

   in der Statusleiste gibt an, dass **Verfolgen** aktiviert ist.

So zeigen Sie die Ergebnisse des **Verfolgen**-Vorgangs an:

1. Klicken Sie auf **Verschiedenes**, und klicken Sie anschließend in der Gruppe **System** auf **Verfolgen**.
2. Klicken Sie auf **Editieren**.

   Die Ergebnisse des **Verfolgen**-Vorgangs werden im Standardtexteditor angezeigt.

Dies ist die `trace`-Datei für `parenth`:

```
Parenth
LET P1 0,0
```

```
LET P2 ( 10,0 ) 10,0
LET X ( 1 ) 1
WHILE ( X 1 < 5 ) 1
LET P1 ( P1 0,0 + 0,10 ) 0,10
LET P2 ( P2 10,0 + 0,10 ) 10,10
LINE TWO_PTS P1 0,10 P2 10,10
END
LET X ( X 1 + 1 ) 2
END_WHILE ( X 2 < 5 ) 1
LET P1 ( P1 0,10 + 0,10 ) 0,20
LET P2 ( P2 10,10 + 0,10 ) 10,20
LINE TWO_PTS P1 0,20 P2 10,20
END
LET X ( X 2 + 1 ) 3
END_WHILE ( X 3 < 5 ) 1
LET P1 ( P1 0,20 + 0,10 ) 0,30
LET P2 ( P2 10,20 + 0,10 ) 10,30
LINE TWO_PTS P1 0,30 P2 10,30
END
LET X ( X 3 + 1 ) 4
END_WHILE ( X 4 < 5 ) 1
LET P1 ( P1 0,30 + 0,10 ) 0,40
LET P2 ( P2 10,30 + 0,10 ) 10,40
LINE TWO_PTS P1 0,40 P2 10,40
END
LET X ( X 4 + 1 ) 5
END_WHILE ( X 5 < 5 ) 0
TRACE
```

Die erste Zeile von `trace`

```
LET P1 0,0
```

entspricht der in unserem Makro.

Die zweite Zeile ist unterschiedlich. Darin wurde der Klammerausdruck berechnet. Das Ergebnis `10,0` wird direkt nach dem Ausdruck angezeigt.

Als nächstes soll der folgende Boolesche Ausdruck näher erläutert werden:

```
WHILE ( X 1 < 5 ) 1
```

Der aktuelle Wert von `X` ist `1`. Dies wird direkt nach `X` angezeigt. Der Boolesche Ausdruck `(1 < 5)` ist wahr, sodass der Wert `1` unmittelbar nach dem Ausdruck angezeigt wird.

Beachten Sie, dass `WHILE` nicht erneut in `trace` angezeigt wird, der Wert des Booleschen Ausdrucks jedoch nach `END_WHILE`. Beachten Sie den letzten Booleschen Ausdruck nach dem letzten `END_WHILE`. Der aktuelle Wert von `X` ist `5`. Der Boolesche Ausdruck ist `(5 < 5)` (falsch), sodass der Wert des Ausdrucks `0` ist.

Die folgende Version des Makro ist funktionsfähig:

```
DEFINE Parenth_2
  LET P1 (0,0)                {parentheses not necessary}
  LET P2 (10,0)               {parentheses not necessary}
  WHILE (1)                   {parentheses necessary}
    LET P1 (P1 + 0,10)        {parentheses necessary}
    LET P2 (P2 + 0,10)        {parentheses necessary}
    LINE TWO_PTS P1 P2 END
  END_WHILE
END_DEFINE
```

Nun ist der Ausdruck nach `WHILE` immer wahr. Dieses Makro zeichnet solange weiter, bis ein Stromausfall auftritt oder bis die Taste `[Break]` gedrückt wird.

Hier wird noch einmal bestätigt, dass bei einem Booleschen Ausdruck immer Klammern verwendet werden müssen (auch, wenn nur mit einfachen Token gearbeitet wird.). Wenn Sie das nicht glauben, versuchen Sie, `WHILE (1)` in `WHILE 1` zu ändern.

Ändern Sie `WHILE (1)` in `WHILE (TRUE)`. Es funktioniert weiterhin.

Das folgende Makro zeigt, dass ein Boolescher Ausdruck immer wahr ist, wenn sein Wert ungleich 0 ist:

```
DEFINE Parenth
  LET P1 0,0
  LET P2 (10,0)
  LET X (5)
  WHILE (X)
    LET P1 (P1 + 0,10)
    LET P2 (P2 + 0,10)
    LINE TWO_PTS (P1) (P2) END
    LET X ( X - 1)
  END_WHILE
  DISPLAY_NO_WAIT 'something has changed'
  WAIT 3
  LET X ( X - 1 )
  WHILE (X)
    LET P1 (P1 + 0,10)
    LET P2 (P2 + 0,10)
    LINE (TWO_PTS) P1 P2 END
    LET X ( X + 1)
  END_WHILE
END_DEFINE
```

Zeilen einrücken

Es wird Ihnen sicherlich aufgefallen sein, dass einige Zeilen im Makro eingerückt sind. Diese Einrückung wird vom Rechner ignoriert und dient lediglich dazu, dem Benutzer das Verständnis für den logischen Aufbau des Makros zu erleichtern.

Es ist allgemein üblich, die auf einen Booleschen Ausdruck folgenden Zeilen einzurücken. Im vorherigen Beispiel werden die Zeilen, die jeder `WHILE`-Anweisung folgen, eingerückt. Die letzte Anweisung des `WHILE`-Konstrukts ist `END_WHILE`. Diese Anweisung wird in derselben Spalte wie `WHILE` angezeigt. Das Ergebnis ist, dass die Schlüsselwörter des Konstrukts visuell wie Markierungen agieren. Es ist sofort ersichtlich, wo jeder Codeabschnitt beginnt und endet. Hierdurch wird die Fehlerbehebung viel einfacher.

Bei folgenden Konstruktionen ist ebenso zu verfahren:

- `IF ... ELSE_IF ... ELSE ... END_IF`
- `LOOP ... EXIT_IF ... END_LOOP`
- `REPEAT ... UNTIL`

Jedes Schlüsselwort eines Konstrukts fängt in derselben Spalte an. Die Anweisungen nach einem Schlüsselwort sind eingerückt.

Beim Verschachteln solcher Konstruktionen sollte in Anlehnung an die jeweiligen Verschachtelungsstufen mit entsprechenden Einrückungsstufen gearbeitet werden.

Defensives Programmieren

Beim Schreiben von Makros können viele Fehler durch sogenanntes defensives Programmieren verhindert werden.Defensives Programmieren heißt, dass beim Programmieren allen möglichen Eventualitäten Rechnung getragen wird. Die nachfolgenden Empfehlungen sollen Ihnen dafür als Anhaltspunkte dienen:

- Verwenden Sie Einrückungen nach einem einheitlichen Muster. Auf diesen Punkt wurde bereits im vorhergehenden Abschnitt eingegangen.
- Verwenden Sie eindeutige Variablennamen. `Radius1` oder `Rad1` ist einfacher zu verstehen als `S`. Wenn Sie versehentlich `TAN(S)` schreiben, wird der Fehler unmittelbar in `TAN(Radius1)` angezeigt.
- Halten Sie den Code so einfach wie möglich. Die beiden folgenden Codefragmente liefern identische Ergebnisse. Welches ist einfacher zu verstehen?

  Fragment 1:
  ```
  LET P3 (P1 - (PNT_RA O.5*(D2 - D1) ANG(P2 -P1)))
  ```

  Fragment 2:
  ```
  LET Shoulder (O.5*(D2 - D1)
  LET Angle (ANG(P2 - P1)
  LET Vect_S (PNT_RA Shoulder Angle)
  LET P3 (P1 - Vect_S)
  ```

  In welchem der Codefragmente wäre ein Fehler am leichtesten zu beheben?
- Halten Sie sich nicht schon beim Schreiben des Makros zu sehr damit auf, den Code so effektiv wie möglich zu gestalten. Konzentrieren Sie sich erst nach Fertigstellung des Makros auf die Stellen, an denen Verbesserungen der Geschwindigkeit möglich erscheinen.
- Vermeiden Sie lange Makros. Verwenden Sie mehrere kürzere Makros. Fehler für jedes Makro können separat behoben werden.
- Versuchen Sie, die Möglichkeiten fehlerhafter Benutzereingaben so weit wie möglich auszuschließen. Wir haben uns bei der Erörterung der `IF`-Anweisungen folgendes Codefragment angesehen:
  ```
  READ "ENTER 'H' FOR HORIZ, OR 'V' FOR VERT, OR 'P' FOR PERP"
  IF (Q='H')
      C_LINE HORIZONTAL P2      {if Q='H', only this statement}
                                                {is executed}
  ELSE_IF (Q='V')
        C_LINE VERTICAL P2   {if Q<>'H', but Q='V', only these}
        LET X 3                   {two statements are executed}
  ELSE
        C_LINE PERPENDICULAR P1 P2     {if Q<>'H', and Q<>'V',}
                             {but Q='P', only this statement is}
                                                    {executed}
  END_IF
  ```

  Dieses Beispiel ist gut geeignet, um `IF`-Anweisungen zu demonstrieren, jedoch nicht sehr gut geeignet, um defensives Programmieren zu demonstrieren. Was geschieht beispielsweise, wenn der Benutzer versehentlich `'Y'` eingibt? Oder wenn er `'v'` statt `'V'` eingibt?

Mit der abgeänderten Version unten sollen mögliche Fehler bei der Benutzereingabe abgefangen werden:

```
LOOP
READ STRING "ENTER 'H' FOR HORIZ, OR 'V' FOR
                                VERT, OR 'P' FOR PERP" Q
EXIT_IF ((Q='H') OR (Q='h') OR (Q='V') OR
                            (Q='v') OR (Q='P') OR (Q='p'))
END_LOOP
IF ((Q='H') OR (Q='h'))
  C_LINE HORIZONTAL P2
ELSE_IF ((Q='V') OR (Q='v'))
  C_LINE VERTICAL P2
ELSE
  C_LINE PERPENDICULAR P1 P2
END_IF
```

- Ihr Makro kann nicht ausgeführt werden, weil die vom Benutzer eingegebenen Zahlen zu groß, zu klein oder negativ sind.Sie können diese Zahlen durch Anweisungen wie im Beispiel unten abfangen:

```
LOOP
  READ NUMBER 'Enter number of sides' Num_sides
EXIT_IF ((Num_sides > 0) AND (Num_sides < 20))
END_LOOP
```

Makro-Befehle

Alle Befehle, die in der Online-Hilfe aufgeführt sind, können im Hauptteil eines Makros verwendet werden. Einige Befehle und Anweisungen sind in Makros sehr hilfreich. Dieser Abschnitt enthält einige Beispiele.

`DISPLAY P1`	Zeigt die X- und Y-Koordinaten des Punktes `P1` und erwartet eine Benutzereingabe.
`BEEP`	gibt ein akustisches Signal über den Systemlautsprecher.

`IF (M>N)`
`...`
`ELSE`
`...`
`END_IF`
	Führt alle Zeilen vor `ELSE if M>N` aus. Andernfalls werden alle Zeilen zwischen `ELSE` und `END_IF` ausgeführt. Die Anzahl der `ELSE`-Anweisungen ist beliebig.

`LET D (L1 + 5)`	Definiert `D` als Summe von `L1` und `5`.

`LOOP`
`...`
`EXIT_IF (N > 50)`
`...`
`EXIT_IF (M < 7)`
`END_LOOP`
	Führt alle Zeilen vor `END_LOOP` so lange aus, bis `N > 50` oder `M < 7`. Die Anzahl der `EXIT`-Anweisungen ist beliebig. Ist keine EXIT-Anweisung vorhanden, wird die Schleife so lange ausgeführt, bis `END` oder ein anderer Befehl eingegeben bzw. ausgewählt wird.

`READ PNT P8`	Hält das System so lange an, bis der Benutzer den Punkt `P8` eingegeben hat.

`REPEAT`
`...`
`UNTIL (N>10)`
	Führt alle Zeilen so lange aus, bis `N` größer ist als `10`.

`TONE 256 2 0.5`	Gibt `2` Sekunden lang einen Ton (`256` Hz) mit der relativen Amplitude `0.5` auf dem Systemlautsprecher aus.

`WHILE (N< 20)`
`...`
`END_WHILE`
	Führt alle Zeilen nach `WHILE` so lange aus, bis `N` kleiner `20` ist.

Ausdrücke sind in Klammern eingeschlossen. Ein Ausdruck enthält normalerweise arithmetische, trigonometrische oder relationale Vorgänge wie in den vorherigen Beispielen. Es gibt auch mehrere spezielle Vorgänge, die verwendet werden können. Diese werden im nächsten Abschnitt beschrieben.

Integrierte Operationen

Die folgenden integrierten Operationen sind in Makros besonders nützlich:

`ANG P6`	Berechnet den Winkel zwischen dem Vektor `P6` und der X-Achse.
`LEN P4`	Berechnet die Länge des Vektors `P4`.
`PNT_XY 20 65`	definiert einen Vektor mit dem Ursprung bei X0, Y0 und dem Kopf bei X20, Y65.
`PNT_RA 16 38`	Definiert einen Vektor mit dem Ursprung bei X0, Y0, der Länge `16` und einem Winkel zur X-Achse von `38` Grad.
`ROT P2 45`	Definiert einen Vektor mit gleicher Länge wir der Vektor `P2` im Winkel von `45` Grad (gegen den Uhrzeigersinn) zum Vektor `P2`.
`SQR 15`	Berechnet das Quadrat von `15`.
`SQRT 15`	Berechnet die Quadratwurzel aus `15`.
`X_OF P1`	Berechnet die X-Koordinate des Punktes `P1`.
`Y_OF P6`	Berechnet die Y-Koordinate des Punktes `P6`.

4

Abfragefunktionen (INQ_ENV, INQ_ELEM)

Beim Schreiben von Makros ist es oft wichtig, die aktuellen Parameter, mit denen eine Zeichnung erstellt wird, zu kennen. Welche Farbe haben die Linien? Inch oder Millimeter? Ist die aktuelle Linienfarbe weiß oder andersfarbig? Ist der Maßstab 1:1 oder 2.5:1?

Beispiel:Sie wollen ein Makro erstellen, mit dem eine gelbe punktierte Linie gezeichnet wird. Es ist einfach, Farbe und Linienart zu ändern. Nach der Ausführung des Makros sollten Sie die Farbe und Linienart wieder in die ursprünglichen Werte ändern. Der Benutzer des Makros sollte diese Werte nicht zurücksetzen müssen.

Das Makro sollte deshalb die aktuellen Werte für Farbe und Linienart speichern und diese vor dem Beenden des Makros wieder aktivieren. Dieses Kapitel zeigt, wie Sie die Funktionen `INQ_ENV` und `INQ_ELEM` verwenden, um Informationen im System-Array zu speichern, und wie Sie `INQ` verwenden, um das System-Array abzufragen. `INQ` ist eine Abkürzung für Inquire (Abfrage).

INQ_ENV verwenden

Das folgende Makro demonstriert die Verwendung von INQ_ENV und INQ:

```
DEFINE Env_check
  LINE HORIZONTAL 0,0 300
  INQ_ENV 3               {load information in system array}
  LET Col (INQ 201)                {save color information}
  LET Ltype (INQ 301)            {save linetype information}
  INQ_ENV 2           {load window position in system array}
  LET Lower_left (INQ 101)        {save lower left point of}
                                             {current window}
  LET Upper_right (INQ 102)              {upper right point}
  YELLOW                                        {new color}
  DOT_CENTER                                 {new linetype}
  LINE HORIZONTAL 0,50 300
  COLOR Col                        {reset to original color}
  LINETYPE Ltype                 {reset to original linetype}
  LINE HORIZONTAL 0,100 300 END
  DISPLAY_NO_WAIT ("Look at the OLD window")
  WAIT 5
  WINDOW -10,-10 350,150               {create a new window}
  DISPLAY_NO_WAIT ("Now look at the NEW window")
  WAIT 5
  WINDOW Lower_left Upper_right
  DISPLAY_NO_WAIT ("Back to the OLD window")
  WAIT 5
END_DEFINE
```

Die zweite Zeile des Makros zeichnet eine Linie mit den aktuellen Werten für Farbe und Linienart. Der Rest des Makros wird im folgenden erläutert.

```
INQ_ENV 3
```

Wenn Sie sich die INQ_ENV-Funktion in der Online-Hilfe ansehen, wird deutlich, dass INQ_ENV Daten in einen Abschnitt des System-Arrays einfügt. Der Abschnitt des Arrays wird durch die Zahl nach INQ_ENV angegeben. Beispiel: Abschnitt 0 enthält Informationen zur Softwareversionsnummer und Softwareversionszeichenfolge. Abschnitt 1 enthält Informationen zum aktuellen Darstellungsfenster usw.

Hier ist besonders Abschnitt 3 interessant, der Angaben über Fangbereich, Zeichnungs- und Hilfslinien, Text usw. enthält.

Jedes Mal, wenn Sie INQ_ENV benutzen, werden die alten Daten im betreffenden Abschnitt des Systemfelds überschrieben.

```
LET Col (INQ 201)
```

INQ fragt den Abschnitt des System-Arrays ab, in den zuletzt mit INQ_ENV geschrieben wurde. In unserem Beispiel schrieb die letzte INQ_ENV-Funktion in Abschnitt 3, sodass INQ Abschnitt 3 abfragt. Die Angaben hängen jeweils von der Zahl hinter INQ ab.

Sehen Sie sich die INQ_ENV-Funktion in der Online-Hilfe an. INQ 201 gibt die aktuelle Geometriefarbe zurück. Dieser Wert wird der Variablen Col zugewiesen.

```
LET Ltype (INQ 301)
```

Die aktuelle Linienart der Geometrie ist der Variable Ltype zugeordnet.

```
INQ_ENV 2
```

Angaben zum aktuellen Fenster werden in Abschnitt 2 des Systemfeldes eingefügt.

```
LET Lower_left (INQ 101)
LET Upper_right (INQ 102)
```

Den Punkten links unten und rechts oben im aktuellen Fenster werden Variablen zugeordnet.

Die folgenden drei Zeilen des Makros ändern Farbe und Linienart und zeichnen eine Linie, um die Änderungen optisch deutlich zu machen.

```
COLOR Col
LINETYPE Ltype
```

Die ursprünglichen Werte für Farbe und Linienart werden wieder aktiviert. Das Makro zeigt die Änderungen, indem es eine weitere Linie zeichnet.

Die letzten Zeilen des Makros ändern die Koordinaten des aktuellen Fensters und zeigen dieses Fenster für etwa fünf Sekunden an, so dass Sie die Änderung sehen können. Anschließend werden die ursprünglichen Koordinaten wieder gültig.

INQ_ELEM verwenden

INQ_ELEM gleicht INQ_ENV und platziert Daten auch im System-Array. Diese Daten hängen von dem Element ab, das sich in der Zeichnung an dem angegebenen Punkt befindet. Wenn es sich bei dem Element z.B. um einen Kreis handelt, werden Kreisdaten in das Systemfeld gestellt.

Wenn Sie beim Auswählen eines Elements INQ 403 benutzen, können Sie den Elementtyp bestimmen. In Ihrem Makro sollen Benutzer evtl. einen Kreis wählen. Sie können INQ 403 verwenden, um sicherzustellen, dass es sich um einen Kreis statt beispielsweise ein Rechteck handelt.

Das folgende Codefragment soll als Beispiel dienen:

```
LOOP
  READ PNT 'Digitize a circle' P
  INQ_ELEM P
  EXIT_IF (INQ 403 = CIRCLE)
END_LOOP
```

In diesem Fragment liest INQ_ELEM Informationen zum digitalisierten Punkt in das System-Array. Wenn Sie sich die INQ_ELEM-Funktion in der Online-Hilfe ansehen, wird deutlich, dass INQ 403 den Elementtyp zurückgibt. Handelt es sich bei dem Elementtyp um einen Kreis, verlässt das Makro die Schleife und fährt mit dem nächsten Makroabschnitt fort.

Mit den folgenden Anweisungen können Sie Kreisradius und Kreismittelpunkt ermitteln:

```
LET Rad (INQ 3)
LET Cen_pt (INQ 101)
```

Wenn kein Element am digitalisierten Punkt gefunden wird, gibt INQ 403 END zurück. Vor der nächsten Eingabeaufforderung könnten Sie beispielsweise den Fangmodus abfragen.

GETENV verwenden

GETENV ruft die Einstellungen für die aktuelle Umgebung ab. Beispiel: Um die Einstellung für die Variable MEDIR abzurufen, geben Sie Folgendes in die Befehlszeile ein:

```
display (getenv('MEDIR'))
```

Creo Elements/Direct Drafting stellt die aktuellen Einstellungen wieder her.

Weitere Abfragen (inquiries)

Die Verwendung anderer Abfragen wie `HL_INQ_Z_VALUE` und `HL_INQ_FACE_COLOR` ähnelt der Verwendung von `INQ_ENV` und `INQ_ELEM`. In jedem Fall werden die Werte in das Abfrage-System-Array geschrieben und mit `INQ` abgefragt.

Ein Beispiel eines Makros mit `HL_INQ_Z_VALUE` finden Sie unter Anwendungsbeispiele auf Seite 91.

5

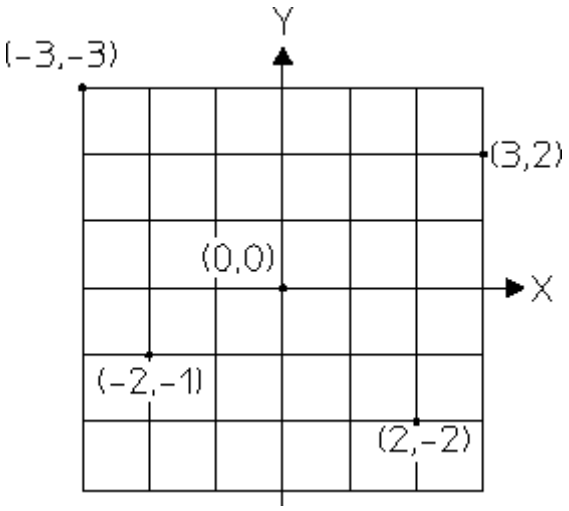
Punkte und Vektoren

Wie die meisten Leser dieses Handbuchs werden Sie wahrscheinlich mit Punkten und Vektoren vertraut sein. Allerdings könnte es schon ein paar Jahre her sein, dass Sie zuletzt damit gearbeitet haben. Dieses sehr kurze Kapitel ist dazu gedacht, das Thema aufzufrischen, bevor Sie Makros zum Zeichnen auf Seite 62 lesen.

Punkte

Ein Punkt wird durch eine X-Koordinate und eine Y-Koordinate definiert. Dies ist in Abbildung 1 unten dargestellt:

Abbildung 1. Darstellung von Punkten



Die X-Achse und die Y-Achse schneiden sich an Punkt (0,0). Dieser Punkt wird als Ursprung bezeichnet. Am Ursprung ist die X-Koordinate 0 und die Y-Koordinate 0.

Sie können erkennen, dass die anderen Punkte durch Schreiben der X-Koordinate gefolgt von der Y-Koordinate identifiziert werden.

Wenn Sie in einem Makro über einen Punkt verfügen und die X-Koordinate oder Y-Koordinate des Punkts kennen müssen, verwenden Sie die Operatoren X_OF oder Y_OF. Beispiel:

```
LET X1 (X_OF P1)
LET Y1 (Y_OF P1)
```

Nehmen wir an, dass P1 (3,2) ist. Dann ist X_OF P1 3 und Y_OF P1 2.

In einem Makro kann mit zwei beliebigen Zahlen und dem Operator PNT_XY ein Punkt beschrieben werden. Beispielsweise mit den Zahlen 3 und 2:

```
LET P1 (PNT_XY 3 2)
```

Dadurch entsteht der Punkt (3,2).

Allgemein gilt:

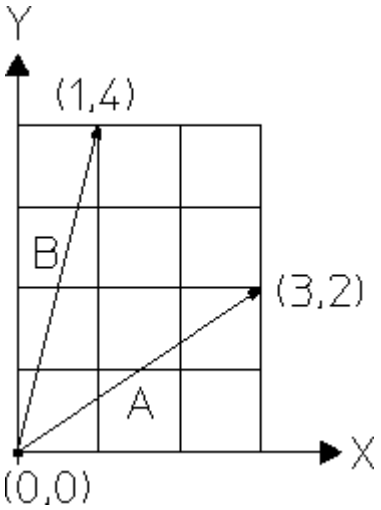
```
LET P1 (PNT_XY X Y)
```

erzeugt den Punkt (X,Y).

Vektoren

Ein Vektor ist eine gerichtete Größe, hat also immer eine Länge und eine Richtung. In der folgenden Abbildung stellt der Vektor A eine Linie vom Ursprung (0,0) zum Punkt (3,2) dar. Der Vektor B stellt eine Linie vom Ursprung (0,0) zum Punkt (1,4) dar.

Abbildung 2. Vektoren mit Ursprung im Nullpunkt

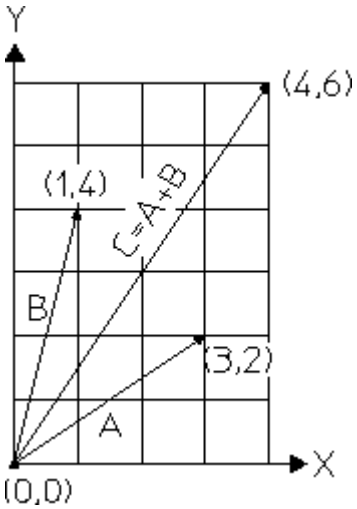


Wenn Vektoren ihren Ursprung im Nullpunkt haben, wie in der vorherigen Abbildung, ist Vektor `A (3,2)` und Vektor `B (1,4)`. In den Zeichnungsmakros, die in den nachfolgenden Kapiteln beschrieben werden, beginnen alle Vektoren im Ursprung.

Addition von Vektoren

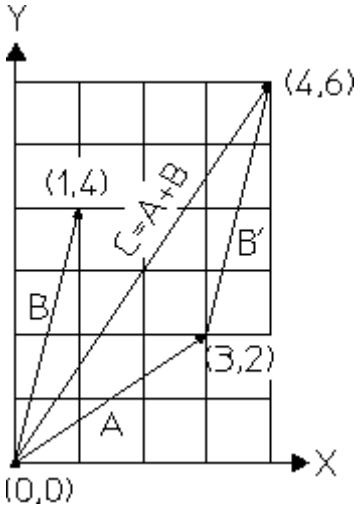
Vektor `A` kann zu Vektor `B` hinzugefügt werden, um Vektor `C` zu ergeben. In der folgenden Abbildung 3 stellt die Gerade vom Ursprung zum Punkt (4,6) den Vektor `C` dar. Die `X`-Koordinate (dargestellt durch `C`) wird durch Hinzufügen der `X`-Koordinate s von (1,4) und (3,2) erzielt, was `4` ergibt. Gleichermaßen wird die `Y`-Koordinate durch Hinzufügen der `Y`-Koordinaten dieser beiden Punkte erzielt, was `6` ergibt.

Abbildung 3. Addition von Vektoren



Es gibt auch eine andere Möglichkeit, das Hinzufügen des Vektors `A` zu Vektor `B` zu visualisieren. Verschieben Sie zunächst Vektor `B` an eine neue Position `B'` am Ende des Vektors `A` (siehe nächste Abbildung). Denken Sie daran, dass ein Vektor durch eine Länge und Richtung definiert wird, sodass der neue Vektor `B'` mit dem alten Vektor `B` identisch ist.

Abbildung 4. Visualisierung der Vektoraddition



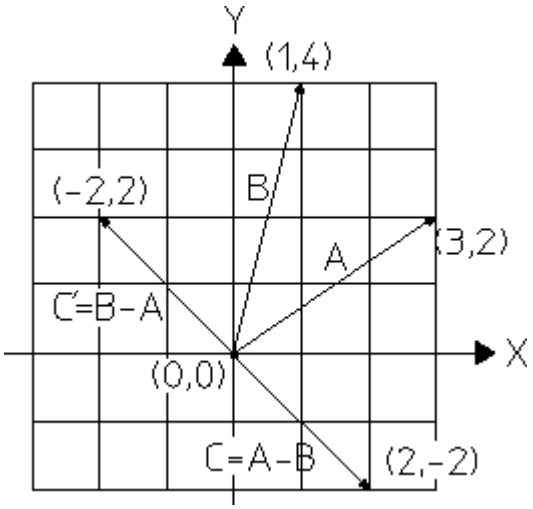
Beginnen Sie nun am Nullpunkt, und verschieben Sie 3 Einheiten in die `X`-Richtung und 2 Einheiten in die `Y`-Richtung. Verschieben Sie dann 1 Einheit in die `X`-Richtung und 4 Einheiten in die `Y`-Richtung. Zu diesem Punkt kommen Sie auch, wenn Sie vom Ursprung aus 4 Einheiten in `X`-Richtung und 6 Einheiten in `Y`-Richtung gehen. Dies beweist Folgendes: `C = A + B`.

Bei der Vektoraddition entspricht `A + B B + A`.

Subtraktion von Vektoren

Um Vektoren zu subtrahieren, subtrahieren Sie ihre `X`- und `Y`-Koordinaten. Dabei spielt die Reihenfolge (d.h. welcher Vektor von welchem subtrahiert wird) eine Rolle. Die folgende Abbildung zeigt das Ergebnis von `A - B` und von `B - A`.

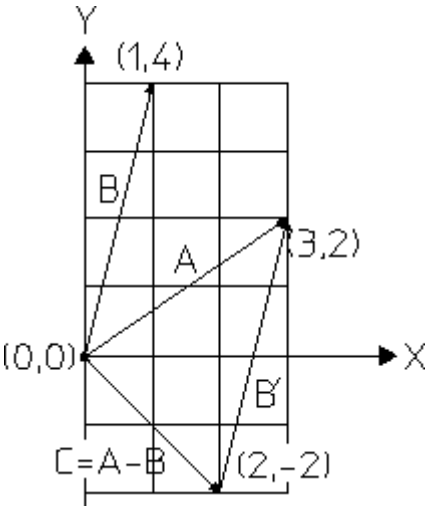
Abbildung 5. Subtraktion von Vektoren



Wie bei normaler Arithmetik entspricht der Vektor `A − B` nicht dem Vektor `B − A`. Die Vektoren haben die gleiche Länge, jedoch unterschiedliche Richtungen.

Wie bei der Addition können Vektoren auch zeichnerisch subtrahiert werden.Dazu verschieben Sie den Anfang des einen Vektors an den Endpunkt des anderen Vektors. Die folgende Abbildung zeigt das Ergebnis von `A − B`:

Abbildung 6. Visualisierung der Vektorsubtraktion



6

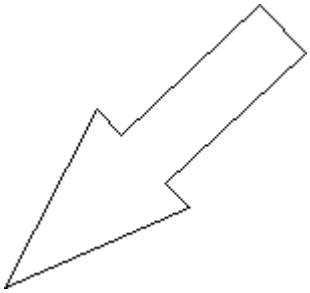
Makros zum Zeichnen

In diesem Abschnitt wird beschrieben, wie im einzelnen ein Makro zum Erstellen einer Geometrie geschrieben wird. Beim Konstruieren treten bestimmte Formen immer wieder (teils identisch, teils mit geringfügigen Abweichungen) auf. Der Aufwand, der für das Erlernen des Makroschreibens investiert werden muss, macht sich durch die erhöhte Produktivität und den durch das Anwenden der Makros gewonnenen Bedienungskomfort schnell bezahlt.

In diesem Kapitel werden die einzelnen Schritte zum Erstellen eines Makros für einen Pfeil und für eine Frontplatte ausführlich beschrieben.

Makro für einen Pfeil

Das folgende Makro soll den unten abgebildeten Pfeil erstellen.



Dabei sind folgende Vorgaben zu beachten:

- Die Pfeilspitze muss zwischen zwei Punkten liegen, die im Bildschirm gewählt werden. Der erste Punkt ist die Spitze des Pfeils. Der zweite Punkt befindet sich in der Mitte des Endes.
- Die Proportionen des Pfeils müssen konstant bleiben.
- Das Makro soll beliebig oft nacheinander ausgeführt werden können, bis es durch Eingabe eines anderen Befehls abgebrochen wird.

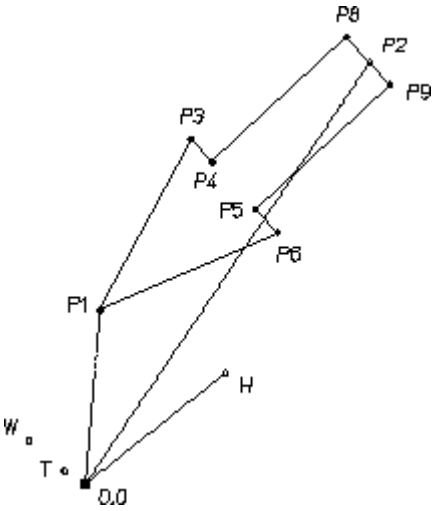
Es gibt verschiedene Möglichkeiten, ein solches Makro zu schreiben. Es gibt keine Regeln hinsichtlich der zu verwendenden Methode. Wählen Sie die Methode, die am besten für Sie geeignet ist.

Die hier verwendete Vorgehensweise basiert auf der klassischen Art der Geometrieerstellung, bei der Punkte als Vektoren aufgefaßt werden. Sie eignet sich besonders für diejenigen, die bereits Erfahrung im Umgang mit der Vektordarstellung haben.

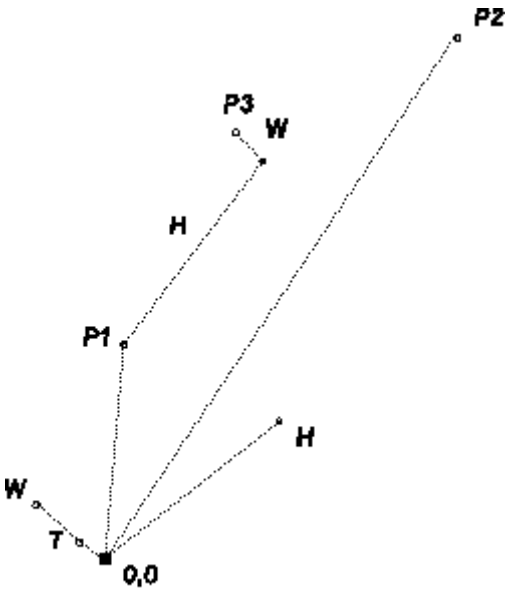
Vorgehensweise

Die schrittweise Vorgehensweise:

1. Pfeil in einem beliebigen Winkel zum Achsenkreuz zeichnen und alle Punkte beschriften. Die im Bildschirm gewählten Punkte sind `P1` und `P2`. Für den Ursprung (0,0) ist ein beliebiger Punkt festzulegen.



2. Bearbeiten Sie zunächst den Hauptteil des Makros. Bei einem Geometriemakro ist dies der Teil, der für die Zeichnung verantwortlich ist. Nehmen wir an, dass die Punkte `P1` und `P2` definiert sind und dass sie jeweils den Kopf eines Vektors darstellen, dessen Ursprung der Nullpunkt ist (siehe vorherige Abbildung).
3. Danach ist das Verhältnis der Flanken zur Gesamtlänge des Pfeils (z.B. 60%) festzulegen. Definieren Sie einen Vektor mit dieser Länge zwischen `P1` und `P2` in Richtung `P2`, und nennen Sie ihn `H`. Wie bei allen anderen Vektoren ist der Ursprung von `H 0,0`.
```
LET H (0.6 * ( P2 - P1 ) )
```
4. Das Verhältnis der Flanken zur Mittelinie beträgt z.B. 30% von `H`. Definieren Sie einen Vektor dieser Länge senkrecht zu `H`, und nennen Sie ihn `W`.
```
LET W ( 0.3 * ( ROT H 90 ) )
```
5. Die halbe Dicke des Endes entspricht einem bestimmten Verhältnis von `W` (z. B. 40%). Definieren Sie einen Vektor in Richtung `W`, und nennen Sie ihn `T`.
```
LET T ( 0.4 * W )
```
6. Nun sind die einzelnen Punkte zu definieren. Fügen Sie `H` und `W` zu `P1` hinzu, um den Punkt `P3` wie in der nächsten Abbildung zu definieren.



LET P3 (P1 + H + W)

```
LET P3 ( P1 + H + W )
```

7. Alle anderen Punkte sind auf die gleiche Weise zu definieren.

```
LET P6 ( P1 + H - W )
LET P4 ( P1 + H + T )
LET P5 ( P1 + H - T )
LET P8 ( P2 + T )
LET P9 ( P2 - T )
```

8. Danach sind die Punkte miteinander zu verbinden.

```
LINE POLYGON P1 P3 P4 P8 P9 P5 P6 P1
```

Der Hauptteil des Makros ist fertig. Nun können wir die im Bildschirm ausgewählten Punkte `P1` und `P2` bearbeiten. Der `READ`-Befehl ermöglicht dem Benutzer, die Koordinaten `P1` und `P2` einzugeben:

```
READ PNT 'Pick the arrow point' P1
```

`READ PNT` teilt dem System mit, dass ein Punkt eingegeben wird. Das System zeigt die Meldung "Pick the arrow point" an. Ein daraufhin eingegebener Punkt wird als `P1` definiert. Danach wird die nächste Makrozeile ausgeführt.

```
READ PNT 'Pick the tail centerpoint' RUBBER_LINE P1 P2
```

Mit READ PNT wird die Eingabe eines weiteren Punktes erwartet. Der nach der Meldung stehende Befehl RUBBER_LINE bewirkt, dass zwischen `P1` und der jeweiligen Position des Cursors eine Linie gezeichnet wird. Dies ist für die Bearbeitung der Pfeilspitzengröße hilfreich. Ein daraufhin eingegebener Punkt wird als `P2` definiert. Danach wird die nächste Makrozeile ausgeführt. In der nächsten Zeile beginnt der Hauptteil des Makros.

Erstellen Sie nun für die `READ`-Anweisungen und den Hauptteil des Makros eine Schleife. Verwenden Sie dazu `LOOP` und `END_LOOP`. Dadurch kann der Benutzer mehrere Pfeilspitzen während des gleichen Vorgangs erzeugen. Geben Sie `LINETYPE` und `COLOR` außerhalb der Schleife an, sodass sie während des Vorgangs geändert werden können (falls erforderlich). Definieren Sie alle Makrovariablen als `LOCAL`.

Das Makro ist somit vollständig aufbereitet und kann ausgeführt werden:

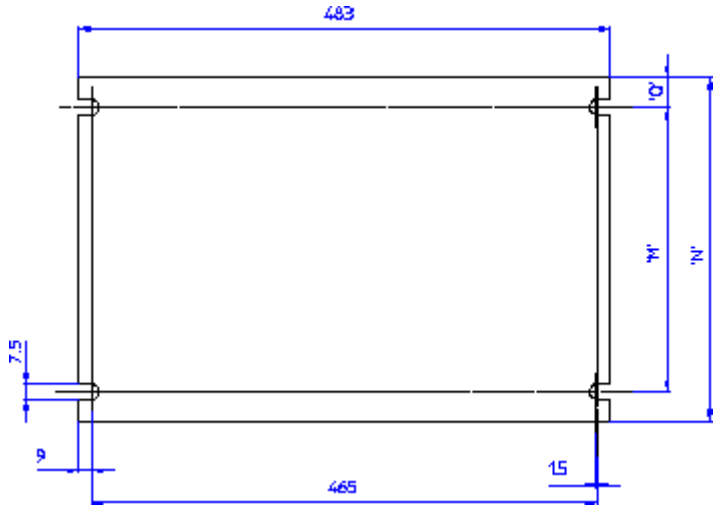
```
DEFINE Arrow_head
  LOCAL P1
```

```
  LOCAL P2
  LOCAL P3
  LOCAL P4
  LOCAL P5
  LOCAL P6
  LOCAL P8
  LOCAL P9
  LOCAL H
  LOCAL W
  LOCAL T
    COLOR WHITE
    LINETYPE SOLID
  LOOP
    READ PNT 'Pick the arrow point' P1
    READ PNT 'Pick the tail centerpoint' RUBBER_LINE P1 P2
    LET H (0.6 * ( P2 - P1 ) )
    LET W ( 0.3 * ( ROT H 90 ) )
    LET T ( 0.4 * W )
    LET P3 ( P1 + H + W )
    LET P6 ( P1 + H - W )
    LET P4 ( P1 + H + T )
    LET P5 ( P1 + H - T )
    LET P8 ( P2 + T )
    LET P9 ( P2 - T )
    LINE POLYGON P1 P3 P4 P8 P9 P5 P6 P1
    END
  END_LOOP
  END
END_DEFINE
```

Dies ist nur eine von vielen Möglichkeiten, ein solches Makro zu schreiben. Als Alternative wäre es z.B. auch möglich, vor der Eingabe der Punkte die Achsen um den gewünschten Winkel zu verschieben. Diese Methode würde Befehle verwenden wie `CS_SET`, `CS_ROTATE`, `CS_REF_PT`, `CS_AXIS`.

Makro für eine Frontplatte

Das Ziel besteht darin, ein Makro zu schreiben, um eine Frontplatte wie die hier abgebildete zu erzeugen. Die Platte passt in eine standardmäßige 19-Zoll-Zahnstange.



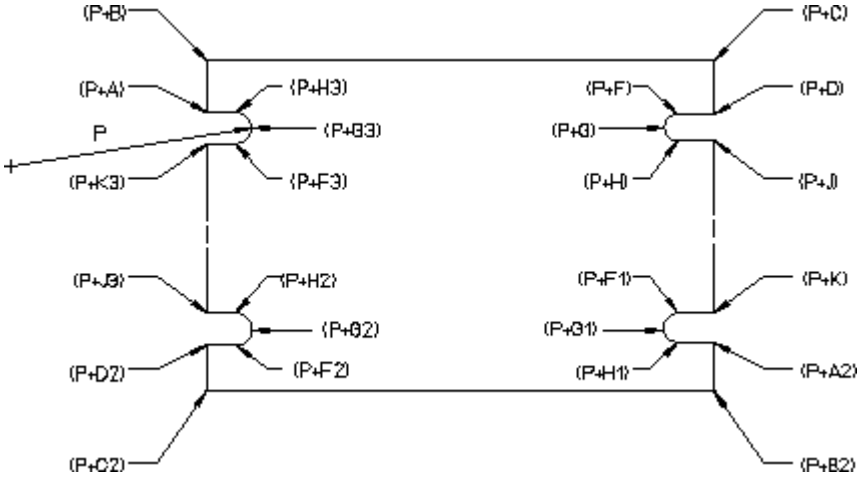
Das Makro muss Folgendes bewirken:

- Die Platte wird durch Anwählen einer der Gewindebohrungen im linken Träger des Rahmens positioniert.
- Die Höhe der Platte wird durch Eingabe des entsprechenden `U`-Werts bestimmt. Die Breite ist konstant.
- Die Ausrichtung der Platte entspricht stets den Linealachsen.

Die Frontplatte basiert auf Fixgeometrie.Aus diesem Grund können die Punkte der Hilfsgeometrie durch Koordinaten statt durch Vektoren definiert werden.

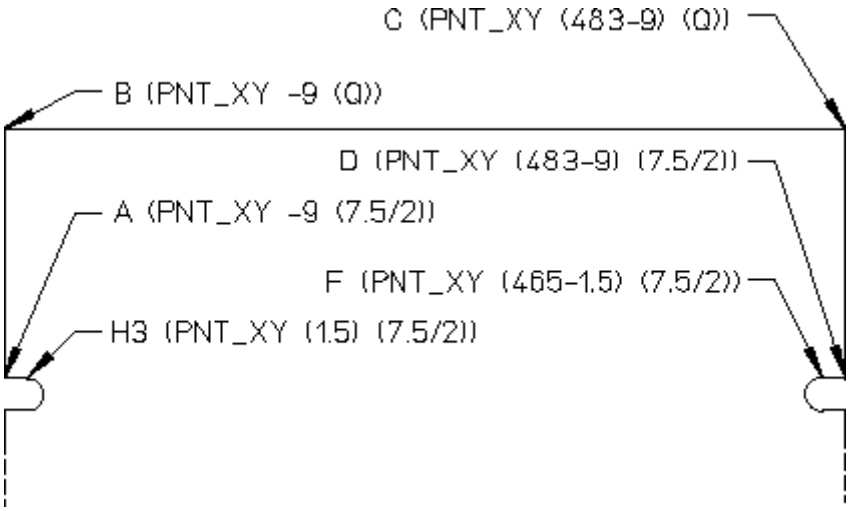
Vorgehensweise

Zuerst wird der Umriß der Frontplatte mit fester Ausrichtung (am zweckmäßigsten 0 °) unter Angabe aller relevanten Koordinaten eingezeichnet.



Ebenso wie beim Pfeil-Makro wird zuerst der Hauptteil des Makros erstellt. Es wird davon ausgegangen, dass der Vektor `P` bereits definiert ist und dass die Daten der Frontplatte der Abbildung entsprechen. `M` und `Q` sind variable Parameter.

Definieren Sie alle Punkte durch ihre Koordinaten vom Endpunkt des Vektors `P`. `P` hat die Koordinaten (0,0). Die Koordinaten werden somit einfach zum Ausgangsvektor `P` addiert. Die nächste Abbildung zeigt die Koordinaten einiger Geometriepunkte der Frontplatte.



Auf dieselbe Weise werden nach und nach alle übrigen Geometriepunkte der Frontplatte definiert:

```
LET A (PNT_XY -9 (7.5/2))
LET B (PNT_XY -9 (Q))
LET C (PNT_XY (483-9) (Q))
LET D (PNT_XY (483-9)(7.5/2))
LET F (PNT_XY (465-1.5)(7.5/2))
LET G (PNT_XY (465-1.5-(7.5/2)) 0)
LET H (PNT_XY (465-1.5)(-7.5/2))
LET J (PNT_XY (483-9)(-7.5/2))
LET K (PNT_XY (483-9)(-M+(7.5/2)))
LET F1 (PNT_XY (465-1.5)(-M+(7.5/2)))
LET G1 (PNT_XY (465-1.5-(7.5/2)) (-M))
LET H1 (PNT_XY (465-1.5)(-M-(7.5/2)))
LET A2 (PNT_XY (483-9)(-m-(7.5/2)))
LET B2 (PNT_XY (483-9)(-M-Q))
LET C2 (PNT_XY -9(-M-Q))
LET D2 (PNT_XY -9(-M-(7.5/2)))
LET F2 (PNT_XY 1.5(-M-(7.5/2)))
LET G2 (PNT_XY (1.5+(7.5/2))(-M))
LET H2 (PNT_XY 1.5(-M+(7.5/2)))
LET J2 (PNT_XY -9(-M+(7.5/2)))
LET K2 (PNT_XY -9(-7.5/2))
LET F3 (PNT_XY 1.5(-7.5/2))
LET G3 (PNT_XY (1.5+(7.5/2)) 1.5)
LET H3 (PNT_XY (1.5)(7.5/2))
```

Zur Vervollständigung des Makro-Hauptteils müssen nun noch die einzelnen Punkte miteinander verbunden werden. Dies erfolgt mithilfe der Befehle `LINE` und `ARC` in beliebiger Reihenfolge. Die hier verwendete Reihenfolge erfolgt von Punkt `(P+H3)` zu Punkt `(P+G3)`:

```
LINE POLYGON (P+H3) (P+A) (P+B) (P+C) (P+D) (P+F)
ARC THREE_PTS (P+F) (P+H) (P+G)
LINE POLYGON (P+H) (P+J) (P+K) (P+F1)
ARC_THREE_PTS (P+F1) (P+H1) (P+G1)
LINE POLYGON (P+H1) (P+A2) (P+B2) (P+C2) (P+D2) (P+F2)
ARC THREE_PTS (P+F2) (P+H2) (P+G2)
LINE POLYGON (P+H2) (P+J2) (P+K2) (P+F3)
ARC THREE_PTS (P+F3) (P+H3) (P+G3)
```

Der Hauptteil des Makros ist somit vollständig. Um das Makro ausführbar zu machen, müssen noch die Dateneingaben erfasst und der Vektorpunkt P positioniert werden. `READ`-Anweisungen werden für die Dateneingabe verwendet. Die `READ`-Anweisungen werden wie unten dargestellt implementiert:

```
READ NUMBER  "Enter the required height of the front panel" U
```

```
READ "Identify the position of the top left bolt hole" P
```

Schließlich muss noch der Vektorpunkt P positioniert werden:

```
READ NUMBER  "Enter the required height of the front panel" U
READ "Identify the position of the top left bolt hole" P
```

Das Makro kann nun ausgeführt werden, enthält jedoch nur sehr wenige Angaben. Die Linienart, die Linienfarbe, der Schluß des Makros, Regeln hinsichtlich der Eingabewerte sowie das Verfahren zur Umsetzung der Daten in die tatsächliche Höhe der Platte sind nicht berücksichtigt. Diese Angaben sind maßgebend für die Benutzerfreundlichkeit des Makros.

Linienart und die Farbe können innerhalb des Makros festgelegt werden. Dies kann z.B. durch folgende Eingabe erfolgen:

```
COLOR YELLOW
LINETYPE SOLID
```

Sie können auch einfach angeben, wo und wann das Makro ausgeführt werden soll. Dies erfolgt durch Einfügen von `END` vor `END_DEFINE` im Makro. Enthält das Makro keine `END`-Anweisung, so kehrt das System automatisch zur Eingabeaufforderung des Befehls zurück, der vor dem Aufrufen des Makros ausgeführt wurde.

Sie können Bedingungen für den `U`-Wert auf verschiedene Weise angeben. In diesem Beispiel wurde die Konstruktion `LOOP...EXIT_IF...END_LOOP` zusammen mit `IF...END_IF` gewählt, um die Eingabe von `U` zu prüfen. Drei Bedingungen werden für den `U`-Wert für dieses Makro festgelegt:

- Es muss sich um eine ganze Zahl handeln.
- Der Wert muss über 0 liegen.
- Der Wert muss unter 17 liegen.

Die Schleifen werden in folgender Form eingegeben:

```
IF ((FRACT U <> 0) OR (U>16) OR (U< 1))
  LOOP
    READ NUMBER "Please re-enter an integer value between 1 and 16"
    EXIT_IF ((FRACT U=0) AND (U>0) AND (U< 17))
  END_LOOP
END_IF
```

In diesem Beispiel wird die Schleife so lange ausgeführt, bis der `U`-Wert die obigen Bedingungen erfüllt.

Nach den Prüfungen muss der Wert so aufbereitet werden, dass er im Makro sinnvoll eingesetzt werden kann. Momentan liegt ein ganzzahliger Wert vor, der der Einheitshöhe der Platte entspricht; er muss daher so umgewandelt werden, dass er die tatsächliche Höhe wiedergibt. Dies erfolgt durch die letzte `IF...END_IF`-Schleife im Makro. Vor der Schleifendefinition wird der Wert `U` von `LET N (((U-1)*44.45)+43.6)` konvertiert, um einen Wert `N` zu erzeugen, der die tatsächliche Höhe der Platte angibt. Dieser Wert wird dann verwendet, um die Variablen `M` und `Q` zu definieren. Die Variable `M` definiert den vertikalen Abstand zwischen den Nuten, und `Q` definiert den vertikalen Abstand zwischen jeder Nut und einer horizontalen Kante, wobei `M` im kompletten Bereich von `U` variabel ist. `Q` ist für separate Werte fest:

- 3 > U > 0 (U kleiner als 3 aber größer als 0).
- 17 > U > 2 (U kleiner als 17 aber größer als 2).

Das Makro ist somit vollständig aufbereitet und kann ausgeführt werden:

```
DEFINE Tline
LOCAL P
READ NUMBER 'Enter the required height of the front panel' U
  COLOR YELLOW
  LINETYPE SOLID
  IF ((FRACT U <> 0) OR (U>16) OR (U< 1))
    LOOP
     READ NUMBER "Please re-enter an integer value between 1 and 16" U
     EXIT_IF ((FRACT U=0) AND (U>0) AND (U< 17))
    END_LOOP
  END_IF
LET N (((U-1)*44.45)+43.6)
        IF (U< 3)
           LET M (N-11.9)
           LET Q 5.95
           LET M (N-75.4)
        ELSE
           LET Q 37.7
        END_IF
        READ "Identify the position of the top left bolt hole" P
        LET A (PNT_XY -9 (7.5/2))
        LET B (PNT_XY -9 (Q))
        LET C (PNT_XY (483-9) (Q))
        LET D (PNT_XY (483-9) (7.5/2))
        LET F (PNT_XY (465-1.5) (7.5/2))
        LET G (PNT_XY (465-1.5-(7.5/2)) 0)
        LET H (PNT_XY (465-1.5) (-7.5/2))
        LET J (PNT_XY (483-9) (-7.5/2))
        LET K (PNT_XY (483-9) (-M+(7.5/2)))
        LET F1 (PNT_XY (465-1.5) (-M+(7.5/2)))
        LET G1 (PNT_XY (465-1.5-(7.5/2)) (-M))
        LET H1 (PNT_XY (465-1.5) (-M-(7.5/2)))
        LET A2 (PNT_XY (483-9) (-M-(7.5/2)))
        LET B2 (PNT_XY (483-9) (-M-Q))
        LET C2 (PNT_XY -9 (-M-Q))
        LET D2 (PNT_XY -9 (-M-(7.5/2)))
        LET F2 (PNT_XY 1.5 (-M-(7.5/2)))
        LET G2 (PNT_XY (1.5+(7.5/2)) (-M))
        LET H2 (PNT_XY 1.5 (-M+(7.5/2)))
        LET J2 (PNT_XY -9 (-M+(7.5/2)))
        LET K2 (PNT_XY -9 (-7.5/2))
        LET F3 (PNT_XY 1.5 (-7.5/2))
        LET G3 (PNT_XY (1.5+(7.5/2)) 1.5)
        LET H3 (PNT_XY 1.5 (7.5/2))
        LINE POLYGON (P+H3) (P+A) (P+B) (P+C) (P+D) (P+F)
        ARC THREE_PTS (P+F) (P+H) (P+G)
        LINE POLYGON (P+H) (P+J) (P+K) (P+F1)
        ARC THREE_PTS (P+F1) (P+H1) (P+G1)
        LINE POLYGON (P+H1) (P+A2) (P+B2) (P+C2) (P+D2) (P+F2)
        ARC THREE_PTS (P+F2) (P+H2) (P+G2)
        LINE POLYGON (P+H2) (P+J2) (P+K2) (P+F3)
        ARC THREE_PTS (P+F3) (P+H3) (P+G3)
        END
END_DEFINE
```

7

Dateiein-/Ausgabe und Zeichenfolgen

In diesem Kapitel wird beschrieben, wie Daten von einem Makro aus einer Datei gelesen und in eine Datei geschrieben werden können. Im nachfolgenden Beispiel wird mit Textdaten gearbeitet.Bei numerischen Daten können Sie nach dem gleichen Prinzip verfahren.

Wie das Makro arbeitet

Es soll ein Makro geschrieben werden, das die Schlüsselwörter aus den Abschnitten der Hilfedatei (`help`) extrahiert und in eine neue Datei schreibt.

Der folgende Ausschnitt aus der Hilfedatei (`help`) verdeutlicht, was unter einem "Schlüsselwort" zu verstehen ist.

```
=====================================================
^AUTO_NEW_SCREEN
AUTO_NEW_SCREEN function
---->(AUTO_NEW_SCREEN)---->+------>(ON)----->+---->
                           |                  |
                           `------>(OFF)---->'
The AUTO_NEW_SCREEN function allows you to select ...
.
.
=====================================================
```

In diesem Beispiel beschreibt der Abschnitt die `AUTO_NEW_SCREEN`-Funktion. Das Schlüsselwort ist `AUTO_NEW_SCREEN`. Das ist die Zeichenfolge, die in die Ausgabedatei geschrieben werden soll.

Nach einem Schlüsselwort kann auf zwei Arten gesucht werden:

- Suchen Sie nach `^`, und ziehen Sie die Zeichenfolge hinter `^`. oder
- Suchen Sie die Zeichenfolge `command` oder `function`. Verwenden Sie dann die vorhergehende Zeichenfolge.

Für die Programmierung ist die erste Methode besser. Der Prozessor muss eine Datei nur vorwärts prüfen. Bei der zweiten Methode muss der Prozessor vorwärts suchen, bis er `command` oder `function` findet und dann rückwärts, um die vorangehende Zeichenfolge zu finden. Wir verwenden die erste Methode.

Nächstes Beispiel:

```
=====================================================
^CANCEL ESC STOP INTERRUPT BREAK
^ABORT
^Meview_cancel
CANCEL command
---->(CANCEL)---->
CANCEL  cancels  the current  activity  ...
.
.
=====================================================
```

In diesem Beispiel ist `^CANCEL`, `^ABORT` und `^Meview_cancel ^` vorangestellt. Wir möchten jedoch nur `CANCEL` in unserer Ausgabedatei verwenden. Wir möchten nicht `ABORT` oder `Meview_cancel` einschließen, da sich diese Zeichenfolgen auf andere Abschnitte der Hilfedatei (`help`) beziehen. Das Makro muss den Rechner also anweisen, in einem Abschnitt nur die erste Zeichenfolge nach `^` als Schlüsselwort zu erkennen und alle weiteren Zeichenfolgen zu ignorieren.

Vor der Beschreibung des Textmakros muss Folgendes erwähnt werden. Unser Makro muss nach Leerzeichen in den Zeilen suchen. Im vorherigen Beispiel ist `^CANCEL` von der folgenden Zeichenfolge `ESC` durch ein Leerzeichen getrennt. Manchmal werden Leerzeichen verwendet, um das Ende von Zeichenfolgen zu markieren. Sehen wir uns nun diesen Abschnitt der Hilfedatei (`help`) erneut an:

```
=====================================================
^AUTO_NEW_SCREEN
```

```
AUTO_NEW_SCREEN function
---->(AUTO_NEW_SCREEN)---->+------>(ON)----->+---->
                           |                 |
                           `------>(OFF)---->'
The AUTO_NEW_SCREEN function allows you to select ...
.
.
=====================================================
```

Die erste Zeile des Abschnitts ist `^AUTO_NEW_SCREEN`. Es ist wichtig zu wissen, dass diese Zeile bei der Anzeige am Bildschirm mit Leerzeichen aufgefüllt wird. Diese Leerzeichen sind jedoch in der Datei nicht enthalten. Direkt hinter dem letzten `N` von `AUTO_NEW_SCREEN` befindet sich ein Zeilenvorschubzeichen, das nicht auf dem Bildschirm, sondern nur in der Datei angezeigt wird. Das Zeilenvorschubzeichen hat den hexadezimalen ASCII-Wert 0A. In der Programmiersprache C wird als Zeilenvorschubzeichen ein "\n" verwendet. Die erste Zeile im Beispiel enthielte also demnach in der Datei folgende Zeilenvorschubzeichen:

```
^AUTO_NEW_SCREEN\nAUTO_NEW_SCREEN function\n\n ... (and so on)
```

"\n" ist hier mit zwei Zeichen dargestellt. In Wirklichkeit besteht das Zeilenvorschubzeichen jedoch nur aus einem Zeichen.

Makroanalyse

Das folgende Makro extrahiert die Schlüsselwörter aus der Hilfedatei (`help`). Es gibt sicherlich noch effizientere Methoden, diese Aufgabe mit Hilfe eines Makros zu erfüllen. Dieses Makro wurde jedoch in diesem Zusammenhang absichtlich gewählt, weil es einfach aufgebaut ist und sich deswegen als Beispiel zum Analysieren besonders gut eignet.

```
DEFINE Keywords_search
{##############################################}
{##  This macro searches for all the keywords  ##}
{##  in the help file, and lists them in   ##}
{##  an output file.                          ##}
{##############################################}

  LOCAL File1
  LOCAL File2
  LOCAL Flag
  LOCAL Filestring1
  LOCAL First_char
  LOCAL Line_pos
  LOCAL String_length
  LOCAL First_string
  LET File1 '\me10\help'
  LET File2 '\john\keywords.out'
  OPEN_INFILE 1 File1
  OPEN_OUTFILE 2 DEL_OLD File2
  LET Flag 0                    {initialize the value of Flag}

  LOOP
    READ_FILE 1 Filestring1     {read the next line of file 1}
  EXIT_IF (Filestring1='END-OF-FILE')
    LET First_char (SUBSTR Filestring1 1 1)
    IF (First_char='^')
      IF (Flag = 0)
        LET Line_pos (POS Filestring1 ' ')
                                {find the position in the line}
                                {of the first blank character}
        IF (Line_pos = 0)       {no blank characters}
```

```
          LET String_length (LEN Filestring1)
                               {find the length of the complete}
                               {line}
          LET First_string (SUBSTR Filestring1 2 (String_length-1))
          WRITE_FILE 2 First_string
          LET Flag 1
        ELSE
          LET First_string (SUBSTR Filestring1 2 (Line_pos-2))
          WRITE_FILE 2 First_string
          LET Flag 1
        END_IF
      END_IF
    ELSE
      LET Flag 0
    END_IF
  END_LOOP
  CLOSE_FILE 1
  CLOSE_FILE 2
END_DEFINE
```

Die erste Zeile des Makros lautet:

```
DEFINE Keywords_search
```

Jedes Makro muss mit `DEFINE` beginnen, gefolgt vom Namen des Makros. Hier ist der Makroname `Keywords_search`.

```
LOCAL File1
LOCAL File2
LOCAL Flag
LOCAL Filestring1
LOCAL First_char
LOCAL Line_pos
LOCAL String_length
LOCAL First_string
```

Dies sind die lokalen Variablen. Diese werden definiert, damit es beim Aufruf von `Keywords_search` von einem anderen Makro aus nicht zu Verwechselungen von Variablennamen kommen kann.

```
LET File1 '\me10\help'
LET File2 '\john\keywords.out'
```

In diesem Makro wird die Variable `File1` verwendet, um den kompletten Pfadnamen für die Hilfedatei (`help`) darzustellen. Gleichermaßen ist `File2` die Datei, in die die Liste der Schlüsselwörter geschrieben wird.

```
OPEN_INFILE 1 File1
```

Ein Makro kann grundsätzlich nur mit geöffneten Dateien arbeiten. Die Funktion `OPEN_INFILE` öffnet `File1` zum Lesen. Dabei zeigt der Dateizeiger auf den ersten Datensatz in der Datei. Die Ziffer `1` ist ein Dateideskriptor. Nach dem Öffnen von `File1` erfolgen alle weiteren Referenzen zu `File1` über den Dateideskriptor `1`. Auf die Verwendung von Dateibeschreibung wird später noch ausführlicher eingegangen. Der Dateideskriptor entspricht dem MS-DOS "Handle".

Die folgende Anweisung ist ebenfalls gültig und würde zum gleichen Ergebnis führen, wie die obengenannte Anweisung:

Plattformabhängigkeiten

```
OPEN_INFILE 1 '\me10\help'
```

Sie könnten denken, dass die Variable File1 nicht unbedingt erforderlich ist. Der Vorteil der Verwendung einer Variable wie File1 ist, dass das Makro schnell geändert werden kann. So können andere Dateien als C:\Program Files\PTC\Creo Elements\Direct Drafting [version]\help gelesen werden. Dazu müsste dann lediglich die folgende Zeile geändert werden:

```
LET File1 '\me10\help'
```

Falls die Variable File1 nicht verwendet würde, müsste der Name der zu bearbeitenden Datei (hier \me10\help) im gesamten Makro durch den Namen der betreffenden Datei ersetzt werden. Unser Makro ist sehr kurz, sodass sich dies einfach gestaltet. Dies kann bei sehr langen Makros einen erheblichen zusätzlichen Aufwand bedeuten.

```
OPEN_OUTFILE 2 DEL_OLD File2
```

Öffnen Sie File2 zum Schreiben. Der Dateideskriptor ist 2. Wenn die Datei bereits vorhanden ist, wird sie aufgrund der DEL_OLD-Anweisung überschrieben.

```
LET Flag 0
```

Flag ist eine Variable mit dem Wert 1 oder 0, je nachdem welche der beiden Verzweigungen bei jedem Makrodurchlauf verwendet wird. Eine Variable, die einen Zustand oder Status anzeigt, wird normalerweise als Kennzeichen (Flag) bezeichnet. Auf die Verwendung von Flag wird später noch ausführlicher eingegangen. Diese Zeile initialisiert den Wert von Flag in 0.

```
LOOP
```

Damit wird eine Schleife begonnen. Diese Schleife wird fortgesetzt, bis eine spätere EXIT_IF-Anweisung erfüllt ist oder END_LOOP ausgeführt wird.

```
READ_FILE 1 Filestring1
```

Lesen Sie die Datei, deren Dateideskriptor 1 ist. Also File1 (/me10/help). Bei neu eröffneten Dateien, die noch nicht gelesen wurden, steht der Dateizeiger auf dem ersten Datensatz in der Datei. Die READ_FILE-Funktion behandelt jede vollständige Zeile als Datensatz. Das heißt, der READ_FILE-Befehl liest die erste Zeile der Datei und bewegt dann den Dateizeiger, sodass er auf den nächsten Datensatz zeigt. Wenn die Datei das nächste Mal gelesen wird, liest der READ_FILE-Befehl die zweite Zeile usw. Wenn die Datei mehrmals durchsucht und dabei jedes Mal am Dateianfang begonnen werden soll, muss vor jeder READ_FILE-Anweisung die OPEN_INFILE-Anweisung ausgeführt werden, damit der Dateizeiger wieder auf der ersten Zeile der Datei positioniert wird.

```
EXIT_IF (Filestring1='END-OF-FILE')
```

Jede Datei weist eine Dateiendemarkierung auf. Die tatsächliche Markierung hängt vom Betriebssystem ab, ist jedoch häufig das NULL-Zeichen. Wenn das Makro dieses Dateiendezeichen erreicht, wird die aktuelle Schleife beendet.

```
LET First_char (SUBSTR Filestring1 1 1)
```

Suchen Sie die Teilzeichenkette, die an Position 1 in der Zeile beginnt und die Länge 1 aufweist. Suchen Sie mit anderen Worten das erste Zeichen in der Zeile.

Nehmen wir als Beispiel für Teilzeichenketten Folgendes an: Filestring1 = Have a nice day!

Die Programmanweisung lautet:

```
LET Substring1 (SUBSTR Filestring1 3 6)
```

Substring1 wäre dann ve a n.

```
IF (First_char='^')
```

Wenn das erste Zeichen ^ ist, führen Sie die folgenden Anweisungen bis zum übereinstimmenden ELSE_IF, ELSE oder END_IF aus. (Das übereinstimmende ELSE tritt 14 Programmzeilen später auf.)

Wenn das erste Zeichen nicht ^ ist, springen Sie zur ersten Anweisung nach dem übereinstimmenden ELSE.

```
IF (Flag = 0)
```

Wenn der Wert von Flag 0 ist, führen Sie die folgenden Anweisungen bis zum übereinstimmenden ELSE_IF, ELSE oder END_IF aus. ELSE_IF oder ELSE sind nicht vorhanden. Das übereinstimmende END_IF tritt 12 Programmzeilen später auf.

(Wir haben noch nicht den Punkt erreicht, an dem wir den Zweck von Flag erläutern können.)

```
LET line_position (POS Filestring1 ' ')
```

Wie bereits zuvor festgestellt wurde, enthält eine Zeile, die mit ^ beginnt und nur ein Schlüsselwort enthält, keine Leerzeichen. Wenn jedoch mehrere Schlüsselwörter in einer Zeile vorhanden sind, werden diese mit Hilfe von Leerzeichen voneinander getrennt. (Ein Leerzeichen wird hier durch ' ' dargestellt. Zwei Leerzeichen wären ' ' usw.) Der Zweck der Programmanweisung ist, zu ermitteln, ob die Zeile Leerzeichen enthält. Die Funktion POS ergibt die Position des ersten Auftretens einer Teilzeichenfolge (zweites Argument) in einer Zeichenfolge (erstes Argument). Als Beispiel könnte die aktuelle Zeile aus der Hilfedatei (help) wie folgt aussehen:

```
^CURRENT_DIRECTORY CD CURRENT DIRECTORY TM_FILE_2 SM_FILE_2
```

Hier ist das erste Argument die komplette Zeile beginnend mit ^CURRENT_DIRECTORY und endend mit SM_FILE_2. Das zweite Argument ist ' '. Daher entspricht Line_pos 19.

Die aktuelle Zeile könnte auch folgendermaßen lauten:

```
^AUTO_NEW_SCREEN
```

Der Wert von Line_pos ist 0. Jede Zeile endet mit einem Zeilenvorschubzeichen.Diese Zeile kann also keine Leerzeichen enthalten.

```
LET String_length (LEN Filestring1)
```

Diese Anweisung berechnet die Länge der gesamten Zeile. Bei der Zeile

```
^AUTO_NEW_SCREEN
```

ist der Wert von String_length 16.

Die nächste Programmzeile macht deutlich, warum die Länge der Zeichenfolge benötigt wird.

```
LET First_string (SUBSTR Filestring1 2 (String_length-1))
```

Sie kennen SUBSTR bereits. Diese Anweisung extrahiert die Teilzeichenkette, die bei Position 2 in der Zeichenfolge beginnt und die Länge String_length-1 aufweist. Bei der Zeichenfolge

```
^AUTO_NEW_SCREEN
```

ist First_string AUTO_NEW_SCREEN. Dies bedeutet, dass ^ vom Anfang der Zeile entfernt wurde. AUTO_NEW_SCREEN kann nun in eine Datei geschrieben werden.

```
WRITE_FILE 2 First_string
```

Hängen Sie `First_string` an die Datei an, deren Dateideskriptor `2` ist. Diese Datei ist `\john\keywords.out`. `First_string` wird als vollständige Zeile in dieser Datei behandelt.

```
LET Flag 1
```

Wenn eine Zeile in die Ausgabedatei geschrieben wird, legen wir `Flag` auf `1` fest. Wenn Sie sich den restlichen Code gründlich ansehen, stellen Sie fest, dass `Flag` auf `0` festgelegt wird, wenn eine Zeile aus `File1` gelesen, jedoch nicht in `File2` geschrieben wird.

Warum möchten wir dies erreichen? Wir möchten aus jedem Abschnitt die erste Zeichenfolge extrahieren, der ^ vorangestellt ist. Im nächsten Abschnitt der Hilfedatei (`help`) möchten wir beispielsweise `CANCEL` extrahieren, jedoch nicht `ABORT` oder `Meview_cancel`:

```
=======================================================
^CANCEL ESC STOP INTERRUPT BREAK
^ABORT
^Meview_cancel
CANCEL command
.
.
.
=======================================================
```

Zur Verdeutlichung ist nachfolgend eine vereinfachte Version des Makros abgebildet. Sie enthält nur Lese- und Schreibanweisungen sowie Anweisungen, die sich auf `Flag` beziehen:

```
LET Flag 0
LOOP
  READ_FILE 1 Filestring1
  ...
  IF (First_char='^')
    IF (Flag = 0)
      IF ...
        ...
        WRITE_FILE 2 First_string
        LET Flag 1
      ELSE
        ...
        WRITE_FILE 2 First_string
        LET Flag 1
      END_IF
    END_IF
  ELSE
    LET Flag 0
  END_IF
END_LOOP
```

Nach der `READ_FILE`-Anweisung gibt es vier mögliche Bedingungen:

1. `First_char` entspricht ^, `Flag` entspricht `0`.

   Ein Schlüsselwort wird in eine Datei geschrieben. `Flag` wird auf `1` festgelegt.

2. `First_char` entspricht ^, `Flag` entspricht `1`.

   Es wird nichts in die Datei geschrieben. `Flag` wird auf `0` festgelegt.

3. `First_char` entspricht ^, `Flag` entspricht `0`.

   Es wird nichts in die Datei geschrieben. `Flag` wird auf `0` festgelegt.

4. `First_char` entspricht ^, `Flag` entspricht `1`.

Es wird nichts in die Datei geschrieben. `Flag` wird auf `0` festgelegt.

Wenn `First_char` nicht `^` entspricht, wird nichts in die Datei geschrieben. Das ist OK. Dadurch wird gewährleistet, dass nur Schlüsselwörter als Zeichenfolgen in die Datei geschrieben werden.

Es wird nur etwas in die Datei geschrieben, wenn `First_char ^` und `Flag 0` ist. Folgendes ist wichtig: `Flag` kann nur `0` sein, wenn die vorherige Zeile nicht mit `^` beginnt. Wenn die vorige Linie mit `^` beginnt, wäre `Flag 1`. Dies bedeutet, dass wenn zwei aufeinanderfolgende Zeilen mit `^` beginnen, nur das Schlüsselwort in der ersten Zeile in die Datei geschrieben wird.

Wenn `Flag 1` ist, kann es nur durch Lesen einer Zeile auf `0` zurückgesetzt werden, die nicht mit `^` beginnt.

Haben Sie bemerkt, dass wir nur eine `Let Flag 1`-Anweisung benötigen? Diese wird nach der ersten `END_IF`-Anweisung im Makro platziert. Durch die Verwendung von zwei Anweisungen und die Platzierung jeder Anweisung nach einer `WRITE_FILE`-Anweisung wird `Flag` nur auf `1` eingestellt, wenn etwas in die Datei geschrieben wird.

```
ELSE
```

Wenn die Zeile Leerzeichen enthält, werden die nach `ELSE` aufgeführten Anweisungen ausgeführt.

```
LET First_string (SUBSTR Filestring1 2 (Line_pos-2))
```

Bei dieser Verzweigung steht vor dem ersten Schlüsselwort ein `^` und danach ein Leerzeichen. Das Schlüsselwort beginnt daher an Position `2`. Es weist eine Länge von `Line_pos-2` auf. Beispiel:

```
^CURRENT_DIRECTORY CD CURRENT DIRECTORY TM_FILE_2 SM_FILE_2
```

Das Schlüsselwort, das wir extrahieren möchten, ist `CURRENT_DIRECTORY`. Das erste Leerzeichen in dieser Zeile befindet sich an Position `19`. Die Länge von `CURRENT_DIRECTORY` ist `19-2` oder `17`.

```
WRITE_FILE 2 First_string
LET Flag 1
```

`First_string` wird in die Ausgabedatei geschrieben und `Flag` auf `1` festgelegt.

```
END_IF
```

Das letzte `END_IF` für `IF (Line_pos = 0).`

```
END_IF
```

Das letzte `END_IF` für `IF (Flag = 0).`

```
ELSE
```

Wenn `First_char` nicht `^` ist, wird der Code nach `ELSE` ausgeführt.

```
LET Flag 0
```

`Flag` wird auf `0` festgelegt, wenn eine Zeile aus der Hilfedatei gelesen, jedoch nicht in der Ausgabedatei gedruckt wird.

Nun haben wir das Ende des Makros erreicht. Die letzten Zeilen sind:

```
END_IF
```

Das letzte `END_IF` für `IF (First_char = '^').`

```
END_LOOP
```

Damit wird die Schleife beendet.

```
CLOSE_FILE 1
CLOSE_FILE 2
```

Die CLOSE_FILE-Anweisungen entsprechen den Anweisungen OPEN_INFILE und OPEN_OUTFILE zu Beginn des Makros.

Es gehört zu einem guten Programmierstil, dass alle durch ein Makro eröffneten Dateien am Ende wieder geschlossen werden. Wenn Sie eine Datei nicht schließen, bleibt sie geöffnet, bis der nächste Benutzer OPEN_INFILE für diese Datei ausführt. Vor dem erneuten Eröffnen wird die betreffende Datei vom Betriebssystem automatisch geschlossen. Wenn niemand die Datei verwendet, bleibt sie geöffnet. Die Anzahl gleichzeitig geöffneter Dateien für Betriebssysteme ist jedoch begrenzt. Wenn diese Grenze erreicht ist, kann das System keine weiteren Dateien für andere Benutzer öffnen. Daher ist es wichtig, nicht verwendete Dateien zu schließen.

Verschachtelter Aufruf

Innerhalb eines Makros können weitere Makros aufgerufen werden. Dieser Vorgang wird als Verschachtelung bezeichnet und bietet folgende Vorteile:

- Bei der Verschachtelung kann auf schon vorhandene fehlerfreie Makros zurückgegriffen werden. Dadurch wird Zeit für das Neuschreiben und das Beheben von Fehlern gespart.
- Lange Makros sind oftmals schwer zu verstehen. Deshalb ist es oft sinvoll, ein langes Makro aufzuteilen und die einzelnen Komponenten automatisch nacheinander aufzurufen. Dabei sollten möglichst aussagekräftige Namen für die einzelnen Bestandteile verwendet werden, damit der Benutzer bereits von den Namen der einzelnen Makros auf deren Funktion schließen kann. Eine durchdachte Systematik bei der Namensvergabe erübrigt zudem manchen Kommentar. Beispiel:

```
DEFINE Create_shell
  Calculate_internal_stress
  Calculate_flange_thickness
  Calculate_bolt_thickness
  Draw_semi_shell
  Draw_flanges
  Draw_bolts
  Draw_washers
  Draw_nuts
END_DEFINE
```

Ersetzen wir einige Zeilen in Keywords_search durch ein inneres Makro. In diesem Beispiel ist die Verwendung eines derartigen Makros wahrscheinlich nicht sehr sinnvoll. Dies soll ein relevantes Prinzip veranschaulichen und gleichzeitig erläutern, wie Parameter an Makros übergeben werden.

Die verschachtelt aufzurufende Komponente soll folgende Zeilen aus dem ersten Makro enthalten:

```
IF (Line_pos = 0)       {no blank characters}
  LET String_length (LEN Filestring1)
                        {find the length of the complete}
                        {line}
  LET First_string (SUBSTR Filestring1 2 (String_length-1))
  WRITE_FILE 2 First_string
  LET Flag 1
ELSE
  LET First_string (SUBSTR Filestring1 2 (Line_pos-2))
```

```
  WRITE_FILE 2 First_string
  LET Flag 1
END_IF
```

Um ein neues Makro namens `Write_to_file` zu erstellen, müssen Sie `DEFINE Write_to_file` am Anfang und `END_DEFINE` am Ende platzieren.

Im aufrufenden Makro müssen die ausgelagerten 10 Zeilen durch eine Zeile mit dem Namen des neuen Makros ersetzt werden.

```
Write_to_file
```

Das Ergebnis sieht dann folgendermaßen aus:

```
DEFINE Keywords_search
{##############################################}
{##  This macro searches for all the keywords  ##}
{##  in the help file, and lists them in       ##}
{##  an output file.                           ##}
{##############################################}
  LOCAL File1
  LOCAL File2
  LOCAL Flag
  LOCAL Filestring1
  LOCAL First_char
  LOCAL Line_pos
  LOCAL String_length
  LOCAL First_string
  LET File1 '\me10\help'
  LET File2 '\john\keywords.out'
  OPEN_INFILE 1 File1
  OPEN_OUTFILE 2 DEL_OLD File2
  LET Flag 0                    {initialize the value of Flag}
  LOOP
    READ_FILE 1 Filestring1     {read the next line of file 1}
  EXIT_IF (Filestring1='END-OF-FILE')
    LET First_char (SUBSTR Filestring1 1 1)
    IF (First_char='^')
      IF (Flag = 0)
        LET Line_pos (POS Filestring1 ' ')
                                {find the position in the line}
                                {of the first blank character}
        Write_to_file
      END_IF
    ELSE
      LET Flag 0
    END_IF
  END_LOOP
  CLOSE_FILE 1
  CLOSE_FILE 2
END_DEFINE
DEFINE Write_to_file
  IF (Line_pos = 0)     {no blank characters}
    LET String_length (LEN Filestring1)
                        {find the length of the complete}
                        {line}
    LET First_string (SUBSTR Filestring1 2 (String_length-1))
    WRITE_FILE 2 First_string
    LET Flag 1
  ELSE
    LET First_string (SUBSTR Filestring1 2 (Line_pos-2))
    WRITE_FILE 2 First_string
    LET Flag 1
  END_IF
END_DEFINE
```

Die neue Datei veranschaulicht die Tatsache, dass nicht als `LOCAL` deklarierte Variablen, automatisch global sind. Globale Variablen sind in beiden Makros bekannt. Beispiel: Der Wert für `Line_pos` wird umgekehrt im äußeren Makro berechnet und im inneren Makro benutzt. Der Wert für `Flag` wird umgekehrt im inneren Makro berechnet und im äußeren Makro benutzt.

Parameter an ein Makro übergeben

Es wurde bereits angesprochen, dass globale Variablen manchmal zu unerwünschten Nebeneffekten führen können. Um dies zu vermeiden, besteht die Möglichkeit, Informationen auch mit Hilfe von Parametern an andere Makros zu übergeben. Schreiben wir das innere Makro `Write_to_file` mit Parametern neu.

Die Variablen `String_length` und `First_string` werden nicht mehr von `Keywords_search` verwendet. Das heißt, sie wurden gelöscht.

```
DEFINE Keywords_search
  LOCAL File1
  LOCAL File2
  LOCAL Flag
  LOCAL Filestring1
  LOCAL First_char
  LOCAL Line_pos
{ LOCAL String_length               deleted }
{ LOCAL First_string                deleted }
  LET File1 '\me10\help'
  LET File2 '\john\keywords.out'
  OPEN_INFILE 1 File1
  OPEN_OUTFILE 2 DEL_OLD File2
  LET Flag 0
  LOOP
    READ_FILE 1 Filestring1
  EXIT_IF (Filestring1='END-OF-FILE')
    LET First_char (SUBSTR Filestring1 1 1)
    IF (First_char='^')
      IF (Flag = 0)
        LET Line_pos (POS Filestring1 ' ')
        Write_to_file Line_pos Filestring1 2
      END_IF
    ELSE
      LET Flag 0
    END_IF
  END_LOOP
  CLOSE_FILE 1
  CLOSE_FILE 2
END_DEFINE
DEFINE Write_to_file
  PARAMETER Column
  PARAMETER String
  PARAMETER File_descriptor
  LOCAL String_length
  LOCAL First_string
  IF (Column = 0)
    LET String_length (LEN String)
    LET First_string (SUBSTR String 2 (String_length-1))
    WRITE_FILE File_descriptor First_string
    LET Flag 1
  ELSE
    LET First_string (SUBSTR String 2 (Column-2))
    WRITE_FILE File_descriptor First_string
    LET Flag 1
  END_IF
END_DEFINE
```

Bei den neuen Makros ist folgendes zu beachten:

- Mit Ausnahme von `Flag` unterscheiden sich die Variablen in `Write_to_file` von denen in `Keywords_search`. Dies sind die üblichen Bedingungen, die bei einem verschachtelten Aufruf von Makros angetroffen werden. Normalerweise ist das aufgerufene Makro Bestandteil einer Makrobibliothek.Der Einsatz eines solchen bereits vorhandenen Makros macht das Schreiben eines eigenen Makros überflüssig. In der Makrobeschreibung wird erklärt, was das Makro macht, und welche Parameter zu übergeben sind. Sie brauchen sich nicht um die interne Arbeitsweise des Makros und um die Verarbeitung der Variablen zu kümmern.
- In `Write_to_file` können die Parameter in beliebiger Reihenfolge deklariert werden. Wenn jedoch `Write_to_file` von `Keywords_search` aufgerufen wird, müssen die Argumente die richtige Reihenfolge aufweisen. Die Anweisung, die `Keywords_search` aufruft, ist:
  ```
  Write_to_file Line_pos Filestring1 2
  ```

  In dieser Anweisung ist folgendes zu beachten:

  - `Line_pos` entspricht `Column`.
  - `Filestring1` entspricht `String`.
  - `2` entspricht `File_descriptor`.

Beim Aufruf des Makros `Write_to_file` weiß das System, dass der erste Parameter nach dem Makronamen der erste im aufgerufenen Makro definierte Parameter, der zweite Parameter der zweite im aufgerufenen Makro definierte Parameter ist usw.

Es ist zu beachten, dass Parameter zwar vom aufrufenden an das aufgerufene Makro, nicht jedoch umgekehrt vom aufgerufenen an das aufrufende Makro übergeben werden können. Wie in allen Programmiersprachen werden Makros bei der Kompilierung erweitert. Dies bedeutet, dass der Makroname durch den Makrocode ersetzt wird. Wenn ein Makro ein anderes Makro aufruft, wird der Code während der Kompilierung inline im aufrufenden Makro ersetzt. Der resultierende Code wird abgearbeitet, als stammte er aus nur einem Makro.

Wenn `Write_to_file` inline ohne `PARAMETER`-Anweisungen ersetzt wurde, meldet der Compiler, dass einige Variablen unbekannt sind. Die `PARAMETER`-Anweisungen informieren den Compiler darüber, dass einige Variablenpaare gleich sind. `PARAMETER`-Variablen verhalten sich jedoch wie `LOCAL`-Variablen: Sie sind nur in dem Makro sichtbar, in dem sie deklariert wurden. In diesem Punkt unterscheiden sich Makros von Funktionen und Unterroutinen in anderen Programmiersprachen wie beispielsweise PL/I oder Fortran. In diesen Sprachen verzweigt das Hauptprogramm zu der aufgerufenen Funktion bzw. Unterroutine und kann von dort aus dann Werte an das aufrufende Programm übergeben.

Da Sie den Wert von `Flag` nach der Ausführung von `Write_to_file` kennen müssen, wurde `Flag` nicht als lokale Variable oder als Parameter in `Write_to_file` deklariert. `Flag` bleibt also eine globale Variable, die in beiden Makros bekannt ist.

Die Schlüsselwörter in der Ausgabedatei `keywords.out` sind wie in der Hilfedatei (`help`) aufgeführt.

Plattformabhängigkeiten

Zum Sortieren wird sinnvollerweise der MS-DOS Befehl `sort` verwendet. Aktivieren Sie in der Windows-Umgebung den "Program Manager", um eine Befehlsaufforderung (DOS-Fenster) zu erhalten und geben Sie folgendes ein:

```
sort < keywords.out > keywords.srt
```

`keywords.srt` ist der Name der sortierten Datei. Wenn sich `keywords.out` nicht im aktuellen Verzeichnis befindet, müssen Sie den vollständigen Pfad zu dieser Datei angeben.

8

Makros und Bemaßungen

Dieses Kapitel beschreibt ein Makro, das Bemaßungen aus einer Datei mit Tabellendaten extrahiert. Das Makro zeichnet dann das Teil an einer bestimmten Position auf dem Bildschirm.

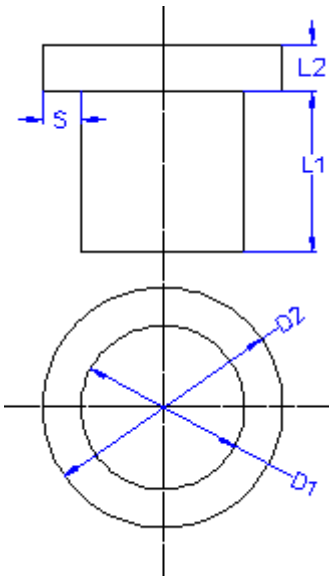
Wie das Makro arbeitet

- Der Benutzer wird aufgefordert, zwei Punkte auf dem Umfang einer Bohrungsfläche anzutippen oder einzugeben. Im Beispielmakro sind dies vom Betrachterstandpunkt aus der Punkt rechts und links.
- Das Makro berechnet anhand dieser beiden Punkte den Bohrungsdurchmesser.
- Das Makro sucht in der Datei, bis der errechnete Bohrungsdurchmesser mit einem Wert in der ersten Spalte übereinstimmt. Mit den restlichen Daten in der Tabellenzeile wird die Größe eines Zapfens berechnet, der ohne Spiel in das Loch passen soll.
- Das Makro zeichnet den Zapfen in seiner richtigen Position und Lage.
- Der Benutzer wird erneut aufgefordert, zwei Punkte oder `END` zu digitalisieren.

Beschreibung des Zapfens

Die folgende Abbildung zeigt die allgemeinen Bemaßungen des Zapfens:

Abbildung 13. Zapfen – allgemeine Bemaßungen

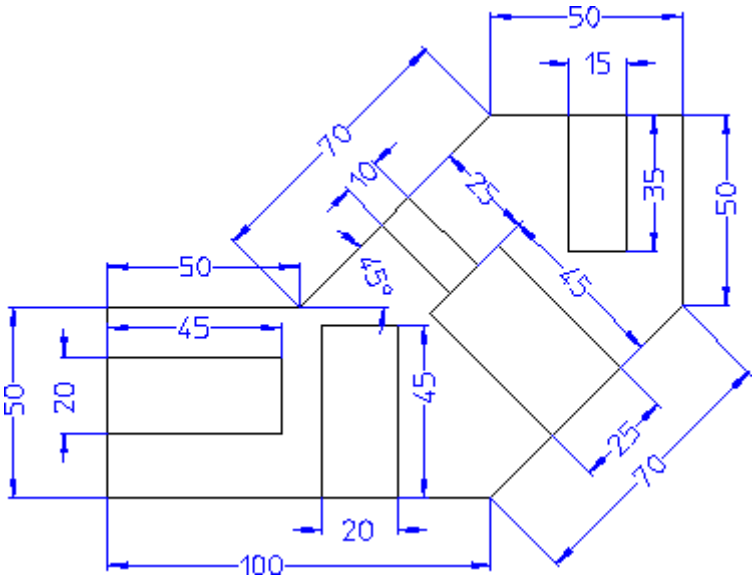


Die Bemaßung `S`, die Breite der Zapfenschulter, kann mit der Hälfte berechnet werden (`D2-D1`).

Die Geometrie des Zapfens ist sehr einfach aufgebaut. Das Makro, das später in diesem Kapitel beschrieben wird, kann jedoch auch an sehr komplexe geometrische Figuren angepasst werden.

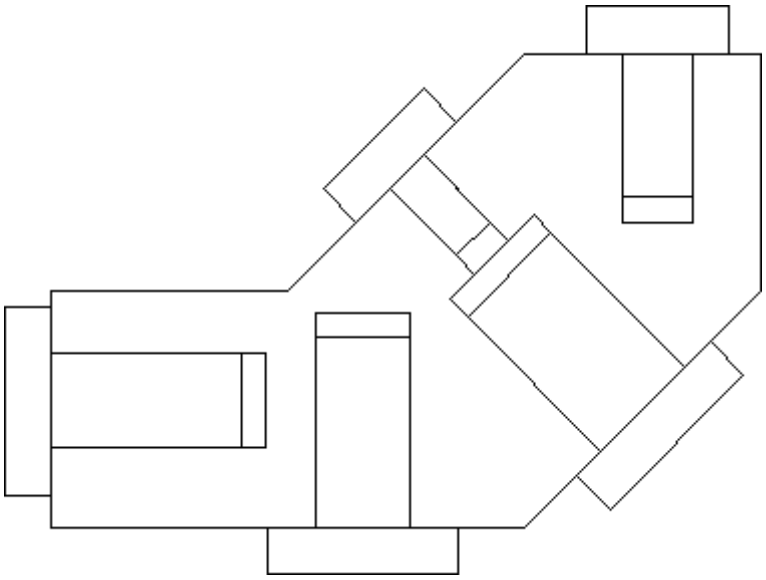
Zum Testen Ihres Makros könnten Sie beispielsweise eine der Schnittansicht ähnliche Anordnung zeichnen (nachfolgend abgebildet).In ihr sind Bohrungen mit verschiedenen Durchmessern in verschiedenen Lagen dargestellt.

Abbildung 14. Testanordnung (Schnittansicht)



Nachdem Sie mit dem Makro den jeweils passenden Zapfen in die einzelnen Bohrungen eingefügt haben, sollte die Zeichnung so aussehen:

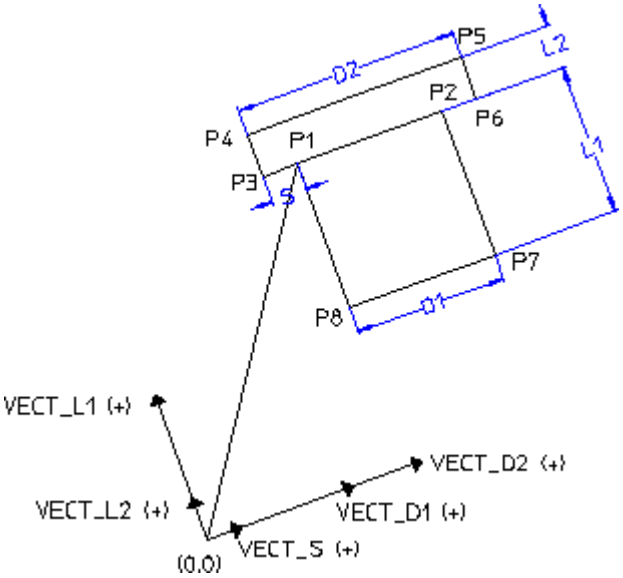
Abbildung 15. Platzierte Zapfen



Vektoranalyse

In der nachfolgenden Abbildung ist ein Zapfen dargestellt. `P1` bis `P8` sind Vektoren, die ihren Ursprung am Nullpunkt haben. Aus Gründen der Übersichtlichkeit wird in der Zeichnung lediglich der Vektor `P1` vollständig als vom Ursprung ausgehende Linie gezeigt. Darüber hinaus sind die zu den Bemaßungen des Zapfens gehörenden Vektoren dargestellt:

Abbildung 16. Zapfen – allgemeine Bemaßungen



Bemaßungen wie `D1` und `L1` (siehe Abbildung 16. Zapfen – allgemeine Bemaßungen auf Seite 87) können nicht für die Vektoranalyse verwendet werden. Für Vektoren ist eine Richtung und eine Länge erforderlich. Es ist jedoch einfach, Bemaßungen in Vektoren umzuwandeln, wenn die Winkel bekannt sind oder berechnet werden können.

Verwenden wir zunächst die Bemaßung `S`. Der Winkel von `S` entspricht dem Winkel von `(P2-P1)`. Da `P2` und `P1` Vektoren sind, kann der Winkel des Vektors `(P2-P1)` aus `ANG(P2-P1)` berechnet werden. `ANG` ist eine integrierte Funktion.

Wenn der von `S` abgeleitete Vektor `Vect_S` heißt, gilt Folgendes:

```
Vect_S = (PNT_RA S ANG(P2_P1))
```

`PNT_RA` ist eine integrierte Funktion, die eine Länge und einen Winkel in einen Vektor konvertiert.

Um `L1` und `L2` in Vektoren zu konvertieren, ist der Winkel 90 Grad größer als der Winkel von `(P2-P1)`. Beispiel:

```
Vect_L1 = (PNT_RA L1 (ANG(P2-P1)+90))
```

Die anderen Vektoren werden ähnlich definiert. In [Missing cross reference text] sind alle Vektoren, die Bemaßungen repräsentieren, als positive Vektoren in der dargestellten Richtung eingezeichnet. In Abbildung 16. Zapfen – allgemeine Bemaßungen auf Seite 87 haben wir den Zapfen in beliebigen Positionen gezeichnet. Das bedeutet, dass die von Bemaßungen abgeleiteten Vektoren in vielen anderen Positionen negativ sind. Dies ist unwesentlich. Solange Sie einheitliche "Bezeichnungskonventionen" für die Vektoren verwenden, müssen Sie sich über das Vorzeichen eines Vektors keine Sorgen machen.

Beschreibung der Datendatei

Die Datendatei, die das Makro verwendet, heißt `\anna\spigot.dat`. Dies sind die in ihr enthaltenen Daten:

```
10 30 20
15 30 30
```

```
20 40 40
25 40 40
```

Von links nach rechts repräsentieren die Daten die Zapfenbemaßungen `D1`, `D2` und `L1` in Millimetern. Diese Bemaßungen wurden frei gewählt, die Werte haben also keine feste Beziehung zueinander. Bei festen Beziehungen wie `D2=3xD1` und `L1=2xD1` ist es einfacher, die Gleichungen in Ihr Makro zu schreiben.

Makroanalyse

Im folgenden Abschnitt ist das Makro vollständig aufgeführt. Viele der darin enthaltenen Befehle und Funktionen kennen Sie bereits aus dem vorangegangenen Kapitel.

```
DEFINE Spigot
  {local variables here}
  LET File1 '\anna\spigot.dat'
  OPEN_INFILE 1 File1
  LET File1 '\anna\spigot.dat'
  OPEN_INFILE 1 File1
  LET L2 10 {height of head is constant for simplicity}
{the main loop is executed once for each pair of points }
{digitized by the user}
  LOOP
    READ PNT 'Digitize first point, or END'  P1
    READ PNT 'Digitize second point' P2
    LET Hole_diam (ABS(P2 - P1)) {we don't want negative }
                                 {lengths}
    OPEN_INFILE 1 File1
    LOOP
      READ_FILE 1 Line_of_data
      LET Line_pos (POS Line_of_data ' ')
               {find the position of the first blank in the line}
      LET Data_1 (SUBSTR Line_of_data 1 (Line_pos -1))
                  {Data_1 finishes just before the first blank}
      LET D1 (VAL Data_1)          {convert from text to numeric}
    EXIT_IF (ABS(D1 - Hole_diam) < 0.0001)   {we want the spigot}
                               {to be a friction fit in the hole}
    END_LOOP
    LET Line_length (LEN Line_of_data)
{mark the start position of the next data item, then}
{step along until the next blank is found}
    LET Start_pos (Line_pos+1)
    LET Line_pos (Line_pos+1)
    LOOP
      LET Next_char (SUBSTR Line_of_data Line_pos 1)
    EXIT_IF (Next_char = ' ')
      LET Line_pos (Line_pos+1)
    END_LOOP
    LET Data_2 (SUBSTR Line_of_data Start_pos (Line_pos-1))
{the last data starts from the position after the blank, and is}
{of length (Line_length-Line_pos)  }
    LET Data_3 (SUBSTR Line_of_data (Line_pos+1)
                       (Line_length-Line_pos))
    LET D2 (VAL Data_2)                        {convert to numeric}
    LET L1 (VAL Data_3)
    LET S (0.5*(D2 - D1))
    LET Angle (ANG(P2-P1))
{convert all lengths to vectors}
    LET Vect_S (PNT_RA S Angle)
    LET Vect_L2 (PNT_RA L2 (Angle+90))
    LET Vect_D2 (PNT_RA D2 Angle)
    LET Vect_L1 (PNT_RA L1 (Angle+90))
    LET Vect_D1 (PNT_RA D1 Angle)
    LET P3 (P1 - Vect_S)
```

```
    LET P4 (P3 + Vect_L2)
    LET P5 (P4 + Vect_D2)
    LET P6 (P5 - Vect_L2)
    LET P7 (P2 - Vect_L1)
    LET P8 (P7 - Vect_D1)
{connect the points}
    LINE POLYGON P1 P3 P4 P5 P6 P2 P7 P8 P1
  END_LOOP
END_DEFINE
```

Zunächst soll folgende Zeile analysiert werden:

```
LET Hole_diam (ABS(P2 - P1))
```

Wir suchen den Wert `D1` in der Datendatei, der `Hole_diam` entspricht . Da `(P2-P1)` auch einen negativen Wert annehmen kann, wird hier der Betrag verwendet.

```
LET D1 (VAL Data_1)
```

`Data_1` ist eine Textzeichenfolge. Wir möchten in der Lage sein, `Data_1` mit `D1` (Zahl) zu vergleichen. Daher konvertieren wir `Data_1` in einen Zahlenwert.

```
EXIT_IF (ABS(D1 - Hole_diam) < 0.0001)
```

Wir möchten die Datei durchsuchen, bis wir einen `D1`-Wert finden, der `Data_1` entspricht. Nun könnten Sie denken, dass hier eine Gleichheitsbedingung wie die folgende verwendet werden sollte:

```
EXIT IF (D1 = Hole_diam)
```

In allen Programmiersprachen werden in solchen Situationen Gleichheitsbedingungen vermieden, da sich Probleme mit der Genauigkeit einstellen würden. Beispiel: Sie digitalisieren die Bohrung mit 20 mm Durchmesser links von Abbildung 14. Testanordnung (Schnittansicht) auf Seite 86. Aufgrund von Zeichnungsungenauigkeiten könnte der Rechner einen Bohrungsdurchmesser von 20.0000000000001 mm errechnen. Wenn das Makro die dritte Zeile in der Datendatei liest, würde die Anweisung (mit der Funktion `trace` sichtbar gemacht) so aussehen:

```
EXIT IF (D1 20 = Hole_diam 20.0000000000001) 0
```

`0` am Ende der Zeile gibt "falsch" an, sodass keine Gleichheit vorhanden ist.

Die tatsächliche Anweisung, die wir im Makro verwenden, besagt, dass wenn der Unterschied zwischen `D1` und `Hole_diam` klein ist, `D1` akzeptiert wird.

```
LET P3 (P1 - Vect_S)
```

Abbildung 16. Zapfen – allgemeine Bemaßungen auf Seite 87 zeigt die positive Richtung von `Vect_S`. Um von `P1` zu `P3` zu gelangen, müssen wir die negative Richtung verwenden. Es gilt `P3 = P1 - Vect_S`. Wenn wir die positive Richtung von `Vect_S` nach links unten definieren, wäre die korrekte Gleichung `P3 = P1 + Vect_S`. Dies zeigt, dass die Vorzeichen von Vektoren immer bestimmt werden können, so lange Sie Ihre eigene Vorzeichenkonvention beachten.

```
LET P4 (P3 + Vect_L2)
```

Um von `P3` zu `P4` gelangen, verwenden wir die positive Richtung von `Vect_L2`. Das heißt, wir fügen `Vect_L2` hinzu. Ob die restlichen Vektoren addiert oder subtrahiert werden müssen, hängt davon ab, welche Richtung als positiv definiert worden ist.

Verbessern des Makros

Dieses Makro wurde so einfach wie möglich gehalten, um die Prinzipien, nach denen es funktioniert, zu verdeutlichen und um Ihren Schreibaufwand zu minimieren. Es ist unerheblich, ob Sie Daten für einfache Teile wie Unterlegscheiben, Dichtungen oder Stiftschrauben oder etwa für sehr komplexe Teile wie mechanische Dichtungen oder Kolbenbaugruppen in Textdateien speichern: das Prinzip ist das gleiche.

Selbst dafür, dass ein Zapfen eine sehr einfache geometrische Figur ist, ist das Makro noch sehr unausgefeilt. Einige Verbesserungsvorschläge:

- In unserem Makro müssen `P1` und `P2` in der richtigen Reihenfolge digitalisiert werden, damit das Makro die korrekte Lage des Zapfens kennt. Werden die Punkte in der falschen Reihenfolge eingegeben, wird der Zapfen spiegelverkehrt gezeichnet (probieren Sie es einmal aus!). Das Makro kann so umgeschrieben werden, dass der Benutzer einen dritten Punkt (z.B. am Ende der Bohrung) eingeben muss, durch den das Makro die Lage der Bohrung erkennt.
- Für den Zapfen wird eine Passung ohne Spiel benötigt, also ist sein Durchmesser mit dem der Bohrung identisch. Wird eine Passung mit Spiel benötigt, muss das Makro den Zapfen in der Bohrung zentrieren können.
- In der Datei `spigot.dat` besteht jedes Datenelement aus zwei Zahlen, wobei der Abstand zwischen den Datenelementen lediglich eine Leerstelle beträgt. Normalerweise sind Datenelemente, die in tabellarischer Form angeordnet sind, unterschiedlich lang, so dass die Spalten rechtsbündig ausgerichtet sind. Deshalb sind die Abstände zwischen den einzelnen Datenelementen unterschiedlich groß. Das Makro muss dies zulassen.
- Das Beispielmakro für den Zapfen enthält nur eine Datenzeile für jeden `D1`-Wert. Wenn für jeden `D1`-Wert mehrere Datenzeilen vorhanden sind (beispielsweise mehrere `D2`-Werte für jeden `D1`-Wert), müssen Sie eine Schleife in einer Schleife hinzufügen, um den erforderlichen `D2`-Wert zu suchen.
- Bei `P1` und `P2` wird angenommen, dass der Wert `(P1-P2)` mit dem Wert `D1` in der Datei `spigot.dat` übereinstimmt. Wenn der Benutzer die Punkte `P1` und `P2` eingibt, die nicht die Bedingung erfüllen, wird die Schleife ausgeführt, bis die Zeichenfolge `END-OF-FILE` gelesen wird. Alle anderen Ausführungen werden unterbrochen. Für diesen Fall sollten Sie eine weitere Zeile hinzufügen, die beim Erreichen des Dateiendes (`END-OF-FILE`) anstatt abzubrechen, eine Meldung über die falsche Dateneingabe ausgibt.

9

Anwendungsbeispiele

Dieses Kapitel enthält einige Makros, die beim Arbeiten mit dem CAD-System häufig eingesetzt werden können. Auch wenn Sie mit keinem der hier vorgestellten Makros arbeiten wollen, können Sie in diesem Kapitel doch einiges über den Aufbau eines Makros lernen.

Wenn Sie diese Makros lediglich einmal ausprobieren wollen, können Sie bei der Eingabe Zeit sparen, indem Sie die lokalen Variablen und die Kommentare weglassen.

Hilfslinien zu vorhandenen Linien zeichnen

Mit diesem Makro können Sie eine Hilfslinie in einem bestimmten Winkel zu einer vorhandenen Linie zeichnen. Bei der vorhandenen Linie kann es sich ebenfalls um eine Hilfslinie oder um eine Zeichnungslinie handeln.

```
DEFINE Ang_c_line
LOCAL P1
LOCAL P2
LOCAL Ang1
LOCAL Ang2
LOOP
  CATCH ELEM
  READ PNT
    'Pick intersection point on existing line' P1
  READ PNT
    'Pick a second point on existing line' P2
  READ NUMBER
    'Enter angle of C_line from existing line' Ang1
  LET Ang2 (ANG(P2-P1))
  C_LINE PT_ANG P1 (Ang1+Ang2)
  END
END_LOOP
END_DEFINE
```

Eine Linie in gleiche Segmente teilen

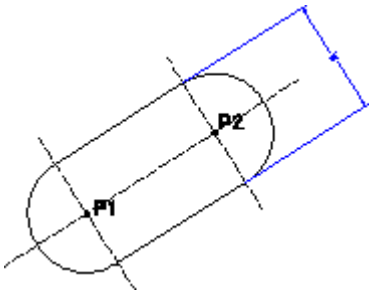
Mit diesem Makro können Sie eine Linie (real oder gedacht) in eine bestimmte Anzahl gleich großer Segmente teilen. Das Makro zeichnet an den Punkten, an denen die Linie geteilt werden soll, Hilfslinien ein. Diese Hilfslinien können waagerecht, senkrecht oder normal zur vorhandenen Linie verlaufen.

```
DEFINE cline_mix
{local variables here}
  LOOP
  READ STRING
    "ENTER 'H', 'V', OR 'P' FOR C_LINE HORIZ, VERT, OR PERP" Q
  EXIT_IF ((Q='H') OR (Q='h') OR (Q='V') OR (Q='v') OR
    (Q='P') OR (Q='p')) {this line must be joined to previous line}
  END_LOOP
  READ NUMBER
    'ENTER NO. OF SECTIONS FOR SPLITTING LINES (2,3,4,..)' N
  READ PNT 'ENTER start point of line' P1
  READ PNT 'ENTER end point of line' P2
  LET Fraction (1/N)
  LOOP
      LET N (N-1)
  EXIT_IF (N=0)
      LET PN (N*Fraction)
      LET PNN (P2-(P2-P1)*PN)
      IF ((Q='H') OR (Q='h'))
        C_LINE HORIZONTAL PNN
        ELSE_IF ((Q='V') OR (Q='v'))
          C_LINE VERTICAL PNN
        ELSE
          C_COLOR BLACK
          C_LINE P1 P2
          C_COLOR RED
          C_LINE PERPENDICULAR P1 PNN
      END_IF
    END_LOOP
  IF (Q='P')
    DELETE SELECT C_LINES BLACK CONFIRM END REDRAW
    ELSE
```

```
    END
  END_IF
END_DEFINE
```

Ein Langloch zeichnen

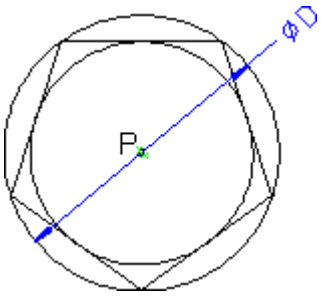
Dieses Makro zeichnet ein Langloch mit beliebigen Abmessungen in einem beliebigen Winkel zum Achsenkreuz.




```
DEFINE A_slot_macro
  LOCAL W
  LOCAL P1
  LOCAL V
  LOCAL P2
  READ NUMBER 'Enter the slot width' W
    LOOP
    FOLLOW OFF
    COLOR WHITE
    LINETYPE SOLID
    READ PNT 'Pick one center point' P1
    READ PNT 'Pick the other center point' RUBBER_LINE P1 P2
    LET V (ROT (( P2 - P1 ) * ( W /2 ) / (LEN (P2 - P1))) 90)
    ARC CEN_BEG_END P1 ( P1 + V ) ( P1 - V )
    ARC CEN_BEG_END P2 ( P2 - V ) ( P2 + V )
    LINE POLYGON ( P1 - V ) ( P2 - V )
    LINE POLYGON ( P1 + V ) ( P2 + V )
    LET V ( V * 1.5 )
    COLOR YELLOW
    LINETYPE DOT_CENTER
    LINE POLYGON ( P1 - V ) ( P1 + V )
    LINE POLYGON ( P2 - V ) ( P2 + V )
    LET V ( ROT V 90 )
    LINE POLYGON ( P1 + V ) ( P2 - V )
    END_LOOP
  COLOR WHITE
  LINETYPE SOLID
END_DEFINE
```

Ein regelmäßiges Polygon zeichnen

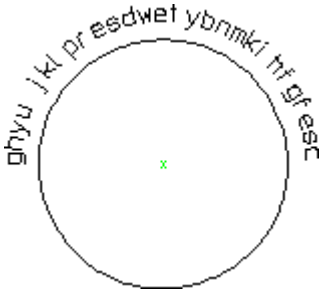
Dieses Makro zeichnet ein regelmäßiges Polygon mit beliebig vielen Seiten zusammen mit dazugehörigem Um- und Inkreis.




```
DEFINE A_polygon_macro
  LOCAL P
  LOCAL D
  LOCAL N
  LOCAL R
  LOCAL A
  LOCAL P1
  LOCAL P2
  LOCAL Pm
    LOOP
    READ 'Pick the center point of the polygon' P
    READ 'Enter the diameter of the circumscribing circle' D
    READ 'Enter the number of sides in the polygon ' N
    LET R (D/2)
    LET A ((180-(360/N))/2)
    LET P1 (P+PNT_XY(R*COS A)(R*SIN A))
    LET P2 (P+PNT_XY(-R*COS A)(R*SIN A))
    LET Pm ((P1 + P2) /2)
    LINE P1 P2
    MODIFY Pm ROTATE COPY (N -1) CENTER P (360/N)
    CIRCLE P (R*SIN A)
    CIRCLE P R
    WINDOW FIT
    END_LOOP
END_DEFINE
```

Text an kreisförmige Objekte anpassen

Dieses Makro passt eine Zeichenfolge an ein kreisförmiges Objekt an.




```
DEFINE T_rot
  LOCAL T
  LOCAL Cp
  LOCAL Sp
  LOCAL A
  LOCAL N
  LOCAL Da
  READ STRING 'Enter text' T
  READ PNT 'Enter center point' Cp
  READ PNT 'Enter start point' Sp
  READ NUMBER 'Enter angle' A
  LET N (LEN T)
  LET Da (A/N)
  LET A (ANG (Sp - Cp))
```

```
  LET N 1
  WHILE (N< =LEN T)
    TEXT_ANGLE (A - 90)
    TEXT (SUBSTR T N 1) Sp
    LET A (A - Da)
    LET Sp (Cp+PNT_RA (LEN (Sp - Cp)) A)
    LET N (N+1)
  END_WHILE
  END
  TEXT_ANGLE 0
END_DEFINE
```

Mehrere z-Wertebenen darstellen

Dieses Makro stellt die einzelnen z-Wertebenen einer Zeichnung nacheinander dar. Im Rechenspeicher (RAM) wird eine logische Tabelle erstellt, die die notwendige Information speichert.

Bevor Sie das Makro starten können, müssen Sie Ihr Modul "Verdeckte Linien" aktivieren und Sie müssen eine Zeichnung verwenden, deren Elemente auf verschiedenen z-Wertebenen liegen.

```
DEFINE Scan_z_levels
  LOCAL Range_min
  LOCAL Range_max
  LOCAL Act_line
  LOCAL Max_line
  LOCAL Act_val
  WINDOW FIT
  HL_INQ_Z_VALUE RANGE
  LET Range_min ( INQ 3 )
  LET Range_max ( INQ 4 )
  LET Act_line 0
  CREATE_LTAB "Range_drawing"     {create a logical table in RAM}
  { fill the logical table with all z values in the drawing}
  REPEAT
    LET Act_line ( Act_line + 1 )
    HL_INQ_Z_VALUE NEXT
    LET Act_val ( INQ 3 )
    WRITE_LTAB "Range_drawing" Act_line 1 Act_val
  UNTIL ( Act_val = Range_max )
  LET Max_line Act_line
  {Switch all geometry off and show only geometry having no Z-values}
  HL_REDRAW_MODE ON CURRENT
  SHOW GLOBAL ALL ON
  SHOW DIMENSIONS OFF
  SHOW HATCHING ALL OFF
  SHOW TEXTS ALL OFF
  HL_VISUALIZE GLOBAL ALL OFF
  DISPLAY "First of all, you see all elements without a z-value"
  LET Act_line 0
  {show elements on each z-value, one at a time }
  REPEAT
    LET Act_line ( Act_line + 1 )
    LET Act_val ( READ_LTAB "Range_drawing" Act_line 1)
    SHOW GLOBAL ALL OFF
    HL_VISUALIZE GLOBAL Act_val Act_val CYAN
    DISPLAY ("Now you see all elements with z-value " + (STR Act_val))
  UNTIL ( Act_line = Max_line )
  SHOW GLOBAL ALL ON
  DELETE_LTAB 'Range_drawing'
END_DEFINE
```

10

Systemoperationen aufzeichnen

Die Eingaben, die über einen bestimmten Zeitraum in das System gemacht werden, können aufgezeichnet und in einer Datei gespeichert werden. Dies ist besonders nützlich:

- Beim schnellen Erstellen von Makros für Geometrien mit fester Bemaßung
- Bei Systemvorführungen

Zwei Funktionen ermöglichen dies: ECHO und TECHO. Die Formelemente sind jeweils in den folgenden Abschnitten skizziert.

Funktion ECHO

Mit dieser Funktion werden alle Systemeingaben in folgender Form in eine Datei geschrieben oder auf einem Drucker ausgegeben:

```
Tm_screen_create_1
LINE POLYGON
-1.74255323667387E+002,-1.91693974675132E+000
-1.68756133685496E+002,1.64136935262188E+001
-1.64631741199077E+002,2.81847384876597E+000
-1.82402271788707E+002,3.73500551241448E+000
-1.78226960876530E+002,1.07617482670530E+001
-1.52869584848922E+002,6.79899966919450E-001
LINE PARALLEL
-1.53429687532262E+002,1.08635851185695E+001
-1.52767747997405E+002,3.88776078968923E+000
LINE HORIZONTAL
-1.67381336190023E+002,-3.44449251949884E+000
-1.50170908283734E+002,-2.27336872705908E+000
ARC THREE_PTS
-1.60100001306593E+002,7.50296901852502E+000
-1.50731010967075E+002,1.59643163056796E+000
-1.52716829571647E+002,1.09654219700860E+001
SPLITTING ON
CIRCLE CENTER
-1.54040708641361E+002,-2.06969502402607E+000
-1.51902134759515E+002,4.75337402757949E+000
END
END
```

Dabei werden alle Befehle und Funktionen sowie über die Tastatur eingegebene Zahlen aufgezeichnet.

Um eine ECHO-Datei zu starten, geben Sie Folgendes ein:

```
ECHO 'echofilename'
```

`echofilename` ist der gewählte Name für Ihre ECHO-Datei.

Von nun an werden alle Systemeingaben aufgezeichnet.

Um eine ECHO-Datei zu schließen, geben Sie Folgendes ein:

```
ECHO OFF
```

Um eine ECHO-Datei wiederzugeben, geben Sie Folgendes ein:

```
INPUT 'echofilename'
```

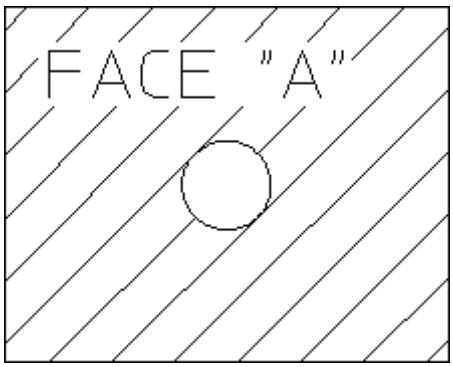
`echofilename` ist der Name, der der ECHO-Datei zuvor gegeben wurde.

Verwendung von ECHO beim Erstellen von Makros

Sie können die ECHO-Datei als Basis zur Erstellung eines Makros verwenden. ECHO ist jedoch für einige Makrotypen nicht sinnvoll. Das Zapfenmakro, das wir uns unter Makros und Bemaßungen auf Seite 84 angesehen haben, ist nahezu komplett mit der Vektoranalyse und dem Lesen von Dateien ausgelastet. ECHO eignet sich jedoch besonders für solche Aufgaben, bei denen viel digitalisiert wird oder bei denen viele Änderungen in Menüs vorgenommen werden.

Beispiel:

Abbildung 20. Mit ECHO erstellter Block



Beim Erstellen des Blocks enthält die ECHO-Datei folgende Daten:

```
TM_CREATE_1
LINE RECTANGLE
-6.263595945324684,8.418272950516371
5.595479044490039,0.7683344359598321
TM_CONSTRUCT_1
-6.230190100282952,8.418272950516371
5.628884889531771,0.7683344359598321
-6.230190100282952,0.7683344359598321
5.562073199448307,8.418272950516371
CIRCLE CENTER
-0.350761372938188,4.610006615758968
1.252719189064929,4.409571545508578
TM_END
Tm_text_1
TEXT_SIZE
1
I_set_font_1 "hp_i3098_v"
TEXT Restore_old_text_font
'FACE "A"'
-5.161203058947541,6.71457485338806
Tm_hatch
HATCH_DIST
1
HATCH AUTO
3.991998482486922,4.409571545508578
TM_END
echo off
```

Die Hilfslinien werden dazu verwendet, den Schnittpunkt der Rechteck-diagonalen zu ermitteln, um damit den Kreismittelpunkt optisch sichtbar zu machen. In einem Makro wären die Koordinaten des Kreismittelpunkts bekannt, so dass in diesem Fall keine Hilfslinien erforderlich wären.

Wir können die ECHO-Datei mit unseren eigenen Koordinaten für Punkte editieren:

```
DEFINE Hatch_block
LINE RECTANGLE
-5,4
5,-4
CIRCLE CENTER
0,0
-1,0
END
TEXT_SIZE
1
I_set_font_1 "hp_i3098_v"
TEXT Restore_old_text_font
'FACE "A"
-4,2
```

```
HATCH_DIST
1
HATCH AUTO
3,0
END
END_DEFINE
```

Vorgehensweise zum Erstellen des Makros:

- Platzieren Sie `DEFINE 'filename'` am Anfang und `END_DEFINE` am Ende.
- Entfernen Sie Menüreferenzen. Beispiel: Entfernen Sie `TM_CREATE_1`, `Tm_text_1` und `Tm_hatch`.
- Entfernen Sie `echo off`.
- Ersetzen Sie `TM_END` durch `END`.

In einem Makro wird `END` öfters verwendet als beim manuellen Arbeiten. Das liegt daran, dass beim manuellen Arbeiten rekursive Befehle und Funktionen durch Aufrufen eines neuen Befehls automatisch beendet werden.

11

Benutzeroberfläche und Befehlssatz

In diesem Kapitel schauen wir "hinter" die Benutzeroberfläche, um zu sehen, welche Befehle aktiviert werden, wenn Sie ein Objekt in einem Bildschirmmenü wählen. Auf diese Weise verbinden Sie eine vertraute Handlung von der Benutzeroberfläche mit einem nicht vertrauten Makrobefehl oder -Funktion.

Welche Befehle verwenden Sie?

Wenn Sie mit dem Schreiben eines Makros beginnen, welche Befehle und Funktionen verwenden Sie dabei? In Systemoperationen aufzeichnen auf Seite 96 wurde deutlich, dass die Verwendung von ECHO und die anschließende Bearbeitung der ECHO-Datei eine gute Methode zum Starten eines Makros ist. Diese Möglichkeit ist bestens geeignet für ein "zeichnendes" Makro, da es dem Benutzer eine Menge Antippvorgänge und Menüwechsel erspart. Viele Makros jedoch enthalten nur wenige Befehle, die direkt sichtbare Ergebnisse hervorbringen, wie beispielsweise das Zeichnen einer Linie oder das Erstellen eines neuen Fensters. Der verbleibende Teil der Befehle führt Berechnungen oder Vektorenanalysen durch. Das Zapfenmakro in Makros und Bemaßungen auf Seite 84 ist ein gutes Beispiel. Bei solchen Makros ist ECHO nicht sehr sinnvoll.

Falls Sie in einem Makro ein Fenster erstellen, welchen Befehl werden Sie verwenden? Sie gehen am Besten genauso vor wie auf der Benutzeroberfläche. Wenn Sie herausfinden können, welcher interne Befehl durch das Drücken des Tablettfeldes **ERSTELLEN** aufgerufen wird, haben Sie den richtigen Befehl für Ihr Makro gefunden.

Bildschirmbefehle und Funktionen herausfinden

Im Folgenden wird erläutert, wie Sie die Befehle ermitteln, die den Bildschirm-Menüfeldern zugrundeliegen. Verwenden wir das INFO-Menü als Beispiel. Tippen Sie ein:

```
EDIT_MACRO
```

Das System reagiert mit der Aufforderung:

```
Enter macro_name or ALL
```

Drücken Sie nun **INFO**. Das folgende Makro wird am Bildschirm angezeigt:

```
DEFINE Tm_info
  Sm_info
END_DEFINE
```

Geben Sie jetzt ein:

```
EDIT_MACRO Sm_info
```

Das Makro Sm_info wird am Bildschirm angezeigt. Es lautet wie folgt:

```
DEFINE Sm_info
  IF (I_port)
    Check_i_port
  END_IF
  IF (NOT I_port)
    CURRENT_MENU '' T_clear_menu
    MENU
    BLACK
    YELLOW '         INFO' ''  1 1
    MENU
    BLACK
    WHITE 'ADD SELECT' 'ADD_ELEM_INFO' 3 1
    MENU 'Element' 'ADD_ELEM_INFO' 3 2
    MENU
    BLACK
    CYAN 'SCREEN' 'SCREEN' 4 2
    MENU
    BLACK
    WHITE 'ADD CURRNT' 'ADD_CURRENT_INFO' 5 1
    MENU
    BLACK
```

```
    WHITE 'EDIT' 'EDIT_ELEM_INFO' 7 1
    MENU 'Element' 'EDIT_ELEM_INFO' 7 2
    MENU 'Current' 'EDIT_CURRENT_INFO' 8 2
    MENU
    BLACK
    WHITE 'DELETE' 'DELETE_ELEM_INFO' 9 1
    MENU 'Element' 'DELETE_ELEM_INFO' 9 2
    MENU 'Current' 'DELETE_CURRENT_INFO' 10 2
    MENU
    BLACK
    WHITE 'CHANGE' 'CHANGE_ELEM_INFO' 11 1
    MENU 'Element' 'CHANGE_ELEM_INFO' 11 2
    MENU 'Global' 'CHANGE_GLOBAL_INFO' 12 1
    MENU 'Current' 'CHANGE_CURRENT_INFO' 12 2
    MENU
    BLACK
    WHITE 'LIST' 'LIST_GLOBAL_INFO' 13 1
  END_IF
END_DEFINE
```

Sie können sehen, dass der angezeigte Text für Zeile 1/Spalte 1 `ADD SELECT` ist. Der entsprechende Befehl (manchmal als "Aktionstext" bezeichnet) ist `ADD_ELEM_INFO`. Der angezeigte Text für Zeile 8/Spalte 2 ist `Current`. Der entsprechende Befehl ist `EDIT_CURRENT_INFO`.

Verwenden Sie diese Methode, um den Inhalt von Menüs zu ermitteln.

Zusammenfassung

Kurz zusammengefaßt, wenn Sie wissen wollen, welche Befehle oder Funktionen Sie in einem Makro verwenden sollen, um eine bestimmte Aufgabe zu erreichen, so beachten Sie nachfolgende Auflistung:

- Entscheiden Sie, wie Sie Aufgabe unter Verwendung der Benutzeroberfläche (interface) lösen würden.
- Finden Sie heraus, welcher Befehl sich "hinter" dem Tablett- oder Menüfeld befindet.
- Lesen Sie falls erforderlich die Beschreibung dieses Befehls in der Online-Hilfe nach.
- Verwenden Sie den Befehl und die korrekten Optionen in Ihrem Makro.

12

Anpassung

Beim Start von Creo Elements/Direct Drafting werden vom Programm mehrere Dateien, in denen unter anderem Bildschirmmenüs und Creo Elements/Direct Drafting Umgebung definiert sind, gelesen und ausgeführt. Bei diesen Dateien handelt es sich um ausführbare Programme (so genannte Makros), die Sie bearbeiten und unter neuem Dateinamen speichern können, um sie bei Bedarf zu laden. Aus diesem Grund ist es möglich, alternative Definitionen für die Bildschirmmenüs und die Umgebung von Creo Elements/Direct Drafting zu erstellen. Dieser Vorgang wird als Anpassung bezeichnet.

Creo Elements/Direct Drafting Umgebung

Die Creo Elements/Direct Drafting Umgebung umfasst beispielsweise Angaben zu Größe und Farbe der Maßangaben, Zeichnungsmaßstab, Farbe der Zeichnungs- und Hilfslinien auf dem Bildschirm, Maßeinheiten usw.

Alle diese Grundeinstellungen werden mit Hilfe der über 70 verfügbaren Umgebungsfunktionen festgelegt. Ein Beispiel für eine Umgebungsfunktion ist `DIM_COLOR`. Sie kann so eingestellt werden, dass Bemaßungen mit den verfügbaren Farben erstellt werden.

Die aktuellen Grundeinstellungen der Umgebungsfunktionen können in einer Datei gespeichert werden. Die folgende Abbildung zeigt einen Auszug aus einer Creo Elements/Direct Drafting Umgebungsdatei:

```
MAX_FEEDBACK 100
CONFIGURE_EDITOR '$' 1 79
UNITS 1 MM
UNITS 1 DEG
CS_REF_PT 0,0
CS_AXIS 1,0  0,1
FOLLOW OFF
GRID_FACTOR 10
CURRENT_FONT 'hp_block_v'
TEXT_FRAME OFF
TEXT_ANGLE 0
TEXT_ADJUST 1
TEXT_LINESPACE 2.2
TEXT_FILL OFF
TEXT_SIZE 1
TEXT_RATIO 1
TEXT_SLANT 0
LINE WHITE SOLID END
C_LINE RED DOTTED END
TEXT WHITE END
SPLITTING ON
ARROW_FILL ON
```

Beim Start werden die Funktionen wie in der Liste dargestellt eingestellt, um eine Standardumgebung bereitzustellen. Während der normalen Verwendung werden die Einstellungen im Bildschirmmenü wie erforderlich geändert.

Die Anpassung kann auch durch Editieren der Creo Elements/Direct Drafting Umgebung erfolgen. Dies hat den Vorteil, dass Sie den Status sämtlicher Umgebungsfunktionen gleichzeitig überblicken können.

Eine angepasste Umgebung kann in einer speziellen Datei gespeichert werden. Mit dieser Datei lässt sich bei Bedarf dieselbe Umgebung wiederherstellen.

Creo Elements/Direct Drafting Umgebung anpassen

Daraufhin wird die aktuelle Systemumgebung angezeigt, in der Sie dann die gewünschten Änderungen vornehmen können. Die eingegebenen Änderungen sind nach dem Drücken der Tastenkombination `[CTRL] [D]` wirksam. Sie können die aktuelle Systemumgebung für die spätere Verwendung in einer Datei speichern. Dazu wählen Sie **KONFIG SP** im Bildschirmmenü aus und geben Folgendes ein:

```
'env_filename'
```

Für env_filename können Sie einen Namen Ihrer Wahl eingeben.

Wenn Sie die gespeicherte Datei ausführen und auf diese Weise die Systemumgebung wieder umdefinieren wollen, geben Sie folgenden Befehl ein:

```
INPUT  'env_filename'
```

Bildschirmmenüs erstellen

Der Bildschirm ist aus einem Menü (über das Befehle eingegeben werden) und einem Arbeitsbereich (in dem Zeichnungen erstellt werden) aufgebaut. Das Menü besteht aus einzelnen Menüfeldern, über die Sie Befehle direkt ausführen können. In diesen Feldern kann der Befehlsname voll ausgeschrieben oder abgekürzt angezeigt werden.

Eine typische Menüauflistung ist am Anfang von Benutzeroberfläche und Befehlssatz auf Seite 100 dargestellt. An diesem Beispiel sieht man, dass für alle Zeilen und Spalten zuerst der auf dem Bildschirm sichtbare Text und anschließend das Befehlswort aufgeführt ist.

Durch Eingabe von EDIT_MACRO und Wahl eines Bildschirmmenüfelds können Sie die Namen der Systemmakros für die Bildschirmmenüs anzeigen. Es wird folgendes angezeigt:

```
  DEFINE action_text
    action
  END_DEFINE
```

In diesem Beispiel ist action der Name des Makros, welches das Bildschirmmenü anzeigt. Jedes dieser Makros kann editiert und für spätere Verwendung in einer Datei gespeichert werden.

Im Folgenden wird erläutert, wie Sie das Makro speichern, das ein Bildschirmmenü anzeigt. Verwenden wir das CREATE 1-Menü als Beispiel:

1. Verwenden Sie EDIT_MACRO, um den Makronamen über das Bildschirmmenü zu erhalten. Sie stellen fest, dass das Makro für das CREATE 1-Menü Sm_create_1 ist.
2. Geben Sie Folgendes in die Befehlszeile ein:
   ```
   SAVE_MACRO Sm_create_1 'filename'
   ```

filename ist ein Name Ihrer Wahl.

Sie können nun die Datei editieren. Sie möchten die Inhalte der Datei evtl. zu Ihrer customize-Datei hinzufügen.

Menüvariablen

Im Folgenden sind die ersten Zeilen des CREATE 1-Menüs mit den Menüvariablen aufgeführt:

```
DEFINE Sm_create_1
  LET Lastmen 'Tm_create_1'
  IF (I_port)
    Check_i_port
  END_IF
  IF (NOT I_port)
    MENU_BUFFER ON
    CURRENT_MENU Sm_create_1_layout_name
```

```
    T_clear_menu
    Menu_control_icons
    MENU Colo0 Bcol5 CENTER 'CREATE 1' '' 1 3
.
.
.
    MENU Colo0 Bcol5 CENTER 'SPLINE' 'Tm_create_3' 25 2
    Eight_menu_slots_add
  END_IF
END_DEFINE
```

Die einzelnen Menüvariablen haben folgende Bedeutung:

- `I_port`

  Die nächste Variable `I_port` bezieht sich auf das Feld **GROSS/KLEIN** unterhalb von **FENST** auf dem zuvor verwendeten Tablett. Bei Auswahl von **LARGE/SMALL** wird das Makro `Tm_port_large_slash_small` ausgeführt. Die Koordinaten des aktuellen Darstellungsfensters werden gespeichert. Daraufhin erscheint das aktuelle Fenster vergrößert, so dass der Zeichenbereich den gesamten Bildschirm einnimmt. Gleichzeitig nimmt die Variable `I_port` den Wert `1` an, wodurch dem System mitgeteilt wird, dass das darstellungsfenster den gesamten Bildschirm einnimmt. Dies bedeutet, dass kein Menü auf dem Bildschirm angezeigt werden kann. Wenn **LARGE/SMALL** erneut ausgewählt wird, wird die ursprüngliche Größe des aktuellen Darstellungsfensters wiederhergestellt. Die Variable `I_port` wird auf `0` eingestellt, sodass Menüs wie gewohnt geändert werden können.

- `CURRENT_MENU Sm_create_1_layout_name`

  Dadurch wird der Name des aktuellen Bildschirmmenüs auf `Sm_create_1` eingestellt. (`Sm_create_1_layout_name` ist ein Makro, das in eine Zeichenfolge erweitert wird.) Alle Menü-Layouts und Menübefehle bleiben bis zur Verwendung eines Menüs mit einem anderen Namen dem aktuellen Menü zugeordnet. Für eigene Menüs sollten Sie " (leere Zeichenfolge) als Layout-Name benutzen.

- `T_clear_menu`

  Dieses Makro löscht den gesamten Text im Menü, sodass neuer Text eingefügt werden kann.

- `Menu_control_icons`

  Dieses Makro fügt die Symbole für das Verankern, Bewegen und Entfernen des Menüs hinzu.

- `MENU Colo0 Bcol5 CENTER 'CREATE 1' 1 3`

  Dies steht am Ende der ersten Menüzeile. `Colo0` und `Bcol5` werden auf eine Vordergrund- und eine Hintergrundfarbe erweitert, die gemäß `MEPELOOK` beim Systemstart definiert wird. Alle Systemmenüs verwenden diese Makros für die Definition der Vorder- und Hintergrundfarbe. Die folgende Tabelle zeigt, wie die Farben festgelegt werden, wenn `MEPELOOK` auf `0` gesetzt ist.

Vordergrund	Hintergrund	COLOR (MEPELOOK=0)
Colo0	Bcol0	BLACK
Colo1	Bcol1	WHITE
Colo2	Bcol2	RED
Colo3	Bcol3	GREEN
Colo4	Bcol4	BLUE
Colo5	Bcol5	YELLOW
Colo6	Bcol6	MAGENTA
Colo7	Bcol7	CYAN

- `Eight_menu_slots_add`

  `Eight_menu_slots_add` ist ein Makro, das die letzten 8 vom Benutzer anpassbaren Felder aktualisiert. Es wird in allen Standardmenüs verwendet. Sie können mit dem `Eight_menu_slots`-Makro eigene Menüelemente in diesen 8 Feldern platzieren. Die Syntax für dieses Makro lautet

  ```
  Eight_menu_slots 'title' 'action' Row_number Column_number
  ```

  Die Zeichenfolge `title` wird im Feld angezeigt. `action` ist der Aktionstext, der über das Menüfeld ausgeführt wird. `Row_number` und `Column_number` sind die entsprechenden Zeilen- und Spaltennummern. Für die Zeilennummer sind Werte von 26 bis 27 und für die Spaltennummer Werte von 1 bis 4 zulässig.

Aufbau der Menüfelder

Wahrscheinlich ist Ihnen schon aufgefallen, dass alle Felder in den Bildschirmmenüs (bis auf wenige Ausnahmen) gleich aufgebaut sind. Der Grund hierfür ist, dass der Aufbau der meisten Menüs auf der gleichen "Schablone" beruht. Diese Vorlage wird unter Verwendung des Makros `Layout_body_1` erzeugt. Dieses Makro wird beispielsweise verwendet, um das Layout des `Sm_create_1`-Menüs zu definieren:

```
DEFINE Layout_body_1
  Headline_height    ' | |               | '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
```

```
  Text_slot_height   '          |          '
  Text_slot_height   '          |          '
  Bottom_slot_height '    |     |     |    '
  Bottom_slot_height '    |     |     |    '
END_DEFINE
```

Die einzelnen Variablen im Makro oben haben folgende Bedeutung:

- `Headline_height`

  Höhe der Titelleiste des Menüblocks

- `Text_slot_height`

  Höhe der Menüfelder, die am Bildschirm sichtbaren Text enthalten

- `Bottom_slot_height`

  Höhe der Kästchen am unteren Rand jedes Menüs

Die Werte für die verschiedenen Höhenangaben der Menüfelder werden beim Systemstart errechnet. Hierdurch passt sich die Größe der Menüfelder der Fenster- bzw. Anzeigegröße in Creo Elements/Direct Drafting an. Die Werte werden in der `hp_macro.m`-Datei definiert. Diese Definitionen können als Beispiele dafür verwendet werden, wie die Feldgrößen sich an verschiedene Fenster- oder Displaygrößen selbst anpassen können.

Dieses Vorlagenmakro wird in folgendem Makro verwendet, das das `CREATE 1`-Menülayout definiert:

```
DEFINE Sm_create_1_layout
  CURRENT_MENU 'Sm_create_1'
  CURRENT_SCREEN 1
  MENU_LAYOUT Menu_position RIGHT
    Layout_body_1
    Menu_home_point_top
  END
  MENU_STATUS ENABLE_INQ
  LET  Sm_create_1_layout_name 'Sm_create_1'
END_DEFINE
```

Die einzelnen Variablen im Makro oben haben folgende Bedeutung:

- `CURRENT_MENU 'Sm_create_1'`

  Dies legt den Namen für das `CREATE 1`-Menülayout auf `Sm_create_1` fest.

- `CURRENT_SCREEN 1`

  Damit wird sichergestellt, dass auch in einer Umgebung mit zwei Bildschirmen das Menü auf dem ersten Bildschirm angezeigt wird.

- `MENU_LAYOUT`

  Mit dem Befehl `MENU_LAYOUT` wird das Menü rechts auf dem Bildschirm angezeigt. Am Ende des `MENU_LAYOUT`-Befehls steht `END`.

- `Menu_position`

  Ein Makro mit dem Kennzeichner `LOWER`. Dieser kann auf `UPPER` eingestellt werden, falls erforderlich.

- `Layout_body_1`

  Diese Zeile ruft das Makro `Layout_body_1` auf.

- `Menu_home_point_top`

  Verschiebt das Menü an die richtige Position auf dem Bildschirm. Diese Position wird entsprechend des Makros `Menu_position` und der Benutzeroberflächenversion (nur Bildschirm) berechnet.
- `MENU_STATUS ENABLE_INQ`

  Dies entfernt den Abfrageschutz im Menü.
- `LET Sm_create_1_layout_name 'Sm_create_1'`

  Das Menü `CREATE 1` verwendet nicht den Layoutnamen direkt. Um das Layout zu aktivieren, wird die Variable `Sm_create_1_layout_name` verwendet. Daher muss `Sm_create_1_layout_name` auf die richtige Zeichenfolge definiert sein. Es ist sinnvoll, die tatsächliche Generierung des Menü-Layouts erst nach dem ersten Aufruf des betreffenden Menüs vorzunehmen. Bis zur ersten Anforderung definiert `Sm_create_1_layout_name` den Makronamen, der das Layout generiert.

Bildschirmmenüs mit Tabellen erstellen

Einige der Bildschirmmenüs werden mit den `TABLE`-Befehlen definiert, wie `TABLE_LAYOUT` und `SHOW_TABLE`. Tabellen werden verwendet, da sie die Darstellung von Systemstatusvariablen ermöglichen. Mithilfe des Tabellenbildlaufs können lange Listen angezeigt werden. Beispiele von Menüs mit Tabellen sind:

- `HIDDEN LINE`

Diese Menüs werden mit den Befehlen `MENU` und `TABLE` erzeugt.

Die Definition dieser Menüs ist in ASCII-Versionen der Dateien `hp_macro.m` und `hp_men_t.m` verfügbar.

Wenn Sie an diesen Menüs Änderungen oder Erweiterungen vornehmen wollen, verwenden Sie dazu Kopien der bestehenden Menüs.

Mit Tabellen definierte Menüs können nicht gelöscht oder überschrieben werden. Dies verhindert potenzielle Probleme, die durch Befehle wie `DELETE_TABLE_ALL` verursacht werden.

Wenn Sie eigene Menütabellen verwenden möchten, ist die Anweisung `SECURE_TABLE` in den zuvor erwähnten Dateien an das Ende der Tabellendefinitionen zu setzen. Die logischen Tabellen für die Standardmenüs sollten nicht verändert werden, da der Creo Elements/Direct Drafting Code auf diese zugreift.

Nähere Angaben zur Verwendung von Tabellen können in der Online-Hilfe nachgelesen werden.

Bildschirmmenüs anpassen

Bildschirmmenüs können folgendermaßen angepasst werden:

- Neue Befehle in vorhandene Menüs integrieren. Fügen Sie den am Bildschirm sichtbaren Text hinzu und geben Sie den neuen Befehl als Befehlswort ein.
- Ein eigenes Makro auf ein leeres Menüfeld legen. Fügen Sie den am Bildschirm sichtbaren Text hinzu und geben Sie den Makronamen als Befehlswort ein.
- Neue Menüs in Anlehnung an den vorhandenen Menüaufbau erstellen.
- Neue Menüs mit neuem Menüaufbau erstellen.
- Neue Fensteranordnungen definieren und bei Bedarf abrufen.

Menüfelder für lokale Verzeichnisse definieren

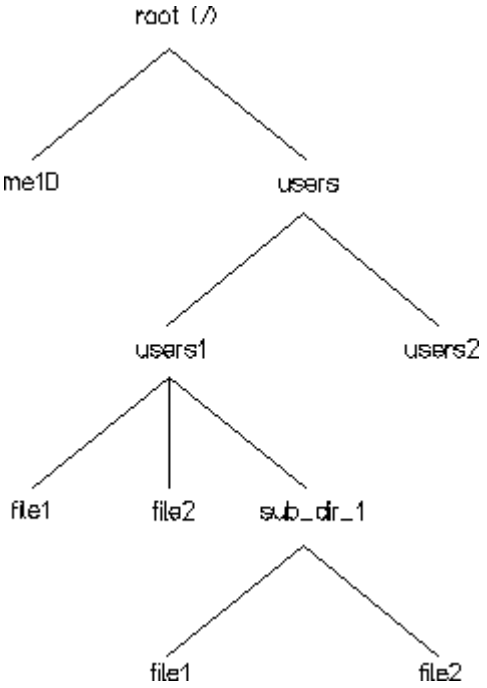
Bei der Arbeit mit Creo Elements/Direct Drafting müssen Sie immer wieder Dateien (mit eigenen Makros, Anpassungen, Zeichnungen, Körpern usw.) auf Platte speichern. Diese Dateien werden normalerweise in Ihrem eigenen Benutzerverzeichnis gespeichert (auch als Basisverzeichnis bezeichnet). Dieses Verzeichnis ist von Verzeichnissen anderer Benutzer getrennt.

Plattformabhängigkeiten

Falls Sie Creo Elements/Direct Drafting aus Ihrem Basisverzeichnis heraus starten, werden alle Dateien beim Speichern in diesem Verzeichnis gespeichert, es sein denn, Sie spezifizieren einen anderen Pfad. In einer Windows Umgebung können Sie Creo Elements/Direct Drafting so installieren, dass es immer aus Ihrem Basisverzeichnis startet. Mehr dazu finden Sie im Handbuch zum Betriebssystem gehörigen Konfigurationshandbuch.

Abbildung 23. Verzeichnishierarchie-Struktur auf Seite 111 zeigt eine typische Hierarchie von Verzeichnissen. Ihr System kann anders aussehen.

Abbildung 23. Verzeichnishierarchie-Struktur



An der Spitze der Baumstruktur steht das Root-Verzeichnis und wird durch einen Schrägstrich (/) dargestellt. Darunter befindet sich ein Verzeichnis namens `users`. Unter `users` befindet sich Ihr eigenes Benutzerverzeichnis, das Ihrem Anmeldenamen entspricht, sowie die Verzeichnisse anderer Benutzer. Das eigene Benutzerverzeichnis kann Dateien und Unterverzeichnisse enthalten. Normalerweise werden Sie Ihre Dateien im eigenen Benutzerverzeichnis oder darin enthaltenen Unterverzeichnissen speichern.Es ist jedoch auch möglich, Dateien in den Verzeichnissen anderer Benutzer zu speichern, wenn diese Verzeichnisse nicht schreibgeschützt sind.

Unter dem Stammverzeichnis befindet sich ein Verzeichnis namens `me10`, das die Creo Elements/Direct Drafting Software beinhaltet. Andere Dateien und Verzeichnisse werden von Ihrem Betriebssystem verwendet. Diese müssen Sie berücksichtigen.

Dateien und Verzeichnisse, die in derselben Verzweigung der Baumstruktur stehen, werden mit einem Schrägstrich voneinander getrennt, wobei der erste Schrägstrich für das Root-Verzeichnis steht. Die vollständige Pfadangabe für eine Datei im eigenen Benutzerverzeichnis sieht folgendermaßen aus:

```
/users/user_directory/filename
```

user_directory ist das Verzeichnis mit dem Namen, den Sie zum Anmelden beim System verwendet haben, und filename ist eine der persönlichen Dateien.

Hinweis

Sie verwenden entweder den Schrägstrich [(/)] oder den umgekehrten Schrägstrich [(\)], um Verzeichnisse und Dateien beim Angeben des Pfades in Creo Elements/Direct Drafting voneinander zu trennen. Die nachfolgenden beiden Pfade spezifizieren dieselbe Datei:

```
/users/user1/file2
\users\user1\file2
```

Beim Anmelden am System ist Ihr eigenes Benutzerverzeichnis das aktuelle Verzeichnis. Wenn Sie zum Aufrufen einer Datei lediglich den Dateinamen angeben, sucht das System zunächst im aktuellen Verzeichnis. Wenn sie nicht vorhanden ist, sucht das System im Verzeichnis `me10\me10` oder in den Verzeichnissen im Pfad `SEARCH`. Der Pfad `SEARCH` wird in der Umgebungsdatei angegeben.

Wenn Sie eine Datei, die nicht im aktuellen Verzeichnis steht, ohne Angabe des `SEARCH`-Pfades aufrufen möchten, müssen Sie den vollständigen Pfad beginnend mit dem Stammverzeichnis angeben.

Zum Erstellen eines Unterverzeichnisses im aktuellen Verzeichnis geben Sie folgenden Befehl in der Creo Elements/Direct Drafting Befehlszeile ein:

```
CREATE_DIRECTORY 'subdirectory'
```

`subdirectory` ist ein Name Ihrer Wahl.

Das Unterverzeichnis rufen Sie auf, indem Sie VERZ AKT im Bildschirmmenü DATEI auswählen und den Namen des neuen Verzeichnisses angeben. Wenn Sie in `subdirectory` eine Datei erstellen, lautet die vollständige Pfadangabe für diese Datei:

```
/users/user_directory/subdirectory/filename
```

Im folgenden wird anhand eines Beispiels gezeigt, wie Sie ein angepasstes Menü zum schnelleren Wechseln zwischen verschiedenen Verzeichnissen erstellen können.

Beispiel – Erzeugen eines eigenen Verzeichnismenüs

Erstellen Sie mit DATEI EDIT im Bildschirmmenü DATEI eine Datei mit einem Namen Ihrer Wahl, und geben Sie den Inhalt eines Makros ein. Beispiel:

```
DEFINE Directory
  IF (NOT I_port)
    CURRENT_MENU ''
    MENU Colo1 Bcol0 '' '' BOX  1 1 38 7
    MENU Colo0 Bcol5 CENTER 'DIRECTORY' 1 1
    MENU Colo1 Bcol0 'users' 'CURRENT_DIRECTORY "/users"' 3 1
    MENU 'me10' 'CURRENT_DIRECTORY "/me10"' 3 2
    MENU 'Jim' 'CURRENT_DIRECTORY "/users/jim"' 4 1
  END_IF
END_DEFINE
```

Die erste MENU-Anweisung definiert die Menü-Überschrift. Die zweite MENU-Anweisung legt für alle Felder unterhalb der Titelleiste Schwarz als Hintergrundfarbe und Weiß für den eingegebenen Text fest.

Die dritte MENU-Anweisung definiert ein Feld, um das Verzeichnis der Benutzer darzustellen. Die darauf folgende Anweisung definiert das Verzeichnis, in dem die Creo Elements/Direct Drafting Software enthalten ist, usw.

Beachten Sie, dass alle CURRENT_DIRECTORY-Anweisungen den vollständigen Pfadnamen für die Verzeichnisse angeben, beginnend mit dem Stammverzeichnis (/). Hierdurch können die Verzeichnisse innerhalb der Verzeichnisbaumstruktur jederzeit gefunden werden.

Die Liste der MENU-Anweisungen kann so lange fortgeführt werden, bis die Felder im Menüaufbau gefüllt sind.

Die Zahlen am Ende jeder MENU-Anweisung geben die Reihen- und Spaltennummer des Feldes an.

In diesem Beispiel definiert die letzte MENU-Anweisung ein Feld zum Anzeigen des aktuellen Katalogs auf dem Bildschirm. Wenn das Menü angezeigt wird, können Sie das aktuelle Verzeichnis wechseln, indem Sie ein beliebiges (beschriftetes) Feld auswählen und das neue Verzeichnis anschließend durch Auswahl von CATALOG anzeigen.

Wenn Sie die Datei laden und das Makro ausführen, wird das Menü angezeigt. Wählen Sie für das gewünschte Verzeichnis das entsprechende Menüfeld aus.

Tastatur anpassen

Beim Arbeiten mit Creo Elements/Direct Drafting werden Sie einige Befehle häufiger verwenden als andere. In diesem Abschnitt wird erläutert, wie Sie über eine Anpassung der Tastatur die zum Aufrufen dieser Befehle benötigte Zeit minimieren können.

Die Tastatur weist 8 oder 12 Funktionstasten auf (F1 bis F8 oder F1 bis F12). Sie können diese Tasten alle selbst mit Funktionen belegen.

Beispielsweise, wenn Sie mit Befehlen arbeiten möchten, die im aktuellen Bildschirmmenü nicht vorhanden sind. Hierdurch ersparen Sie sich das Aufrufen des entsprechenden Bildschirmmenüs. Sehen Sie sich die Erläuterung zu DEFINE_KEY in der Online-Hilfe an.

Die Funktionstasten können auch mit Zeichen belegt werden, die auf der Tastatur nicht vorhanden sind. Jedem Zeichen ist eine Nummer, die sogenannte ASCII-Nummer, zugeordnet. Diese Nummern, von denen die meisten darstellbaren Zeichen zugeordnet sind, gehen von 0 bis 255. Einige Nummern sind bestimmten Steuerfunktionen zugeordnet (wie z.B. ESC, RETURN, TAB oder DEL). Eine Liste aller darstellbaren Zeichen finden Sie am Ende dieses Kapitels.

Den ASCII-Nummern 0-31, 127-159 und 255 sind keine darstellbaren Zeichen zugeordnet.

Zum Anzeigen eines darstellbaren Zeichens in der Eingabezeile geben Sie folgendes ein:

```
DISPLAY (CHR ascii_number)
```

ascii_number steht für die ASCII-Nummer des gewünschten Zeichens.

Zum Belegen einer Funktionstaste mit einem Zeichen geben Sie den folgenden Befehl ein:

```
DEFINE_KEY key_number (CHR ascii_number)
```

key_number ist eine Zahl von 1 bis 8 oder von 1 bis 12, die die anzupassende Funktionstaste darstellt. Beim nächsten Drücken der Funktionstaste wird das entsprechende Zeichen in der Eingabezeile angezeigt.

Hinweis

Wenn Sie den auf eine Funktionstaste gelegten Text von einem der Menüs **TEXT**, **SYMBOLE** oder **BEMASSEN** aus in eine Zeichnung einfügen möchten, kann sich das Aussehen des Texts in der Zeichnung von dem in der Eingabezeile unterscheiden. Es hängt davon ab, wie die Schriftart angepasst wurde.

Geben Sie zum Belegen einer Taste mit einem Befehl folgendes ein:

```
DEFINE_KEY key_number 'action_text'
```

action_text ist der Name des gewünschten Befehls. Durch Drücken der Taste erscheint der Befehl in der Eingabezeile. Um ihn auszuführen, drücken Sie [Return].

Der Befehlsname muss in Hochkommata eingeschlossen sein, da das Programm die Eingabe von Text erwartet. Wenn Sie einen Schlüssel anpassen, um ein Zeichen zu erzeugen, sind keine Anführungszeichen erforderlich, da (CHR ascii_number) vom System als Text betrachtet wird.

Die Funktionstasten können auch über eine Datei belegt werden. Wählen Sie **DATEI EDIT** im Bildschirmmenü **DATEI** aus, und geben Sie den Namen, unter dem die Datei auf Platte gespeichert werden soll, in Anführungszeichen ein. Geben Sie die erforderlichen Anweisungen ein, und speichern Sie die Datei mit [CTRL] [D].

Wählen Sie nun **INPUT** aus, und geben Sie den Dateinamen erneut ein. Daraufhin werden alle in der Datei definierten Tasten belegt.

Nach dem Abschalten des Systems werden die Definitionen der Funktionstasten gelöscht, unabhängig davon, ob Sie die Funktionstasten über die Tastatur oder eine Datei belegt haben. Wenn Sie die DEFINE_KEY-Anweisungen in einer Datei speichern, müssen Sie beim Systemstart zum Belegen der Funktionstasten lediglich diese Datei laden.

Es folgt ein Beispiel für eine Datei, mit der acht Funktionstasten belegt werden.

```
DEFINE_KEY 1 (#14 '3' #15)
DEFINE_KEY 2 (#14 '1' #15)
DEFINE_KEY 3 (#14 '2' #15)
DEFINE_KEY 4 'LOAD'
DEFINE_KEY 5 'STORE'
DEFINE_KEY 6 'CATALOG' "" SCREEN'
DEFINE_KEY 7 'INPUT'
DEFINE_KEY 8
   'INPUT "dir_file" DEFINE Tm_macros_1 directory END_DEFINE'
```

- Mit den ersten drei Tasten werden die Zeichen Grad (°), Phi und Plus-Minus (±) aus der Schriftart `hp_symbols` in die Zeichnung eingefügt.
- Mit Taste 4 und 5 werden Zeichnungen geladen und gespeichert.
- Taste 6 wird verwendet, um das aktuelle Verzeichnis auf dem Bildschirm zu katalogisieren. Taste 7 wird verwendet, um eine Datei zu laden.
- Mit Taste 8 wird eine Datei geladen und gleichzeitig das erste benutzerdefinierte Makrofeld definiert, so dass bei Auswahl des Feldes ein Makro ausgeführt wird. In diesem Beispiel ist `dir_file` die Datei, die im vorhergehenden Abschnitt "Für lokale Verzeichnisse anpassen" beschrieben wurde. Dieses Makro (`directory`) zeigt, wenn es ausgeführt wird, das Menü mit den Benutzerverzeichnissen auf dem Bildschirm an. Wenn die Datei im aktuellen Verzeichnis steht und Dateiname und Makroname die gleichen wie in der `DEFINE_KEY`-Anweisung sind, müssen Sie, nachdem Sie die Funktionstaste und anschließend `[Return]` gedrückt haben, zum Anzeigen des Menüs mit den Benutzerverzeichnissen lediglich das erste benutzerdefinierte Makrofeld auf dem Tablett auswählen.

Hinweis

Wenn Sie eine Funktionstaste so belegen, dass mit ihr gleich mehrere Makros hintereinander ausgeführt werden, darf nur das letzte Makro eine `READ`-Anweisung enthalten, in der der Benutzer zur Eingabe aufgefordert wird. Dies liegt daran, dass das nächste Wort im Makro vom Programm als Eingabe für die `READ`-Anweisung interpretiert wird.

Schriftarten - Allgemeines

Eine Schriftart ist ein Satz vordefinierter geometrischer Muster, die in eine Zeichnung eingefügt werden können. Jedes Muster in einer Schriftart besteht aus geraden Linien, die an Verbindungspunkten in einem Gitter zusammenlaufen und wird zum leichteren Aufrufen einem Zeichen auf der Tastatur zugeordnet.

Obwohl eine Schriftart primär zum Erstellen von Texten dient (z.B. für Standardbeschriftungen oder Maßangaben), können sie auch Sonderzeichen enthalten. Mit einem feinmaschigen Gitter können entsprechend detaillierte Sonderzeichen definiert werden.

Sie können Sonderzeichen innerhalb Ihrer eigenen angepassten Schriftart definieren und für spätere Verwendung auf Platte speichern. So könnten beispielsweise Warenzeichen von Firmen, Logos usw. selbst erstellt werden.

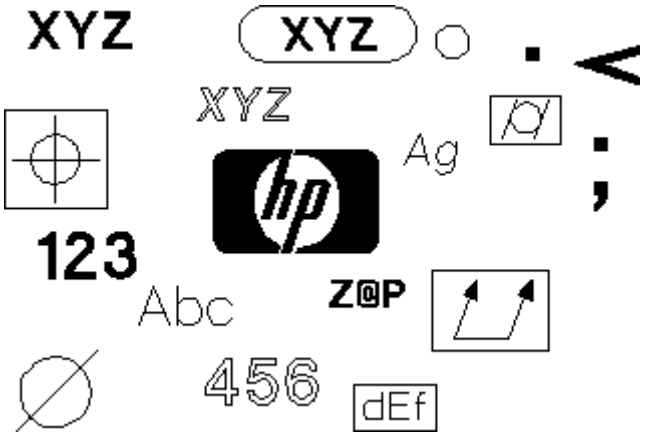
Textschriftarten sind sehr vielseitig. Die meisten Schriftarten haben folgende Haupteigenschaften:

- Zeichen können mit oder ohne Füllung dargestellt werden.
- Das Verhältnis Zeichenbreite/Zeichenhöhe kann festgelegt werden.

- Der Neigungswinkel der Zeichen kann festgelegt werden.
- Der Drehwinkel kann festgelegt werden (0 bis 360 Grad).

Beispiele:

Abbildung 24. Beispiele für Text und Symbole



Die Systemschriftarten

Beim Starten von Creo Elements/Direct Drafting werden mehrere Schriftartdateien in den Arbeitsspeicher geladen, davon ist jedoch nur eine Schriftart aktuell verfügbar. Über SCHRFT LST im Menü TEXT 2 können Sie ermitteln, welche Schriftarten im Arbeitsspeicher geladen sind.

Es erscheint eine Liste der geladenen und benutzten Schriftarten, aus der auch die aktuelle Schriftart ersichtlich ist. In dieser Aufstellung sind die Namen der Schriftarten im Hauptspeicher aufgeführt (nicht die Namen der Dateien, aus denen die Schriftarten erstellt werden!). Die Schriftarten `hp_Y14.5` und `hp_i3098_v` werden aus Dateien gleichen Namens geladen, die Schriftart `hp_def_font` wird jedoch innerhalb des Systems definiert. Bevor bei dieser Schriftart ein Zeichen erzeugt wird, wird ein Kästchen auf dem Bildschirm angezeigt, das angibt, wie viel Platz die Zeichen einnehmen werden.

Bei dieser Schriftart erscheint vor dem Erzeugen der Zeichen ein Kästchen auf dem Bildschirm, das anzeigt, wieviel Platz die Zeichen einnehmen werden.

Die folgende Tabelle enthält den Creo Elements/Direct Drafting und DOS Schriftartnamen und eine Beschreibung für jede Creo Elements/Direct Drafting Zeichnungsschriftart:

Font-Name	DOS-Dateiname	Beschreibung
`hp_block_c`	`hp_blk_c.fnt`	Schriftart mit gefüllten Zeichen (feste Zeichenbreite)
`hp_block_v`	`hp_blk_v.fnt`	Schriftart mit gefüllten Zeichen (variable Zeichenbreite)
`hp_d17_c`	`hp_d17_c.fnt`	Schriftart DIN17 (feste

Font-Name	DOS-Dateiname	Beschreibung
		Zeichenbreite)
hp_d17_v	hp_d17_v.fnt	Schriftart DIN17 (variable Zeichenbreite)
hp_jasc_c	hp_jas_c.fnt	Zeichensätze Japanisch (Kanji) und Griechisch (feste Zeichenbreite)
hp_jasc_v	hp_jas_v.fnt	Zeichensätze Japanisch (Kanji) und Griechisch (variable Zeichenbreite)
hp_kanji_c	hp_kan_c.fnt	
hp_symbols	hp_syms.fnt	Sonderzeichen für Bemaßungen: Grad, Plus-Minus, Phi, Minuten, usw.
hp_symbols2		Sonderzeichen für Bemaßungstoleranzen
hp_Y14.5	hp_y14_5.fnt	Nur für Symbole verwendet
hp_i3098_c	i3098_c.fnt	Schriftart ISO 3098 (feste Zeichenbreite)
hp_i3098_v	i3098_v.fnt	Schriftart ISO 3098 (variable Zeichenbreite)

Beispiel – Anzeigen der Zeichen in einer Schriftart

Mit dem folgenden Makro können Sie die Zeichen in der aktuellen Schriftart zusammen mit den zugehörigen ASCII-Nummern am Bildschirm anzeigen:

- Starten Sie den SCHRIFT-EDITOR im Menü **TEXT 2** (siehe auch Konstruieren und Zeichnen mit Creo Elements/Direct Drafting).
- Starten Sie das folgende Makro.

Nähere Angaben zu den ASCII-Nummern können Sie in dem Abschnitt "Tastatur anpassen" nachlesen.

Hinweis

Das Makro verwendet die **FENSTER**-Funktion, um die Koordinaten des aktuellen Fensters zu definieren. Soll die Anzeige der Zeichen nicht im aktuellen Fenster erfolgen, verwenden Sie AKTUELL, und wählen Sie das gewünschte Darstellungsfenster aus.

Daraufhin löscht das Makro die gesamte 2D-Geometrie.

```
DEFINE char_font
  LOCAL N
  LOCAL Y
  CHAR_LAYOUT
  (CHR 0) 24,27 23,21 21,17 19,15 16,13 13,12 11,12 8,13 5,15 3,17 1,21 0,27
          0,37 1,43 3,47 5,49 8,51 11,52 13,52 16,51 19,49 21,47 23,43 24,37
          24,27 36
  (CHR 1) 0,40 12,52 12,12 24
  (CHR 2) 0,46 2,49 5,51 9,52 15,52 19,51 22,49 24,46 24,42 22,38 0,12 24,12
          36
  (CHR 3) 6,36 16,36 BREAK 0,52 15,52 20,51 23,49 24,46 24,43 23,40 21,38
          19,37 16,36 20,35 22,33 23,31 24,27 24,22 23,18 21,15 18,13 15,12
          0,12 36
  (CHR 4) 18,52 0,20 24,20 BREAK 18,28 18,12 36
  (CHR 5) 22,52 0,52 0,36 15,36 20,35 23,32 24,28 24,22 23,18 21,15 18,13
          15,12 0,12 36
  (CHR 6) 0,26 1,30 3,33 6,35 9,36 15,36 18,35 21,33 23,30 24,26 24,22 23,18
          21,15 18,13 15,12 9,12 6,13 3,15 1,18 0,22 0,26 1,34 3,38 7,43
          18,52 36
  (CHR 7) 0,48 0,52 24,52 4,12 36
  (CHR 8) 10,36 5,34 2,31 0,27 0,21 2,17 5,14 9,12 15,12 19,14 22,17 24,21
          24,27 22,31 19,34 14,36 BREAK 7,51 10,52 14,52 17,51 20,49 22,46
          22,42 20,39 17,37 14,36 10,36 7,37 4,39 2,42 2,46 4,49 7,51 36
  (CHR 9) 23,30 9,30 6,31 3,33 1,36 0,40 0,42 1,46 3,49 6,51 9,52 15,52
          18,51 21,49 23,46 24,42 24,35 23,30 22,26 19,22 12,16 3,12 36
  END
  TEXT_ADJUST 1
  TEXT_ANGLE 0
  TEXT_FILL OFF
  TEXT_FRAME OFF
  TEXT_LINESPACE 2.2
  TEXT_RATIO 1
  TEXT_SIZE 3.5
  TEXT_SLANT 0
  DELETE ALL CONFIRM  (Caution! This line deletes all 2D geometry )
  WINDOW (-20,10) (80,-150)
  LET N 31
  LET Y 0
  DISPLAY_NO_WAIT 'Preparing character table'
  REPEAT
    LET N (N+1)
    IF ((N<127) OR (N>159))
      LET Y (Y-20)
      TEXT ( (CHR(N DIV 100)) + (CHR((N MOD 100) DIV 10)) + (CHR(N MOD 10))
      ( PNT_XY 0 Y ) (CHR N) ( PNT_XY 50 Y )
    END_IF
  UNTIL (N=254)
  END
END_DEFINE
```

Dieses Makro zeigt alle Zeichen in der aktuellen Schriftart an. Die Zeichen können über die Menüs TEXT, SYMBOLE und BEMASSEN in einer Zeichnung angezeigt werden.

Sie können ein einer beliebigen ASCII-Nummer (0 - 255) zugeordnetes Zeichen in einer Schriftart definieren und über ein Makro in einer Zeichnung anzeigen, wenn Sie aufgefordert werden, Text in eine Zeichnung einzufügen. Dies kann auch durch Eingabe von `(CHR n)` (ohne Anführungszeichen) in die Befehlszeile geschehen. (n gibt die ASCII-Nummer an.)

In der Regel sind Sie jedoch nur an den Zeichen interessiert, die durch die Eingabe von Text angezeigt werden können, wenn Sie von den Menüs **TEXT**, **SYMBOLE** und **BEMASSEN** zur Eingabe aufgefordert werden. Aus diesem Grund werden im Makro nur Zeichen aufgelistet, die mit normalen Tastaturzeichen oder angepassten Funktionstasten eingegeben werden können. Die ASCII-Zahlen, die Steuerzeichen entsprechen, wurden in der Liste ausgelassen. Dies sind die Zahlen `0-31`, `127-159` und `255`.

Das Makro beginnt, indem es lokale Variablen und anschließend die ASCII-Zahlen `0` bis `9` definiert, um die Zeichen `0` bis `9` zu erzeugen. Diesen Nummern werden zwar normalerweise keine darstellbaren Zeichen zugeordnet, wurden im Makro jedoch benutzt, um Überschneidungen mit den Zeichen in der aktuellen Schriftart zu vermeiden. Die Zeichen `0` bis `9` werden verwendet, um die Liste der ASCII-Zahlen in der Anzeige zu erzeugen.

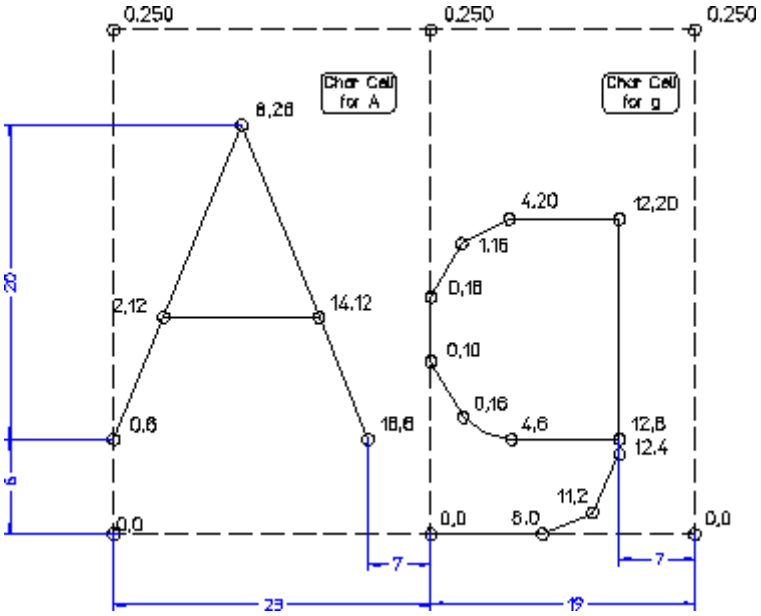
Die Textparameter werden auf ihre Standardwerte gesetzt und alle 2D-Zeichnungslinien gelöscht. Anschließend wird ein Ausschnitt im aktuellen Fenster definiert, in dem die Zeichen zusammen mit den zugehörigen ASCII-Nummern angezeigt werden. Wenn das Makro mit `hp_i3098_v` (Standard-Textschriftart) als aktueller Schriftart ausgeführt wird, sieht die Anzeige wie folgt aus:

```
032   SPACE

033      !

034      "

035      #

036      $
```

Eine Schriftart erstellen

Jedes Zeichen in einer Schriftart hat eine eigene Zelle. Die Zellengröße (in Gitterpunkten) kann bis zu 250 × 250 betragen. Die Anzahl der benötigten Gitterpunkte hängt von der gewünschten Auflösung ab (je höher die Auflösung, desto mehr Punkte werden benötigt). Die folgende Abbildung zeigt ein Beispiel, wie die Buchstaben `A` und `g` in einer Textschriftart definiert werden können. Auf die gleiche Art ist es möglich, jedes beliebige Zeichen oder Sonderzeichen zu erstellen.

Abbildung 27. Schriftart erzeugen



Die Zeichen werden in Zellen auf einem Gitter je nach gewünschter Auflösung aufgebaut. Die Zeichenhöhe entspricht dabei der Anzahl der vertikalen Gitterpunkte und dem mit der Funktion `TEXT_SIZE` festgelegten Wert. Die Zeichenbreite entspricht der Anzahl der horizontalen Gitterpunkte und kann darüber hinaus jedoch auch mit der Funktion `TEXT_RATIO` verändert werden. Diese Funktion beeinflusst das Verhältnis Breite/Höhe der Gitterpunkte und ist normalerweise auf den Wert `1` gesetzt. Details zu diesen Funktionen finden Sie in der Hilfedatei (`help`).

Der Befehl, der eine Textschriftart erzeugt, ist `DEFINE_FONT`. Die oben abgebildete Schriftart wird wie folgt definiert:

```
DEFINE_FONT 'example' 6 20 20
  CHAR_LAYOUT
  'A' 0,6 2,12 8,26 14,12 16,6 BREAK 14,12 2,12 23
  'g' 12,6 4,6 1,8 0,10 0,16 1,18 4,20 12,20 12,4 11,2 8,0 0,0 19
END
```

Der Name der Schriftart ist `Example`. Die erste Zahl danach (`6`) wird als Unterlänge bezeichnet. Die Unterlänge ist der Abstand des unteren Zellenrands zur Bezugslinie. In Textanwendungen kann sie als maximaler Abstand für das Ende eines Zeichens verwendet werden, wie `g` oder `p`.

Die zweite Zahl (`20`) wird als Höhe bezeichnet. Die Zeichenhöhe entspricht dem mit der Funktion `TEXT_SIZE` festgelegten Wert geteilt durch den Höhenwert. Bei Textanwendungen kann dieser Wert der Höhe eines Großbuchstabens entsprechen.

Wenn für `TEXT_SIZE` in obigem Beispiel der Standardwert 3.5 mm beibehalten wird, ist die Höhe des Buchstabens `A` 3.5 mm, da 20 Gitterpunkte die Höhe des Buchstabens darstellen und der Wert für die Höhe in der `DEFINE_FONT`-Funktion auch `20` ist.

Die dritte Zahl in dieser Zeile (`20`) wird als Breite bezeichnet. Die Breite entspricht der Anzahl der Gitterpunke in horizontaler Richtung, die dem Wert für die `TEXT_SIZE`-Funktion entspricht. In diesem Beispiel wurde für die Breite der

gleiche Wert wie für die Höhe verwendet, so dass die im Gitter definierten Proportionen erhalten bleiben. Wenn Sie die Angaben für die Breite auf `10` reduzieren, werden die Zeichen auf doppelte Breite vergrößert.

Der `CHAR_LAYOUT`-Befehl beginnt mit der Zeichendefinition. Jedes Zeichen wird mit einer Reihe von Gitterpunktkoordinaten definiert. Die Gitterpunkte werden durch Linien miteinander verbunden und bilden so die äußere Umrandung des Zeichens. Die `BREAK`-Funktion wird verwendet, um die gezeichnete Linie zu unterbrechen.

Die Zahl am Ende jeder Zeichendefinition gibt die Zeichenbreite in Gitterpunkten an. Hiermit wird der Abstand zwischen den einzelnen Zeichen gesteuert.

Der Befehl `END` wird verwendet, um die Schriftartdefinition zu verwenden.

Sie können die Befehle und Angaben zum Definieren einer neuen Schriftart über die Tastatur eingeben. Es ist jedoch wesentlich bequemer, eine Schriftart mit allen gewünschten Zeichendefinitionen mit dem Editor zu erstellen und als Datei zu speichern, die jederzeit GELADEN werden kann. Die Schriftart, die Sie auf eine dieser Arten definieren, wird zur aktuellen Schriftart und nur die in ihr enthaltenen Zeichen sind verfügbar. Wenn Sie Zeichen aus einer Systemschriftart oder einer anderen selbstdefinierten Schriftart benutzen wollen, muss die aktuelle Schriftart entsprechend neu angegeben werden.

Über **SCHR SET** im Bildschirmmenü **TEXT 1** können Sie aus einer Reihe von im System verfügbaren Schriftarten auswählen. Sie könne jedoch auch eine selbstdefinierte Schriftart zur aktuellen Schriftart machen, indem Sie folgenden Befehl eingeben:

```
CURRENT_FONT 'fontname'
```

`fontname` stellt dabei den Namen Ihrer Schriftart dar.

Schriftarten speichern – ASCII-Dateien und Binärdateien

Zum Erstellen einer Textdatei, die alle Anweisungen aus einer Systemschriftart enthält, müssen Sie zunächst überprüfen, ob die Schriftart geladen ist. Wählen Sie SCHRFT LST aus dem TEXT-Bildschirmmenü aus, um zu prüfen, ob die Schriftart in der Liste vorhanden ist. Drücken Sie dann `[ESC]`, um zum Menü zurückzukehren.

Wenn die Schriftart nicht in der Liste steht, muss sie zunächst geladen werden. Geben Sie dazu folgenden Befehl ein:

```
LOAD_FONT 'bin_filename'
```

`bin_filename` ist der Name der Binärdatei, die die Schriftart enthält.

Nun kann die Schriftart als ASCII-Datei auf Platte geschrieben werden.Geben Sie dazu folgenden Befehl ein:

```
SAVE_FONT 'fontname' 'ascii_filename'
```

`fontname` ist der Name der Schriftart und `ascii_filename` der Name für die Textdatei, die die Schriftart enthält. Die Datei kann nun in der üblichen Art editiert werden.

Eine Textdatei kann aus mehreren Schriftarten bestehen. Beim Laden einer Textdatei werden alle in ihr enthaltenen Schriftarten geladen.Die letzte Schriftart wird zur aktuellen Schriftart.

Zum Erstellen einer Binärdatei, die eine der selbstdefinierten Schriftarten enthalten soll, gehen Sie folgendermaßen vor:

1. Erstellen Sie eine Textdatei (wie oben beschrieben).
2. Laden Sie die Datei mit dem Befehl INPUT, so dass alle in ihr enthaltenen Schriftarten in das System geladen sind.
3. Speichern Sie jede einzelne Schriftart in einer separaten Binärdatei auf Platte. Geben Sie dazu folgenden Befehl ein:
   ```
   STORE_FONT 'fontname' 'bin_filename'
   ```
   `fontname` ist der Name der Schriftart und `bin_filename` der Name, unter dem die Binärdatei auf der Festplatte gespeichert werden soll.

Wie bereits in diesem Abschnitt beschrieben kann die Schriftart mit `LOAD_FONT` dann zurück in das System geladen werden.

Binärdateien können nicht im Editor geprüft werden. Der Vorteil von Binärdateien ist der schnelle Zugriff.

Systemstart anpassen

Bevor Sie damit beginnen, die im folgenden beschriebenen Schritte auszuführen, sollten Sie den Abschnitt über die Creo Elements/Direct Drafting Startdatei (startup) im zum Betriebssystem gehörigen Konfigurationshandbuch lesen.

Die `custom.m`-Datei wird nur beim Start geladen, wenn die geschweiften Klammern aus der folgenden Zeile am Ende der Datei `\me10\startup` entfernt wurden:

```
{ INPUT 'custom.m' }
```

Ist dies der Fall, sucht das System nach der Datei `custom.m` gemäß `SEARCH`-Pfad.

Dies bedeutet normalerweise, dass es zuerst im aktuellen Verzeichnis und dann im `\me10`-Verzeichnis sucht. Nähere Angaben zum `SEARCH`-Pfad können Sie im Kapitel zu den Umgebungsvariablen im zum Betriebssystem gehörigen Konfigurationshandbuch nachlesen. Nach dem Start erfolgt die Suche des `SEARCH`-Pfads durch Auswahl von KONFIG EDI im Bildschirmmenü KONFIGUR.

Beim Systemstart wird das aktuelle Verzeichnis zum Hauptverzeichnis (HOME) des angemeldeten Benutzers. Möglichkeiten für einen angepassten Systemstart:

Plattformabhängigkeiten

- Erstellen Sie die Datei `custom.m` im Verzeichnis, von dem aus Sie Creo Elements/Direct Drafting normalerweise starten, und editieren Sie die Datei nach Ihren speziellen Wünschen. Hierdurch werden die Anpassungsvorgänge beim Systemstart gesteuert.

- Verwenden Sie die Anpassungsdatei der Gruppe. In diesem Fall darf keine Datei namens `custom.m` in Ihrem Creo Elements/Direct Drafting Startverzeichnis vorhanden sein. In diesem Fall sucht das System nach der `\me10\custom.m`-Datei, die vom Systemadministrator erzeugt wurde. Die Anpassungsprozeduren gelten dann für die gesamte Gruppe, unabhängig aus welchem Verzeichnis heraus gestartet wird.

Systemstart für die ganze Gruppe anpassen

Die Datei `\me10\custom.m` wird in der Regel von einem Systemadministrator erzeugt. Diese Datei kann so editiert werden, dass sie Prozeduren für einen angepassten Systemstart enthält, die für alle Mitglieder Ihrer Gruppe, die keine `custom.m`-Datei im eigenen Creo Elements/Direct Drafting Startverzeichnis erstellt haben, gelten.

Die `\me10\custom.m`-Datei kann mit einem beliebigen Texteditor editiert werden. Es ist jedoch wahrscheinlicher, dass der Systemadministrator die Anpassungsprozeduren entwickeln und testen möchte, während er die Creo Elements/Direct Drafting Software als normaler Benutzer ausführt, und die Dateien dann in `\me10\custom.m` kopieren oder anhängen möchte.

Systemstart für die ganze Gruppe und Einzelbenutzer anpassen

Es wurde bereits erläutert, wie Benutzer ihre eigenen Startprozeduren implementieren können, indem Sie eine Datei namens `custom.m` in ihren Creo Elements/Direct Drafting Startverzeichnissen erstellen. Die folgenden Optionen sind für Benutzer verfügbar, die diese Datei erzeugt haben.

- Schließen Sie folgende Zeile als erste Zeile in Ihre `custom.m`-Datei ein:
  ```
  INPUT '\me10\custom.m'
  ```
  Die Anpassungsprozeduren, die der Systemverwalter für die ganze Gruppe erstellt hat, werden zuerst ausgeführt.Anschließend werden die eigenen Prozeduren ausgeführt.
- Wenn Sie den Systemstart ausschließlich nach den eigenen Wünschen anpassen wollen (und keine der Prozeduren für die Gruppe verwenden wollen), fügen Sie in Ihre Anpassungsdatei keine Zeile wie zuvor dargestellt ein.
- Wenn das System mit den Standardwerten (also ohne Anpassungen) arbeiten soll, erstellen Sie eine Datei `custom.m` im eigenen Basisverzeichnis und lassen diese jedoch leer.

Menü für angepassten Systemstart erstellen

Wenn Sie im eigenen Basis-/Startverzeichnis eine `custom.m`-Datei erstellt haben, können Sie alle Funktionen oder Befehle, die beim Systemstart ausgeführt werden sollen, in diese Datei stellen. Mit dieser Datei können jedoch auch Funktionstastenbelegungen definiert werden, so dass Sie andere Dateien nach Wunsch laden können.

Das folgende Beispiel zeigt, wie Sie das System anpassen können, damit nach dem Systemstart ein Menü mit verschiedenen Anpassungsprozeduren (die Sie unter Umständen noch benötigen) angezeigt wird. Editieren Sie die Datei `custom.m`, sodass sie die folgenden Zeilen enthält:

```
DEFINE_KEY 1 ((CHR 4)+'INPUT"project1"'+(CHR 13))
DEFINE_KEY 2 ((CHR 4)+'INPUT"project2"'+(CHR 13))
DEFINE_KEY 3 ((CHR 4)+'INPUT"project3"'+(CHR 13))
DEFINE_KEY 4 ((CHR 4)+'INPUT"project4"'+(CHR 13))
DEFINE_KEY 5 ((CHR 4)+'INPUT"project5"'+(CHR 13))
DEFINE_KEY 6 ((CHR 4)+'INPUT"project6"'+(CHR 13))
DEFINE_KEY 7 ((CHR 4)+'INPUT"project7"'+(CHR 13))
DEFINE_KEY 8 ((CHR 4)+'INPUT"project8"'+(CHR 13))
EDIT_FILE"menu" {Must be last line in file}
```

In diesem Beispiel sind die Namen `project1` bis `project7` die Namen der Dateien mit Anpassungsprozeduren für verschiedene Projekte. Die Dateien können beliebig benannt werden.

Die Funktionstasten können benutzt werden, wenn ein Menü im Bildschirm für die Eingabe im Textmodus angezeigt wird und auch, wenn eine Grafikanwendung läuft. Die Funktionstasten werden folgendermaßen definiert:

- `(CHR 4)` bedeutet `[CTRL] [D]`. Dies beendet die Alpha-Anzeige, sodass die Eingabeanweisung ausgeführt werden kann. Wenn Grafiken angezeigt werden und das System auf einen Befehl wartet, hat `[CTRL] [D]` keine Auswirkungen.
- `(CHR 13)` bedeutet `RETURN`. Dies verhindert, dass nach Drücken der Funktionstaste `[RETURN]` gedrückt werden muss.
- Die ersten sieben Tasten werden zum Laden der benötigten Dateien verwendet und Taste `[f8]` zum erneuten Anzeigen des Menüs. Der Name `menu` ist lediglich ein Beispiel. Sie können einen eigenen Namen für die Datei wählen, die Ihr Bildschirmmenü enthält.

Die `EDIT_FILE`-Anweisung am Ende der Anpassungsdatei zeigt das Bildschirmmenü an. Diese Anweisung sollte in der letzten Zeile stehen, da der Benutzer an dieser Stelle die Auswahl hat, ob er mit den Funktionstasten andere Dateien laden möchte oder nicht.

Die Menüdatei könnte z.B. folgendes Aussehen haben:

```
Press function key f1 to customize for project 1
Press function key f2 to customize for project 2
Press function key f3 to customize for project 3
Press function key f4 to customize for project 4
Press function key f5 to customize for project 5
Press function key f6 to customize for project 6
Press function key f7 to customize for project 7
Press ESC to display graphics
```

Durch Drücken einer Funktionstaste verschwindet das Menü und die Zeichnung wird angezeigt. Die für die betreffende Funktionstaste erstellte Datei wird mit `INPUT` geladen und die in ihr enthaltenen Anpassungsprozeduren ausgeführt. Dateien mit Anpassungsprozeduren sollten die folgende Zeile als letzte Anweisung enthalten:

```
DISPLAY 'Press function key f8 to display customizing menu'
```

Der Benutzer kann das Menü erneut anzeigen oder mit anderen Operationen weitermachen.

Hinweis

Geben Sie nichts auf dem Bildschirm ein, wenn das Anpassungsmenü angezeigt wird. `(CHR 4)` in den Funktionstastendefinitionen speichert eine Kopie der Menüdatei und überschreibt die alte Datei.

Schraffurmuster anpassen

Die `defaults.m`-Datei enthält die Schraffur für die Optionen Eisen, Stahl und Kupfer im Bildschirmmenü SCHRAFFUR. Mehr über die `defaults.m`-Datei erfahren Sie im zum Betriebssystem gehörigen Konfigurationshandbuch.

Die Schraffurmuster werden mit den folgenden Makros erzeugt:

```
DEFINE I_hatch_iron
  HATCH_ANGLE 45
  HATCH_DIST 5
  CURRENT_HATCH PATTERN 0 1 0 CYAN SOLID CONFIRM
END_DEFINE
DEFINE I_hatch_steel
  HATCH_ANGLE 45
  HATCH_DIST 15
  CURRENT_HATCH PATTERN 0 1 0 CYAN SOLID (1/3 ) 1 0 CYAN SOLID
  CONFIRM
END_DEFINE
DEFINE I_hatch_copper
  HATCH_ANGLE 45
  HATCH_DIST 5
  CURRENT_HATCH PATTERN 0 1 0 CYAN 0.5 1 0 CYAN DASHED CONFIRM
END_DEFINE
```

Es ist sehr einfach, diese Makros zu ändern oder neue Makros zum Erzeugen anderer Schraffurmuster zu schreiben.

Ihr Systemadministrator kann die `\me10\defaults.m`- Datei mit einem beliebigen Texteditor editieren. Alle Änderungen an der `\me10\defaults.m`-Datei werden jedoch überschrieben, wenn Sie Ihre Creo Elements/Direct Drafting Software neu installieren oder aktualisieren. Gehen Sie deshalb folgendermaßen vor:

1. Starten Sie Creo Elements/Direct Drafting.
2. Verwenden Sie den `COPY`-Befehl im Bildschirmmenü **DATEI**, um die `\me10\defaults.m`-Datei in das aktuelle Verzeichnis zu kopieren.

3. Editieren Sie die Datei mit dem Editor von Creo Elements/Direct Drafting. Löschen Sie alle Zeilen, die nicht geändert werden müssen, und ändern Sie die restlichen Zeilen.
4. Laden Sie die Datei mit `INPUT` zum Überprüfen der eingegebenen Änderungen.
5. Wenn die Ergebnisse zufriedenstellend sind, können Sie die folgende Zeile in die `\me10\custom.m`-Datei einschließen:
   ```
   INPUT 'filename'
   ```
   `filename` steht für die vollständige Pfadangabe und den Namen Ihrer Datei.

   Editieren Sie bei Bedarf `\me10\startup`, um die geschweiften Klammern (`{ }`) aus der Zeile zu entfernen:
   ```
   {INPUT 'custom.m'}
   ```
   Anschließend sollte die Zeile folgendermaßen aussehen:
   ```
   INPUT 'custom.m'
   ```

Wenn Sie Creo Elements/Direct Drafting starten, lädt die `startup`-Datei die Inhalte der `custom.m`-Datei. Dadurch wird Ihre Datei geladen.

Weitere Angaben zu Schraffuren erhalten Sie im Kapitel zum Konstruieren und Zeichnen mit Creo Elements/Direct Drafting.

ASCII-Code-Tabelle (ohne Steuerzeichen)

CHAR 32 = SPACE	CHAR 64 = @	CHAR 96 = `
CHAR 33 = !	CHAR 65 = A	CHAR 97 = a
CHAR 34 = "	CHAR 66 = B	CHAR 98 = b
CHAR 35 = #	CHAR 67 = C	CHAR 99 = c
CHAR 36 = $	CHAR 68 = D	CHAR 100 = d
CHAR 37 = %	CHAR 69 = E	CHAR 101 = e
CHAR 38 = &	CHAR 70 = F	CHAR 102 = f
CHAR 39 = '	CHAR 71 = G	CHAR 103 = g
CHAR 40 = (	CHAR 72 = H	CHAR 104 = h
CHAR 41 =)	CHAR 73 = I	CHAR 105 = i
CHAR 42 = *	CHAR 74 = J	CHAR 106 = j
CHAR 43 = +	CHAR 75 = K	CHAR 107 = k
CHAR 44 = ,	CHAR 76 = L	CHAR 108 = l
CHAR 45 = -	CHAR 77 = M	CHAR 109 = m
CHAR 46 = .	CHAR 78 = N	CHAR 110 = n
CHAR 47 = /	CHAR 79 = O	CHAR 111 = o
CHAR 48 = 0	CHAR 80 = P	CHAR 112 = p
CHAR 49 = 1	CHAR 81 = Q	CHAR 113 = q
CHAR 50 = 2	CHAR 82 = R	CHAR 114 = r
CHAR 51 = 3	CHAR 83 = S	CHAR 115 = s
CHAR 52 = 4	CHAR 84 = T	CHAR 116 = t
CHAR 53 = 5	CHAR 85 = U	CHAR 117 = u
CHAR 54 = 6	CHAR 86 = V	CHAR 118 = v
CHAR 55 = 7	CHAR 87 = W	CHAR 119 = w
CHAR 56 = 8	CHAR 88 = X	CHAR 120 = x
CHAR 57 = 9	CHAR 89 = Y	CHAR 121 = y
CHAR 58 = :	CHAR 90 = Z	CHAR 122 = z
CHAR 59 = ;	CHAR 91 = [	CHAR 123 = {
CHAR 60 = <	CHAR 92 = \	CHAR 124 = |
CHAR 61 = =	CHAR 93 =]	CHAR 125 = }
CHAR 62 = >	CHAR 94 = ^	CHAR 126 = ~
CHAR 63 = ?	CHAR 95 = _	

ASCII-Code-Tabelle (ohne Steuerzeichen)

CHAR 160 = SPACE	CHAR 192 = â	CHAR 224 = Á
CHAR 161 = À	CHAR 193 = ê	CHAR 225 = Ã
CHAR 162 = Â	CHAR 194 = ô	CHAR 226 = ã
CHAR 163 = È	CHAR 195 = û	CHAR 227 = Đ
CHAR 164 = Ê	CHAR 196 = á	CHAR 228 = đ
CHAR 165 = Ë	CHAR 197 = é	CHAR 229 = Ĩ
CHAR 166 = Î	CHAR 198 = ó	CHAR 230 = ĩ
CHAR 167 = Ï	CHAR 199 = ú	CHAR 231 = Õ
CHAR 168 = ´	CHAR 200 = à	CHAR 232 = Ò
CHAR 169 = `	CHAR 201 = è	CHAR 233 = õ
CHAR 170 = ^	CHAR 202 = ò	CHAR 234 = ő
CHAR 171 = ¨	CHAR 203 = ù	CHAR 235 = Š
CHAR 172 = ~	CHAR 204 = ä	CHAR 236 = š
CHAR 173 = Ù	CHAR 205 = ë	CHAR 237 = Ú
CHAR 174 = Û	CHAR 206 = ö	CHAR 238 = Ÿ
CHAR 175 = £	CHAR 207 = ü	CHAR 239 = ÿ
CHAR 176 = ¯	CHAR 208 = Å	CHAR 240 = Þ
CHAR 177 = ₰	CHAR 209 = î	CHAR 241 = þ
CHAR 178 = ₱	CHAR 210 = Ø	CHAR 242 = °
CHAR 179 = °	CHAR 211 = Æ	CHAR 243 = ø
CHAR 180 = Ç	CHAR 212 = å	CHAR 244 = ₹
CHAR 181 = ç	CHAR 213 = í	CHAR 245 = ₺
CHAR 182 = Ñ	CHAR 214 = ø	CHAR 246 = -
CHAR 183 = ñ	CHAR 215 = œ	CHAR 247 = ₹
CHAR 184 = ¡	CHAR 216 = Ä	CHAR 248 = ₹
CHAR 185 = ¿	CHAR 217 = ì	CHAR 249 = ₾
CHAR 186 = ¤	CHAR 218 = Ö	CHAR 250 = ₾
CHAR 187 = £	CHAR 219 = Ü	CHAR 251 = «
CHAR 188 = ¥	CHAR 220 = É	CHAR 252 = ▫
CHAR 189 = §	CHAR 221 = ï	CHAR 253 = »
CHAR 190 = ƒ	CHAR 222 = ß	CHAR 254 = ±
CHAR 191 = ç	CHAR 223 = Ô	CHAR 255 = ▯

13

Befehle und Funktionen (Überblick)

Dieses Kapitel enthält für alle Befehle und Funktionen jeweils eine kurze Definition.Die Erklärungen sind alphabetisch sortiert. Ausführlichere Beschreibungen zu den einzelnen Einträgen finden Sie in der Creo Elements/Direct Drafting Hilfe. Dazu geben Sie beispielsweise folgendes ein:

```
help catch
```

Daraufhin wird der Hilfetext zur Funktion FANGEN in Creo Elements/Direct Drafting angezeigt.

`ABS` (arithmetische Funktion)

Bei einem Zahlenargument: Gibt den absoluten Wert des Arguments zurück. Bei einem Vektorargument: Gibt die Länge des Vektors vom Ursprung bis zum Argumentpunkt zurück.

`ADD_CURRENT_INFO` (Befehl)

Bewirkt, dass der angegebene Text den aktuellen Informationen hinzugefügt wird.

`ADD_DIM_POSTFIX` (Befehl)

Fügt einer bestehenden Bemaßung ein Postfix hinzu.

`ADD_DIM_PREFIX` (Befehl)

Fügt einer bestehenden Bemaßung ein Präfix hinzu.

`ADD_DIM_SUBFIX` (Befehl)

Fügt einer bestehenden Bemaßung ein Subfix hinzu.

`ADD_DIM_SUPERFIX` (Befehl)

Fügt einer bestehenden Bemaßung ein Superfix hinzu.

`ADD_DIM_TOLERANCE` (Befehl)

Fügt einer ausgewählten Bemaßung eine Toleranz hinzu.

`ADD_ELEM_INFO` (Befehl)

Bewirkt, dass der angegebene Text den Informationen zu den angegebenen Elementen hinzugefügt wird.

ADU_ACCURACY (Funktion)

Gibt die Genauigkeit beim Vergleichen von Layouts an.

ADU_CHECK (Befehl)

Vergleicht zwei Layouts und versucht, das neue Layout neu zu beschriften.

ADU_CONFIRM_ANNOS (Befehl)

Markiert nach einer automatischen Aktualisierung von Beschriftungen jede einzelne Beschriftung als 'übernommen' (Standardfarbe = grün), 'neu erstellt' (Standardfarbe = rot) oder 'aktualisiert' (Standardfarbe = blau). Mit ADU_CONFIRM_ANNOS können Sie die Vorschläge des Systems übernehmen.

ADU_UPDATE_ANNOS (Befehl)

Ermöglicht das Austauschen von Bezugselementen für Beschriftungen. Damit können Sie beispielsweise das Bezugselement einer Bemaßung austauschen, ohne die Bemaßung löschen zu müssen.

ANALYZE_BSPLINE (Funktion)

Ermöglicht das Analysieren bestimmter Elemente und Werte, für eine ausgewählte B-Spline-Kurve.

AND (arithmetische Funktion)

Ergibt 1, wenn beide Argumente ungleich 0 sind. Ergibt sonst 0.

ANG (arithmetische Funktion)

Ergibt den Winkel zwischen der X-Achse und dem Vektor vom Ursprung zum Argument-Punkt.

ARC (Befehl)

Erstellt einen Bogen.

ARCCOS (arithmetische Funktion)

Ergibt den Hauptwert des Winkels, dessen Kosinus gleich dem Argument ist.

ARCSIN (arithmetische Funktion)

Ergibt den Hauptwert des Winkels, dessen Sinus gleich dem Argument ist.

ARCTAN (arithmetische Funktion)

Ergibt den Hauptwert des Winkels, dessen Tangens gleich dem Argument ist.

ARC_RESOLUTION (Funktion)

Legt die Auflösung von Bögen bei der Anzeige am Bildschirm fest (bei Verwendung von Anzeigelisten).

AREA_PROPERTY (Befehl)

Mißt die physikalischen Merkmale ausgewählter Bereiche.

ARROW_CURSOR (Funktion)

Legt die Form des Cursors in Tabellen und Menüs fest.

ARROW_FILL (Funktion)

Legt die Füllung der Pfeile bei Maß- und Hinweislinien fest.

ASSIST (Befehl)

Schaltet die Benutzerunterstützung (Copilot) ein.

AUTO_NEW_SCREEN (Funktion)

Gibt an, ob die Funktion NEW_SCREEN nach Anzeigen eines Creo Elements/ Direct Drafting Fensters wirksam sein soll.

AUTO_STORE_TIME (Funktion)

Speichert die Zeichnung automatisch. Die Häufigkeit wird vom Benutzer festgelegt.

BEEP (Funktion)

Gibt ein akustisches Signal über den Lautsprecher aus.

BLACK (Funktion)

Legt die Linienfarbe fest.

BLUE (Funktion)

Legt die Linienfarbe fest.

BSPLINE (Befehl)

Erstellt eine B-Spline Kurve.

BSPL_ADD_C_PNT (Befehl)

Fügt einen Stützpunkt zwischen zwei vorhandenen Stützpunkten hinzu.

BSPL_ADD_I_PNT (Befehl)

Fügt einen Interpolationspunkt zwischen zwei vorhandenen Stützpunkten hinzu.

BSPL_DEL_C_PNT (Befehl)

Löscht einen Stützpunkt aus einer Spline-Kurve.

BSPL_DEL_I_PNT (Befehl)

Löscht einen Interpolationspunkt aus einer Spline-Kurve.

BSPL_DEL_TANGENT (Befehl)

Löscht eine Tangente (Steigung).

BSPL_MOVE_C_PNT (Befehl)

Bewegt einen Stützpunkt an eine andere Position.

BSPL_MOVE_I_PNT (Befehl)

Bewegt einen Interpolationspunkt an eine andere Position.

BSPL_MOVE_PNT (Befehl)

Bewegt eine B-Spline-Kurve an eine andere Position.

BSPL_POINT_LENGTH (Befehl)

Unterteilt eine B-Spline-Kurve in gleichlange Abschnitte.

BSPL_POLYGON_FEEDBACK (Funktion)

Legt die Art der Rückmeldung beim Erstellen von B-Spline-Kurven fest.

CANCEL (Befehl)

Bricht die aktuelle Systemaktion ab. Kehrt zum Enter-Befehl zurück.

CANCEL_EDIT_DIM_TEXT (Befehl)

Bricht EDIT_DIM_TEXT ab. Der ursprüngliche Zustand einer editierten Bemaßung wird wiederhergestellt.

CATALOG (Funktion)

Gibt eine Auflistung des angegebenen Verzeichnisses (beispielsweise auf dem Drucker) aus.

CATALOG_LAYOUT (Befehl)

Bestimmt die äußere Form der von CATALOG angegebenen Verzeichnisinformationen.

CATCH (Funktion)

Legt den Fangmodus fest.Damit kann ein Punkt auf dem Bildschirm angewählt werden, ohne dass sich der Cursor genau auf diesem Punkt befindet.

CENTER (Befehl)

Zentriert den Menütext.

CENTERLINE (Befehl)

Erstellt eine Mittellinie für ein kreisförmiges Element. Die Mittellinie gehört zum kreisförmigen Element und wird beim Verschieben oder Löschen des Elements ebenfalls bewegt bzw. gelöscht.

CENTER_DASH_DASH (Funktion)

Legt die Linienart fest.

CHAMFER (Befehl)

Erstellt eine Fase.

CHANGE_COLOR (Befehl)

Ändert die Farbe der ausgewählten Elemente.

CHANGE_CURRENT_INFO (Funktion)

Globales Suchen und Ersetzen im aktuellen Informationstext.

CHANGE_DIM_ARROW (Befehl)

Ändert die aktuelle Begrenzung der Maßlinien.

CHANGE_DIM_COLOR (Befehl)

Ändert die Farben der Maß- und der Maßhilfslinien für die ausgewählten Bemaßungen.

`CHANGE_DIM_FORMAT` (Befehl)

Ändert das Format einer bestehenden Bemaßung.

`CHANGE_DIM_FRAME` (Befehl)

Ändert den ausgewählten Rahmentyp für die Maßangabe.

`CHANGE_DIM_LINEWIDTH` (Befehl)

Ändert die Stiftbreite (Linienstärke) der Maß- und der Maßhilfslinien für die ausgewählten Bemaßungen (siehe `CHANGE_DIM_PENSIZE`).

`CHANGE_DIM_PENSIZE` (Befehl)

Ändert die Stiftbreite (Linienstärke) der Maß- und der Maßhilfslinien für die ausgewählten Bemaßungen.

`CHANGE_DIM_TEXTS` (Befehl)

Ändert die Attribute für bestehende Maßangaben.

`CHANGE_DIM_TEXT_COLOR` (Befehl)

Ändert die Farbe für Maßangaben.

`CHANGE_DIM_TEXT_LOCATION` (Befehl)

Ändert die Position des Bemaßungstexts (über, auf oder unter der Linie).

`CHANGE_DIM_TEXT_ORIENTATION` (Befehl)

Ändert die Ausrichtung des Bemaßungstexts.

`CHANGE_DIM_VERTEX` (Befehl)

Bewegt die Maßhilfslinie zu einem anderen Eckpunkt.

`CHANGE_ELEM_INFO` (Befehl)

Globales Suchen und Ersetzen im angegebenen Informationstext.

`CHANGE_FILLET` (Befehl)

Ändert den Radius ausgewählter Rundungen.

`CHANGE_GLOBAL_INFO` (Funktion)

Globales Suchen und Ersetzen im Informationstext zu jedem Element im Speicher.

`CHANGE_HATCH_ANGLE` (Befehl)

Ändert den Neigungswinkel der ausgewählten Schraffur.

`CHANGE_HATCH_COLOR` (Befehl)

Ändert die Farbe der ausgewählten Schraffur.

`CHANGE_HATCH_DIST` (Befehl)

Ändert den Abstand zwischen den Schraffurlinien.

CHANGE_HATCH_LINETYPE (Befehl)

Ändert die Linienart der ausgewählten Schraffur.

CHANGE_HATCH_PATTERN (Befehl)

Ändert das Muster der ausgewählten Schraffur.

CHANGE_HATCH_REF_PT (Befehl)

Ändert den Bezugspunkt der ausgewählten Schraffur.

CHANGE_LEADER_ARROW (Befehl)

Ändert die Begrenzungen der ausgewählten Hinweislinien.

CHANGE_LEADER_ARROW_SIZE (Befehl)

Ändert die Größe der ausgewählten Hinweislinienbegrenzungen.

CHANGE_LINETYPE (Befehl)

Ändert die Linienart ausgewählter Elemente.

CHANGE_LINEWIDTH (Befehl)

Ändert die Linienstärke (Stiftbreite) ausgewählter Elemente (Geometrie, keine Hilfsgeometrie oder Punkt). (siehe CHANGE_DIM_PENSIZE).

CHANGE_LINESIZE (Befehl)

Ändert die Linienbreite ausgewählter Elemente (Geometrie, keine Hilfsgeometrie oder Punkt).

CHANGE_DIM_PENSIZE (Befehl)

Ändert die Stiftbreite (Linienstärke) der Maß- und der Maßhilfslinien für die ausgewählten Bemaßungen.

CHANGE_PART_REF_PT (Funktion)

Ändert den Bezugspunkt des aktiven Teils.

CHANGE_TABLE_SIZE (Funktion)

Ändert die Größe bestehender Tabellen, die nicht gegen Größenänderung gesichert sind.

CHANGE_TEXT (Befehl)

Ändert einen oder mehrere Texte zu einem neuen Text.

CHANGE_TEXT_ADJUST (Befehl)

Ändert die Ausrichtungsparameter des ausgewählten Textes.

CHANGE_TEXT_ANGLE (Befehl)

Ändert den Neigungswinkel des ausgewählten Texts.

CHANGE_TEXT_FILL (Befehl)

Schaltet die Füllung des ausgewählten Textes ein- bzw. aus.

CHANGE_TEXT_FONTNAME (Befehl)

Ändert die Schriftart des ausgewählten Texts.

CHANGE_TEXT_FRAME (Befehl)

Ändert den Rahmen des aktuellen Texts.

CHANGE_TEXT_LINESPACE (Befehl)

Ändert den Zeilenabstand bei mehrzeiligen Texten.

CHANGE_TEXT_RATIO (Befehl)

Ändert im ausgewählten Text das Verhältnis zwischen Zeichenbreite und Zeichenhöhe.

CHANGE_TEXT_SIZE (Befehl)

Ändert im ausgewählten Text die Größe der Zeichen.

CHANGE_TEXT_SLANT (Befehl)

Ändert im ausgewählten Text den Neigungswinkel der Zeichen.

CHANGE_VIEWPORT_COLOR (Funktion)

Ändert die Farbe für den Hintergrund des aktuellen Darstellungsfensters.

CHANGE_VIEWPORT_SIZE (Funktion)

Ändert die Größe des aktuellen Darstellungsfensters.

CHAR_LAYOUT (Befehl)

Definiert die Zeichen der aktuellen Schriftart.

CHECK_3D_GEO_MODIFY (Funktion)

Gibt an, ob beim Ändern eines Creo Elements/Direct Modeling Layouts (ADU) eine Warnung ausgegeben wird.

CHECK_BREAK (arithmetische Funktion)

Gibt 1 zurück, wenn die BREAK-Taste gedrückt wurde und IGNORE_BREAK aktiv war.

CHECK_DIM_DETAIL (Funktion)

Aktiviert/Inaktiviert die integrierte Bemaßungsprüfung.

CHECK_ERROR (arithmetische Funktion)

Ergibt 1, wenn seit dem letzten Aufruf der Funktion TRAP_ERROR mindestens ein Fehler aufgetreten ist (sonst 0).

CHECK_FONT_FILLABLE (Funktion)

Schreibt Zeichen der aktuellen Schriftart an die nicht füllbare, angegebene Zielposition.

CHECK_WINDOW (Funktion)

Aktiviert/Inaktiviert die integrierte Fensterprüfung.Dabei werden standardmäßig keine extrem große oder kleine Fenster zugelassen.

CHG_PIXEL_COLOR (Befehl)

Ändert die Pixelfarbe in die unter SET_COLOR angegebene Farbe.

CHR (arithmetische Funktion)

Wandelt eine Dezimalzahl in das entsprechende ASCII-Zeichen um. Beispiel: CHR(35) ist '#'.

CIRCLE (Befehl)

Erstellt einen Kreis.

CL_ABS_OFFSET (Befehl)

Legt den Abstand für die Mittellinie fest.

CL_COLOR (Befehl)

Legt die Farbe für die Mittellinie fest.

CL_LINETYPE (Befehl)

Legt die Linienart für die Mittellinie fest.

CL_LINEWIDTH (Befehl)

Legt die Linienstärke (Stiftbreite) für die Mittellinie fest. (Siehe CL_PENSIZE).

CL_PENSIZE (Befehl)

Legt die Linienstärke (Stiftbreite) für die Mittellinie fest.

CL_REL_OFFSET (Befehl)

Legt den Abstand für die Mittellinie (relativ zur Radiusbemaßung) fest.

CLEAN_DRAWING (Befehl)

Räumt Zeichnung auf (beispielsweise bei Überschneidungen).

CLIPBOARD_SIZE

Setzt die Größe des Windows Clipboard (Plotbereich) für nachfolgende Plots auf die Größe des Clipboard.

CLOSE_FILE (Funktion)

Schließt die angegebene Datei.

CMD_BG_COLOR (Funktion)

Ändert die Hintergrundfarbe der Befehlszeile.

CMD_TXT_COLOR (Funktion)

Ändert die Farbe des Befehlszeilentexts.

COLOR (Funktion)

Legt die aktuelle Farbe für Geometrie und Text fest.

COLOR_LTAB (Funktion)

Ändert die Farbe der angezeigten Position im Titelfeld und im Datenbereich der logischen Tabelle.

CONFIGURE_EDITOR (Funktion)

Konfiguriert den integrierten Bildschirmeditor.

`CONNECT_TABLE` (Funktion)

Verbindet eine Anzeigetabelle mit einer logischen Tabelle.

`CONTOUR` (Befehl)

Stutzt alle ausgewählten Elemente untereinander, so dass eine geschlossene Kontur entsteht.

`CONTROLZ_IS_EOF` (Funktion)

Gibt an, ob die Tastenkombination `Ctrl-Z` als Dateiendezeichen oder als normales Datenbyte interpretiert werden soll.

`CONVERT_C_TO_B_SPLINE` (Befehl)

Wandelt eine ausgewählte Spline-Kurve im alten Format ("C-Spline") in das neue Format ("B-Spline") um.

`CONVERT_DIM_TOLERANCE` (Befehl)

Ändert den Typ einer bestehenden Toleranz.

`CONVERT_DIM_UNIT` (Befehl)

Ändert die aktuellen Einheiten für die Bemaßungen von Linien und Winkeln.

`CONVERT_SPLINE` (Befehl)

Wandelt die ausgewählte Spline-Kurve in eine Reihe von Bögen und Linien um.

`COPY_FILE` (Funktion)

Kopiert die angegebene Quellendatei an die gewünschte Stelle.

`COS` (arithmetische Funktion)

Ergibt den Kosinus des Arguments.

`CREATE_DETAIL` (Befehl)

Erstellt mit dem angegebenen Vergrößerungsfaktor eine Detailansicht einer bestehenden Zeichnung.

`CREATE_DIRECTORY` (Funktion)

Erstellt ein neues Verzeichnis.

`CREATE_LTAB` (Funktion)

Erstellt eine neue logische Tabelle. Wenn diese Tabelle bereits vorhanden ist, wird der Befehl ohne Ausgabe einer Fehlermeldung ignoriert.

`CREATE_POLY` (Befehl)

Ermöglicht Ihnen, eine Polylinie aus den vorhandenen Elementen zu erzeugen. Anschließend können Sie die Polylinie als Einzelelement behandeln und entsprechend ändern.

`CREATE_SUBPART` (Befehl)

Erstellt ein neues Teil innerhalb des aktiven Teils unter Verwendung bestehender Elemente.

CREATE_VIEWPORT (Funktion)

Erstellt ein neues Darstellungsfenster.

CS_AXIS (Funktion)

Definiert die Achsen des relativen Koordinatensystems neu.Dabei bleibt die Lage des Ursprungs unverändert.

CS_MIRROR (Funktion)

Spiegelt die angegebene Achse des relativen Koordinatensystems an der anderen Achse.

CS_REF_PT (Funktion)

Ändert den Ursprung des relativen Koordinatensystems.

CS_ROTATE (Funktion)

Rotiert das relative Koordinatensystem um seinen Ursprung.

CS_SET (Funktion)

Ändert im relativen Koordinatensystem den Ursprung und den Winkel der Achsen zueinander.

CURRENT_DIM_TEXTS (Funktion)

Ändert die aktuellen Attribute für alle Maßangaben.

CURRENT_DIM_UNITS (Funktion)

Legt die aktuellen Einheiten für die Bemaßungen von Linien und Winkeln fest.

CURRENT_DIRECTORY (Funktion)

Macht das angegebene Verzeichnis zum aktuellen Verzeichnis.

CURRENT_FONT (Funktion)

Wählt die Schriftart für neuen Text aus.

CURRENT_HATCH_PATTERN (Funktion)

Legt das aktuelle Muster für die neu erstellten Schraffuren fest.

CURRENT_MENU (Funktion)

Legt den Namen für das aktuelle Bildschirmmenü fest.

CURRENT_SPOTLIGHT_ATTR (Funktion)

Ändert die Standardwerte für Farbe und Linienart bei SPOTLIGHT ON.

CURRENT_VERTEX_COLOR (Funktion)

Legt die Standardfarbe für Eckpunkte bei SHOW VERTEX ON fest.

CURRENT_VIEWPORT (Funktion)

Legt das aktuelle Darstellungsfenster fest.

CURSOR (Funktion)

Auswahl:Block- oder Strichcursor.

CURSOR_COORDINATES (Funktion)

Zeigt die aktuellen Koordinaten für den Cursor über dem Eingabebereich an.

CUT_MIDDLE

Schneidet an den Schnittpunkten zu zwei anderen Elementen ein Stück aus der Mitte eines Elements heraus und löscht dieses Mittelstück.

CYAN (Funktion)

Ändert die Standardfarbe in Cyan.

C_CIRCLE (Befehl)

Erstellt einen Hilfskreis.

C_COLOR (Funktion)

Legt die aktuelle Farbe für die Hilfslinien fest.

C_LINE (Befehl)

Erstellt Hilfslinien.

C_LINETYPE (Funktion)

Legt die aktuelle Linienart für Hilfsgeometrie (Hilfskreise und Hilfslinien) fest.

DASHED (Funktion)

Legt die Linienart fest.

DASH_CENTER (Funktion)

Legt die Linienart fest.

DATE (arithmetische Funktion)

Gibt Datum und Uhrzeit aus. Beispiel: '26-Oct-99 14:26:58'.

DA_DB_ADD (Befehl)

Fügt ein neues Teil zur Bemaßungsdatenbank hinzu.

DA_DB_DELETE (Befehl)

Löscht einen Eintrag aus der Bemaßungsdatenbank.

DA_DB_EXPORT (Befehl)

Kopiert einen Eintrag aus der Bemaßungsdatenbank in die aktuelle Zeichnung. Der neue Eintrag wird unter dem obersten Teil platziert.

DA_DB_FILL_TABLE (Befehl)

Löscht und füllt die Bemaßungsdatenbanktabelle. (Nicht für die Ausführung über die Befehlszeile bestimmt.)

DA_DB_INQ (Funktion)

Fragt Informationen über die Bemaßungsdatenbank ab. Diese Funktion unterstützt die Bildschirmschnittstelle des Bemaßungsmoduls. Sie ist im Allgemeinen nicht für die Ausführung über die Befehlszeile bestimmt.

DA_DB_LOAD (Befehl)

Läd die Bemaßungsdatenbank in den Arbeitsspeicher.

`DA_DB_MATCH` (Befehl)

Überträgt Bemaßungen aus der Bemaßungsdatenbank auf Zeichnungselemente.

`DA_DB_STORE` (Befehl)

Speichert die Bemaßungsdatenbank in eine Datei im MI-Format.

`DA_DB_UNLOAD` (Befehl)

Entläd die aktuelle Bemaßungsdatenbank aus dem Arbeitsspeicher.

`DA_DB_WIN_CREATE` (Funktion)

Erzeugt ein Fenster für die Bemaßungsdatenbank. (Nicht für die Ausführung über die Befehlszeile bestimmt.)

`DA_DB_WIN_LOC` (Funktion)

Legt die Position für das Bemaßungsdatenbankfenster fest. (Nicht für die Ausführung über die Befehlszeile bestimmt.)

`DA_DIM_ANGLE` (Befehl)

Erstellt Winkelbemaßungen.

`DA_DIM_ARC` (Befehl)

Erstellt Bogenbemaßungen.

`DA_DIM_AUTO_LOC` (Funktion)

Legt Parameter für die automatische Bemaßungspositionierung fest.

`DA_DIM_AUTO_STRATEGY` (Funktion)

Legt die Art der Prüfung auf Überschneidungen bei automatischer Bemaßungspositionierung fest.

`DA_DIM_CHAIN` (Befehl)

Erstellt Kettenbemaßungen.

`DA_DIM_CHAMFER` (Befehl)

Erstellt Fasenbemaßungen.

`DA_DIM_COORD` (Befehl)

Erstellt Bezugsbemaßung mit Koordinaten.

`DA_DIM_DATUM_LONG` (Befehl)

Erstellt Bezugsbemaßung mit langen Maßlinien.

`DA_DIM_DATUM_LONG_SYM` (Befehl)

Erstellt symmetrische Bezugsbemaßung mit langen Maßlinien.

`DA_DIM_DATUM_SHORT` (Befehl)

Erstellt symmetrische Bezugsbemaßung mit kurzen Maßlinien.

`DA_DIM_DELETE` (Befehl)

Löscht ein oder mehrere Segmente aus dem Bezugsbemaßungsstapel Löscht auch gesamte Bemaßungen.

`DA_DIM_DIAMETER` (Befehl)

Erstellt Bemaßungen für Durchmesser.

`DA_DIM_GEO_SENSE` (Befehl)

Ordnet geometrieabhängige Bemaßungen für ausgewählte Elemente zu.

`DA_DIM_HOLE_INSERTION` (Funktion)

Legt das Verhalten bei neuen Bemaßungen innerhalb von Schraffuren fest.

`DA_DIM_INCLINE` (Befehl)

Neigt Bemaßungen mit den Attributen Senkrecht oder Waagerecht.

`DA_DIM_INSERT` (Befehl)

Fügt neue Linienabschnitte in den Bezugsbemaßungspuffer ein.

`DA_DIM_LINE` (Befehl)

Erstellt einfache Linienbemaßungen.

`DA_DIM_LINE_SYM` (Befehl)

Erstellt symmetrische Linienbemaßungen.

`DA_DIM_PD_SCAN` (Befehl)

Ordnet Bemaßungen für ausgewählte Elemente anhand parametrischer Vorgaben zu.

`DA_DIM_RADIUS` (Befehl)

Erstellt Bemaßungen für Radien.

`DA_DIM_SHORT_SPACE` (Funktion)

Legt den Abstand zwischen Einträgen im Bezugsbemaßungspuffer fest.

`DA_FILTER_ACTIVATE` (Funktion)

Schaltet den Auswahlfilter für Bemaßungen ein.

`DA_FILTER_ADD` (Funktion)

Fügt neue Auswahlkriterien für Bemaßungsfilter hinzu. Diese Funktion ist für die Ausführung über die Bemaßungsmenüs bestimmt.

`DA_FILTER_CLEAR_GEOTYPES` (Funktion)

Löscht das Auswahlkriterium "Geometrieart" für Bemaßungsfilter. Diese Funktion ist für die Ausführung über die Bemaßungsmenüs bestimmt.

`DA_FILTER_CLEAR_LINETYPES` (Funktion)

Löscht das Auswahlkriterium "Linienart" für Bemaßungsfilter. Diese Funktion ist für die Ausführung über die Bemaßungsmenüs bestimmt.

`DA_FILTER_DEL_COLOR` (Funktion)

Löscht das Auswahlkriterium "Farbe" für Bemaßungsfilter.

DA_FILTER_DEL_ORIENT (Funktion)

Löscht das Auswahlkriterium "Linienneigung" für Bemaßungsfilter.

DA_FILTER_DEL_WIDTH (Funktion)

Löscht das Auswahlkriterium "Linienbreite" für Bemaßungsfilter.

DA_FILTER_DEL_LINESIZE (Funktion)

Löscht das Auswahlkriterium "Linienbreite" für Bemaßungsfilter.

DA_FILTER_DEL_PENSIZE (Funktion)

Löscht das Auswahlkriterium "Stiftbreite" für Bemaßungsfilter. (siehe DA_FILTER_DEL_WIDTH).

DA_FILTER_INQ (Funktion)

Fragt Informationen über Auswahlfilter für Bemaßungen ab. Diese Funktion unterstützt die Bildschirmschnittstelle des Bemaßungsmoduls. Sie ist im Allgemeinen nicht für die Ausführung über die Befehlszeile bestimmt.

DA_FILTER_REFRESH_LINEWIDTH (Funktion)

Aktualisiert den Inhalt der Auswahlfiltertabelle für die Stiftbreite. (Nicht für die Ausführung über die Befehlszeile bestimmt.) (siehe DA_FILTER_REFRESH_PENSIZE).

DA_FILTER_REFRESH_LINEWIDTH (Funktion)

Aktualisiert den Inhalt der Auswahlfiltertabelle für die Stiftbreite. (Nicht für die Ausführung über die Befehlszeile bestimmt.) (siehe DA_FILTER_REFRESH_PENSIZE).

DA_FILTER_REFRESH_LINESIZE (Funktion)

Aktualisiert den Inhalt der Auswahlfiltertabelle für die Linienbreite. (Nicht für die Ausführung über die Befehlszeile bestimmt.)

DA_FILTER_REFRESH_PENSIZE (Funktion)

Aktualisiert den Inhalt der Auswahlfiltertabelle für die Stiftbreite. (Nicht für die Ausführung über die Befehlszeile bestimmt.)

DA_FILTER_REFRESH_ORIENT (Funktion)

Aktualisiert den Inhalt der Auswahlfiltertabelle für die Linienneigung. (Nicht für die Ausführung über die Befehlszeile bestimmt.)

DA_FILTER_SET_NAME (Funktion)

Benennt den aktuellen Auswahlfilter für Bemaßungen.

DA_FILTER_STORE (Funktion)

Speichert den aktuellen Auswahlfilter für Bemaßungen in einer Datei.

DA_LINESIZE (Funktion)

Legt die Linienbreite der Bemaßung fest.

DA_PENSIZE (Funktion)

Legt die Stiftbreite der Bemaßung fest.

DA_MOVE_DIMENSION (Befehl)

Bewegt die Bemaßungen.

DA_NULL (Funktion)

Unterstützung der Bildschirmschnittstelle des Bemaßungsmoduls. Bewirkt nichts.

DA_STYLE_APPLY (Funktion)

Übernimmt die aktuellen Werksnormparameter für die ausgewählte Bemaßung.

DA_STYLE_DEFER_UPDATE (Funktion)

Verhindert eine Aktualisierung des Werksnormfensters.

DA_STYLE_ENABLE_UPDATE (Funktion)

Ermöglicht eine Aktualisierung des Werksnormfensters.

DA_STYLE_GET (Funktion)

Macht die Werksnorm der ausgewählten Bemaßung zur aktuellen Werksnorm.

DA_STYLE_INQ (Funktion)

Fragt Informationen über die aktuelle Werksnorm ab. Diese Funktion unterstützt die Bildschirmschnittstelle des Bemaßungsmoduls. Sie ist im Allgemeinen nicht für die Ausführung über die Befehlszeile bestimmt. (Nicht für die Ausführung über die Befehlszeile bestimmt.)

DA_STYLE_TYPE (Funktion)

Wählt eine neue Werksnorm aus.

DA_STYLE_UPDATE (Funktion)

Aktualisiert das Werksnormfenster.

DA_STYLE_WIN_CREATE (Funktion)

Erstellt ein Fenster, in dem die aktuelle Werksnorm grafisch darstellt wird.

DA_STYLE_WIN_LOC (Funktion)

Legt die Position für das Werksnormfenster fest. Unterstützung der Bildschirmschnittstelle des Bemaßungsmoduls. (Nicht für die Ausführung über die Befehlszeile bestimmt.)

DA_STYLE_WIN_RAISE (Funktion)

Ruft das Werksnormfenster NORMEN auf.

DA_WRITE_DIM_SETTINGS_MACRO (Funktion)

Schreibt die Einstellungen der aktuellen Werksnorm in eine Makrodatei.

DDE_ADD_TOPIC (Funktion)

Fügt ein neues Datenthema (string) zur Liste der geöffneten DDE-Themen.

DDE_CLOSE (arithmetische Funktion)

Schließt, beendet eine benannte DDE-Konversation.

DDE_ENABLE (Funktion)

Befähigt Creo Elements/Direct Drafting als DDE-Server zu agieren und auf Anforderungen von DDE-Clients zu reagieren. DDE steht für "Dynamic Data Exchange".

`DDE_EXECUTE` (arithmetische Funktion)

Schickt einen Befehl an die Applikation, die durch den angelegten Konversationsmoderator gewährleistet wird.

`DDE_INITIATE` (arithmetische Funktion)

Beginnt eine DDE-Konversation für die gewünschte Applikation zu einem gegebenen Thema. Etabliert einen Konversationsmoderator für nachfolgende DDE-Befehle.

`DDE_REMOVE_TOPIC` (Funktion)

Entfernt ein angegebenes Thema aus der Liste der geöffneten DDE-Themen.

`DDE_REQUEST` (arithmetische Funktion)

Fragt die entfernte DDE-Applikation nach dem Wert des gewünschten Datenthemas.

`DDE_SEND_ACK` (arithmetische Funktion)

Schickt eine bestätigende DDE-Meldung für eine etablierte Konversationsmoderation.

`DDE_WITHHOLD_ACK` (arithmetische Funktion)

Verhindert das Senden einer bestätigenden DDE-Meldung, die der Ausführung eines Aktionstexts folgt. Der aktive Konversationsmoderator wird stattdessen zurückgegeben.

`DEFINE` (Funktion)

Definiert ein Makro mit dem angegebenen Namen.

`DEFINE_CATALOG` (Funktion)

Gibt an, welche Informationen mit der Funktion CATALOG gedruckt werden sollen.

`DEFINE_FONT` (Funktion)

Definiert eine neue Schriftart.

`DEFINE_KEY` (Funktion)

Definiert die Belegung der angegebenen Funktionstaste. Eine derartige Taste heißt "Hotkey", "Smartkey" oder "Softkey".

`DEFINE_MOUSE_KEY` (Funktion)

Definiert die Belegung der Maustasten 1, 2 und 3.

`DELETE` (Befehl)

Löscht die ausgewählten Elemente.

`DELETE_CURRENT_INFO` (Funktion)

Löscht die aktuellen Informationen.

DELETE_DIMENSION (Befehl)

Löscht Bemaßungen.

DELETE_DIM_POSTFIX (Befehl)

Löscht ein angegebenes Postfix.

DELETE_DIM_PREFIX (Befehl)

Löscht ein Präfix einer Bemaßung.

DELETE_DIM_SUBFIX (Befehl)

Löscht ein Subfix einer Bemaßung.

DELETE_DIM_SUPERFIX (Befehl)

Löscht ein Superfix einer Bemaßung.

DELETE_DIM_TOLERANCE (Befehl)

Löscht die Toleranz einer angegebenen Bemaßung.

DELETE_ELEM_INFO (Befehl)

Löscht die einem Element zugeordneten Informationen.

DELETE_FONT (Funktion)

Löscht die Definition der angegebenen Schriftart.

DELETE_HATCH (Befehl)

Löscht die ausgewählte Schraffur.

DELETE_LABEL (Befehl)

Löscht im aktiven Teil alle mit dem Befehl LABEL erzeugten Kenndaten.

DELETE_LTAB_ROW (Funktion)

Löscht die angegebene Zeile aus der angegebenen Benutzertabelle.

DELETE_MACRO (Funktion)

Löscht ein angegebenes Makro oder alle Makros.

DELETE_MENU (Funktion)

Löscht die Definition des Bildschirmmenüs.

DELETE_TABLE (Befehl)

Löscht eine bestehende ungesicherte Tabelle.

DELETE_VIEWPORT (Funktion)

Löscht das angegebene Darstellungsfenster.

DIM_ANGLE (Befehl)

Erstellt eine Bemaßung für einen Winkel.

DIM_ARC (Befehl)

Erstellt eine Bemaßung für einen Bogen.

DIM_ARROW (Funktion)

Legt die Art des Maßpfeiles fest.

DIM_BREAK_RESTORE (Funktion)

Legt den Umgang mit Maßlinen und Maßhilfslinien nach Bewegung fest.

DIM_BROKEN (Funktion)

Legt fest, wie die Maßlinien dargestellt werden, wenn der Bemaßungstext nicht zwischen die Maßhilfslinien passt.

DIM_CATCH_LINES (Funktion)

Legt auswählbare Bemaßungselemente fest.

DIM_CATCH_RANGE (Funktion)

Setzt den Fangbereich der Maßangabe auf die Mitte der Maßhilfslinien.

DIM_CHAIN (Befehl)

Erstellt eine Kettenbemaßung.

DIM_CHAMFER (Funktion)

Erstellt eine Bemaßung für eine Fase nach JIS.

DIM_COLOR (Funktion)

Legt die Farbe für Maß- und Maßhilfslinien fest.

DIM_CONVERT_UNIT (Befehl)

Legt die Maßeinheiten fest.

DIM_COORD (Befehl)

Erstellt eine Koordinatenbemaßung.

DIM_CURSOR_POSITION (Funktion)

Legt die Position des Cursors beim Suchen von Bemaßungen fest.

DIM_DATUM (Befehl)

Legt die Position der Bemaßung fest.

DIM_DATUM_LONG (Befehl)

Erstellt einer Bezugsbemaßung mit langen Grundlinien.

DIM_DATUM_SHORT (Befehl)

Erstellt eine Bezugsbemaßung mit kurzen Grundlinien.

DIM_DATUM_STEP (Funktion)

Gibt den Abstand zwischen den Maßlinien einer Bezugsbemaßung an.

DIM_DEC_PLACE (Funktion)

Legt die Anzahl der Dezimalstellen für Bemaßungen fest.

DIM_DEG_MIN_SEC (Funktion)

Legt die Grade, Minuten und Sekunden für Bemaßungen fest.

DIM_DIAMETER (Befehl)

Erstellt eine Bemaßung für einen Durchmesser.

DIM_DIAMETER_LINE (Funktion)

Legt die Durchmesserlinien fest.

DIM_EXTENSION_LENGTH (Funktion)

Legt die aktuelle Länge von der Pfeilspitze bis zum Ende der Maßhilfslinien fest.

DIM_FONT (Funktion)

Gibt Bemaßungsschriftarten an.

DIM_FORMAT (Funktion)

Gibt die Einheiten für Bemaßungen von Linien und Winkeln an.

DIM_FRAME (Funktion)

Legt fest, ob die Maßangabe eingerahmt sein soll.

DIM_FT_INCH_SIGN (Funktion)

Gibt Optionen für die Bemaßung mit Fuß/Zoll an.

DIM_LINE (Befehl)

Erstellt eine Bemaßung für eine einzelne Linie.

DIM_LINEWIDTH (Funktion)

Legt die Linienstärke (Stiftbreite) für Maß- und Maßhilfslinien fest. (Siehe DIM_PENSIZE.)

DIM_PENSIZE (Funktion)

Legt die Linienstärke (Stiftbreite) für Maß- und Maßhilfslinien fest.

DIM_LINES_COLOR (Funktion)

Legt die Farbe für Bemaßungslinien fest.

DIM_MIN_SPACE (Funktion)

Gibt den aktuellen minimalen Abstand zwischen Geometrie und Maßlinien an.

DIM_NUMBER_FORMAT (Funktion)

Gibt das Zahlenformat für Bemaßungen an.

DIM_OFFSET_LINE (Funktion)

Gibt den Abstand zwischen Maß- und Maßhilfslinien an.

DIM_OFFSET_POINT (Funktion)

Gibt den Abstand zwischen Maßhilfslinien und Zeichnungslinien an.

DIM_POSTFIX (Funktion)

Fügt Bemaßungstext eine Postfixzeichenfolge hinzu.

DIM_PREFIX (Funktion)

Fügt Bemaßungstext eine Postfixzeichenfolge hinzu.

DIM_RADIUS (Befehl)

Erstellt eine Bemaßung für einen Radius.

DIM_RADIUS_LINE (Funktion)

Schaltet die Linie für die Bemaßung eines Radius ein/aus.

DIM_SCALE (Funktion)

Legt den Maßstab für die Bemaßung fest.

DIM_SELECT_BY_TEXTBOX (Befehl)

Legt die Art der Auswahl beim Einrahmen fest.

DIM_STAGGER_RESTORE (Funktion)

Legt den Umgang mit Maßlinen und Maßhilfslinien nach Bewegung fest.

DIM_SUBFIX (Funktion)

Fügt einer Maßangabe ein Subfix hinzu.

DIM_SUPERFIX (Funktion)

Fügt einer Maßangabe ein Superfix hinzu.

DIM_TEXT_COLOR (Funktion)

Legt die Farbe für den Bemaßungstext fest.

DIM_TEXT_FRAME_COLOR_MODE (Funktion)

Legt fest, ob die Farbe für den Rahmen um einen Bemaßungstext von der Farbe des Bemaßungstextes oder von der Farbe der Bemaßungslinie abhängig sein soll.

DIM_TEXT_GAP (Funktion)

Legt den Zwischenraum zwischen Maßangabe und Maßlinien fest, wenn eine Maßlinie zum Aufnehmen einer Maßangabe unterbrochen ist.

DIM_TEXT_HOLE (Befehl)

Fügt eine Lücke in eine bestehende Schraffur zum Aufnehmen einer ausgewählten Maßangabe ein.

DIM_TEXT_LOCATION (Funktion)

Legt fest, ob sich die aktuellen Maßangaben über, auf oder unter der Maßlinie befinden sollen. Bei senkrechten Maßlinien bedeutet "unter", dass sich die Maßangabe näher an der Zeichnung befindet.

DIM_TEXT_ORIENTATION (Funktion)

Gibt die aktuelle Lage der Maßangabe an.

DIM_TEXT_RATIO (Funktion)

Legt das Verhältnis für den Bemaßungstext fest.

DIM_TEXT_SIZE (Funktion)

Legt die Größe für den Bemaßungstext fest.

DIM_TEXT_SPACE (Funktion)

Gibt den Abstand zwischen Maßangabe und Maßlinie an, wenn sich die Maßangabe über oder unter einer Maßlinie befindet.

DIM_TOLERANCE (Funktion)

Legt den aktuellen Toleranztyp fest.

DIM_UNDERLINE_EDITED (Befehl)

Unterstreicht geänderte Bemaßungen.

DIM_UNITS (Funktion)

Gibt Bemaßungseinheiten an.

DIM_UPDATE (Befehl)

Ermittelt, markiert und verarbeitet Bemaßungskomponenten, deren Darstellung sich nach einer Neuberechnung ändern würden.

DISPLAY (Funktion)

Berechnet ein Token und zeigt das Ergebnis in der Befehlszeile an. Anschließend muss eine beliebige Taste gedrückt oder ein beliebiger Punkt angetippt werden.

DISPLAY_LIST (Funktion)

Aktiviert/deaktiviert die Verwendung der Anzeigeliste für die ausgewählten Fenster.

DISPLAY_NO_WAIT (Funktion)

Berechnet ein Token und zeigt das Ergebnis in der Befehlszeile an. Keine Benutzeraktion erforderlich.

DIV (arithmetische Funktion)

Ergebnis einer Division, bei dem die Stellen hinter dem Komma abgerundet sind.

DOTTED (Funktion)

Legt als Standardlinienart DOTTED (Punktlinie) fest.

DOT_CENTER (Funktion)

Legt als Standardlinienart DOT_CENTER (Strichpunkt) fest.

DOT_GRID (Funktion)

Schaltet in den angegebenen Darstellungsfenstern die Punktgitter ein- bzw. aus.

DRAWING_SCALE (Befehl)

Skaliert die Zeichnung mit dem angegebenen Faktor (in Abhängigkeit von der Größe des Druckerpapiers).

DRAW_CURR_PART_ON_TOP (Funktion)

Legt die Art des Neuzeichnens des aktuellen Teils (Top oder z-Ebene) fest.

DUMP_SCREEN (Funktion)

Speichert den Inhalt der Zeichnung in der angegebenen Datei.

DUMP_SCREEN_DEFAULTS (Funktion)

Legt die Parameter für die Funktion DUMP_SCREEN fest.

DUMP_SCREEN_LANG (Funktion)

Ermöglicht Ihnen, die Druckersprache für die DUMP_SCREEN-Funktion auszuwählen (siehe DUMP_SCREEN).

ECHO (Funktion)

Öffnet oder schließt die Datei ECHO.

EDIT_CURRENT_INFO (Funktion)

Editiert die aktuellen Informationen, die jedem neuen Element zugeordnet werden sollen.

EDIT_DIM_POSTFIX (Befehl)

Editiert das Postfix für die ausgewählte Maßangabe.

EDIT_DIM_PREFIX (Befehl)

Editiert das Präfix für die ausgewählte Maßangabe.

EDIT_DIM_SUBFIX (Befehl)

Editiert das Subfix für die ausgewählte Maßangabe.

EDIT_DIM_SUPERFIX (Befehl)

Editiert das Superfix für die ausgewählte Maßangabe.

EDIT_DIM_TEXT (Befehl)

Editiert die ausgewählte Maßangabe.

EDIT_DIM_TOLERANCE (Befehl)

Editiert die Toleranz der ausgewählten Maßangabe.

EDIT_ELEM_INFO (Befehl)

Editiert die Informationen zum angegebenen Element.

EDIT_ENVIRONMENT (Funktion)

Ermöglicht das Editieren einer Liste mit Befehlen, durch die die aktuelle Umgebung definiert ist. Die Umgebung besteht aus Einheiten, Standardbemaßungen, Schraffurmustern usw.

EDIT_FILE (Funktion)

Editiert die angegebene Datei mit dem integrierten Texteditor.

EDIT_MACRO (Funktion)

Editiert das angegebene Makro mit dem integrierten Texteditor.

EDIT_PART (Befehl)

Macht das angegebene Teil zum aktiven Teil.

EDIT_PORT (Funktion)

Legt fest, in welchem Darstellungsfenster mit dem integrierten Texteditor gearbeitet wird.

EDIT_TEXT (Befehl)

Ermöglicht das Editieren des angegebenen Texts.

ELLIPSE (Makro)

Zeichnet eine Spline-Kurve, die einer Ellipse stark ähnelt.

ELSE (Pseudo-Befehl)

Bedingungsoperator.

ELSE_IF (Pseudo-Befehl)

Bedingungsoperator.

ENABLE_BREAK (Funktion)

Setzt die Unterbrechungsverarbeitung auf den Standardwert zurück. (Makros können durch Drücken der Taste BREAK unterbrochen werden.)

END (Befehl)

Beendet den aktuellen Befehl/die aktuelle Funktion.

END_DEFINE (Pseudo-Befehl)

Beendet eine Makrodefinition.

END_IF (Pseudo-Befehl)

Beendet eine "IF-Anweisung".

END_LOOP (Pseudo-Befehl)

Beendet eine Schleife in einem Makro.

END_PART (Befehl)

Siehe EDIT_PART PARENT.

ENTER (Funktion)

Wertet ein Token aus. Das Ergebnis wird in der Befehlszeile angezeigt.

EQUIDISTANCE (Befehl)

Erstellt eine abstandsgleiche Umrandung.

ERROR_LOG (Funktion)

Sichert vom System ausgegebene Warnungen und Fehlermeldungen.

ERROR_STR (arithmetische Funktion)

Zeigt die erste vom System ausgegebene Fehlermeldung nach Einschalten der Fehleraufzeichnung durch TRAP_ERROR an.

EXIT (Befehl)

Beendet die Sitzung. Die Kontrolle wird wieder an das Hostbetriebssystem gegeben. Um ein versehentliches Beenden der Sitzung zu verhindern, muss der Befehl mit CONFIRM noch einmal bestätigt werden.

EXIT_IF (Pseudo-Befehl)

Anweisung zum bedingten Beenden einer Schleife.

EXOR (arithmetische Funktion)

Exclusive OR. Ergibt 1, wenn genau ein Argument (von zweien) 0 ist. Ergibt sonst 1.

EXP (arithmetische Funktion)

Gibt e (2.718...) hoch Argument zurück.

FALSE (arithmetische Funktion)

Liefert den Wert 0.

FBROWSER (Funktion)

Ruft die Dateiliste zur Dateiverwaltung auf.

FILLET (Befehl)

Erstellt eine Rundung mit dem angegebenem Radius.

FOLLOW (Funktion)

Macht den zuletzt eingegebenen Punkt zum Ursprung des Koordinatensystems.

FONT_EDITOR (Befehl)

Erstellt und ändert die Schriftarten von Creo Elements/Direct Drafting.

FRACT (arithmetische Funktion)

Ergibt die Stellen hinter dem Komma des Arguments.

GATHER (Befehl)

Fügt bestehende Elemente in das aktive Teil ein.

GET_ELEM_INFO (Funktion)

Macht die Informationen zum ausgewählten Element zu aktuellen Informationen.

GET_PID (arithmetische Funktion)

Zeigt die Verarbeitungskennung des aktiven Programms an.

GET_PROPERTIES (Funktion)

Gleicht die aktuellen Merkmale denen des ausgewählten Elements an.

GET_TYPE (Funktion)

Gibt den Typ des angegebenen Tokens zurück. Es handelt sich um eine Verbesserung des TYPE-Befehls.

GREEN (Funktion)

Ändert die Standardfarbe in Grün.

GRID_FACTOR (Funktion)

Gibt den Abstand zwischen Gitterpunkten bzw. den Unterteilungsstrichen auf dem Lineal an.

HATCH (Befehl)

Erstellt Schraffuren.

HATCH_ANGLE (Funktion)

Legt den aktuellen Neigungswinkels der Schraffur fest.

HATCH_COLOR (Funktion)

Legt die aktuelle Farbe für die Schraffur fest.

HATCH_DIST (Funktion)

Legt den aktuellen Abstands zwischen den Schraffurlinien fest.

HATCH_LINETYPE (Funktion)

Legt die aktuelle Linienart für die Schraffur fest.

HATCH_REF_PT (Funktion)

Legt den aktuellen Bezugspunkt für die Schraffur fest.

HELP (Funktion)

Zeigt den Abschnitt der Hilfedatei an, der die Beschreibung des angegebenen Schlüsselwortes enthält.

HELP_PORT (Funktion)

Legt ein Darstellungsfenster für die Anzeige der Hilfetexte fest.

HIGHLIGHT_LTAB (Funktion)

Ändert den Hervorhebungsstatus der angegebenen Position im Datenbereich der angegebenen Benutzertabelle.

HL_CHANGE_COLOR (Makro)

Ändert die Farbe aller verdeckten Linien, die mit dem Befehl HL_GENERATE_HIDDEN erzeugt wurden.

HL_CHANGE_LTYPE (Makro)

Ändert die Linienart aller verdeckten Linien, die mit dem Befehl HL_GENERATE_HIDDEN erzeugt wurden.

HL_DEFAULT_FACE_COLOR (Funktion)

Legt die Standardfarbe für Flächen fest.

HL_DELETE_FACE (Befehl)

Löscht überlagernde Flächen. Die ausgewählten Flächen müssen sich im aktuellen Teil befinden.

HL_GENERATE_FACE (Befehl)

Gibt überlagernde Flächen an. Eine überlagernde Fläche verdeckt alle Linien mit einem niedrigeren Z-Wert.

HL_GENERATE_HIDDEN (Befehl)

Ermöglicht das Arbeiten mit verdeckten Linien.

HL_GEN_ALL_PART (Befehl)

Legt Einstellungen für Z-Werte und für die Flächengenerierung bezogen auf das gesamte Teil fest.

`HL_INQ_CURR_Z_VALUE` (Funktion)

Zeigt den aktuellen z-Wert, der jeder neu erstellten Komponente standardmäßig zugeordnet wird.

`HL_INQ_FACE_COLOR` (Funktion)

Zeigt den Wert für die RGB-Farbe der angegebenen Fläche.

`HL_INQ_LOAD_OFFSET` (Funktion)

Zeigt den Wert des Ladeabstands in z-Richtung.

`HL_INQ_LOAD_VALUE` (Funktion)

Zeigt den Modus, in dem der Ladewert bzw. Ladeabstand angegeben wurde, und den Wert selbst.

`HL_INQ_RELATION_OFFSET` (Funktion)

Zeigt den Wert des Ladeabstands in z-Richtung.

`HL_INQ_Z_VALUE` (Funktion)

Zeigt den z-Wert des angegebenen Elements oder den höchsten oder niedrigsten z-Wert der gesamten Baugruppe.

`HL_REDRAW_MODE` (Funktion)

Schaltet zwischen dem Neuzeichnen im normalen Modus und dem Modus für verdeckte Linien in Creo Elements/Direct Drafting hin und her.

`HL_SET_COLOR` (Funktion)

Legt die Farbe für die verdeckten Linien fest.

`HL_SET_CURR_Z_VALUE` (Funktion)

Definiert einen z-Wert, der jeder neu erstellten Komponente standardmäßig zugeordnet wird.

`HL_SET_FACE_COLOR` (Befehl)

Legt die Hintergrundfarbe für überlagernde Flächen fest oder ändert sie.

`HL_SET_KEEP_COLOR` (Funktion)

Legt fest, ob Elemente beim Generieren von verdeckten Linien ihre Farbe behalten.

`HL_SET_LINETYPE` (Funktion)

Legt die Linienart für verdeckte Linien fest.

`HL_SET_LOAD_VALUE` (Funktion)

Legt einen Ladeabstand in z-Richtung fest.

`HL_SET_RELATION_OFFSET` (Funktion)

Legt einen Abstand in z-Richtung fest.Dieser Abstand wird zum Berechnen von z-Werten verwendet, die als ÜBER/UNTER/ZWISCHEN-Beziehung zu einem anderen Element angegeben wurden.

`HL_SET_Z_VALUE` (Befehl)

Ordnet z-Werte zu oder ändert sie.

`HL_SHOW_HIDDEN` (Makro)

Blendet verdeckte Linien ein oder aus.

`HL_VISUALIZE` (Funktion)

Ermöglicht es, alle Elemente mit bestimmten z-Werten anzuzeigen oder die Flächen bestimmter Stufen in verschiedenen Farben darzustellen.

`HSL_COLOR` (Funktion)

Gibt eine neue TSL-Farbe (Farbton, Sättigung, Helligkeit) an.

`ICONIFY_WINDOW` (Funktion)

Stellt das Creo Elements/Direct Drafting Fenster als Symbol dar.

`IF` (Pseudo-Befehl)

Boolescher Ausdruck.

`IGNORE_BREAK` (Funktion)

Ignoriert Unterbrechungen durch Drücken der Taste BREAK.

`INIT_PART` (Befehl)

Erstellt ein neues, leeres Teil direkt unter dem aktiven Teil. Dieses neue Teil wird aktiviert.

`INIT_SUBPART` (Befehl)

Erstellt ein neues, leeres Teil direkt unter dem aktiven Teil. Dieses neue Teil wird aktiviert.

`INPUT` (Funktion)

Leitet über Tastatur oder Maus eingegebene Daten vorübergehend in die angegebene Textdatei um. (Wenn Sie den `INPUT`-Befehl in einem Makro verwenden, muss er mit dem Kennzeichner `IMMEDIATE` verwendet werden, siehe auch INPUT auf Seite 33.)

`INQ` (arithmetische Funktion)

Ergibt ein Element eines Systemfeldes. Diese Werte können mit INQ_ELEM oder INQ_ENV festgelegt werden.

`INQ_ELEM` (Funktion)

Schreibt Informationen zum angegebenen Element in das Systemfeld. Diese Angaben können mit `INQ` abgerufen werden.

`INQ_ENV` (Funktion)

Schreibt Informationen zur Systemumgebung in das Systemfeld. Diese Angaben können mit `INQ` abgerufen werden.

INQ_PART (Funktion)

Schreibt Informationen über den Vergrößerungsfaktor des Teils in das Systemfeld. Diese Angaben können mit INQ abgerufen werden.

INQ_SELECTED_ELEM (Funktion)

Schreibt Informationen über das angegebene Element in das Systemfeld. Er kann anschließend mit INQ aufgerufen werden (siehe INQ).

INQ_TABLE (Funktion)

Ruft Informationen über die angegebene Tabelle ab.

INSERT_LTAB_ROW (Funktion)

Fügt Zeilen in die angegebene Benutzertabelle ein.

INT (arithmetische Funktion)

Ergibt den ganzteiligen Wert eines Argument. INT(PI) ist beispielsweise gleich 3.

ISOMETRIC (Befehl)

Erstellt eine isometrische Ansicht.

KEEP_CORNER (Funktion)

Durch Auswahl von AUS werden die Linien oder Bögen zwischen der Ecke und der Rundung oder der Fase gelöscht. Durch Auswahl von EIN bleibt die Ecke bestehen.

KNOB_BOX_FACTOR (Funktion)

Legt die Empfindlichkeit der Drehknöpfe fest.

LABEL (Befehl)

Generiert Kenndaten für ausgewählte Punkte, Linien, Bögen und Kreise im aktiven Teil.

LAST_POSTFIX (Funktion)

Verwendet das vorherige Postfix als aktuelles Postfix.

LAST_PREFIX (Funktion)

Verwendet das Präfix der vorherigen Bemaßung als aktuelles Präfix.

LAST_SUBFIX (Funktion)

Verwendet das Subfix der vorherigen Bemaßung als aktuelles Subfix.

LAST_SUPERFIX (Funktion)

Verwendet das Superfix der vorherigen Bemaßung als aktuelles Superfix.

LAST_TOLERANCE (Funktion)

Verwendet die vorherige Toleranz als aktuelle Toleranz.

LAST_WINDOW (Funktion)

Stellt das vorherige Fenster des aktuellen Darstellungsfensters wieder her.

LEADER_ARROW (Funktion)

Gibt die Begrenzung für neue Hinweislinien an.

LEADER_LINE (Befehl)

Erstellt eine Hinweislinie.

LEN (arithmetische Funktion)

Bei einem Zeichenfolgenargument: Gibt die Länge der Zeichenfolge zurück. Bei einem Vektorargument: Gibt die Länge des Vektors vom Ursprung bis zum Argumentpunkt zurück.

LET (Funktion)

Definiert ein Makro oder eine Variable.

LG (arithmetische Funktion)

Ergibt den Zehnerlogarithmus (Basis 10) des Arguments.

LINE (Befehl)

Erstellt eine Linie.

LINEPATTERN (Funktion)

Legt die aktuelle Linienart für die Komponenten, die mit dem aktuellen Befehl erstellt werden sollen, fest.

LINESIZE (Funktion)

Legt die aktuelle Linienbreite für alle Zeichnungslinien außer für Hilfslinien und POINT fest.

LINETYPE (Funktion)

Legt die aktuelle Linienart fest.

LINEWIDTH (Funktion)

Legt die aktuelle Stiftbreite für die neue reale Geometrie außer für die Hilfsgeometrie und POINT fest. (siehe PENSIZE).

LINE_GRID (Funktion)

Schaltet das Liniengitter ein- bzw. aus.

LIST_FONTS (Funktion)

Zeigt eine Liste aller definierten und verwendeten Schriftarten an. Zeigt außerdem die aktuell verwendete Schriftart an.

LIST_GLOBAL_INFO (Funktion)

Listet die Informationen zu sämtlichen Elementen im Speicher auf.

LIST_MACRO_NAMES (Funktion)

Gibt die Namen aller zur Zeit für das angegebene Ziel definierten Makros aus.

LN (arithmetische Funktion)

Liefert den natürlichen Logarithmus (Basis e) des angegebenen Werts.

LOAD (Befehl)

Lädt einen Teil der angegebenen Datei in den Speicher.

`LOAD_FONT` (Befehl)

Lädt alle in der angegebenen Datei verwendeten Schriftarten in den Speicher.

`LOAD_MACRO` (Funktion)

Lädt alle in der angegebenen Datei gespeicherten Makros in den Speicher.

`LOAD_MODULE` (Befehl)

Aktiviert das angegebene Anwendungsmodul.

`LOCAL` (Pseudo-Befehl)

Definiert eine lokale Variable innerhalb eines Makros.

`LONG_DASHED` (Funktion)

Legt als Standardlinienart LONG_DASHED (Langstrichlinie) fest.

`LOOP` (Pseudo-Befehl)

Definiert eine Schleife, die wiederholt wird, bis der Boolesche Ausdruck in einer EXIT_IF-Klausel wahr ist.

`LOWER_WINDOW` (Funktion)

Bewirkt, dass das Creo Elements/Direct Drafting Fenster in der Windows® Umgebung als unterstes Fenster angezeigt wird.

`LTAB_COLUMNS` (arithmetische Funktion)

Zeigt die Anzahl der Spalten in der angegebenen logischen Tabelle an.

`LTAB_ROWS` (arithmetische Funktion)

Zeigt die Anzahl der Zeilen in der angegebenen logischen Tabelle an.

`LTAB_TITLES` (arithmetische Funktion)

Zeigt die Anzahl der Titel in der angegebenen logischen Tabelle an.

`LWC` (arithmetische Funktion)

Kleinschreibung. Wandelt Großbuchstaben in Kleinbuchstaben um.

`MAGENTA` (Funktion)

Ändert die Standardfarbe für Linien in Magenta.

`MAKE_TMP_NAME` (arithmetische Funktion)

Liefert einen eindeutigen Dateinamen zur vorübergehenden Verwendung.

`MATCH` (arithmetische Funktion)

Ergibt 1, wenn die erste Zeichenfolge mit dem in der zweiten Zeichenfolge angegebenen Muster übereinstimmt (sonst 0).

`MAX_FEEDBACK` (Funktion)

Gibt an, in welchem Umfang Elemente bei den Befehlen `MODIFY` und `STRETCH` mit einbezogen werden.

`MEASURE_ANGLE` (Funktion)

Mißt den Winkel zwischen zwei angegebenen Elementen.

MEASURE_AREA (Funktion)

Mißt die von einem Kreis, einem Bogen, einer Spline-Kurve oder einer Rundung eingeschlossene Fläche.

MEASURE_COORDINATE (Funktion)

Mißt die Koordinaten eines ausgewählten Punktes.

MEASURE_DISTANCE (Funktion)

Mißt den Abstand zwischen zwei angegebenen Punkten.

MEASURE_LENGTH (Funktion)

Mißt die Länge einer Linie, eines Kreises, eines Bogens, einer Rundung oder einer Spline-Kurve.

MEASURE_RADIUS (Funktion)

Mißt den Radius eines Kreises, eines Bogens oder einer Rundung.

MENU (Funktion)

Definiert Aussehen und Funktion der Menüfelder des Bildschirmmenüs.

MENU_LAYOUT (Funktion)

Definiert Form und Lage eines Bildschirmmenüs.

MENU_STATUS (Funktion)

Definiert den Status eines Bildschirmmenüs.

MERGE (Befehl)

Vereinigt zwei Elemente zu einem Element.

MIRR (arithmetische Funktion)

Ergibt das Bild eines Vektors, der an einer gedachten Linie gespiegelt wurde.

MOD (arithmetische Funktion)

Ergibt den Rest einer Division.

MODIFY (Befehl)

Ermöglicht das Verändern von Elementen mit MOVE, MIRROR, SCALE, SIMILAR oder AFFINE.

MODIFY_DIM_LINES (Befehl)

Ersetzt bei den bestehenden Maß- und Maßhilfslinen Vollinien durch unterbrochene Linien.

MOVE_DIMENSION (Befehl)

Verschiebt die ausgewählte Bemaßung.

MOVE_TABLE (Funktion)

Versetzt eine Tabelle in einen bestimmten Bildschirmbereich. Nur Tabellen, die nicht gegen Versetzen gesichert sind, können neu versetzt werden.

NEW_SCREEN (Funktion)

Zeichnet den gesamten Bildschirm neu.

NOT (arithmetische Funktion)

Gibt 1 zurück, wenn das Argument gleich 0 ist. Ergibt sonst 0.

NUM (arithmetische Funktion)

Wandelt das erste ASCII-Zeichen einer Zeichenfolge in die entsprechende Dezimalzahl um. Beispiel: NUM('hullo') ist 104.

ON_ERROR (Funktion)

Benutzt die angegebene Zeichenfolge beim Auftreten des nächsten Fehlers als Eingabe.

OPEN_INFILE (Funktion)

Öffnet die benannte Datei mit der Funktion READ_FILE zum Lesen.

OPEN_OUTFILE (Funktion)

Öffnet die benannte Datei mit der Funktion WRITE_FILE zum Schreiben.

OR (arithmetische Funktion)

Ergibt 1, wenn beide Argumente ungleich 0 sind. Ergibt sonst 0.

ORIGIN (Funktion)

Aktiviert bzw. inaktiviert im angegebenen Darstellungsfenster das Symbol für den Ursprung des relativen Koordinatensystems.

OUTPUT_HP15 (Funktion)

Schaltet den Ausgabemodus für Kanji-Text auf Code HP15.

OUTPUT_HP16 (Funktion)

OUTPUT_HP16 schaltet den Ausgabemodus für Kanji-Text auf Code HP16.

OUTPUT_STRING (Funktion)

Gibt den Inhalt von |string| auf einem Ausgabegerät aus.

OVERDRAW (Befehl)

Zeichnet Hilfsgeometrie nach. Dieser Befehl ersetzt den Inhalt des Nachzeichnen-Makros.

PARAMETER (Pseudo-Befehl)

Kennzeichnet Makroparameter in Makrodefinitionen.

PART_DRW_SCALE (Befehl)

Legt einen Zeichnungsmaßstab für das angegebene Teil fest.

PART_DRW_SCALE_REF (Funktion)

Legt einen Bezugspunkt fest, von dem aus das Teil skaliert wird.

PARTS_LIST (Funktion)

Zeigt alle Teile innerhalb des aktiven Teils an, die nicht mit "." anfangen. Zeigt an, wie oft jedes Teil vorkommt.

PASSWORD (Funktion)

Dient zur Eingabe des Kennwortes für den Systemzugriff.

PENSIZE (Funktion)

Legt die aktuelle Stiftbreite für die neue reale Geometrie außer für die Hilfsgeometrie und POINT fest.

PHANTOM (Funktion)

Ändert die Linienart der ausgewählten Schraffur in PHANTOM.

PI (arithmetische Funktion)

Ergibt den Wert für pi (3.14159265358979).

PICK_UTAB_ROW_BY_NAME (Funktion)

Unterstützung der Bildschirmschnittstelle des Bemaßungsmoduls. (Nicht für die Ausführung über die Befehlszeile bestimmt.)

PICK_VP_PNT (Funktion)

Emuliert eine interaktive Benutzerauswahl in einem bestimmten Darstellungsfenster durch Angabe einer Zahl und des Namens des Darstellungsfensters.

PICTURE_BROWSER (Funktion)

Ruft das Fenster PICTURE BROWSER auf.

PICTURE_LIST (Funktion)

Ruft des Fenster mit der Liste der in der aktuellen Sitzung geladenen Abbildungen auf.

PLOT (Befehl)

Plottet eine Zeichnung.

PLOTTER_TYPE (Funktion)

Gibt den angeschlossenen Plotter an.

PLOT_AUTO_ROTATE (Funktion)

Inaktiviert die Auto-Rotierfunktion, die bei manchen Rasterplottern Bestandteil der zugehörigen Firmware ist.

PLOT_CENTER (Funktion)

Gibt an, ob die Zeichnung im Darstellungsfenster für die Ausgabe an den Plotter zentriert werden soll.

PLOT_DESTINATION (Funktion)

Gibt an, wo die Plot-Ausgabe erfolgen soll.

PLOT_FORMAT (Funktion)

Gibt die maximale Größe des Plotbereichs an (absolute Grenzen).

PLOT_IMAGE_QUALITY (Funktion)

Definiert Farbpalette, Auflösung und Maßstabsänderungen.

PLOT_LINETYPE_LENGTH (Funktion)

Gibt die Länge des Musters für jede Linienart an.

PLOT_PEN_TABLE (Funktion)

Steuert die Zuordnung der Linienarten und Farben in der Zeichnung zur Plotter-Hardware.

PLOT_SCALE (Funktion)

Gibt den Skalierungsfaktor der Zeichnung vor dem Plotten an.

PLOT_STOP_ON_ERROR (Funktion)

Steuert das Verhalten bei der Plotausgabe, wenn die Zeichnung nicht in den Plotbereich passt.

PLOT_TRANSFORMATION (Funktion)

Gibt an, mit welchem Plotterstift bestimmte Elemente gezeichnet werden sollen.

PLOT_VIEWPORT (Funktion)

Sucht das Darstellungsfenster für die Ausgabe an den Plotter innerhalb des (mit PLOT_FORMAT angegebenen) maximalen Plotting-Bereichs.

PNT_RA (arithmetische Funktion)

Ergibt bei einer gegebenen Länge und einem gegebenen Winkel einen 2D-Punkt.

PNT_XY (arithmetische Funktion)

Ergibt bei gegebener X- und Y-Koordinate einen 2D-Punkt (Vektor).

PNT_XYZ (arithmetische Funktion)

Ergibt bei gegebener X-, Y- und Z-Koordinate einen 3D-Punkt (Vektor).

POINT (Befehl)

Erstellt Punktelemente.

POLYELEM (Befehl)

Ermöglicht Ihnen, eine Polylinie aus den vorhandenen Elementen zu erzeugen. Anschließend können Sie die Polylinie als Einzelelement behandeln und entsprechend ändern.

POP_DOWN_LTAB (Funktion)

Blendet die angegebene logische Tabelle aus.

POP_UP_LTAB (Funktion)

Blendet die angegebene logische Tabelle ein.

POS (arithmetische Funktion)

Ergibt die erste Position einer Unterzeichenfolge innerhalb einer Zeichenfolge.

PRE_VIEW (Befehl)

Ermöglicht das Ansehen einer Zeichnung vor dem Laden (Vorschau).

`PRINT_TABLE` (Funktion)

Druckt eine Anzeigetabelle in eine Datei.

`PROMPT_LIST` (Funktion)

Sichert bis zu 500 Eingabeaufforderungen im Systemspeicher.

`PURGE_FILE` (Funktion)

Löscht die angegebenen Dateien von der Festplatte.

`PUT_PROPERTIES` (Befehl)

Gleicht die Merkmale des ausgewählten Elements den aktuellen Merkmalen an.

`RAC_CHECK` (Befehl)

Aktualisiert Creo Elements/Direct Modeling Layoutdaten durch Hinzufügen von Dokumentationsdaten zu einer älteren Version dieses Layouts.

`RAISE_WINDOW` (Funktion)

Bewirkt, dass das Creo Elements/Direct Drafting Fenster in der Windows Umgebung als oberstes Fenster angezeigt wird.

`RC_ACCURACY` (Funktion)

Legt die Genauigkeit beim Vergleichen zweier MI-Dateien fest.

`RC_CHECK` (Befehl)

Vergleicht zwei Teile einschließlich aller untergeordneten Teile miteinander und speichert das Ergebnis als Informationstexte für die einzelnen Elemente.

`READ` (Funktion)

Liest Benutzereingaben über die Befehlszeile und ordnet diese einer Variable zu.

`READ_FILE` (Funktion)

Liest eine Textzeile aus der angegebenen Datei und ordnet diese der angegebenen Variable zu.

`READ_LTAB` (arithmetische Funktion)

Liest den Wert aus der angegebenen logischen Tabelle.

`RECALL_BUFFER` (Funktion)

Sichert bis zu 63 eingegebene Zeilen. Diese Zeilen können mit den Tasten `[Prev]` und `[Next]` erneut aufgerufen werden.

`RECALL_WINDOW` (Funktion)

Das Fenster des aktuellen Darstellungsfensters wird in das Fenster geändert, das mit `STORE_WINDOW` gespeichert wurde.

`RED` (Funktion)

Ändert die Standardfarbe für Linien in Rot.

`REDRAW` (Funktion)

Zeichnet den Inhalt des aktuellen Darstellungsfensters neu.

`REDRAW_SCENE` (Funktion)

Zeichnet ein Effektbildfenster neu.

`RENAME_ELEMENT` (Befehl)

Kopiert oder benennt ein Element einschließlich aller Änderungsstände und Versionen um.

`RENAME_PART` (Befehl)

Benennt das aktive Teil um.

`RENOVATE` (Funktion)

Baut das ausgewählte Darstellungsfenster neu auf.

`REPEAT` (Pseudo-Befehl)

Führt eine Schleife so lange aus, bis der Boolesche Ausdruck in der Klausel `UNTIL` wahr ist.

`REQUEST_PRINT_SETUP` (Funktion)

Setzt den "Windows Print Manager" derart, dass er seinen eigenen Dialog ein- oder ausschaltet, sowie Plots oder Screendumps an den Print Manager geschickt werden.

`RESET_PART_NUMBER` (Befehl)

Sortiert alle "eindeutigen" Teilenummern beginnend bei dem obersten Teil (Erzeugnis).

`RESTORE` (Befehl)

Lädt archivierte Elemente zusammen mit den zugehörigen Dateien vom Archivdatenträger zurück in die Datenbank.

`RGB_COLOR` (Funktion)

Gibt eine neue RGB-Farbe (Rot, Grün, Blau) an.

`RND` (arithmetische Funktion)

Gibt eine pseudozufällige Xnumber im Bereich 0 < = X < 1 zurück.

`ROT` (arithmetische Funktion)

Ergibt einen Punkt (Vektor), der um den angegebenen Winkel um den Ursprung gedreht wurde.

`ROTATE_DIM_TEXT` (Befehl)

Dreht die ausgewählte Maßangabe um den angegebenen Winkel.

`ROUND` (arithmetische Funktion)

Ergibt das Argument auf die nächste ganze Zahl auf- bzw. abgerundet. Beispiel: Beispiel: ROUND(4.4999) entspricht `4`.

`RPT` (arithmetische Funktion)

Ergibt die gewünschte Anzahl von Kopien einer Zeichenfolge.

RTL_COLOR (Befehl)

Legt die Farbe für Bezugslinien fest.

RTL_DST_GAP (Befehl)

Legt den Bezugslinienabstand zur Zielposition fest.

RTL_LINETYPE (Befehl)

Legt die Linienart für Bezugslinien fest.

RTL_LINEWIDTH (Befehl)

Legt die Linienstärke (Stiftbreite) für Bezugslinien fest. (siehe RTL_PENSIZE).

RTL_PENSIZE (Befehl)

Legt die Stiftbreite (Linienstärke) für Bezugslinien fest.

RTL_SRC_GAP (Befehl)

Legt den Bezugslinienabstand zur Ausgangsposition fest.

RULER (Funktion)

Schaltet das Lineal in den angegebenen Darstellungsfenstern ein- bzw. aus.

RUN (Funktion)

Bewirkt das Zurückkehren in die Betriebssystem-Eingabeaufforderung.

SAVE (Befehl)

Sichert die Zeichnung im Speicher in der angegebenen Datei.

SAVE_ENVIRONMENT (Funktion)

Sichert die Umgebung.

SAVE_FONT (Funktion)

Sichert alle Schriftarten (oder eine angegebene Schriftart) in der angegebenen Datei.

SAVE_LTAB (Funktion)

Speichert die angegebene logische Tabelle in "output spec" (siehe OUTPUT_SPEC in der Hilfe).

SAVE_MACRO (Funktion)

Sichert das angegebene Makro oder alle Makros.

SAVE_MENU (Funktion)

Sichert die Definition des aktuellen Bildschirmmenüs.

SAVE_TABLE (Funktion)

Sichert die Konfiguration einer Anzeigetabelle in einer Datei.

SAVE_VIEWPORT (Funktion)

Sichert die Definitionen des aktuellen Darstellungsfensters.

SCREEN_TRANSFORMATION (Funktion)

Definiert, während die Zeichnung neu gezeichnet wird, eine Zuordnung für bestimmte Elemente.

`SCROLL_LTAB` (Funktion)

Blättert in den zur angegebenen logischen Tabelle gehörigen Anzeigetabellen, sodass die angegebene Zeile in der Anzeige ganz oben angezeigt wird.

`SEARCH` (Befehl)

Gibt die Verzeichnisse in der Suchliste an.

`SECURE_LTAB` (Funktion)

Schützt die angegebene Benutzertabelle. Geschützte Benutzertabellen können nicht gelöscht werden.

`SECURE_MACRO` (Funktion)

Verhindert, dass Makros aufgelistet, editiert, angesehen oder seine einzelnen Anweisungen mit Hilfe der Ablaufverfolgung am Bildschirm angezeigt werden.

`SECURE_TABLE` (Funktion)

Schützt eine Anzeigetabelle gegen Löschen. Anschließend kann die Tabelle nicht mehr gelöscht oder neu definiert werden.

`SELECT_DIM_ARROW` (Funktion)

Gibt die aktuelle Maßlinienbegrenzung an.

`SELECT_FROM_LTAB` (Funktion)

Führt eine ausgewählte Operation für die Ausgangstabelle aus.

`SGN` (arithmetische Funktion)

Ergibt -1, wenn die eingegebene Zahl negativ ist, 0, wenn Zahl 0 ist und 1, wenn Zahl positiv ist.

`SHARE_PART` (Befehl)

Macht das angegebene Teil zu einem gemeinsam benutzten Teil.

`SHOW` (Funktion)

Aktiviert bzw. inaktiviert die Anzeige von Elementen. Zeigt Elemente in verschiedenen Farben an.

`SHOW_CPOLY` (Funktion)

Zeigt ein Stützpolygon für eine B-Spline–Kurve an.

`SHOW_PART` (Funktion)

Ändert die Art, in der Teile im aktuellen Darstellungsfenster angezeigt werden.

`SHOW_TABLE` (Funktion)

Zeigt eine bestehende Tabelle an oder löscht sie.

`SHOW_TABLE_PAGE` (Funktion)

Zeigt eine der angegebenen Anzeigetabelle zugeordnete Abfrageseite an.

`SIN` (arithmetische Funktion)

Ergibt den Sinus des Arguments.

SL_COLOR (Befehl)

Legt die Farbe für Symmetrielinien fest.

SL_LINETYPE (Befehl)

Legt die Linienart für Symmetrielinien fest.

SL_LINEWIDTH (Befehl)

Legt die Linienstärke (Stiftbreite) für Symmetrielinien fest. (siehe SL_PENSIZE).

SL_PENSIZE (Befehl)

Legt die Linienstärke (Stiftbreite) für Symmetrielinien fest.

SL_OFFSET (Befehl)

Legt den Abstand für Symmetrielinien fest.

SMASH_POLY (Befehl)

Zerlegt ein Polygon, so dass seine Bestandteile danach als einzelne Elemente vorliegen.

SMASH_SUBPART (Befehl)

Fügt alle Elemente des ausgewählten Teils in das aktive Teil ein und löscht das untergeordnete Teil. Das ausgewählte Teil muss dem aktiven Teil untergeordnet sein.

SNID (arithmetische Funktion)

Gibt die Produkt- und Seriennummer eines HPHIL-Sicherheitssteckers an. Wenn keine Sicherheitseinheit vorhanden ist, wird die Produkt- und Seriennummer des Computers angegeben.

SOLID (Funktion)

Legt als Standardlinienart SOLID fest.

SORT_LTAB (Funktion)

Sortiert eine Benutzertabelle nach den angegebenen Spalten.

SPLINE (Befehl)

Erstellt eine Spline-Kurve.

SPLINE_CONVERSION (Funktion)

Wandelt Spline-Kurven im alten Format ("C-Spline") in das neue Format ("B-Spline") um.

SPLIT (Befehl)

Zerlegt Linien, Kreise und Rundungen.

SPLITTING (Funktion)

Schaltet die Funktion zum automatischen Zerlegen und Vereinigen von Elementen ein- bzw. aus.

SPOTLIGHT (Funktion)

Das aktive Teil bleibt weiterhin aktiv. Zeichnet den Rest der Zeichnung in der Farbe Magenta und mit gestrichelten Linien neu.

SQR (arithmetische Funktion)

Ergibt das Quadrat des Arguments.

SQRT (arithmetische Funktion)

Ergibt die Quadratwurzel aus dem Argument.

STATLINE_RESET (Funktion)

Reaktiviert die Statuszeile nachdem sie automatisch inaktiviert wurde.

STORE (Befehl)

Speichert die Zeichnung in der festgelegten Datei (MI-Format 2.30).

STORE_240 (Befehl)

Speichert die Zeichnung im MI-Format 2.40.

STORE_250 (Befehl)

Speichert die Zeichnung im MI-Format 2.50.

STORE_260 (Befehl)

Speichert die Zeichnung im MI-Format 2.60.

STORE_270 (Befehl)

Speichert die Zeichnung im MI-Format 2.70.

STORE_280 (Befehl)

Speichert die Zeichnung im MI-Format 2.80.

STORE_290 (Befehl)

Speichert die Zeichnung im MI-Format 2.90.

STORE_300 (Befehl)

Speichert die Zeichnung im MI-Format 3.00.

STORE_310 (Befehl)

Speichert die Zeichnung im MI-Format 3.10.

STORE_320 (Befehl)

Speichert die Zeichnung im MI-Format 3.20.

STORE_330 (Befehl)

Speichert die Zeichnung im MI-Format 3.30.

STORE_340 (Befehl)

Speichert die Zeichnung im MI-Format 3.40.

STORE_350 (Befehl)

Speichert die Zeichnung im MI-Format 3.50.

STORE_360 (Befehl)

Speichert die Zeichnung im MI-Format 3.60.

STORE_370 (Befehl)

Speichert die Zeichnung im MI-Format 3.70.

STORE_380 (Befehl)

Speichert die Zeichnung im MI-Format 3.80.

STORE_FONT (Funktion)

Speichert alle Schriftarten (oder die angegebene Schriftart) in der benannten Datei.

STORE_IN_RECALL_BUFFER (Funktion)

Speichert die angegebene Zeichenfolge zur weiteren Verwendung im Recall-Puffer.

STORE_MACRO (Funktion)

Speichert das benannte Makro oder alle Makros.

STORE_WINDOW (Funktion)

Speichert die Fensterkoordinaten des aktuellen Darstellungsfensters.

STR (arithmetische Funktion)

Ergibt das Argument in ASCII-Darstellung. Beispiel:

STRETCH (Befehl)

Streckt Linien, Kreise, Bögen, Spline-Kurven und Hinweislinien.

STRUCTURE (Funktion)

Legt hierarchische Strukturen an.

SUBSTR (arithmetische Funktion)

Ergibt die Unterzeichenfolge einer Zeichenfolge.

SYMBOL_PART (Befehl)

Bewirkt, dass das ausgewählte Teil als Symbol dargestellt wird.

SYMLINE (Befehl)

Erstellt eine Symmetrielinie.

TABLE_COLUMN (Funktion)

Legt die Feldattribute in den Datenspalten einer bestehenden ungesicherten Tabelle fest.

TABLE_LAYOUT (Befehl)

Erlaubt das Erstellen einer neuen Tabelle in einer bestimmten Form an einer bestimmten Stelle.

TABLE_SCROLL_STEP (Funktion)

Legt die Größe der Schritte fest, in denen mit der Blätterleiste in einer Anzeigetabelle gesprungen werden kann.

`TABLE_TITLE` (Funktion)

Legt die Attribute für das Titelfeld einer bestehenden Tabelle, die nicht gegen Änderung des Titels gesichert ist, fest.

`TAN` (arithmetische Funktion)

Ergibt den Tangens des Arguments.

`TEXT` (Befehl)

Erstellt Texte.

`TEXT_ADJUST` (Funktion)

Gibt an, wie neuer Text ausgerichtet werden soll.

`TEXT_ANGLE` (Funktion)

Gibt den aktuellen Neigungswinkel des Texts an.

`TEXT_FILL` (Funktion)

Schaltet die Füllung einzelner Zeichen ein- bzw. aus.

`TEXT_HOLE_INSERTION` (Funktion)

Legt den Rahmentyp des aktuellen Texts fest.

`TEXT_FRAME` (Funktion)

Fügt ein Textfenster in einen schraffierten Bereich ein.

`TEXT_LINESPACE` (Funktion)

Legt den Zeilenabstand für neue Texte fest.

`TEXT_RATIO` (Funktion)

Legt das Verhältnis von Zeichenbreite zu Zeichenhöhe fest.

`TEXT_SIZE` (Funktion)

Legt die aktuelle Zeichenhöhe fest.

`TEXT_SLANT` (Funktion)

Legt den aktuellen Neigungswinkel der Zeichen fest.

`TEXT_TO_GEO` (Befehl)

Wandelt Text in Geometrie.

`TIME` (arithmetische Funktion)

Zeigt die Anzahl der seit Mitternacht verstrichenen Sekunden an.

`TONE` (Funktion)

Erzeugt ein akustisches Signal mit der angegebenen Frequenz, Dauer und Amplitude.

`TRACE` (Funktion)

Öffnet oder schließt die Datei `TRACE`.

TRAP_ERROR (Funktion)

Legt das Verhalten bei Auftreten eines Fehlers fest. Ohne TRAP_ERROR werden bei Auftreten eines Fehlers alle Ausführungen abgebrochen. Mit TRAP_ERROR laufen die Ausführungen trotz Auftreten eines Fehlers weiter. Der Fehler wird registriert.

TRIM_ONE (Befehl)

Verkürzt oder erweitert ein Element, so dass es genau zu einem anderen Element passt.

TRIM_TWO (Befehl)

Verkürzt oder erweitert zwei Elemente, so dass sie genau zueinander passen.

TRIM (arithmetische Funktion)

Ergibt eine Zeichenfolge, die durch Stripping aller führenden und nachfolgenden Leerzeichen vom Argument entsteht.

TRIMMING (Befehl)

Schaltet das automatische Entfernen nicht sichtbarer Spline-Kurventeile nach einer Zerlegung ein bzw. aus.

TRUE (arithmetische Funktion)

Liefert den Wert 1.

TRUE_COLOR_PLOTTING (Befehl)

Ermöglicht das Plotten mit echten Farben und schaltet die Plottransformation aus.

TRUNC (arithmetische Funktion)

Ergibt den ganzzahligen Teil einer reellen Zahl.

TYPE (arithmetische Funktion)

Ergibt den Typ des angegebenen Token.

TXT_WINDOW (Befehl)

Definiert ein Textfenster in einem schraffierten Bereich.

UA_ANGLE_GRID (Funktion)

Legt den von COPILOT verwendeten Winkelzuwachs fest.

UA_CENTER_CATCH_RANGE (Funktion)

Legt den Teil der Zeilenlängen fest, die von COPILOT beim Fangen als Mitte verwendet wird.

UA_DISTANCE_GRID (Funktion)

Legt den von COPILOT verwendeten Längenzuwachs fest.

UA_GET_DESIGN_INTENT (arithmetische Funktion)

Zeigt an, ob das Konstruktionsprotokoll (DESIGN_INTENT) ein- oder ausgeschaltet ist.

UA_PERPENDICULAR_CATCH_RANGE (Funktion)

Legt den Teil der Zeilen, Kreise und Bögen fest, der von COPILOT beim Fangen als Senkrechte verwendet wird.

`UA_TANGENT_CATCH_RANGE` (Funktion)

Legt den Teil der Kreise und Bögen fest, der von COPILOT beim Fangen als Tangente verwendet wird.

`UA_SET_CATCH_DELAY` (Funktion)

Legt die Zeit fest, die ohne Cursorbewegung verstreichen muss, bevor die Fanginformationen von COPILOT angezeigt werden.

`UNITS` (Funktion)

Gibt die aktuellen Einheiten für Abstände und Winkel an.

`UNLOAD_MODULE` (Befehl)

Entlädt Module.

`UNSHARE_PART` (Befehl)

Bewirkt, dass ein zuvor gemeinsam verwaltetes Teil nicht mehr gemeinsam verwaltet wird.

`UNTIL` (Pseudo-Befehl)

Boolescher Ausdruck.

`UPC` (arithmetische Funktion)

Wandelt Kleinbuchstaben in Großbuchstaben um.

`UPDATE_SCREEN` (Funktion)

Alle Geräte- oder Treiberpuffer werden auf dem Bildschirm freigegeben. Die Statuszeile und das Menü werden aktualisiert.

`USE_MULTILINE_HATCH` (Funktion)

Erlaubt die Auswahl zweier verschiedener Schraffierverfahren für Schraffuren mit einem Abstand von 0.

`VAL` (arithmetische Funktion)

Wandelt eine numerische Zeichenfolge in eine Zahl um.

`VERSION` (Funktion)

Zeigt Angaben zur Version des CAD-Programms an.

`VIEW` (Funktion)

Zeigt das ausgewählte Teil im aktuellen Darstellungsfenster an.

`WAIT` (Funktion)

Bewirkt, dass das System in der (in Sekunden) angegebenen Zeit keine Aktionen ausführt.

`WHILE` (Funktion)

Bewirkt ein bedingtes und wiederholtes Ausführen der Schleife, solange der Boolesche Ausdruck in der Klausel WHILE wahr ist.

`WHITE` (Funktion)

Ändert die Standardfarbe für Linien in Weiß.

`WINDOW` (Funktion)

Legt fest, welcher Teil der Zeichnung im aktuellen Darstellungsfenster angezeigt wird.

`WINEXEC` (arithmetische Funktion)

Startet die gewünschte Windows-Applikation mit dem entsprechenden Befehl.

`WRITE_FILE` (Funktion)

Schreibt eine Textzeile in eine angegebene Datei.

`WRITE_LTAB` (Funktion)

Schreibt neue Werte in die angegebene Benutzertabelle.

`X_OF` (arithmetische Funktion)

Ergibt die X-Koordinate eines Vektors.

`YELLOW` (Funktion)

`Y_OF` (arithmetische Funktion)

Ergibt die Y-Koordinate eines Vektors.

`Z_OF` (arithmetische Funktion)

Ergibt die Z-Koordinate eines Vektors.

A

Logische Tabellen und Anzeigetabellen

Dieser Anhang befaßt sich mit logischen Tabellen und Anzeigetabellen.

Der Aufbau von logischen Tabellen und Anzeigetabellen wird beschrieben und es wird dabei erläutert, wie eine Anzeigetabelle mit einer logischen Tabelle verbunden wird.

Anschließend werden die Zugriffsfunktionen für logische Tabellen sowie die zum Definieren von benutzerspezifischen Tabellen benötigten Befehle und Funktionen beschrieben.

Darüber hinaus werden die zum Definieren und Anwenden von Anzeigetabellen benötigten Befehle und Funktionen beschrieben.

Schließlich wird anhand von zwei Beispielen gezeigt, wie logische Tabellen und Anzeigetabellen definiert werden, wie aus einer logischen Tabelle eine Anzeigetabelle erstellt wird und wie mit den Tabellen gearbeitet wird.

Hinweis

Die Beschreibungen und Beispiele in diesem Kapitel setzen voraus, dass Sie sich bereits mit dem Programmieren von Makros in Creo Elements/Direct Drafting auskennen.

Was versteht man unter logischen Tabellen und Anzeigetabellen?

Logische Tabellen und Anzeigetabellen sind Hilfsmittel zum Ausgeben großer Datenmengen in übersichtlicher Form. Diese Darstellungsform ermöglicht dem Benutzer das Lesen und Verstehen der Daten. Ein weiterer Vorteil ist, dass die Tabelleneinträge direkt als Eingabewerte für Befehle und Funktionen ausgewählt werden können.

Der Einsatz von logischen Tabellen und Anzeigetabellen eignet sich besonders, wenn häufig Datenlisten bearbeitet und angezeigt werden müssen. Näheres dazu ist in den Abschnitten "Mit logischen Tabellen und Anzeigetabellen arbeiten - Beispiel 1" und "Mit logischen Tabellen und Anzeigetabellen arbeiten - Beispiel 2" nachzulesen.

In den folgenden Abschnitten wird der Aufbau von logischen Tabellen und Anzeigetabellen beschrieben und es wird erläutert, wie eine Anzeigetabelle mit einer logischen Tabelle verbunden wird.

Logische Tabellen

Eine logische Tabelle stellt eine system- oder benutzerdefinierte interne Datenstruktur dar.

Systemdefinierte logische Tabellen enthalten Systemdaten. Obwohl die Möglichkeit besteht, die Daten zu ändern, dürfen systemdefinierte logische Tabellen nur gelesen werden. Weder das Format noch die Daten in systemdefinierten logischen Tabellen dürfen geändert werden.

Benutzerdefinierte logische Tabellen, auch Benutzertabellen, enthalten benutzerbezogene Daten. Die Daten sind sowohl lesbar als auch überschreibbar. Formate für benutzerdefinierte logische Tabellen können erstellt und bestehende Formate dürfen geändert werden. Darüber hinaus ist neben dem Lesen der Daten auch das Eintragen von Daten in benutzerdefinierte logische Tabellen gestattet.

Eine logische Tabelle besteht normalerweise aus drei Hauptkomponenten:

1. Ein Array von N Datensätzen identifiziert durch eine Zahl von 1 bis N, wobei jeder Datensatz M Werte umfasst, die auch durch eine Zahl von 1 bis M identifiziert werden. Mit anderen Worten ist dies mit einer zweidimensionalen N x M-Matrix von Werten vergleichbar. Die Anzahl der Datensätze oder Einträge in einer solchen Tabelle kann unterschiedlich sein, aber die Anzahl der in einem Datensatz angegebenen Werte muss bei allen Datensätzen gleich sein.

> **Hinweis**
>
> Die Werte in einer logischen Tabelle können entweder Buchstaben- oder Zahlenfolgen sein.

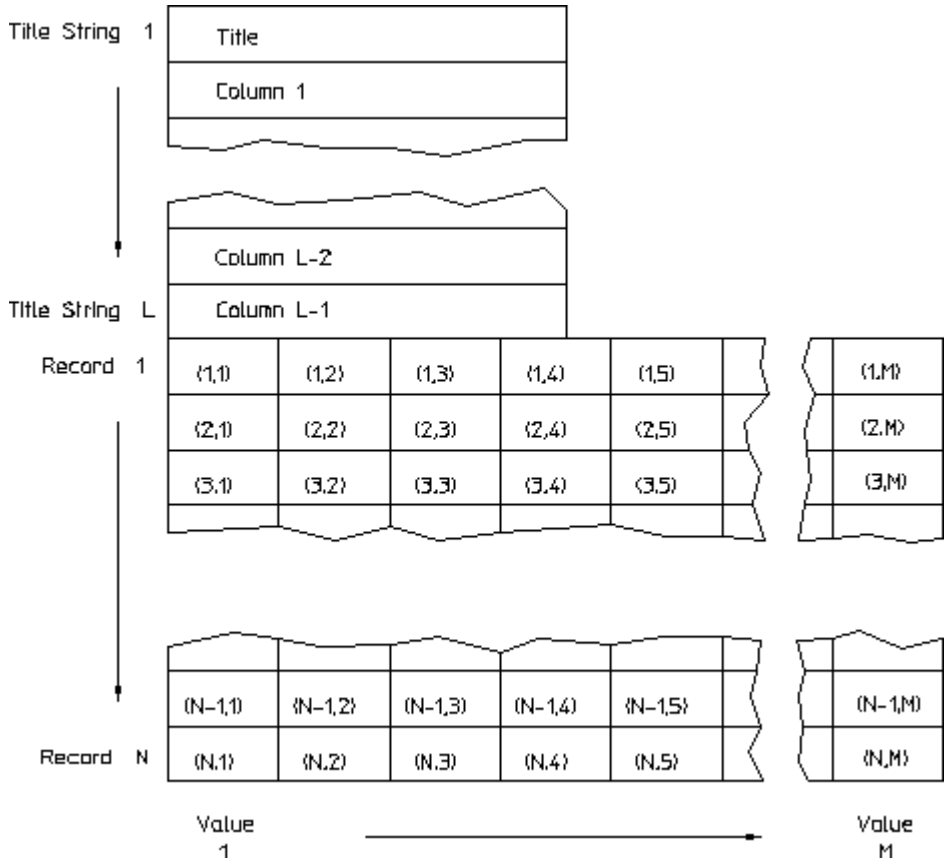
2. L Titelzeichenfolgen, die durch Zahlen von 1 bis L gekennzeichnet werden. Titelzeichenfolgen sind Buchstabenfolgen, mit denen zusätzliche Informationen angegeben werden können. Normalerweise stehen hier der Titel und die Spaltenüberschriften einer Anzeigetabelle. Näheres dazu kann im Abschnitt "Anzeigetabellen" nachgelesen werden.

> **Hinweis**
>
> Wenn keine Titelzeichenfolgen benötigt werden, müssen auch keine definiert werden. Es können aber auch leere Kopfzeichenfolgen definiert werden.

3. Zustandskennzeichnungen für die Werte in allen Datensätzen. Jeder Wert weist einen zugeordneten Zustand auf, der ihn als ausgewählt oder aktiv kennzeichnet. Diese Zusatzinformation kann z.B. folgendermaßen verwertet werden: Wenn der Benutzer Daten aus einer Anzeigetabelle als Eingabewerte für einen Befehl verwenden möchte, kann er diese Daten optisch hervorheben, um die getroffene Auswahl anzuzeigen.
4. Vordergrund- (Anzeige-) und Hintergrundfarben für die Tabellenfelder.

Die folgende Abbildung zeigt schematisch den Aufbau einer logischen Tabelle:

Abbildung 31. Logische Tabelle



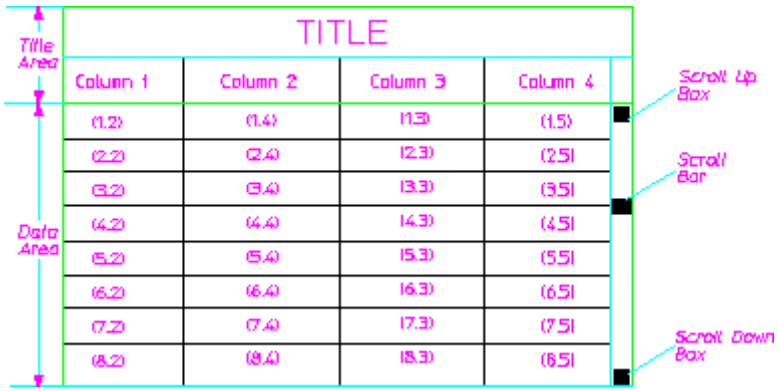
Anzeigetabellen

Eine Anzeigetabelle ist eine Tabelle, in der der Inhalt einer logischen Tabelle auf dem Bildschirm angezeigt werden kann. Es gibt systemdefinierte und benutzerdefinierte Anzeigetabellen. Normalerweise besteht eine Anzeigetabelle aus den folgenden Komponenten:

- Kopfbereich mit Tabellenüberschrift, Zwischenüberschriften und Spaltenüberschriften.
- Datenbereich mit Spalten für Werte.
- Vertikaler Auswahlbalken (wahlweise), der am rechten Rand der Tabelle erscheint. Horizontale Auswahlbalken sind nicht zulässig.

Die folgende Abbildung zeigt ein Beispiel für eine Anzeigetabelle:

Abbildung 32. Anzeigetabelle



Eine logische Tabelle mit einer Anzeigetabelle verbinden

In den einleitenden Abschnitten wurde gesagt, dass es zwei Arten von Tabellen gibt: Logische Tabellen und Anzeigetabellen. Eine logische Tabelle ist eine interne Datenstruktur, in der die eigentlichen Daten gespeichert sind.Dagegen ist eine Anzeigetabelle mit einem Fenster vergleichbar, durch das sich der Benutzer die Daten in einer logischen Tabelle ansehen kann. Ferner ermöglicht eine Anzeigetabelle das Ändern der Daten in der entsprechenden logischen Tabelle.

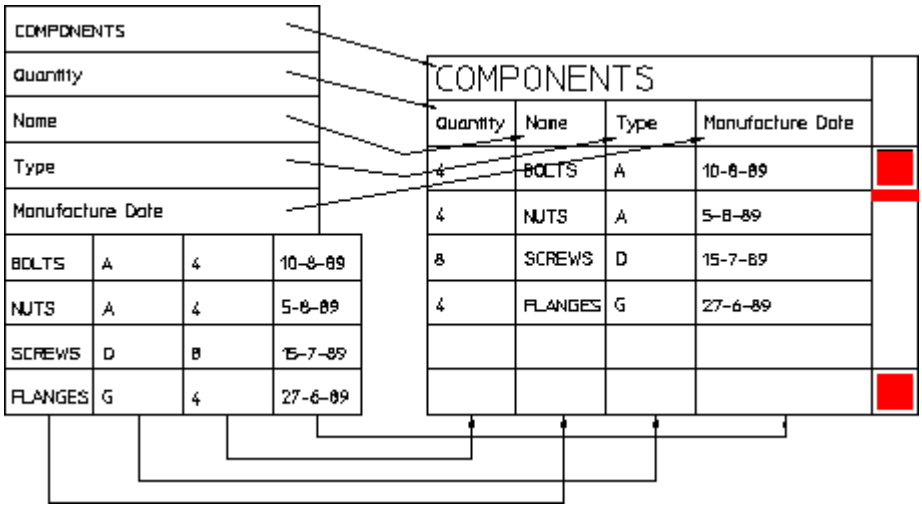
Jede logische Tabelle bzw. Anzeigetabelle erhält bei ihrer Definition einen systemweit nur einmal vorkommenden Namen.Es können beliebig viele logische Tabellen oder Anzeigetabellen definiert werden.

Über eine Anzeigetabelle kann erst dann auf eine logische Tabelle zugegriffen werden, wenn eine entsprechende Verbindung hergestellt wurde. Eine logische Tabelle kann mit mehreren Anzeigetabellen verbunden werden.Dadurch ist es möglich, verschiedene Ausschnitte aus einer umfangreichen logischen Tabelle separat anzuzeigen.

Diese Methode besitzt den Vorteil, dass die in einer logischen Tabelle gespeicherten Daten in einheitlicher Form dargestellt werden.Ferner werden die Anzeigetabellen automatisch aktualisiert, sobald sich die Daten in der logischen Tabelle ändern. Auch der Zeitaufwand für das Definieren gleichartiger Anzeigetabellen entfällt.

Die folgende Abbildung zeigt, wie die Informationen aus einer logischen Tabelle in eine Anzeigetabelle übertragen werden:

Abbildung 33. Zuordnen einer Anzeigetabelle zu einer logischen Tabelle



Die Pfeile in der obigen Abbildung zeigen, aus welchen Positionen in der logischen Tabelle die Elemente in die Anzeigetabelle übertragen werden.

Die Kopfzeichenfolgen der logischen Tabelle werden normalerweise zur Tabellenüberschrift und zu den Spaltenüberschriften der Anzeigetabelle. Die Datenspalten in der logischen Tabelle können den Datenspalten der Anzeigetabelle in beliebiger Reihenfolge zugewiesen werden. Es ist auch möglich, nur einige Datenspalten aus der logischen Tabelle in die Anzeigetabelle zu übernehmen.

Zugriffsfunktionen für logische Tabellen

Die Daten in system- und benutzerdefinierten logischen Tabellen können mit Hilfe von Zugriffsfunktionen angesprochen werden. Die folgenden Funktionen stehen zur Verfügung:

- `LTAB_COLUMNS`
- `LTAB_ROWS`
- `LTAB_TITLES`
- `POP_DOWN_LTAB`
- `POP_UP_LTAB`
- `READ_LTAB`
- `SAVE_LTAB`

- SCROLL_LTAB
- SELECT_FROM_LTAB

Diese Funktionen werden in den folgenden Abschnitten ausführlich behandelt.

LTAB_COLUMNS

Mit diesem Befehl kann die Anzahl der Spalten in der angegebenen logischen Tabelle abgefragt und in eine andere Anweisung eingebaut werden.

Die Syntax des Befehls sieht folgendermaßen aus:
```
LTAB_COLUMNS 'Logical table name'
```

Ein Beispiel für den Befehl:
```
LTAB_COLUMNS  'logtable1'
```

LTAB_COLUMNS erfordert einen Parameter, bei dem es sich um den Namen der logischen Tabelle handelt, die abgefragt werden soll. In diesem Beispiel ist die logische Tabelle logtable1.

Da der Befehl LTAB_COLUMNS einen Wert liefert, muss dieser Wert einer Variablen zugewiesen werden. Dies wird im folgenden Beispiel mit einer typischen Anwendung des Befehls LTAB_COLUMNS demonstriert:
```
LET  num_columns  (LTAB_COLUMNS  'logtable1')
```

Das bedeutet, dass der LTAB_COLUMNS-Befehl die Anzahl der Spalten in der logischen Tabelle logtable1 zurückgibt und LET sie dann der Makrovariable num_columns zuweist.

Der gelieferte Wert ist eine Zahl.

LTAB_ROWS

Mit diesem Befehl kann die Anzahl der Zeilen in der angegebenen logischen Tabelle abgefragt und in eine andere Anweisung eingebaut werden.

Die Syntax des Befehls sieht folgendermaßen aus:
```
LTAB_ROWS 'Logical table name'
```

Ein Beispiel für den Befehl:
```
LTAB_ROWS  'logtable1'
```

LTAB_ROWS erfordert einen Parameter, bei dem es sich um den Namen der logischen Tabelle handelt, die abgefragt werden soll. In diesem Beispiel ist die logische Tabelle logtable1.

Da der Befehl LTAB_ROWS einen Wert liefert, muss dieser Wert einer Variablen zugewiesen werden. Dies wird im folgenden Beispiel mit einer typischen Anwendung des Befehls LTAB_ROWS demonstriert:
```
LET  num_rows  (LTAB_ROWS  'logtable')
```

Das bedeutet, dass der LTAB_ROWS-Befehl die Anzahl der Zeilen in der logischen Tabelle logtable1 zurückgibt und LET sie dann der Makrovariable num_rows zuweist.

Der gelieferte Wert ist eine Zahl.

LTAB_TITLES

Mit diesem Befehl kann die Anzahl der Kopfzeichenfolgen in der angegebenen logischen Tabelle abgefragt und in eine andere Anweisung eingebaut werden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
LTAB_TITLES 'Logical table name'
```

Ein Beispiel für den Befehl:

```
LTAB_TITLES  'logtable1'
```

`LTAB_TITLES` erfordert einen Parameter, bei dem es sich um den Namen der logischen Tabelle handelt, die abgefragt werden soll. In diesem Beispiel ist die logische Tabelle `logtable1`.

Da der Befehl `LTAB_TITLES` einen Wert liefert, muss dieser Wert einer Variablen zugewiesen werden. Dies wird im folgenden Beispiel mit einer typischen Anwendung des Befehls `LTAB_TITLES` demonstriert:

```
LET  num_titles  (LTAB_TITLES  'logtable1')
```

Das bedeutet, dass der `LTAB_TITLES`-Befehl die Anzahl der Titel in der logischen Tabelle `logtable1` zurückgibt und `LET` sie dann der Makrovariable `num_titles` zuweist.

Der gelieferte Wert ist eine Zahl.

POP_DOWN_LTAB

Dieser Befehl bewirkt, dass die angegebene logische Tabelle bei nächster Gelegenheit ausgeblendet wird. Daraufhin verschwinden alle damit verbundenen Anzeigetabellen vom Bildschirm.

Pop-down-Aufrufe werden zunächst in einem Puffer gehalten, bis das System neue Benutzereingaben verarbeiten kann oder `UPDATE_SCREEN` aufgerufen wird. Erst danach wird mit dem Abarbeiten der Pop-down-Aufrufe begonnen.

Die Syntax des Befehls sieht folgendermaßen aus:

```
POP_DOWN_LTAB 'Logical table name'
```

Ein Beispiel für den Befehl:

```
POP_DOWN_LTAB  'logtable1'
```

`POP_DOWN_LTAB` erfordert einen Parameter, bei dem es sich um den Namen der logischen Tabelle handelt, die ausgeblendet werden soll. In diesem Beispiel ist die logische Tabelle `logtable1`.

Hinweis

- Wenn mit dem Befehl POP_DOWN_LTAB eine logische Tabelle ausgeblendet wird, werden alle mit ihr verbundenen Anzeigetabellen im Bildschirm entfernt. Wenn von mehreren Anzeigetabellen nur eine entfernt werden soll, kann der Befehl SHOW_TABLE unter Angabe von OFF verwendet werden.
- Wenn eine Anzeigetabelle durch den Befehl POP_DOWN_LTAB vom Bildschirm entfernt wird, wird der zwischengespeicherte Grafikausschnitt für den betreffenden Bildschirmbereich unter der entfernten Anzeigetabelle neu angezeigt. Das System weiß jedoch nichts von dieser Anzeige. Der zwischengespeicherte Grafikausschnitt enthält jetzt eine Anzeigetabelle (oder einen Ausschnitt davon), die bereits einmal vom Bildschirm entfernt wurde. Beim Neuaufbau des Grafikausschnitts erscheint demnach die Anzeigetabelle (bzw. der Rest der Anzeigetabelle) wieder auf dem Bildschirm. Das System weiß jedoch nichts von dieser Anzeige. Sie haben jetzt das Problem, eine ganze Tabelle oder einen Tabellenausschnitt nicht mit dem Befehl TABLES OFF entfernen zu können. In diesem Fall können Sie den Befehl NEW_SCREEN erteilen. Daraufhin wird der tatsächliche Inhalt des gesamten Bildschirms mit dem Befehl NEW_SCREEN neu erzeugt.

POP_UP_LTAB

Dieser Befehl bewirkt, dass die angegebene logische Tabelle bei nächster Gelegenheit angezeigt wird. Wenn eine logische Tabelle erscheint, werden alle damit verbundenen Anzeigetabellen auf dem Bildschirm angezeigt.

Pop-up-Aufrufe werden zunächst in einem Puffer gehalten, bis das System neue Benutzereingaben verarbeiten kann oder UPDATE_SCREEN aufgerufen wird. Erst danach wird mit dem Abarbeiten der Pop-up-Aufrufe begonnen.

Die Syntax des Befehls sieht folgendermaßen aus:

```
POP_UP_LTAB 'Logical table name'
```

Ein Beispiel für den Befehl:

```
POP_UP_LTAB  'logtable1'
```

POP_UP_LTAB erfordert einen Parameter, bei dem es sich um den Namen der logischen Tabelle handelt, die eingeblendet werden soll. In diesem Beispiel ist die logische Tabelle logtable1.

Hinweis

- Wenn mit dem Befehl POP_UP_LTAB eine logische Tabelle eingeblendet wird, werden alle mit ihr verbundenen Anzeigetabellen auf dem Bildschirm angezeigt. Wenn von mehreren Anzeigetabellen nur eine angezeigt werden soll, kann der Befehl SHOW_TABLE unter Angabe von ON verwendet werden.
- Wenn der Befehl POP_UP_LTAB eine Anzeigetabelle auf dem Bildschirm anzeigt, wird ein Grafikausschnitt des Bildschirmbereichs unter der Anzeigetabelle gespeichert. Wenn die Anzeigetabelle mit POP_DOWN_LTAB vom Bildschirm entfernt wird, wird der ursprüngliche Bereich wiederhergestellt.

READ_LTAB

Mit diesem Befehl können die Werte in einem bestimmten Bereich der angegebenen logischen Tabelle gelesen werden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
READ_LTAB 'Logical table name'
(Row-number Column-number) or (TITLE Title-string-number)
```

Ein Beispiel für den Befehl:

```
READ_LTAB  'logtable1'
2  3
```

READ_LTAB erfordert zwei Parameter. Der erste Parameter ist der Name der logischen Tabelle, in der gelesen werden soll. In diesem Beispiel ist die logische Tabelle logtable1. Der zweite Parameter ist die Position für die zu lesenden Daten innerhalb der angegebenen logischen Tabelle. Hierfür wurden in diesem Beispiel die Werte 2 3 eingesetzt, d.h. Zeile 2 und Spalte 3.

Wenn Sie einen Wert aus der Titelzeichenfolge der angegebenen logischen Tabelle lesen möchten, geben Sie TITLE 5 statt 2 3 an, um die Titelzeichenfolge 5 anzugeben.

Da der Befehl READ_LTAB einen Wert liefert, muss dieser Wert einer Variablen zugewiesen werden. Dies wird im folgenden Beispiel mit einer typischen Anwendung des Befehls READ_LTAB demonstriert:

```
LET  my_value  (READ_LTAB  'logtable1' 1  4)
```

Das bedeutet, dass der READ_LTAB-Befehl den Wert aus der 4. Spalte der ersten Zeile in der logischen Tabelle logtable1 liest und LET ihn dann der Makrovariable my_value zuweist.

Der angezeigte Wert kann eine Zeichenfolge oder eine Zahl sein. Wenn im angegebenen Feld kein Wert steht, ergibt die gelieferte Information ein Leerfeld.

SAVE_LTAB

Mit diesem Befehl wird Inhalt der angegebenen logischen Tabelle gesichert.

Die Syntax des Befehls sieht folgendermaßen aus:

```
SAVE_LTAB 'Logical table name'
SCREEN or (( or DEL_OLD or APPEND) 'filename')
```

Ein Beispiel für den Befehl:

```
SAVE_LTAB  'logtable1'
DEL_OLD  'file1.tbl'
```

`SAVE_LTAB` erfordert zwei Parameter. Der erste Parameter ist der Name der logischen Tabelle, die gesichert werden soll. In diesem Beispiel ist die logische Tabelle `logtable1`. Der zweite Parameter gibt das Ausgabeziel für die Sicherungskopie der angegebenen logischen Tabelle an. In diesem Beispiel ist das Ausgabenziel die Datei `file1.tbl` mit der Option `DEL_OLD`. Das bedeutet, wenn `file1.tbl` bereits vorhanden ist, wird sie überschrieben.

Wenn Sie eine Datei als Ausgabeziel angeben, können Sie alternativ die Option `APPEND` oder keine Option verwenden. `APPEND` bedeutet, dass die angegebene logische Tabelle an das Ende der angegebenen Datei angefügt wird. Falls noch keine Datei vorhanden ist, wird eine Datei erstellt. Wenn Sie gar keinen Parameter angeben, wird eine Datei mit dem angegebenen Namen erstellt, sofern keine andere Datei mit gleichem Namen vorhanden ist. Wenn die angegebene Datei bereits vorhanden ist, wird eine Fehlermeldung ausgegeben und der Befehl abgebrochen.

Alternativ können Sie den Bildschirm als Ausgabeziel angeben, indem Sie `SCREEN` statt `DEL_OLD` und `file1.tbl` in obigem Beispiel angeben. Das Speichern einer Tabelle in einer Datei hat jedoch den Vorteil, dass Sie anschließend über den Befehl `INPUT` die Datei laden und neu definieren können.

SCROLL_LTAB

Dieser Befehl gestattet das Blättern in Anzeigetabellen, die mit der angegebenen logischen Tabelle verbunden sind.Die angegebene Zeile wird als oberste Zeile angezeigt.

Die Syntax des Befehls sieht folgendermaßen aus:

```
SCROLL_LTAB 'Logical table name'
Row-number
```

Ein Beispiel für den Befehl:

```
SCROLL_LTAB  'logtable1'
3
```

`SCROLL_LTAB` erfordert zwei Parameter. Der erste Parameter ist der Name der logischen Tabelle, in der geblättert werden soll. In diesem Beispiel ist die logische Tabelle `logtable1`. Der zweite Parameter gibt die Zeile an, die in der obersten Zeile in der Anzeigetabelle angezeigt werden soll. In diesem Beispiel wird hierfür der Wert `3` gewählt.

SELECT_FROM_LTAB

Mit diesem Befehl werden in einer logischen Tabelle, der Quelltabelle, Zeilen nach bestimmten Kriterien ausgewählt. Die entsprechenden Tabellenpositionen werden in die Tabelle sys_select geschrieben. Wahlweise ist es mit diesem Befehl auch möglich, den Inhalt der den Auswahlkriterien entsprechenden Zeilen in eine andere logische Tabelle, die Zieltabelle, zu kopieren.

Die Syntax des Befehls sieht folgendermaßen aus:

```
SELECT_FROM_LTAB 'Source Logical table name'
COLUMN Column-number
or ( = or <> or > or < or >= or <= )
'Selection-string' or Selection-number
END or ( or APPEND 'Destination Logical table name')
```

Ein Beispiel für den Befehl:

```
SELECT_FROM_LTAB  'logtable1'
COLUMN  1
 >=
10
END
```

`SELECT_FROM_LTAB` erfordert fünf Parameter. Der erste Parameter ist der Name der Quelltabelle, aus der Daten ausgewählt werden sollen. Im Beispiel oben ist die logische Tabelle `logtable1`. Der zweite Parameter ist die Position der Spalte für die Auswahl. Das ist in diesem Beispiel `COLUMN 1`, also die erste Spalte. Der dritte Parameter ist der Operator für die Auswahl, in diesem Beispiel >= (größer als oder gleich). Andere mögliche Operatoren sind =, <>, >, < und <=. Der vierte Parameter ist die Auswahlzeichenfolge oder der Auswahlwert: `10`. D.h. im Beispiel oben die Zahl `10` und nicht die Zeichenfolge "10". Der fünfte Parameter ist entweder `END` oder der Name der Zieltabelle, in die die ausgewählten Zeilen kopiert werden sollen. Im obigen Beispiel wird `END` angegeben (Ende des Befehls).

Zusammengefasst werden in obigem Beispiel Zeilen aus der logischen Quelltabelle `logtable1` ausgewählt, wenn `COLUMN 1` einer Zeile den Wert = `10` aufweist, und die Positionen der ausgewählten Zeilen in die logische Tabelle `sys_select` geschrieben.

Beispiel: Die logische Tabelle `logtable1` enthält die folgenden Daten:

```
1    20
5    25
10   11
22   17
43   13
```

und Sie führen das Makroprogramm aus obigem Beispiel aus. Die Folge ist, dass die logische Tabelle sys_select die folgenden Daten enthält:

```
3
4
5
```

da nur die Zeilen 3, 4 und 5 von `logtable1` die Auswahlkriterien erfüllen.

Ein weiteres Beispiel für den Befehl:

```
SELECT_FROM_LTAB  'logtable1'
COLUMN  2
=
'PARTS'
APPEND  'logtable2'
```

Zusammengefasst werden in obigem Beispiel Zeilen aus der logischen Quelltabelle `logtable1` ausgewählt, wenn `COLUMN 2` einer Zeile die Zeichenfolge = `'PARTS'` aufweist, und die Positionen der ausgewählten Zeilen in die logische Tabelle `sys_select` geschrieben. Darüber hinaus wird `APPEND` ausgeführt. D.h., die ausgewählten Zeilen werden an das Ende der logischen Zieltabelle `logtable2` kopiert.

Wenn Sie die Option `APPEND` angeben, nimmt der Befehl an, dass die angegebene logische Tabelle bereits vorhanden ist. Wenn Sie die Option `APPEND` nicht angeben, geht der Befehl davon aus, dass die angegebene logische Tabelle nicht vorhanden ist, und es wird eine neue logische Tabelle mit dem angegebenen Namen erstellt.

Die logischen Tabellen `logtable1` und `logtable2` enthalten beispielsweise die folgenden Daten:

```
PISTON        DRAWING     4
BOLTS         PARTS       10
NUTS          PARTS       10
CRANKSHAFT    DRAWING     1
WASHERS       PARTS       10
```

und

```
WHEEL         ASSEMBLY    2
FRAME         ASSEMBLY    1
HANDLE        ASSEMBLY    1
```

und Sie führen das Makroprogramm aus obigem Beispiel aus. Dies führt dazu, dass die logischen Tabellen `sys_select` und `logtable2` die folgenden Daten enthalten:

```
2
3
5
```

und

```
WHEEL         ASSEMBLY    2
FRAME         ASSEMBLY    1
HANDLE        ASSEMBLY    1
BOLTS         PARTS       10
NUTS          PARTS       10
WASHERS       PARTS       10
```

da nur die Zeilen 2, 3 und 5 von `logtable1` das Auswahlkriterium erfüllen. Sie können sehen, dass die Inhalte der Zeilen 2, 3 und 5 von `logtable1` kopiert und an das Ende von `logtable2` angehängt werden.

Funktionen für Anzeigetabellen

In einer Anzeigetabelle wird der Inhalt einer logischen Tabelle je nach Auswahl vollständig oder teilweise angezeigt. Näheres dazu ist im Abschnitt "Anzeigetabellen" nachzulesen.

In den folgenden Abschnitten werden die Befehle und Funktionen beschrieben, mit denen das Layout einer Anzeigetabelle erstellt werden kann.Auch auf die Handhabung von Anzeigetabellen wird eingegangen. Ferner wird ein Beispiel für das Definieren einer Anzeigetabelle geliefert.

Hinweis

Für die Befehlsbeispiele gilt Folgendes: In normaler Schrift gesetzter Text muss buchstabengetreu eingegeben werden, damit der jeweilige Befehl funktioniert. Kursiv gesetzter Text ist durch geeignete Zeichenfolgen oder Werte zu ersetzen. Die Leerzeilen wurden nur aus Gründen der Übersichtlichkeit eingefügt und sollen Ihnen helfen, den Aufbau des Befehls zu verstehen. Das Wort "oder" bedeutet, dass zwei oder mehr Parameter vorhanden sind, die Sie als Parameter wählen können.

Eine Anzeigetabelle kann mit den folgenden Befehlen definiert werden:

- `TABLE_COLUMN`
- `TABLE_LAYOUT`
- `TABLE_TITLE`

Näheres dazu kann unter "Eine Anzeigetabelle definieren" im Abschnitt "Mit logischen Tabellen und Anzeigetabellen arbeiten - Beispiel 1" nachgelesen werden.

Die oben aufgeführten Befehle werden in den folgenden Abschnitten ausführlich beschrieben.

TABLE_COLUMN

Mit diesem Befehl werden die Attribute (Überschriften) für die Datenspalten einer vorhandenen Anzeigetabelle gesetzt.

Hinweis

Dieser Befehl funktioniert nur bei Anzeigetabellen, die nicht mit dem Befehl `SECURE_TABLE` vor Änderungen geschützt wurden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
TABLE_COLUMN 'table name'
COLUMN column-number
display color
background color
Logical-table-column-number
FORMAT Format-precision or 'Numeric string'
CENTER or LEFT or RIGHT
'Action Text'
END
```

Ein Beispiel für den Befehl:

```
TABLE_COLUMN 'table1'
  COLUMN 1
  blue
  yellow
  1
  FORMAT 2
  LEFT
```

```
  'LINE'

  COLUMN 2
  red
  green
  2
  FORMAT 3
  RIGHT
  'CIRCLE'
END
```

Hinweis

Im obigen Beispiel gehören die Zahlen am linken Rand NICHT zur Befehlssyntax. Diese Zahlen dienen nur zur leichteren Identifizierung der einzelnen Zeilen. Die Leerzeilen sind ebenfalls nur Lese- und Verständnishilfen.

Zeile 1 enthält `TABLE_COLUMN` und den Namen der Anzeigetabelle, deren Spalten definiert werden müssen.

Zeile 2 gibt `COLUMN 1` an. D.h., Spalte 1 der Anzeigetabelle muss definiert werden.

Zeile 3 gibt die Anzeigefarbe für Text in der Spalte an. In diesem Beispiel wird `blue` angegeben. Wenn Sie eine andere Farbe benötigen, können Sie nach der Beschreibung für Zeile 3 von `TABLE_LAYOUT` vorgehen.

Zeile 4 gibt die Hintergrundfarbe der Spalte an. In diesem Beispiel wird `yellow` angegeben. Wenn Sie eine andere Farbe benötigen, können Sie nach der Beschreibung von `TABLE_LAYOUT` vorgehen.

Zeile 5 gibt die Position der Spalte in der verbundenen logischen Tabelle an, die in dieser Spalte in der Anzeigetabelle angezeigt werden soll.

Zeile 6 gibt `FORMAT 2` an. D.h., dass die Formatgenauigkeit für Gleitkommazahlen zwei signifikante Stellen umfasst. Beispiele:Für die Dezimalzahl 123,0 wird der Wert 120 angezeigt. Für die Dezimalzahl 12,6 wird der Wert 13 angezeigt. Standardmäßig wird die halbe Feldbreite der Spalte für signifikante Stellen verwendet.

Sie können auch eine numerische Zeichenfolge angeben, beispielsweise `'+1.2345'`. Dann erhält der Anzeigewert ein Vorzeichen und vier Dezimalstellen mit Nullenunterdrückung links und rechts vom Komma. Mit dem Befehl `HELP` können Sie Näheres über die Funktion `DIM_FORMAT` und die Darstellungsform dieser numerischen Zeichenfolge erfahren.

Zeile 7 gibt `LEFT`an. D.h., die Daten in der Spalte müssen linksbündig ausgerichtet sein. Es stehen noch zwei andere Werte zur Auswahl: `CENTER` für zentriert und `RIGHT` für rechtsbündig.

Zeile 8 gibt den Aktionstext `LINE` in der Spalte an. Hier kann eine beliebige Textzeichenfolge angegeben werden, die ein gültiges Systemkommando darstellt.

Sie können das Schema `@s#` angeben, d.h. die Textzeichenfolge stammt von der Titelzeichenfolge Nummer # der logischen Tabelle. Die Angabe `@s3` bedeutet, dass die zusätzliche Zeichenfolge 3 der verknüpften logischen Tabelle die

tatsächliche angegebene Textzeichenfolge ist. Wenn Sie `@t3` angeben, handelt es sich bei der zusätzlichen Zeichenfolge 3 der verknüpften logischen Tabelle in einfachen Anführungszeichen um die tatsächliche angegebene Textzeichenfolge. `@s3` und `@t3` sind gleich. `@t3` weist jedoch zwei einfache Anführungszeichen auf, eins vor und eins nach der zusätzlichen Zeichenfolge 3.

Sie können auch das Format `@v#` verwenden, d.h. die Textzeichenfolge stammt aus den Daten in Spalte `#` der logischen Tabelle, die mit dieser Anzeigetabelle verknüpft ist. Beispiel: Die Angabe `@v4` bedeutet, dass es sich bei den Daten in Spalte 4 der verknüpften logischen Tabelle um die tatsächliche angegebene Textzeichenfolge handelt. Die Angabe `@q4` bedeutet, dass es sich bei den Daten in Spalte 4 der verknüpften logischen Tabelle in einfachen Anführungszeichen um die tatsächliche angegebene Textzeichenfolge handelt. `@v4` und `@q4` sind gleich. `@q4` weist jedoch zwei einfache Anführungszeichen auf, eins vor und eins nach den Daten in Spalte 4.

Sie können sogar eine Textzeichenfolge im Format `LINE @s4 @v2` angeben. Wenn die zusätzliche Zeichenfolge 4 und die Daten in Spalte 2 der verknüpften logischen Tabelle `TWO_PNTS` und `33,33` sind, ist die tatsächliche Textzeichenfolge `LINE TWO_PNTS 33,33`.

Die Zeilen 10 bis 16 bilden einen weiteren Parameterblock, der ähnlich aufgebaut ist wie die Zeilen 2 bis 8. Damit wird eine weitere Spalte im Titel der Anzeigetabelle definiert.

Zeile 17 gibt `END` an, um das Ende des `TABLE_COLUMN`-Befehls zu definieren.

Hinweis

Sie können beliebig viele Parameterblöcke ähnlich dem aus Zeile 2 bis 8 einfügen, sofern diese Blöcke zwischen den Befehlen `TABLE_COLUMN` und `END` stehen.

Das obige Beispiel kann auch folgendermaßen umgestellt werden:

```
TABLE_COLUMN 'table1'
  COLUMN 1 blue yellow 1 FORMAT 2 LEFT  'LINE'
  COLUMN 2 red  green  2 FORMAT 3 RIGHT 'CIRCLE'
END
```

Wenn ein Parameterblock in eine Zeile passt, ist dieses Format besser lesbar und leichter vergleichbar. Wichtig ist hierbei, dass die Parameter mindestens durch einen Leer- oder Tabulatorschritt getrennt werden.

TABLE_LAYOUT

Mit diesem Befehl wird eine Tabelle im gewünschten Layout erstellt.

Die Syntax des Befehls sieht folgendermaßen aus:

```
TABLE_LAYOUT 'table name'
'Logical table name'
display color
background color
WIDTH width
HEIGHT height
ROWS row
```

```
FRAME_WIDTH frame-width
HORIZONTAL Line-color Linetype
VERTICAL Line-color Linetype
SCROLL_BAR Foreground Color background color width
Point 1 and/or Point 2
TITLE_LAYOUT
Height of Row 1 'Layout String'
Height of Row 2 'Layout String'
:
END
COLUMN_LAYOUT
Height of Each Data Row 'Layout String'
END
```

Ein Beispiel für den Befehl:

```
TABLE_LAYOUT 'table1'
  'query_results'
  white
  black
  WIDTH  30.0
  HEIGHT 28.0
  ROWS   10
  FRAME_WIDTH 1
  HORIZONTAL white solid
  VERTICAL white solid
  SCROLL_BAR blue white 30
  0,0
  TITLE_LAYOUT
    40 '                                                '
    20 '              |      |   |             |         '
END
  COLUMN_LAYOUT
    20 '              |      |   |             |         '
END
```

Hinweis

Im obigen Beispiel gehören die Zahlen am linken Rand NICHT zur Befehlssyntax. Diese Zahlen dienen nur zur leichteren Identifizierung der einzelnen Zeilen. Die Leerzeilen sind ebenfalls nur Lese- und Verständnishilfen.

Zeile 1 enthält den Befehl `TABLE_LAYOUT` und den Namen, den Sie für diese Anzeigetabelle angeben.

In Zeile 2 steht der Name der logischen Tabelle, aus der die Daten für diese Anzeigetabelle entnommen werden sollen. In diesem Beispiel wird `query_results` angegeben. Standardmäßig wird der Name der logischen Tabelle auch für die Anzeigetabelle verwendet.

Zeile 3 gibt die Anzeigefarbe für Rahmen und Text der Anzeigetabelle an. In diesem Beispiel wird `white` angegeben. Sie können jedoch eine andere Farbe angeben, die bereits im System definiert ist, wie `red`, `yellow`, `green`, `cyan`, `magenta`, `blue` oder `black`.

Wenn Sie eine Farbe benötigen, die nicht zu den acht im System definierten Farben gehört, können Sie den Befehl `rgb_color` und anschließend die Dezimalwerte für die Farben Rot, Grün und Blau in der angegebenen Reihenfolge eingeben. Wenn Sie beispielsweise ein reines Rot, Grün oder Blau benötigen, geben Sie `rgb_color 1 0 0`, `rgb_color 0 1 0` oder `rgb_color 0 0 1` an.

Wenn Sie diese drei Farben kombinieren, können Sie eine bestimmte erforderliche Farbe wie `rgb_color 0.5 0.42 0.8` definieren (Rot: `0.5`, Grün: `0.42` und Blau: `0.8`).

Zeile 4 gibt die Hintergrundfarbe für die Anzeigetabelle an. In diesem Beispiel wird `black` angegeben.

Wenn Sie eine andere Farbe benötigen, können Sie nach der Beschreibung für Zeile 3 vorgehen.

Zeile 5 umfasst `WIDTH` und die Tabellenbreite `30.0` Zeichen. Dieser Wert kann ein Dezimalbruch sein (z.B. `35.5`) und muss eine bestimmte Mindestbreite überschreiten. Standardmäßig wird die zum Unterbringen der Tabelle benötigte Mindestbreite eingesetzt.

Zeile 6 umfasst `HEIGHT` und die Tabellenhöhe `28.0` Zeichen. Dieser Wert kann ein Dezimalbruch sein (z.B. `35.5`) und muss eine bestimmte Mindesthöhe überschreiten. Standardmäßig wird die zum Unterbringen der Tabelle benötigte Mindesthöhe eingesetzt.

Zeile 7 umfasst `ROWS` und die Anzahl der Datenzeilen (`10`) für die Anzeigetabelle. Dieser Parameter kann ein Dezimalbruch sein, zum Beispiel `10.5`. Dieser Parameter sollte nur angegeben werden, wenn Breite und Höhe der Tabelle nicht ausdrücklich definiert werden.

Zeile 8 umfasst `FRAME_WIDTH` und die Rahmenbreite `1`. Dieser Parameter kann die Werte 0, 1 oder 2 annehmen.

Zeile 9 umfasst `HORIZONTAL`, die horizontale Zeilenfarbe `white` und den Typ `solid`. Wenn Sie eine andere Farbe für die horizontalen Linien benötigen, können Sie nach der Beschreibung für Zeile 3 vorgehen. Wenn Sie eine anderen Linientyp benötigen, können Sie einen der anderen Linientypen angeben, die bereits im System definiert sind (wie `dotted`, `dashed`, `long dashed`, `dot center`, `dash center`, `phantom` oder `long dotted`). Wenn Sie nur `OFF` nach `HORIZONTAL` angeben, sind keine horizontalen Linien in der Anzeigetabelle vorhanden.

Zeile 10 umfasst `VERTICAL`, die vertikale Zeilenfarbe `white` und den Typ `solid`. Andere Farben und Linienarten können Sie nach der Beschreibung für Zeile 9 angeben. Wenn Sie nur `OFF` nach `VERTICAL` angeben, sind keine vertikalen Linien in der Anzeigetabelle vorhanden.

Zeile 11 umfasst `SCROLL_BAR`, die Vordergrundfarbe `blue`, die Hintergrundfarbe `white` und die Bildlaufleistenbreite von `30` Pixeln.

Zeile 12 gibt die Koordinaten (0,0) in Pixeln der unteren linken Ecke der Anzeigetabelle an. Wahlweise können auch die Koordinaten der oberen rechten Ecke der Anzeigetabelle angegeben werden.

Zeile 14 enthält `TITLE_LAYOUT`, den Parameter für das Layout des Titels in der Anzeigetabelle.

Zeile 15 enthält die Höhenangabe von `40` Pixeln für die erste Titelzeile sowie die Feldeinteilung `' '`. Daraus geht die Feldanzahl und -länge hervor. In diesem Beispiel wird nur ein Feld definiert, das Platz für 50 Zeichen bietet.

Zeile 16 enthält die Höhenangabe von `20` Pixeln für die erste Titelzeile sowie die Feldeinteilung `' | | | | '`. Daraus geht die Feldanzahl und -breite hervor. In diesem Beispiel werden fünf Spalten definiert, die jeweils Platz für 14, 6, 3, 14 und 9 Zeichen bieten. Der Balken (`|`) trennt die Felder.

Hinweis

In diesem Beispiel enthält der Tabellentitel nur zwei Zeilen. Weitere Zeilen können nach obigem Schema programmiert werden.

Zeile 18 enthält `END`, um das Ende des `TITLE_LAYOUT`-Parameters zu markieren.

Zeile 20 enthält `COLUMN LAYOUT`, den Parameter für das Layout der Datenspalten und -zeilen in der Anzeigetabelle.

Zeile 21 enthält die Höhenangabe von `20` Pixeln für jede Datenzeile sowie die Feldeinteilung `' | | | | '`. Daraus geht die Spaltenanzahl und -breite hervor. In diesem Beispiel werden fünf Spalten definiert, die jeweils Platz für 14, 6, 3, 14 und 9 Zeichen bieten. Der Balken (`|`) trennt die Spalten.

Zeile 23 enthält `END`, um das Ende des `COLUMN_LAYOUT`-Parameters und des `TABLE_LAYOUT`-Befehls zu markieren.

Hinweis

Sie können die Größe einer Anzeigetabelle durch eine Kombination der Parameter `WIDTH`, `HEIGHT`, `Point 1` und `Point 2` im `TABLE_LAYOUT`-Befehl angeben. Darüber hinaus können Sie mit `TITLE_LAYOUT` die Breite des Titelbereichs und mit `COLUMN_LAYOUT` die Spaltenbreite angeben. Wenn sich die in `TABLE_LAYOUT` angegebene Tabellenbreite von der Breite in `TITLE_LAYOUT` oder `COLUMN_LAYOUT` unterscheidet, hat die Breite in `TABLE_LAYOUT` Vorrang. Demnach richtet sich die Breite einer Anzeigetabelle stets nach dem in `TABLE_LAYOUT` angegebenen Wert.

Wenn die Breite in `TABLE_LAYOUT` beispielsweise 18 Spalten und die Breite in `TITLE_LAYOUT` oder `COLUMN_LAYOUT` 9 Spalten ist, gilt Folgendes:

```
40 '   |   |   '
```

Dann wird die Breite der Anzeigetabelle auf 18 Zeichen gesetzt. Daraufhin ändert sich die Breite der Titel und Spalten auf 18 Zeichen, doch die Proportionen bleiben gleich:

```
40 '      |      |      '
```

TABLE_TITLE

Mit diesem Befehl werden die Attribute (Felder) im Kopf einer vorhandenen Anzeigetabelle gesetzt.

Hinweis

Dieser Befehl funktioniert nur bei Anzeigetabellen, die nicht mit dem Befehl `SECURE_TABLE` vor Änderungen geschützt wurden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
TABLE_TITLE 'table name'
display color
background color
'display text' 'Action Text'
(Row, Column of Slot) or (BOX Row1, Column1 to Row2, Column2 of Slots)
:
Repeat the above four lines if you need to define other title-slots
:
END
```

Ein Beispiel für den Befehl:

```
TABLE_TITLE 'table1'
  blue
  yellow
  'LINE' 'LINE'
  1 1

  green
  white
  '@s2' '@s1'
  BOX  1 2  2 3
END
```

Hinweis

Im obigen Beispiel gehören die Zahlen am linken Rand NICHT zur Befehlssyntax. Diese Zahlen dienen nur zur leichteren Identifizierung der einzelnen Zeilen. Die Leerzeilen sind ebenfalls nur Lese- und Verständnishilfen.

Zeile 1 enthält den `TABLE_TITLE`-Befehl und den Namen der Anzeigetabelle, deren Titel definiert werden muss.

Zeile 2 gibt die Anzeigefarbe für den Feldtext an. In diesem Beispiel wird `blue` angegeben. Wenn Sie eine andere Farbe benötigen, können Sie nach der Beschreibung für Zeile 3 von "TABLE_LAYOUT" vorgehen.

Zeile 3 gibt die Hintergrundfarbe für das Feld an. In diesem Beispiel wird `yellow` angegeben. Wenn Sie eine andere Farbe benötigen, können Sie nach der Beschreibung für Zeile 3 von "TABLE_LAYOUT" vorgehen.

Zeile 4 gibt den Anzeigetext `LINE` im Feld und den zugeordneten Aktionstext `LINE` an.

Der Anzeigetext ist eine Textzeichenfolge in der Anzeigetabelle. Der dazugehörige Befehlswort ist ebenfalls eine Textzeichenfolge, die mit einem gültigen Systemkommando identisch ist.

Durch Auswählen am Bildschirm kann dieser Befehl aus der Anzeigetabelle heraus aktiviert werden. Sie können auch das Schema @s# angeben, d.h. die Textzeichenfolge stammt von der Kopfzeichenfolge Nummer # der logischen Tabelle. Beispiel: Die Angabe `@s3` in einem dieser Parameter bedeutet, dass die zusätzliche Zeichenfolge 3 der verknüpften logischen Tabelle die tatsächliche angegebene Textzeichenfolge ist.

Sie können sogar eine Textzeichenfolge im Format `UNITS=@s2` angeben. Wenn die zusätzliche Zeichenfolge 2 der verknüpften logischen Tabelle `Meters` ist, ist die tatsächliche Textzeichenfolge `UNITS=Meters`.

Wenn Sie eine Textzeichenfolge im Format `UNITS=@t2` angeben und die zusätzliche Zeichenfolge 2 der verknüpften logischen Tabelle `Meters` ist, ist die tatsächliche Textzeichenfolge `UNITS='Meters'`.

Zeile 5 gibt `1 1` an, also Zeile 1 und Spalte 1 als Position des zu definierenden Feldes. Mit dem Parameter BOX kann auch ein Kasten mit mehreren Feldern angegeben werden. Beispielsweise bedeutet `BOX 1 2 3 3` wie in Zeile 10, dass die Felder von Zeile 1 und Spalte 2 bis Zeile 3 und Spalte 3 das erforderliche Feld bilden.

Die Zeilen 7 bis 10 bilden einen weiteren Parameterblock, der ähnlich aufgebaut ist wie die Zeilen 2 bis 5. Damit werden weitere Felder für den Titel der Anzeigetabelle definiert.

Zeile 11 gibt `END` an, um das Ende des `TABLE_TITLE`-Befehls zu definieren.

Hinweis

Sie können beliebig viele Parameterblöcke ähnlich dem aus Zeile 2 bis 5 einfügen, sofern diese Blöcke zwischen den Befehlen `TABLE_TITLE` und `END` stehen.

Das obige Beispiel kann auch folgendermaßen umgestellt werden:

```
TABLE_TITLE 'table1'
  blue yellow 'LINE'  'LINE' 1 1
  green white '@s2' '@s1' BOX  1 2  2 3
END
```

Wenn ein Parameterblock in eine Zeile passt, ist dieses Format besser lesbar und leichter vergleichbar. Wichtig ist hierbei, dass die Parameter mindestens durch einen Leer- oder Tabulatorschritt getrennt werden.

Nachfolgend sind die Befehle für das Arbeiten mit vorhandenen Anzeigetabellen aufgeführt:

- `CHANGE_TABLE_SIZE`
- `CONNECT_TABLE`
- `DELETE_TABLE`

- MOVE_TABLE
- PRINT_TABLE
- SAVE_TABLE
- SECURE_TABLE
- SHOW_TABLE
- TABLE_SCROLL_STEP

In den folgenden Abschnitten werden diese Befehle ausführlich beschrieben.

CHANGE_TABLE_SIZE

Dieser Befehl ändert die Größe von vorhandenen Anzeigetabellen, die programmintern nicht vor Größenänderungen geschützt wurden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
CHANGE_TABLE_SIZE 'table name' Point 1 Point 2
```

Ein Beispiel für den Befehl:

```
CHANGE_TABLE_SIZE  'disptable1'  100,100  1000,800
```

CHANGE_TABLE_SIZE erfordert drei Parameter (in diesem Beispiel `disptable1`, `100,100` und `1000,800`).

`disptable1` ist der Name der Anzeigetabelle, deren Größe geändert werden soll. `100,100` und `1000,800` sind die X- und Y-Koordinaten der unteren linken und der oberen rechten Ecke der Anzeigetabelle.

Hinweis

Breite und Höhe der Anzeigetabelle müssen bestimmten Mindestabmessungen entsprechen.

CONNECT_TABLE

Dieser Befehl verbindet eine Anzeigetabelle mit einer logischen Tabelle.

Die Syntax des Befehls sieht folgendermaßen aus:

```
CONNECT_TABLE 'table name' 'Logical table name'
```

Ein Beispiel für den Befehl:

```
CONNECT_TABLE  'disptable1'  'logtable1'
```

CONNECT_TABLE erfordert zwei Parameter (in diesem Beispiel `disptable1` und `logtable1`).

`disptable1` ist der Name der Anzeigetabelle, die mit der logischen Tabelle `logtable1` verknüpft werden soll.

Wenn versucht wird, eine Anzeigetabelle mit einer zweiten logischen Tabelle zu verbinden, wird die Verbindung mit der ersten logischen Tabelle automatisch unterbrochen. Es kann immer nur eine logische Tabelle mit einer Anzeigetabelle verbunden sein.

DELETE_TABLE

Mit diesem Befehl kann eine vorhandene, nicht geschützte Anzeigetabelle gelscht werden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
DELETE_TABLE
'table name' or (ALL CONFIRM)
```

Ein Beispiel für den Befehl:

```
DELETE_TABLE
ALL  CONFIRM
```

oder einfach

```
DELETE_TABLE  ALL  CONFIRM
```

`DELETE_TABLE` erfordert einen Parameter, der in diesem Beispiel `ALL CONFIRM` ist. `ALL` bedeutet, dass alle im System definierten Anzeigetabellen gelöscht werden. Mit `CONFIRM` können Sie bestätigen, dass der Befehl korrekt ist.

Es besteht auch die Möglichkeit, den Namen einer Anzeigetabelle wie folgt anzugeben:

```
DELETE_TABLE  'disptable1'
```

um anzugeben, dass die Anzeigetabelle `disptable1` gelöscht werden soll.

MOVE_TABLE

Mit diesem Befehl kann eine vorhandene Anzeigetabelle in eine bestimmte Position auf dem Bildschirm geschoben werden. Die Anzeigetabelle darf nicht programmintern vor dem Verschieben geschützt worden sein.

Die Syntax des Befehls sieht folgendermaßen aus:

```
MOVE_TABLE 'table name1'
(Point 1 Point 2) or
((UPPER or LOWER or RIGHT or LEFT) and/or (OF 'table name2'))
:
END
```

Ein Beispiel für den Befehl:

```
MOVE_TABLE  'disptable1'
UPPER  OF  'disptable2'
100,100  150,400
END
```

Hinweis

Im obigen Beispiel gehören die Zahlen am linken Rand NICHT zur Befehlssyntax. Diese Zahlen dienen nur zur leichteren Identifizierung der einzelnen Zeilen.

Zeile 1 enthält den `MOVE_TABLE`-Befehl und den Namen `disptable1` der Anzeigetabelle, die verschoben werden muss.

Zeile 2 gibt UPPER an, um eine Verschiebung nach oben im Bildschirm anzugeben. Sie können auch die Verschiebungsrichtungen LOWER, RIGHT oder LEFT angeben.

Optional können Sie OF und den Namen einer Anzeigetabelle 'disptable2' nach der Richtung angeben, um darauf hinzuweisen, dass die Verschiebung relativ zu einer anderen Anzeigetabelle ist.

Zeile 3 gibt 100,100 und 150,400 an. Hierbei handelt es sich um die X- und Y-Koordinaten der Referenz- und Zielpunkte einer anderen Verschiebung.

Zeile 4 gibt END an, um das Ende des MOVE_TABLE-Befehls zu definieren.

Hinweis

Zeile 2 und Zeile 3 enthalten jeweils gültige Parameter für eine Verschiebung, die Angabe erfolgt jedoch in unterschiedlicher Form. In diesem Beispiel werden also zwei Verschiebungen ausgeführt.

> Es ist möglich, mehrere Verschiebungen im MOVE_TABLE-Befehl festzulegen, sofern die Parameter in MOVE_TABLE und END enthalten sind. Bei der Ausführung werden die einzelnen Verschiebungen in einer Gesamtverschiebung kombiniert.

PRINT_TABLE

Mit diesem Befehl wird der sichtbare Inhalt einer Anzeigetabelle über den Bildschirm ausgegeben oder in einer Datei gespeichert.

Die Syntax des Befehls sieht folgendermaßen aus:

```
PRINT_TABLE 'table name' or ALL
SCREEN or (( or DEL_OLD or APPEND) 'filename')
```

Ein Beispiel für den Befehl:

```
PRINT_TABLE  'disptable1'
DEL_OLD  'file1.tbl'
```

oder einfach

```
PRINT_TABLE  'disptable1'  DEL_OLD 'file1.tbl'
```

PRINT_TABLE erfordert zwei Parameter. Der erste Parameter ist der Name der auszugebenden Anzeigetabelle. In diesem Beispiel ist die Anzeigetabelle disptable1. Der zweite Parameter gibt das gewünschte Ausgabeziel für die angegebene Anzeigetabelle an. In diesem Beispiel ist das Ausgabenziel die Datei file1.tbl mit der Option DEL_OLD. Das bedeutet, wenn file1.tbl bereits vorhanden ist, wird sie überschrieben.

Wenn Sie eine Datei als Ausgabeziel angeben, können Sie alternativ die Option APPEND oder keine Option verwenden. APPEND bedeutet, dass die angegebene Anzeigetabelle an das Ende der angegebenen Datei angefügt wird. Falls noch keine Datei vorhanden ist, wird eine Datei erstellt. Wenn Sie gar keinen Parameter angeben, wird eine Datei mit dem angegebenen Namen erstellt, sofern keine

andere Datei mit gleichem Namen vorhanden ist. Wenn die angegebene Datei bereits vorhanden ist, wird eine Fehlermeldung ausgegeben und der Befehl abgebrochen.

Alternativ können Sie den Bildschirm als Ausgabeziel angeben, indem Sie SCREEN statt DEL_OLD und file1.tbl in obigem Beispiel angeben.

SAVE_TABLE

Mit diesem Befehl wird der Aufbau einer Anzeigetabelle in einer Datei gespeichert. Anschließend kann die Anzeigetabelle mit einem INPUT-Befehl neu definiert werden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
SAVE_TABLE 'table name' or ALL
SCREEN or (( or DEL_OLD or APPEND) 'filename')
```

Ein Beispiel für den Befehl:

```
SAVE_TABLE  'disptable1'
DEL_OLD  'file1.tbl'
```

oder einfach

```
SAVE_TABLE  'disptable1'  DEL_OLD  'file1.tbl'
```

SAVE_TABLE erfordert zwei Parameter. Der erste Parameter ist der Name der zu sichernden Anzeigetabelle. In diesem Beispiel ist die Anzeigetabelle disptable1. Der zweite Parameter gibt das gewünschte Ausgabeziel für die zu sichernde Anzeigetabelle an. In diesem Beispiel ist das Ausgabenziel die Datei file1.tbl mit der Option DEL_OLD. Das bedeutet, wenn file1.tbl bereits vorhanden ist, wird sie überschrieben.

Wenn Sie eine Datei als Ausgabeziel angeben, können Sie alternativ die Option APPEND oder keine Option verwenden. APPEND bedeutet, dass die angegebene Anzeigetabelle an das Ende der angegebenen Datei angefügt wird. Falls noch keine Datei vorhanden ist, wird eine Datei erstellt. Wenn Sie gar keinen Parameter angeben, wird eine Datei mit dem angegebenen Namen erstellt, sofern keine andere Datei mit gleichem Namen vorhanden ist. Wenn die angegebene Datei bereits vorhanden ist, wird eine Fehlermeldung ausgegeben und der Befehl abgebrochen.

Alternativ können Sie den Bildschirm als Ausgabeziel angeben, indem Sie SCREEN statt DEL_OLD und file1.tbl in obigem Beispiel angeben.

SECURE_TABLE

Mit diesem Befehl wird eine Anzeigetabelle geschützt, damit sie nicht unbeabsichtigt gelöscht oder geändert werden kann.

Die Syntax des Befehls sieht folgendermaßen aus:

```
SECURE_TABLE
'table name' or (ALL CONFIRM)
```

Ein Beispiel für den Befehl:

```
SECURE_TABLE
ALL  CONFIRM
```

oder einfach

```
SECURE_TABLE ALL CONFIRM
```

SECURE_TABLE erfordert einen oder zwei Parameter (in diesem Beispiel ALL CONFIRM). ALL bedeutet, dass alle im System definierten Anzeigetabellen geschützt werden sollen. Mit CONFIRM können Sie bestätigen, dass der Befehl korrekt ist.

Es besteht auch die Möglichkeit, den Namen einer Anzeigetabelle wie folgt anzugeben:

```
SECURE_TABLE  'disptable1'
```

um anzugeben, dass die Anzeigetabelle disptable1 geschützt werden soll.

SHOW_TABLE

Mit diesem Befehl können Sie eine einzelne Anzeigetabelle oder alle Anzeigetabellen ein- und ausblenden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
SHOW_TABLE
ON or OFF
'table name' or ALL
```

Ein Beispiel für den Befehl:

```
SHOW_TABLE
ON
ALL
```

oder einfach

```
SHOW_TABLE  ON  ALL
```

SHOW_TABLE erfordert zwei Parameter (in diesem Beispiel ON und ALL).

ON bedeutet, dass die Anzeigetabellen angezeigt werden. Sie können alternativ OFF angeben, um sie nicht anzuzeigen.

ALL bedeutet, dass sich der Befehl auf alle im System definierten Anzeigetabellen auswirken soll. Alternativ kann auch der Name einer Anzeigetabelle angegeben werden.

Hinweis

Sie können auch die Befehle POP_UP_LTAB und POP_DOWN_LTAB verwenden, um eine Anzeigetabelle anzuzeigen und auszublenden. Ausführliche Informationen dazu finden Sie in den Abschnitten POP_UP_LTAB und POP_DOWN_LTAB.

Die Unterschiede zwischen SHOW_TABLE ON (oder OFF) und POP_UP_LTAB (oder POP_DOWN_LTAB) sind wie folgt:

- Der Befehl SHOW_TABLE zeigt eine angegebene Anzeigetabelle oder alle Anzeigetabellen, während der Befehl POP_UP_LTAB alle mit einer logischen Tabelle verbundenen Anzeigetabellen anzeigt.
- Der Befehl POP_UP_LTAB speichert die Grafikinformationen unter der angezeigten Anzeigetabelle. Beim Befehl SHOW_TABLE ist dies nicht der Fall.

TABLE_SCROLL_STEP

Mit diesem Befehl wird eingestellt, um wieviele Zeilen eine Anzeigetabelle bei einmaligem Umblättern mit den Kästen am oberen und unteren Ende des Auswahlbalkens weiterlaufen soll.

Die Syntax des Befehls sieht folgendermaßen aus:

```
TABLE_SCROLL_STEP
'table name' or ALL
DEFAULT or number
```

Ein Beispiel für den Befehl:

```
TABLE_SCROLL_STEP
ALL
DEFAULT
```

oder einfach

```
TABLE_SCROLL_STEP  ALL  DEFAULT
```

TABLE_SCROLL_STEP erfordert zwei Parameter (in diesem Beispiel ALL und DEFAULT).

ALL bedeutet, dass sich der Befehl auf alle im System definierten Anzeigetabellen auswirkt. Sie können auch den Namen einer Anzeigetabelle angeben, z.B. disptable1. Dann gilt der Befehl nur für diese Tabelle.

DEFAULT bedeutet, dass um die Anzahl der Datenzeilen in der Anzeigetabelle weitergeblättert wird. Sie können auch eine bestimmte Anzahl von Datenzeilen angeben, z.B. 5. Daraufhin wird jeweils um 5 Zeilen weitergeblättert.

Funktionen für benutzerspezifische Tabellen

Eine Benutzertabelle ist eine logische Tabelle, die im Gegensatz zu einer systemdefinierten Tabelle vom Benutzer definiert wird. Sie hat die gleiche Datenstruktur wie jede logische Tabelle. Außerdem können Sie eine Anzeigetabelle mit einer Benutzertabelle wie mit einer logischen Tabelle verbinden.

Der Befehl zum Definieren einer benutzerspezifischen Tabelle lautet:

- `CREATE_LTAB`

Nachfolgend sind die Befehle für die Handhabung von benutzerspezifischen Tabellen aufgeführt:

- `COLOR_LTAB`
- `DELETE_LTAB`
- `DELETE_LTAB_ROW`
- `HIGHLIGHT_LTAB`
- `SECURE_LTAB`
- `SORT_LTAB`
- `WRITE_LTAB`

Die oben aufgeführten Befehle werden in den folgenden Abschnitten ausführlich beschrieben.

Hinweis

Für die Befehlsbeispiele gilt Folgendes: In normaler Schrift gesetzter Text muss buchstabengetreu eingegeben werden, damit der jeweilige Befehl funktioniert. Kursiv gesetzter Text ist durch geeignete Zeichenfolgen oder Werte zu ersetzen. Das Wort "oder" bedeutet, dass zwei oder mehr Parameter vorhanden sind, die Sie als Parameter wählen können.

COLOR_LTAB

Mit diesem Befehl werden die Vordergrund- (Anzeige-) und Hintergrundfarben der Tabellenfelder angegeben bzw. geändert.

Hinweis

Der `COLOR_LTAB`-Befehl kann für Benutzertabellen verwendet werden, die mit dem `SECURE_LTAB`-Befehl und der Option `READ_ONLY` geschützt wurden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
COLOR_LTAB 'User table name'
(Row-no Column-no) or (ROW Row-no) or (COLUMN Column-no) or ALL
or (TITLE title string-no or ALL)
(Foreground Color or DEFAULT)
(Background Color or DEFAULT)
```

Ein Beispiel für den Befehl:

```
COLOR_LTAB  'usertable1'
3  5
white
blue
```

`COLOR_LTAB` erfordert vier Informationen. Die erste Angabe ist der Name der Benutzertabelle, auf die sich der Befehl auswirken soll. Die zweite Angabe ist der Bereich in der angegebenen benutzerspezifische Tabelle, dessen Farben geändert werden sollen.Die dritte und vierte Angabe sind die Vordergrund- bzw. Hintergrundfarben für das Feld.

In obigem Beispiel müssen der Zeile `3` und der Spalte `5` in der Benutzertabelle `usertable1` die Vordergrundfarbe `white` und die Hintergrundfarbe `blue` zugewiesen werden.

Für die Positionsangabe bestehen noch fünf andere Möglichkeiten, z.B.:

- `ROW 5` (gesamte Zeile 5)
- `COLUMN 3` (gesamte Zeile 3)
- `ALL` (alle Zeilen und Spalten)
- `TITLE 2` (zweite Titelzeichenfolge)
- `TITLE ALL` (alle Titelzeichenfolgen)

Darüber hinaus können alle gültigen Anwendungsfarben als Vorder- und Hintergrundfarben angegeben werden.

Ein weiteres Beispiel für den Befehl:

```
HIGHLIGHT_LTAB 'usertable1'
TITLE 2
black
yellow
```

Dabei wird `black` als Vordergrundfarbe für die Titelzeichenfolge 2 und `yellow` als Hintergrundfarbe zugewiesen.

CREATE_LTAB

Mit diesem Befehl wird eine benutzerspezifische Tabelle neu erstellt.Parameter dienen dabei zum Abschätzen der benötigten Zeilen- und Spaltenzahl.

Näheres dazu kann unter "Eine benutzerspezifische Tabelle definieren" im Abschnitt "Mit logischen Tabellen und Anzeigetabellen arbeiten - Beispiel 1" nachgelesen werden.

Die Parameter für den Umfang der neuen benutzerspezifschen Tabelle gewährleisten den effizienten Einsatz der Tabelle.

Wenn für die zu erstellende benutzerspezifische Tabelle der Name einer bereits vorhandenen, durch READ_ONLY geschützten systemdefinierten Tabelle angegeben wird, erscheint eine Fehlermeldung ausgegeben.

Wenn für die zu erstellende benutzerspezifische Tabelle der Name einer bereits vorhandenen, ungeschützten benutzerspezifischen Tabelle angegeben wird, werden alle Daten in der vorhandenen Tabelle gelöscht und ihre Zeilen- und Spaltenzahl an die im Befehl angegebenen Werte angeglichen.

Wenn jedoch für die zu erstellende Benutzertabelle der Name einer vorhandenen, durch den Befehl `SECURE_LTAB` und die Option `READ_ONLY` geschützten Benutzertabelle angegeben wird, wird eine Fehlermeldung angezeigt.

Die Syntax des Befehls sieht folgendermaßen aus:

```
CREATE_LTAB (or Rows) (or Columns)
'User table name'
```

Ein Beispiel für den Befehl:

```
CREATE_LTAB
20 6
'usertable1'
```

`CREATE_LTAB` erfordert zwei Informationen. Die erste Angabe ist ein Schätzwert für die Zeilen- und Spaltenzahl der neuen benutzerspezifischen Tabelle. In obigem Beispiel wird die geschätzte Größe mit 20 Zeilen und 6 Spalten angegeben. Diese Angabe ist nur ein Schätzwert und kein Grenzwert. Es können also auch mehr Daten in die Tabelle geschrieben werden als in 20 Zeilen und 6 Spalten passen. Die zweite Angabe ist der Name der neuen Benutzertabelle, in obigem Beispiel `usertable1`.

DELETE_LTAB

Mit diesem Befehl wird eine benutzerspezifische Tabelle gelöscht.

Wenn die angegebene Benutzertabelle durch den Befehl `SECURE_LTAB` geschützt wurde oder blockiert ist, weil gerade auf sie zugegriffen wird, wird eine Fehlermeldung angezeigt.

Die Syntax des Befehls sieht folgendermaßen aus:

```
DELETE_LTAB ALL or 'User table name'
```

Ein Beispiel für den Befehl:

```
DELETE_LTAB  ALL
```

`DELETE_LTAB` erfordert einen Parameter, der in diesem Beispiel `ALL` ist. `ALL` bedeutet, dass der Befehl alle Benutzertabellen löscht.

Es besteht auch die Möglichkeit, den Namen der zu löschenden benutzerspezifischen Tabelle folgendermaßen einzugeben:

```
DELETE_LTAB  usertable1
```

um anzugeben, dass die Benutzertabelle `usertable1` gelöscht werden soll.

DELETE_LTAB_ROW

Mit diesem Befehl wird eine Zeile aus der benutzerspezifische Tabelle gelöscht.

Wenn die angegebene Benutzertabelle durch den Befehl SECURE_LTAB und die Option READ_ONLY geschützt wurde, wird eine Fehlermeldung angezeigt.

Die Syntax des Befehls sieht folgendermaßen aus:

```
DELETE_LTAB_ROW 'User table name'
ALL or Row-number
```

Ein Beispiel für den Befehl:

```
DELETE_LTAB_ROW  'usertable1'
ALL
```

DELETE_LTAB_ROW erfordert zwei Informationen. Die erste Angabe ist der Name der Benutzertabelle, auf die sich der Befehl auswirken soll. Die zweite Angabe ist die Nummer der zu löschenden Zeile in der angegebenen benutzerspezifischen Tabelle.

Im obigen Beispiel sollen ALL Zeilen in der Benutzertabelle usertable1 gelöscht werden.

Sie können auch eine Zeilennummer angeben.Dann sieht der Befehl folgendermaßen aus:

```
DELETE_LTAB_ROW  'usertable1'
5
```

um anzugeben, dass die Zeile 5 in der Benutzertabelle usertable1 gelöscht werden soll.

HIGHLIGHT_LTAB

Mit diesem Befehl wird die Intensivanzeige eines Datenbereichs in der benutzerspezifischen Tabelle ein- oder ausgeschaltet.

Hinweis

Der HIGHLIGHT_LTAB-Befehl kann für Benutzertabellen verwendet werden, die mit dem SECURE_LTAB-Befehl und der Option READ_ONLY geschützt wurden.

Die Syntax des Befehls sieht folgendermaßen aus:

```
HIGHLIGHT_LTAB 'User table name'
(Row-no Column-no) or (ROW Row-no) or (COLUMN Column-no) or ALL
or (TITLE title string-no or ALL)
ON or OFF or MARK
```

Ein Beispiel für den Befehl:

```
HIGHLIGHT_LTAB  'usertable1'
3  5
ON
```

HIGHLIGHT_LTAB erfordert drei Informationen. Die erste Angabe ist der Name der Benutzertabelle, auf die sich der Befehl auswirken soll. Die zweite Angabe ist der Datenbereich in der angegebenen benutzerspezifischen Tabelle, in dem der Schaltzustand der Intensivanzeige geändert werden soll.Die dritte Angabe ist der gewünschte Schaltzustand für die Intensivanzeige (ON oder OFF).

Im Beispiel oben sollen Zeile 3 und Spalte 5 in der Benutzertabelle usertable1 hervorgehoben werden, das heißt ON.

Für die Positionsangabe bestehen noch fünf andere Möglichkeiten, z.B.:

- ROW 5 (gesamte Zeile 5)
- COLUMN 3 (gesamte Zeile 3)
- ALL (alle Zeilen und Spalten)
- TITLE 2 (zweite Titelzeichenfolge)
- TITLE ALL (alle Titelzeichenfolgen)

Außerdem können Sie OFF angeben, um die Hervorhebung zu deaktivieren, oder MARK , um das Feld mit einem inneren Feld zu markieren.

Ein weiteres Beispiel für den Befehl:

```
HIGHLIGHT_LTAB 'usertable1'
TITLE 2
MARK
```

Dabei wird die Titelzeichenfolge 2 durch MARK (inneres Feld) hervorgehoben.

SECURE_LTAB

Mit diesem Befehl wird eine angegebene benutzerspezifische Tabelle geschützt, damit sie nicht gelöscht werden kann.Es ist aber nach wie vor möglich, den Tabelleninhalt zu ändern. Wenn eine Benutzertabelle sowohl vor dem Löschen als auch vor Änderungen geschützt werden soll, muss die Option READ_ONLY angegeben werden.

```
The format of the command is as follows:
SECURE_LTAB (or READ_ONLY) 'User table name'
```

Ein Beispiel für den Befehl:

```
SECURE_LTAB  'usertable1'
```

SECURE_LTAB erfordert einen Parameter, um die zu schützende Benutzertabelle anzugeben. In diesem Beispiel ist die Benutzertabelle usertable1.

Die Option READ_ONLY kann folgendermaßen angegeben werden:

```
SECURE_LTAB  READ_ONLY  'usertable1'
```

damit der Inhalt von usertable1 nicht geändert werden kann.

SORT_LTAB

Mit dem Befehl SORT_LTAB werden die Daten in einer benutzerspezifischen Tabelle anhand der angegebenen Spalten sortiert. Mit REVERSE_SORT wird die Sortierrichtung für die nächste angegebene Spalte umgekehrt.

Die Syntax des Befehls sieht folgendermaßen aus:

```
SORT_LTAB 'User table name'
Column-no or REVERSE_SORT Column-no
:
(Repeat the last line if you need to specify other columns to sort)
:
CONFIRM
```

Ein Beispiel für den Befehl:

```
SORT_LTAB 'usertable1'
REVERSE_SORT 2
CONFIRM
```

SORT_LTAB erfordert drei Informationen. Die erste Angabe ist der Name der benutzerspezifischen Tabelle, auf die sich die Funktion auswirken soll. Die zweite Angabe ist die Sortierreihenfolge (vorwärts oder rückwärts). Die Standardvorgabe für den Sortiervorgang ist vorwärts. Die dritte Angabe ist die Nummer der Spalte, deren Daten sortiert werden sollen.

Die Prioritäten beim Sortiervorgang innerhalb einer Spalte sind folgendermaßen festgelegt:

1. NULL-Werte
2. Zahlen
3. Zeichenketten

In obigem Beispiel sollen die Daten in Spalte 2 der Benutzertabelle usertable1 in umgekehrter Reihenfolge sortiert werden. Mit CONFIRM wird die Funktion SORT_LTAB beendet.

Es können auch mehrere Spalten zum Sortieren angegeben werden.In diesem Fall wird die erste angegebene Spalte vorrangig sortiert. Beispiel:

```
SORT_LTAB  'usertable1'
REVERSE_SORT 3
1
REVERSE_SORT 2
CONFIRM
```

Das Sortieren von Spalte 3 in umgekehrter Reihenfolge hat Priorität vor Spalte 1 und 2.

Wenn die Tabelle schreibgeschützt ist, wird eine entsprechende Fehlermeldung ausgegeben.

WRITE_LTAB

Mit diesem Befehl werden neue Werte in die angegebene benutzerspezifische Tabelle geschrieben.

Die Syntax des Befehls sieht folgendermaßen aus:

```
WRITE_LTAB 'User table name'
(Row-number Column-number) or (TITLE Title-string-number)
Value
```

Ein Beispiel für den Befehl:

```
WRITE_LTAB  'usertable1'
3  5
26.1
```

WRITE_LTAB erfordert drei Informationen.

Die erste Angabe ist der Name der Benutzertabelle, auf die sich der Befehl auswirken soll. Die zweite Angabe ist die Position des Datenfelds, in das der Wert geschrieben werden soll.Die dritte Angabe ist der Wert selbst.

Im Beispiel oben soll der Wert 26.1 in die Zeile 3 und die Spalte 5 in der Benutzertabelle usertable1 geschrieben werden.

Es sind auch andere Positionsangaben möglich. Beispiel: `TITLE 2`, d.h. Titelzeichenfolge 2.

Sie können eine "Textzeichenfolge" oder eine Zahl als gültigen Wert angeben, beispielsweise `'WHEEL'` oder `26.1`. Ferner können Sie einen Variablennamen oder einen anderen Befehl als Wert eingeben, beispielsweise `RADIUS` (Wert der Variable `RADIUS` ist der angegebene Wert) oder `(READ_LTAB 'logtable1' 2 3)` (Ausgabe dieses Befehls ist der angegebene Wert).

Logische Tabellen und Anzeigetabellen – Beispiel 1

Beim Arbeiten mit logischen Tabellen und Anzeigetabellen müssen einige grundsätzliche Regeln beachtet werden.

Benutzerdefinierte logische Tabellen müssen im Gegensatz zu systemdefinierten logischen Tabelle zunächst definiert werden. Anschließend muss die benötigte Anzeigetabelle definiert werden.Die Daten aus der logischen Tabelle können dann in die Anzeigetabelle übertragen und auf diese Weise sichtbar gemacht werden.

Für definierte logische Tabellen und Anzeigetabellen sind alle verfügbaren Befehle und Funktionen zulässig.

In den folgenden Abschnitten werden eine benutzerspezifische Tabelle und eine Anzeigetabelle definiert.

Hinweis

Näheres über die in den folgenden Abschnitten verwendeten Befehle und Funktionen ist in den Abschnitten "Zugriffsfunktionen für logische Tabellen", "Funktionen für Anzeigetabellen" und "Funktionen für benutzerspezifische Tabellen" weiter oben in diesem Kapitel nachzulesen.

Eine benutzerspezifische Tabelle definieren

Das Definieren einer benutzerspezifischen Tabelle vollzieht sich in zwei Hauptschritten:

- Die benutzerspezifische Tabelle mit einem angegebenen Namen erstellen und die Anzahl der Zeilen und Spalten abschätzen.
- Die benutzerspezifische Tabelle mit den benötigten Daten füllen.

Unten finden Sie eine Beispiel-Liste zum Definieren einer benutzerspezifischen Tabelle:

```
{
 This is the Macro 'UTABLE1' to define a user table called 'Mach_Op'.
 It then fills the user table with data in the title string 1 and
 from rows 1 to 5 in column 1.
}
DEFINE UTABLE1
    CREATE_LTAB 5 2 'Mach_Op'
    WRITE_LTAB 'Mach_Op' 1 1 'Type of Machining Operation'
    WRITE_LTAB 'Mach_Op' 2 1 'Machine Tool Parameters'
```

```
    WRITE_LTAB 'Mach_Op' 3 1 'Cutting Tool Parameters'
    WRITE_LTAB 'Mach_Op' 4 1 'Workpart Characteristics'
    WRITE_LTAB 'Mach_Op' 5 1 'Other Operating Parameters'
    WRITE_LTAB 'Mach_Op' TITLE 1 'Characteristics of a Machining Operation'
END_DEFINE
```

Hinweis

Der Text innerhalb der geschweiften Klammern ({ }) in obiger und folgender Liste ist nur Kommentartext und gehört nicht zum Makro.

Sie können die obige Liste mit einem Texteditor erstellen und in der Datei `utable1.mac` speichern.

Eine Anzeigetabelle definieren

Das Definieren einer Anzeigetabelle vollzieht sich in drei Schritten:

- Das Layout der Anzeigetabelle mit Kopfbereich und Datenbereich erstellen.
- Die Parameter für den Kopfbereich angeben.
- Die Parameter für den Datenbereich angeben.

Unten finden Sie eine Beispiel-Liste zum Definieren einer Anzeigetabelle:

```
{
 This is the Macro 'DTABLE1' to define a display table called 'Mach_Op'
 which maps to the user table 'Mach_Op'. The user table and display table
 are called by the same name 'Mach_Op' in this example, but they can be
 different.
 It then specifies the layout of the title and data areas of the display
 table, and also specifies the data and their formats to put into the
 title and data areas.
}
DEFINE DTABLE1
  TABLE_LAYOUT  'Mach_Op'
    'Mach_Op'
    WHITE BLACK
    width 60.250000 rows 10.058824
    FRAME_WIDTH 2
    HORIZONTAL WHITE SOLID
    VERTICAL WHITE SOLID
    SCROLL_BAR WHITE BLUE 32
    TITLE_LAYOUT
    18 '                                                          '
    1  '                                                          '
    END
    COLUMN_LAYOUT
        '                                                         '
  END
  TABLE_TITLE 'Mach_Op'
    BLACK YELLOW '@s1' ''  1 1
    WHITE WHITE '' ''  2 1
  END
  TABLE_COLUMN 'Mach_Op'
    COLUMN 1 GREEN BLACK 1 FORMAT 10 LEFT '@v1'
  END
END_DEFINE
```

Die obige Liste kann mit einem Texteditor erstellt werden. Wenn Sie jedoch bereits über eine Anzeigetabelle mit ähnlichem Layout verfügen, ist es einfacher, die Liste der vorhandenen Anzeigetabelle mit dem Befehl SAVE_TABLE in einer Datei zu speichern. Anschließend können Sie diese Datei mit einem Texteditor ändern, um das gewünschte Layout zu erstellen.

Die fertige Liste wird in der Datei dtable1.mac gespeichert.

Mit einer Anzeigetabelle interagieren

Zunächst muss Creo Elements/Direct Drafting auf dem System laufen. Anschließend müssen die Definitionen für die Benutzer- und die Anzeigetabelle geladen werden. Geben Sie dazu die folgenden Befehle über die Tastatur ein, nachdem in Creo Elements/Direct Drafting die Eingabeaufforderung ENTER COMMAND angezeigt wurde:

```
INPUT 'utable1.mac' [Return]
UTABLE1 [Return]
INPUT 'dtable1.mac' [Return]
DTABLE1 [Return]
```

Mit den beiden ersten Befehlen wird die Definition der benutzerspezifischen Tabelle geladen.Mit den beiden zweiten Befehlen wird die Definition der Anzeigetabelle geladen.

Nun können Sie die Arbeit mit den Tabellen beginnen. Zunächst können Sie die Anzeigetabelle Mach_Op auf dem Bildschirm anzeigen:

```
SHOW_TABLE ON 'Mach_Op' [Return]
```

Anschließend verschieben Sie die Anzeigetabelle Mach_Op in die Mitte des Bildschirms:

```
MOVE_TABLE 'Mach_Op' LOWER LEFT 0,0 300,300 END [Return]
```

Hinweis

Es ist wichtig, die LOWER LEFT-Optionen im Befehl zu verwenden, um sicherzustellen, dass die Verschiebung in der unteren linken Ecke des Bildschirms beginnt. Andernfalls beginnt die Verschiebung an der aktuellen Position der Tabelle. Dies bedeutet, dass die Tabelle außerhalb des Bildschirms verschoben werden kann.

Anschließend drucken Sie die Anzeigetabelle Mach_Op wie sie auf dem Bildschirm in der Datei dtable1.prt angezeigt wird:

```
PRINT_TABLE 'Mach_Op' 'dtable1.prt' [Return]
```

Danach kann die Datei über einen Drucker ausgegeben werden.

Hinweis

Mit diesem Befehl erhalten Sie einen Ausdruck der Anzeigetabelle, so wie sie auf dem Bildschirm dargestellt ist.

Anschließend speichern Sie die Anzeigetabelle `Mach_Op` in einer Datei `dtable1.sav`:

```
SAVE_TABLE 'Mach_Op' 'dtable1.sav' [Return]
```

Hinweis

Mit diesem Befehl sichern Sie sich das Layout dieser Tabelle für eine weitere Anzeigetabelle. Dieses Layout kann mit einem Texteditor auch modifiziert werden.

Anschließend entfernen Sie die Anzeigetabelle `Mach_Op` aus dem Bildschirm:

```
SHOW_TABLE OFF 'Mach_Op' [Return]
```

Logische Tabellen und Anzeigetabellen – Beispiel 2

Dieser Abschnitt ist ähnlich aufgebaut wie der Abschnitt "Mit logischen Tabellen und Anzeigetabellen arbeiten - Beispiel 1".Das zweite Beispiel ist jedoch komplexer. Es werden zwei benutzerspezifische Tabellen und zwei Anzeigetabellen verwendet. Die ersten Benutzer- und Anzeigetabellen stammen aus dem Beispiel unter "Logische Tabellen und Anzeigetabellen – Beispiel 1". Die zweiten Benutzer- und Anzeigetabellen sind neu und werden in diesem Abschnitt definiert.

Die folgenden Abschnitte enthalten Beispiele für die Definition der erforderlichen Benutzer- und Anzeigetabellen und die Interaktion mit ihnen.

Hinweis

Näheres über die in den folgenden Abschnitten verwendeten Befehle und Funktionen ist in den Abschnitten "Zugriffsfunktionen für logische Tabellen", "Funktionen für Anzeigetabellen" und "Funktionen für benutzerspezifische Tabellen" weiter oben in diesem Kapitel nachzulesen.

Das erste Tabellenpaar definieren

Die ersten Benutzer- und Anzeigetabellen entsprechen denen unter "Logische Tabellen und Anzeigetabellen – Beispiel 1" mit einigen kleineren Änderungen, sodass Sie die Dateien `utable1.mac` und `dtable1.mac` in `utable21.mac` und `dtable21.mac` kopieren können. Anschließend können Sie das Layout der Tabellen mit Hilfe eines Texteditors anpassen.

Die folgende Liste erzeugt die erste Benutzertabelle in der Datei `utable21.mac`:

```
{
 This is the Macro 'UTABLE21' to define the first user table called
 'Mach_Op2'. It then fills the user table with data in the title
 string 1 and from rows 1 to 5 in column 1.
}
```

```
DEFINE UTABLE21
    CREATE_LTAB 5 2 'Mach_Op2'
    WRITE_LTAB 'Mach_Op2' 1 1 'Type of Machining Operation'
    WRITE_LTAB 'Mach_Op2' 2 1 'Machine Tool Parameters'
    WRITE_LTAB 'Mach_Op2' 3 1 'Cutting Tool Parameters'
    WRITE_LTAB 'Mach_Op2' 4 1 'Workpart Characteristics'
    WRITE_LTAB 'Mach_Op2' 5 1 'Other Operating Parameters'
    WRITE_LTAB 'Mach_Op2' TITLE 1 'Characteristics of a Machining Operation'
END_DEFINE
```

Hinweis

Der Text innerhalb der geschweiften Klammern ({ }) in obiger und folgender Liste ist nur Kommentartext und gehört nicht zum Makro.

Das Makro `UTABLE21` entspricht dem Makro `UTABLE1`, nur der Benutzertabellenname ist `Mach_Op2`.

Die folgende Liste erzeugt die erste Benutzertabelle in der Datei `dtable21.mac`:

```
{
 This is the Macro 'DTABLE21' to define the first display table called
 'Mach_Op2' which maps to the user table 'Mach_Op2'. The first user and
 display tables are called by the same name 'Mach_Op2' in this example,
 but they can be different.
 It then specifies the layout of the title and data areas of the display
 table, and also specifies the data and their formats to put into the
 title and data areas.
}
DEFINE DTABLE21
  TABLE_LAYOUT  'Mach_Op2'
    'Mach_Op2'
    WHITE BLACK
    width 60.250000 rows 10.058824
    FRAME_WIDTH 2
    HORIZONTAL WHITE SOLID
    VERTICAL WHITE SOLID
    SCROLL_BAR WHITE BLUE 32
    TITLE_LAYOUT
    18 '                                                          |     '
    1  '                                                                '
    END
    COLUMN_LAYOUT
       '                                                                 '
  END
  TABLE_TITLE 'Mach_Op2'
    BLACK YELLOW '@s1' ''  1 1
    BLACK YELLOW 'END' 'SHOW_TABLE OFF 'Mach_Op2' SHOW_TABLE OFF 'Op_Para' END'  1
2
    WHITE WHITE '' ''   2 1
  END
  TABLE_COLUMN 'Mach_Op2'
    COLUMN 1 GREEN BLACK 1 FORMAT 10 LEFT '@v1'
  END
END_DEFINE
```

Das Makro `DTABLE21` entspricht `DTABLE1`,

- nur diese Anzeigetabelle heißt `Mach_Op2` und ist einer Benutzertabelle namens `Mach_Op2` zugeordnet
- In der ersten Kopfzeile sind zwei Spalten vorhanden.

- In der zweiten Spalte dieser Titelzeile steht `END` und der dazugehörige Aktionstext lautet:
  ```
  SHOW_TABLE OFF 'Mach_Op2' SHOW_TABLE OFF 'Op_Para' END
  ```
  Dabei werden die Anzeigetabellen `Mach_Op2` und `Op_Para` aus dem Bildschirm entfernt und das aktuelle Makroprogramm abgebrochen.

Das zweite Tabellenpaar definieren

Das zweite Tabellenpaar muss neu definiert werden. Die folgende Liste erzeugt die zweite Anzeigetabelle:

```
{
 This is the Macro 'DTABLE2' to define the second display table called
 'Op_Para' which maps to an unknown user table ''. Mapping
 this display table to a user table is done later when the second user
 table is defined.
 It then specifies the layout of the title and data areas of the display
 table, and also specifies the data and their formats to put into the
 title and data areas.
}
DEFINE DTABLE2
  TABLE_LAYOUT  'Op_Para'
    ''
    WHITE BLACK
    width 60.250000 rows 10.058824
    FRAME_WIDTH 2
    HORIZONTAL WHITE SOLID
    VERTICAL WHITE SOLID
    SCROLL_BAR WHITE BLUE 32
    TITLE_LAYOUT
    18 '                                                  |         '
    1  '                                                            '
    END
    COLUMN_LAYOUT
       '                                                            '
  END
  TABLE_TITLE 'Op_Para'
    BLACK YELLOW '@s1' ''  1 1
    BLACK YELLOW 'OFF' 'SHOW_TABLE OFF' 'Op_Para'  1 2
    WHITE WHITE '' ''  2 1
  END
  TABLE_COLUMN 'Op_Para'
    COLUMN 1 GREEN BLACK 1 FORMAT 10 LEFT '@v1'
  END
END_DEFINE
```

Das Makro `DTABLE2` entspricht `DTABLE21` mit folgender Ausnahme:

- Diese Anzeigetabelle heißt `Op_Para` und wird erst dann mit Daten gefüllt, wenn die zweite Benutzertabelle definiert ist.
- In der zweiten Spalte der ersten Titelzeile steht `OFF` und der dazugehörige Aktionstext lautet:
  ```
  SHOW_TABLE OFF 'Op_Para'
  ```
  Dabei wird die Anzeigetabelle `Op_Para` aus dem Bildschirm entfernt.

Die fertige Liste wird in der Datei `dtable2.mac` gespeichert.

Die folgende Liste erzeugt die zweite benutzerspezifische Tabelle und enthält Makrobefehle für den Datenaustausch zwischen allen vier Tabellen:

```
{
 This is the macro 'UTABLE2' to define the second user table called 'Op_Para'
 and to provide the Macro commands to interact with all four tables.
```

```
  }
  DEFINE UTABLE2
  LOCAL OPARA
  MOVE_TABLE 'Mach_Op2' LOWER LEFT 0,0 300,300 END
  POP_UP_LTAB 'Mach_Op2'
  CREATE_LTAB  6 2 'Op_Para'
  CONNECT_TABLE 'Op_Para' 'Op_Para'
  LOOP
  READ STRING 'Select option from CHARACTERISTICS OF A MACHINING OPERATION table:'
OPARA
  LET OPARA (UPC OPARA)
  IF(OPARA="TYPE OF MACHINING OPERATION")
    WRITE_LTAB 'Op_Para' TITLE 1 'Type of Machining Operation:'
    WRITE_LTAB 'Op_Para' 1 1 'Turning'
    WRITE_LTAB 'Op_Para' 2 1 'Drilling'
    WRITE_LTAB 'Op_Para' 3 1 'Tapping'
    WRITE_LTAB 'Op_Para' 4 1 'Milling'
    WRITE_LTAB 'Op_Para' 5 1 'Boring'
    WRITE_LTAB 'Op_Para' 6 1 'Grinding
  ELSE_IF(OPARA="MACHINE TOOL PARAMETERS")
    WRITE_LTAB 'Op_Para' TITLE 1 'Machine Tool Parameters:'
    WRITE_LTAB 'Op_Para' 1 1 'Size and Rigidity'
    WRITE_LTAB 'Op_Para' 2 1 'Horsepower'
    WRITE_LTAB 'Op_Para' 3 1 'Spindle Speed and Feedrate Levels'
    WRITE_LTAB 'Op_Para' 4 1 'Conventional or NC'
    WRITE_LTAB 'Op_Para' 5 1 'Accuracy and Precision Capabilities'
    WRITE_LTAB 'Op_Para' 6 1 'Operating Time Data'
  ELSE_IF(OPARA="CUTTING TOOL PARAMETERS")
    WRITE_LTAB 'Op_Para' TITLE 1 'Cutting Tool Parameters:'
    WRITE_LTAB 'Op_Para' 1 1 'Material Type'
    WRITE_LTAB 'Op_Para' 2 1 'Material Composition'
    WRITE_LTAB 'Op_Para' 3 1 'Physical and Mechanical Properties'
    WRITE_LTAB 'Op_Para' 4 1 'Type'
    WRITE_LTAB 'Op_Para' 5 1 'Geometry'
    WRITE_LTAB 'Op_Para' 6 1 'Cost'
  ELSE_IF(OPARA="WORKPART CHARACTERISTICS")
    WRITE_LTAB 'Op_Para' TITLE 1 'Workpart Characteristics:'
    WRITE_LTAB 'Op_Para' 1 1 'Material Type'
    WRITE_LTAB 'Op_Para' 2 1 'Hardness of Material'
    WRITE_LTAB 'Op_Para' 3 1 'Geometric Size and Shape'
    WRITE_LTAB 'Op_Para' 4 1 'Tolerances'
    WRITE_LTAB 'Op_Para' 5 1 'Surface Finish'
    WRITE_LTAB 'Op_Para' 6 1 'Initial Surface Condition'
  ELSE_IF(OPARA="OTHER OPERATING PARAMETERS")
    WRITE_LTAB 'Op_Para' TITLE 1 'Other Operating Parameters:'
    WRITE_LTAB 'Op_Para' 1 1 'Depth of Cut'
    WRITE_LTAB 'Op_Para' 2 1 'Cutting Fluid'
    WRITE_LTAB 'Op_Para' 3 1 'Workpart Rigidity'
    WRITE_LTAB 'Op_Para' 4 1 'Fixtures and Jigs'
    DELETE_LTAB_ROW 'Op_Para' 6
    DELETE_LTAB_ROW 'Op_Para' 5
  ELSE
    BEEP
    DISPLAY "UNKNOWN OPTION"
  END_IF
  SHOW_TABLE ON 'Op_Para'
  END_LOOP
END_DEFINE
```

Die fertige Liste wird in der Datei `utable2.mac` gespeichert.

Obwohl die obige Liste hauptsächlich die zweite benutzerspezifische Tabelle definieren soll, enthält sie auch Makrobefehle für den Datenaustausch zwischen allen vier Tabellen.

Nachfolgend finden Sie Erläuterungen zu obigem Makro:

- `DEFINE UTABLE2` markiert den Anfang des Makros `UTABLE2`.

- `LOCAL OPARA` deklariert die lokale Variable `OPARA`.
- `MOVE_TABLE 'Mach_Op2' LOWER LEFT 0,0 300,300 END` verschiebt die Anzeigetabelle `Mach_Op2` an Position 300,300 auf dem Bildschirm.
- `POP_UP_LTAB 'Mach_Op2'` zeigt die Benutzertabelle `Mach_Op2` an.
- `CREATE_LTAB 6 2 'Op_Para'` definiert die Benutzertabelle `Op_Para` mit sechs Zeilen und zwei Spalten.
- `CONNECT_LTAB 'Op_Para' 'Op_Para'` verbindet die Anzeigetabelle `Op_Para` mit der Benutzertabelle `Op_Para`.
- `LOOP` kennzeichnet den Anfang der Schleife, deren Ende durch `END_LOOP` gekennzeichnet ist.
- Innerhalb dieser Schleife wird der Benutzer nach einem Parameter aus einer Tabelle gefragt:
  ```
  READ STRING 'Select option from CHARACTERISTICS OF A MACHINING
  OPERATION table:' OPARA
  ```
  Der Benutzer kann eine Option aus der angegebenen Tabelle auswählen. Der entsprechende Aktionstext wird als Eingabe für die Variable `OPARA` verwendet.
- `LET OPARA (UPC OPARA)` konvertiert den Eingabetext in Großbuchstaben.
- Zudem gibt es die Aussagen `IF`, `ELSE_IF` und `END_IF`, um den Inhalt der Variable `OPARA` zu prüfen, die einen von fünf möglichen Werten aufweisen kann. Je nach Wert werden die entsprechenden Daten in die Benutzertabelle `Op_Para` geschrieben.
- Beachten Sie die beiden `DELETE_LTAB_ROW`-Anweisungen. Sie müssen die Daten der vorherigen Auswahl in den Zeilen 5 und 6 der Benutzertabelle `Op_Para` löschen. Andernfalls werden diese Daten auch in der Anzeigetabelle `Op_Para` angezeigt.
- Wenn `OPARA` keinen der möglichen Werte enthält, wird die Meldung `UNKNOWN OPTION` angezeigt.
- END_IF kennzeichnet das Ende der IF-Anweisung.
- SHOW_TABLE ON 'Op_Para' zeigt die Anzeigetabelle Op_Para an.
- END_LOOP kennzeichnet das Ende der LOOP-Anweisung.
- END_DEFINE kennzeichnet das Ende der DEFINE-Anweisung.

Mit der benutzerspezifischen Tabelle und der Anzeigetabelle arbeiten

Zunächst muss Creo Elements/Direct Drafting auf dem System laufen. Anschließend müssen die Definitionen für die Benutzer- und die Anzeigetabelle geladen werden. Geben Sie dazu die folgenden Befehle über die Tastatur ein, nachdem in Creo Elements/Direct Drafting die Eingabeaufforderung `ENTER COMMAND` angezeigt wurde:

```
INPUT 'utable21.mac' [Return]
UTABLE21 [Return]
```

```
INPUT 'dtable21.mac' [Return]
DTABLE21 [Return]
INPUT 'dtable2.mac' [Return]
DTABLE2 [Return]
INPUT 'utable2.mac' [Return]
```

Mit den ersten vier Befehlen werden die Definitionen der ersten benutzerspezifischen Tabellen und Anzeigetabellen geladen.Mit den zweiten drei Befehlen werden die Definitionen des zweiten Tabellensatzes geladen.

Die benutzerspezifischen Tabellen und die Anzeigetabellen können Sie mit dem folgenden Befehl aufrufen:

```
UTABLE2 [Return]
```

Das Makroprogramm `UTABLE2` zeigt zunächst die Tabelle `Characteristics of a Machining Operation` und die Eingabeaufforderung `Select option from CHARACTERISTICS OF A MACHINING OPERATION table:` an. Sie können dann einen der fünf Parameter aus dieser Tabelle auswählen.In einer zweiten Tabelle können Sie unter mehreren Alternativen für den gewählten Parameter wählen.

Auf dem Bildschirm befinden sich jetzt zwei Anzeigetabellen.Das Makro kehrt an den Schleifenanfang zurück und fordert eine weitere Eingabe an. Sie können alle fünf Parameter ausprobieren.Dabei können Sie verfolgen, wie sich der Inhalt der zweiten Anzeigetabelle entsprechend ändert. Sie können auch eine leere Option verwenden. Dann wird die Meldung `UNKNOWN OPTION` angezeigt.

In der oberen rechten Ecke der zweiten Anzeigetabelle steht `OFF`. Wenn Sie dieses Feld auswählen, wird die zweite Anzeigetabelle ausgeblendet. Das Makro läuft jedoch weiter und fordert eine weitere Eingabe an. Wenn Sie jetzt wieder einen Parameter aus der ersten Anzeigetabelle auswählen, wird die zweite Anzeigetabelle mit dem neuen Inhalt automatisch wieder eingeblendet.

In der oberen rechten Ecke der ersten Anzeigetabelle steht `END`. Wenn Sie dieses Feld auswählen, werden beide Anzeigetabellen gelöscht und das Makro wird beendet.

Kommentare

Die gezeigten Beispiele sollen Ihnen einige Anregungen für den Einsatz von logischen Tabellen und Anzeigetabellen vermitteln. Sie können diese Vorschläge weiterführen und eigene Tabellensysteme entwickeln.

Index

A

B

C

D

E

F

G

H

I

K

L

M

O

P

R

S

T

U

V

W

X

Y

Z