AutoLISP

zu AutoCAD Realease 14

Stand 12/97

AutoLISP

AutoLISP, eine Implementierung der Programmiersprache LISP, ist in das AutoCAD-Paket integriert. Mit AutoLISP können Sie Makroprogramme und -funktionen in einer leistungsfähigen und hochentwickelten Programmiersprache schreiben, die besonders für Grafikanwendungen geeignet ist. AutoLISP ist leicht zu handhaben und sehr flexibel.

Der Handbuchteil *AutoLISP* ist als Nachschlagewerk für die Programmiersprache AutoLISP gedacht. AutoLISP ist ein Bestandteil von Common LISP und lehnt sich eng an die Syntax und Konventionen von LISP an, verfügt aber über viele zusätzliche Funktionen, die speziell auf AutoCAD zugeschnitten sind. Über die Programmierung in LISP sind zahlreiche Bücher erschienen. Insbesondere sei an dieser Stelle auf die Bücher *LISP* von Winston und Horn (zweite Auflage) sowie *Looking at LISP* von Tony Hasemer hingewiesen, die beide bei Addison-Wesley erschienen sind.

Im AutoCAD-Verzeichnis *sample* befinden sich zahlreiche AutoLISP-Programme, darunter viele der Beispiele in diesem Handbuch. Darüber hinaus sind weitere Programme als Shareware oder von anderen Entwicklern erhältlich. Da der AutoLISP-Code direkt in der Befehlszeile eingegeben werden kann, ist die Programmierung leicht zu erlernen. Wenn Sie sich AutoLISP einmal angeeignet haben, können Sie Ihre Kenntnisse zum Ausbauen der AutoCAD-Befehle verwenden.

Einen allgemeinen Überblick über AutoLISP und Informationen über das Laden und die Verwendung von vorhandenen AutoLISP-Anwendungen finden Sie in Kapitel 6, "Programmierschnittstellen". Die Kapitel 7-16 konzentrieren sich auf die Verwendung von AutoLISP-Funktionen und das Schreiben von AutoLISP-Anwendungen.

Inhaltsverzeichnis

6 Programmierschnittstellen

Außer den Schnittstellen Befehlszeile und Menü verfügt AutoCAD über die Einrichtung von Befehlsskripten und über benutzerdefinierbare Schnittstellen, die Sie verwenden können, um Zeichnungen und Datenbanken zu manipulieren. Die in diesem Kapitel vorgestellten benutzerdefinierbaren Schnittstellen sind ActiveX Automatisierung (früher OLE Automation genannt) und AutoLISP. Die ADS-Schnittstelle ist veraltet. Sie wird jedoch an dieser Stelle für Anwender von Vorgängerversionen des Release14 beschrieben. Welche Art Schnittstelle Sie benutzen, hängt von den Erfordernissen Ihrer Anwendung und Ihren Programmiererfahrungen ab. AutoCAD setzt auch die ARX-Anwendungsschnittstelle ein, um den Standardbefehlssatz von AutoCAD zu erweitern.

In diesem Kapitel werden die Konzepte und Einsatzbereiche für die verschiedenen Programmierschnittstellen beschrieben. Außerdem wird erklärt, wie Sie Anwendungen von anderen Entwicklern laden und einsetzen.

Genauere Informationen zur Programmierung mit AutoLISP finden Sie in Teil II dieses Handbuchs, "AutoLISP". Die ARX-Programmiersprache wird im *ARX Developer's Guide* beschrieben. AutoLISP-Programme können für ihre Anwendungen Dialogfelder verwenden. Programmierbare Dialogfelder werden in Teil III dieses Handbuchs, "Programmierbare Dialogfelder", beschrieben.

6.1 Befehlsskripte

AutoCAD verfügt über eine Funktion zur Verarbeitung von *Skripten*, mit deren Hilfe Befehle aus einer Textdatei gelesen werden können. Die Vorteile dieser Funktion kommen vor allem dann zum Tragen, wenn eine Folge von Befehlen ausgeführt werden soll. Sie können diese Befehle aufrufen, wenn Sie AutoCAD starten (mit Hilfe einer Sonderform des Befehls **acad**), oder ein Skript von AutoCAD aus mit dem Befehl SCRIPT aufrufen. Die Skriptfunktion stellt eine einfache Möglichkeit dar, fortlaufende Anzeigen für Produktdemonstrationen und Messen zu erstellen.

Um Skriptdateien außerhalb der AutoCAD-Umgebung zu schreiben, benötigen Sie ein Texteditorprogramm (zum Beispiel Windows Notepad) oder ein Textverarbeitungsprogramm (zum Beispiel Word), mit dem Sie die Datei im ASCII-Format speichern können. Die Datei muß die Dateinamenerweiterung *.scr* haben.

Skriptdateien können Kommentare enthalten. Jede Zeile, die mit einem Semikolon (`;`) beginnt, wird als Kommentar angesehen. Solche Zeilen werden beim Verarbeiten der Skriptdatei von AutoCAD ignoriert.

Wird der auszuführende Befehl aus einem Skript eingelesen, so werden die Werte der Systemvariablen PICKADD und PICKAUTO als 1 bzw. 0 angenommen. Auf diese Weise ist die Kompatibilität zu früheren Releases von AutoCAD gewährleistet, und die Benutzeranpassung wird erleichtert (denn Sie müssen die Definition dieser Variablen nicht überprüfen).

Der AutoCAD-Befehl ZURÜCK sieht ein Skript als *Gruppe* an, die durch Eingabe des Buchstabens U rückgängig gemacht werden kann. Jeder Befehl im Skript wird allerdings im Protokoll des Befehls ZURÜCK festgehalten, wodurch die Skriptverarbeitung verlangsamt werden kann. Sie können jedoch auch die Option Steuern Nichts wählen, um den Befehl ZURÜCK vor Ausführung des Skripts (oder am Anfang des Skripts selbst) auszuschalten. Denken Sie daran, die Option nach Beendigung des Skripts wieder einzuschalten (mit ZURÜCK Ganz).

Alle Verweise auf lange Dateinamen, die eingebettete Leerzeichen enthalten, müssen in doppelte Anführungszeichen gesetzt sein. Um beispielsweise die Zeichnung *Mein Haus.dwg* über ein Skript zu öffnen, müssen Sie die folgende Syntax verwenden:

```
open "Mein Haus"
```

6.1.1 Aufrufen eines Skripts beim Laden von AutoCAD

Wenn Sie beim Starten von AutoCAD ein Skript aufrufen möchten, müssen Sie die folgende Befehlsform benutzen:

acad [*bestehende_Zeichnung*] [/t *Vorlage*] [/v *Ansicht*] /b *Skriptdatei*

Die Skriptdatei muß der letzte in der **acad** Programmaufrufzeile genannte Parameter sein. Eine Datei des Typs *.scr* wird vorausgesetzt. Kann AutoCAD die Skriptdatei nicht finden, erfolgt die Meldung, daß die Datei nicht geöffnet werden kann.

Stellen Sie sich folgendes Beispiel vor. Jedesmal, wenn Sie eine neue Zeichnung beginnen, lassen Sie sich das Raster anzeigen, setzen den globalen Linientypfaktor auf 3.0, definieren Layer 0 als aktuellen Layer und wählen die Farbe Rot. Zu diesem Zweck können Sie eine Prototyp-Zeichnung einsetzen. Schreiben Sie jedoch für dieses Beispiel folgendes Skript, das Sie in der Datei *setup.scr* speichern.

`Raster ein`	*Raster anzeigen lassen*
`ltscale 3.0`	*Linientypfaktor setzen*

`layer setzen 0 Farbe Rot 0`	*Aktuellen Layer und seine Farbe wählen Leerzeile zur Beendigung des Befehls* `LAYER`

Um eine Zeichnung mit der Vorlage *Meine_Vorlage.dwt* zu erstellen, rufen Sie AutoCAD folgendermaßen auf:

acad /t MeineVorlage /b setup

Dieser Befehl erstellt eine neue Zeichnung und fährt fort, eine Folge von Setup-Befehlen aus der Datei *setup.scr* auszugeben. Wenn das Skript abgeschlossen ist, wird die Eingabeaufforderung angezeigt.

Anmerkung Sie können keine neue Zeichnung mehr mit dem gleichen Namen öffnen. Der Dateiname wird der Zeichnung zugewiesen, wenn sie gespeichert wird.

Sie müssen mit der Folge von AutoCAD-Eingabeaufforderungen sehr gut vertraut sein, um in der Skriptdatei eine geeignete Folge von Antworten anzugeben. Bedenken Sie immer, daß die Eingabeaufforderungen und die Namen von Befehlen in AutoCAD sich in zukünftigen Releases ändern können, so daß Sie Ihr Skript möglicherweise überarbeiten müssen, wenn Sie zu einem späteren Release von AutoCAD wechseln. Ebenso sollten Sie die Verwendung von Abkürzungen vermeiden, denn zukünftige neue Befehle könnten Mehrdeutigkeiten verursachen. Beachten Sie außerdem, daß jedes Leerzeichen in einer Skriptdatei eine Bedeutung hat. AutoCAD erkennt entweder ein Leerzeichen oder RETURN als Endzeichen eines Befehls oder eines Datenfelds.

Die folgende Syntax einer Systemeingabeaufforderung bestimmt ein Skript zur Neukonfiguration.

acad -r *Zeichnungs_name Skriptdatei*

Verwenden Sie die folgende Syntax, wenn Sie ein Skript für eine Zeichnung starten möchten, die den vorgegebenen Prototyp (neue Zeichnung ohne Namen) benutzt:

acad /b make_dwg

6.1.2 Erstellen von Diashows

Skripte lassen sich gut für die Erstellung von Diashows einsetzen. Normalerweise ist die Geschwindigkeit, mit der Sie Dias anzeigen können, durch die Anzahl der Festplattenzugriffe begrenzt, die zum Lesen der Diadatei erforderlich sind. Allerdings können Sie das nächste Dia bereits von der Festplatte in den Speicher einlesen, während das Publikum noch das aktuelle Dia betrachtet; und dann das neue Dia schnell aus dem Speicher anzeigen.

Um ein Dia vorab in den Speicher zu laden, müssen Sie im Befehl ZEIGDIA ein Sternchen vor dem Dateinamen eingeben. Der nächste ZEIGDIA-Befehl erkennt, daß bereits ein Dia in den Speicher geladen wurde, und zeigt dieses an, ohne nach einem Dateinamen zu fragen. Betrachten Sie zum Beispiel das folgende Skript:

`ZEIGDIA DIA1`	*Beginn der Diashow, DIA1 wird geladen*
`ZEIGDIA *DIA2`	*DIA2 vorab in Speicher geladen*
`VERZÖG 2000`	*Publikum betrachtet DIA1*
`ZEIGDIA`	*DIA2 wird angezeigt*
`ZEIGDIA *DIA3`	*DIA3 vorab in Speicher geladen*
`VERZÖG 2000`	*Publikum betrachtet DIA2*
`ZEIGDIA`	*DIA3 wird angezeigt*
`VERZÖG 3000`	*Publikum betrachtet DIA3*
`RSCRIPT`	*Weiter*

Die Zeit für den Plattenzugriff beim Laden des nächsten Dias überschneidet sich mit der Betrachtungszeit für das aktuelle Dia. Außerdem werden zusätzliche Verzögerungen eingesetzt.

6.2 ActiveX Automatisierung

Bei ActiveX Automatisierung (früher OLE Automation genannt) handelt es sich um eine neue Programmierschnittstelle für AutoCAD. Mit Automatisierung können Sie Skripte, Makros und vollständige Anwendungen von Drittherstellern entwickeln, indem Sie auf Automatisierung basierende Programmierumgebungen wie beispielsweise Visual Basic 4.0 verwenden. Über Automatisierung stellt AutoCAD programmierbare Objekte zur Verfügung, die mit Automatisierung-Steuerungssoftware (z.B. Visual Basic und Excel) bearbeitet werden können.

Mit Hilfe von Automatisierung können Sie AutoCAD-Objekte von jeder Anwendung aus, die als Automatisierung-Steuerungssoftware dient, erstellen oder verändern. So können Sie mit Automatisierung anwendungsübergreifend Makros programmieren. Diese Möglichkeit besteht bei AutoLISP nicht. Mit Automatisierung können Sie die Funktionalitäten vieler Anwendungen in einer einzigen Anwendung kombinieren.

Die verfügbaren Objekte werden als Automatisierungs-Objekte bezeichnet. Automatisierungs-Objekte verfügen über Methoden und Eigenschaften. Unter Methoden versteht man Funktionen, die eine Operation an einem Objekt ausführen. Eigenschaften sind Funktionen, mit denen Informationen über den Zustand eines Objekts festgelegt oder zurückgegeben werden.

Informationen zum Erstellen von Anwendungen, die die von AutoCAD zur Verfügung gestellten Automatisierungs-Objekte verwenden, finden Sie unter *ActiveX Automatisierung* in der Online-Hilfe.

6.2.1 Verwenden von Automatisierungs-Anwendungen

Praktisch alle Arten von Anwendungen können auf die Automatisierungs-Objekte in AutoCAD zugreifen. Bei diesen Anwendungen kann es sich um einzelne ausführbare Programme, DLL-Dateien oder Makros in Anwendungen wie Word oder Excel handeln. In den meisten Fällen werden es einzelne ausführbare Programme sein. Wenn Sie Anwendungen von Anwendungsentwicklern einsetzen, folgen Sie den jeweiligen Anweisungen für Installation und Gebrauch des Produkts. Im folgenden Abschnitt werden einige Möglichkeiten beschrieben, wie ein einzelnes ausführbares Programm in AutoCAD gestartet werden kann.

6.2.1.1 Starten Ihrer Anwendung von der Befehlszeile aus

Sie können die Datei *acad.pgp* verwenden, um einen neuen AutoCAD-Befehl zu definieren, der einen externen Befehl zum Starten Ihrer Anwendung ausführt. Im folgenden Beispiel wird der Befehl RUNAPP1 definiert, der die Anwendung *app1.exe* im Verzeichnis *C:\vbapps* startet. (Fügen Sie diesen Code dem Abschnitt mit den externen Befehlen in Ihrer Datei *acad.pgp* hinzu).

```
RUNAPP1, start c:\vbapps\app1, 0
```

Wenn Ihre Anwendung Befehlszeilenparameter benötigt, können Sie die folgenden verwenden:

```
RUNAPP2, start c:\vbapps\app2, 0, *Parameter: ,
```

Im vorherigen Beispiel wird der Befehl RUNAPP2 definiert, nach dessen Eingabe Sie aufgefordert werden, Parameter festzulegen, die dann an Ihre Anwendung weitergeleitet werden. Weitere Informationen zur Datei *acad.pgp* finden Sie unter "Programmparameterdatei acad.pgp."

Sie können zum Starten einer Anwendung auch die Funktion **`startapp`** von AutoLISP verwenden. Mit dieser Funktion kann eine AutoLISP-Routine eine Anwendung starten, die Automatisierung nutzt. Natürlich kann AutoLISP nach dem Start dieser externen Anwendung deren Operationen nicht steuern; aber Sie können AutoLISP eventuell nutzen, um Anwendungen mit Hilfe bestimmter Parameter zu suchen und auszuführen.

6.2.1.2 Starten Ihrer Anwendung von einem Menü aus

Nachdem Sie, wie im vorigen Abschnitt beschrieben, einen neuen Befehl zum Starten Ihrer Anwendung definiert haben, können Sie diesen Befehl in einem Menümakro verwenden. Dieses Menümakro kann von einer Menüoption in einem beliebigen Menübereich aufgerufen werden.
Wenn Sie nicht mehr als zwei Anwendungen einsetzen, können Sie sie einem der Standard-Pull-Down-Menüs hinzufügen. Wenn Sie eine Gruppe von Anwendungen einsetzen, möchten Sie vielleicht speziell für diese Anwendungen Ihr eigenes Pull-Down-Menü oder Ihren eigenen Werkzeugkasten hinzufügen. In Kapitel 4, "Benutzerspezifische Menüs", werden alle verfügbaren Optionen für die Anpassung von Menüs beschrieben.

Das folgende Beispiel zeigt eine komplette Menüdatei, die ein neues Pull-Down-Menü mit dem Namen MeinAnw definiert. Diese Menüdatei definiert die Menügruppe `MeinAnw` und zwei Gruppen von Menüoptionen. Die ersten drei Menüoptionen verwenden die AutoLISP-Funktion **`startapp`** zum Starten der zugehörigen Anwendung. Die letzen beiden Optionen gehen davon aus, daß die Befehle APP4 und APP5 definiert wurden (in der Datei *acad.pgp*). Diese Datei enthält auch einen Abschnitt Helpstrings, der eine Statuszeilenhilfe zur Verfügung stellt, wenn die zugehörige Menüoption markiert ist. Im Abschnitt Accelerators sind Tastatur-Kurzbefehle definiert, mit denen alle Menümakros gestartet werden können. Diese Menüabschnitte verwenden Menübezeichner (wie beispielsweise `ID_Anw1`), um eine Verbindung zu den mit ihnen verknüpften Operationen herzustellen.

```
// Menü MEINANW
  //
  ***MENUGROUP= MEINANW

  // In diesem Abschnitt wird das neue Pull-Down-Menü definiert.
  ***POP1
  ID_MeinAnwMnu [Mein&Anw]
  ID_MeinAnw1    [Anw1   ]^C^C^P(startapp "app1")(princ) ^P
  ID_MeinAnw2    [Anw2   ]^C^C^P(startapp "app2")(princ) ^P
  ID_MeinAnw3    [Anw3   ]^C^C^P(startapp "app3")(princ) ^P
                 [--]
  ID_MeinAnw4    [Anw4   ]^C^CAPP4
  ID_MeinAnw5    [Anw5   ]^C^CAPP5

  // In diesem Abschnitt werden die Meldungen der Statuszeile
  // definiert, die angezeigt werden, wenn eine entsprechende
  // Menüoption markiert ist.
  ***HELPSTRINGS
  ID_MeinAnw1    [Dies ist ANW1]
  ID_MeinAnw2    [Dies ist ANW2]
  ID_MeinAnw3    [Dies ist ANW3]
  ID_MeinAnw4    [Dies ist ANW4]
  ID_MeinAnw5    [Dies ist ANW5]

  // In diesem Abschnitt werden die Tastatur-Kurzbefehle definiert, // die
  Menümakros in den zugehörigen Pull-Down-Menüs starten.
  ***ACCELERATORS
  ID_MeinAnw1    [CONTROL+SHIFT+"1"]
  ID_MeinAnw2    [CONTROL+SHIFT+"2"]
  ID_MeinAnw3    [CONTROL+SHIFT+"3"]
  ID_MeinAnw4    [CONTROL+SHIFT+"4"]
  ID_MeinAnw5    [CONTROL+SHIFT+"5"]
```

Nachdem Sie diese Daten in einer Datei mit dem Namen *meinanw.mnu* gespeichert haben, verwenden Sie den Befehl MENÜLAD, um dieses Menü in die Menüleiste aufzunehmen. Wenn Sie diese Datei laden, müssen Sie den MNU-Dateityp (*.mnu*) festlegen.

6.3 AutoLISP

AutoLISP basiert auf der Programmiersprache LISP, die einfach zu erlernen ist und dennoch vielseitig eingesetzt werden kann. AutoCAD verfügt über ein internes Interpretierprogramm für AutoLISP, mit dessen Hilfe Sie AutoLISP-Code in die Befehlszeile eingeben oder aus externen Dateien laden können. AutoLISP-Anwendungen oder -Routinen können auf vielfache Weise interaktiv mit AutoCAD arbeiten. Diese Routinen können den Benutzer direkt zur Eingabe auffordern, unmittelbar auf interne AutoCAD-Befehle zugreifen und Objekte in der Datenbank einer Zeichnung modifizieren oder erstellen. Beim Erstellen von AutoLISP-Routinen können Sie AutoCAD fachspezifische Befehle hinzufügen. Viele der AutoCAD-Standardbefehle sind in Wirklichkeit AutoLISP-Anwendungen.

Da AutoCAD AutoLISP-Code direkt liest, ist keine Kompilierung nötig. Wenn Sie Code in die Befehlszeile eingeben, können Sie die Ergebnisse sofort sehen. Darum ist die AutoLISP-Sprache unabhängig von Ihrer Programmiererfahrung hervorragend zum Experimentieren geeignet. Wenn Sie einmal mit AutoLISP vertraut sind, werden Sie feststellen, daß es sich hervorragend als Erweiterung der grundlegenden AutoCAD-Befehle verwenden läßt.

Selbst wenn Sie keine eigenen AutoLISP-Anwendungen schreiben möchten, enthält Ihr AutoCAD-Paket viele nützliche Routinen in den Verzeichnissen *sample* und *bonus*. Routinen sind auch als Shareware-Produkte und über Drittentwickler erhältlich. Wenn Sie wissen, wie Sie diese Routinen laden und anwenden, können Sie Ihre Produktivität steigern.

Anmerkung Geht die Befehlseingabe von der AutoLISP-Funktion **`command`** aus, wird angenommen, daß die Systemvariablen PICKADD und PICKAUTO auf 1 bzw. 0 gesetzt sind. Auf diese Weise ist die Kompatibilität zu früheren Releases von AutoCAD gewährleistet, und die Benutzeranpassung wird erleichtert (denn Sie müssen die Definition dieser Variablen nicht überprüfen).

6.3.1 Verwenden von AutoLISP-Anwendungen

AutoLISP-Anwendungen werden als ASCII-Textdateien mit der Dateinamenerweiterung *.lsp* gespeichert. Diese Dateien haben im allgemeinen einen Header, der die verwendete(n) Routine(n) und etwaige besondere Instruktionen beschreibt. Dieser Header kann auch Kommentare enthalten, die den Autor nennen, und rechtliche Informationen zum Gebrauch der Routine geben. Vor Kommentaren steht ein Semikolon (;). Mit einem Texteditor- oder einem Textverarbeitungsprogramm, das ASCII-Textdateien erstellen kann, können Sie diese Dateien editieren oder anzeigen lassen.

Bevor Sie eine AutoLISP-Anwendung einsetzen können, muß sie zunächst geladen werden. Zum Laden einer Anwendung können Sie den Befehl APPLOAD (der selbst wieder eine AutoLISP-Anwendung ist) oder die AutoLISP-Funktion **`load`** verwenden (siehe auch „APPLOAD" in der *AutoCAD Befehlsreferenz*). Beim Laden einer AutoLISP-Anwendung wird der AutoLISP-Code von der *.lsp*-Datei in den Speicher Ihres Systems geladen.

Um eine Anwendung mit der Funktion **`load`** zu laden, müssen Sie in der Befehlszeile einen AutoLISP-Code eingeben. Wenn die Funktion **`load`** erfolgreich ausgeführt wurde, wird der Wert des letzten Ausdrucks in der Datei in der Befehlszeile angezeigt. Dabei handelt es sich gewöhnlich um den Namen der letzten in der Datei definierten Funktion oder aber um Anweisungen zum Gebrauch der neu geladenen Funktion. Konnte die Funktion **`load`** nicht erfolgreich ausgeführt werden, gibt sie eine AutoLISP-Fehlermeldung zurück. Eine mißlungene Ausführung der Funktion **`load`** kann durch fehlerhafte Codierungen in der Datei verursacht werden oder darauf zurückzuführen sein, daß in der Befehlszeile der falsche Dateiname eingegeben wurde. Die Syntax für die Funktion **`load`** lautet folgendermaßen:

`(load` *Dateiname [beiFehler]*`)`

Diese Syntax verdeutlicht, daß die Funktion **`load`** zwei Argumente hat*:* `Dateiname` (obligatorisch) und `beiFehler` (fakultativ). Wenn Sie eine AutoLISP-Datei über die Befehlszeile laden, geben Sie normalerweise nur das Argument `Dateiname` an. Mit dem folgenden Beispiel wird die AutoLISP-Datei *meindat.lsp* geladen.

Befehl: **(load "meindat")**

Die Dateinamenerweiterung *.lsp* muß nicht eingegeben werden. Dieses Format ist auf jede beliebige *.lsp*-Datei im aktuellen Bibliothekspfad anwendbar (siehe "Der Bibliotheksuchpfad").

Wenn Sie eine AutoLISP-Datei laden möchten, die nicht in dem Bibliothekspfad enthalten ist, müssen Sie den kompletten Pfad nennen und den Dateinamen als Argument `Dateiname` angeben. Bei der Angabe des Verzeichnispfades müssen Sie einen Schrägstrich (`/`) oder zwei umgekehrte) als Trennzeichen eingeben, da der einfache umgekehrte Schrägstrich in AutoLISP eine besondere Bedeutung hat.

Befehl: **(load "d:/Dateien/mehrlisp/neueDatei")**

6.3.2 Automatisches Laden und Ausführen

Wenn Sie sich eine Bibliothek nützlicher AutoLISP-Routinen erstellen, möchten Sie vielleicht, daß sie mit jedem Aufruf von AutoCAD automatisch geladen wird. Weiterhin möchten Sie vielleicht, daß gewisse Befehle oder Funktionen zu bestimmten Zeitpunkten in einer Zeichnungssitzung ausgeführt werden.

AutoCAD lädt den Inhalt zweier benutzerdefinierbarer Dateien automatisch: *acad.lsp* und die das aktuelle Menü begleitende *.mnl*-Datei. Definiert eine dieser Dateien eine Funktion des speziellen Typs **`S::STARTUP`**, so läuft diese Routine unmittelbar nach der vollständigen Initialisierung der Zeichnung ab. Die Funktion **`S::STARTUP`** wird in "S::STARTUP-Funktion - Automatische Ausführung" beschrieben.

6.3.2.1 Die Datei acad.lsp - Automatisches Laden

Die Datei *acad.lsp* ist nützlich, wenn Sie eine Bibliothek mit AutoLISP-Routinen bei jedem Starten von AutoCAD automatisch laden möchten. Jedesmal, wenn Sie eine Zeichnung beginnen, durchsucht AutoCAD den Bibliothekspfad nach einer *acad.lsp*-Datei. Wenn das Programm eine Datei findet, lädt es sie in den Speicher.

Wenn die AutoLISP-Neuinitialisierung aktiviert ist, wird die Datei *acad.lsp* jedes Mal neu geladen, wenn eine neue Zeichnung begonnen oder eine vorhandene Zeichnung geöffnet wird. Ist die Neuinitialisierung deaktiviert, wird die Datei *acad.lsp* nur geladen, wenn AutoCAD gestartet wird. Die AutoLISP-Neuinitialisierung wird durch die Option AutoLISP für Zeichnungen neu laden der Registerkarte Kompatibilität im Dialogfeld Voreinstellungen oder durch die Systemvariable LISPINIT gesteuert.

Anmerkung Nehmen Sie an der Datei *acadr14.lsp keine* Änderungen vor. Die Datei *acadr14.lsp* enthält AutoLISP-definierte Funktionen, die von AutoCAD benötigt werden. Diese Datei wird unmittelbar vor dem Laden der Datei *acad.lsp*in den Speicher geladen.

Die Datei *acad.lsp* kann den AutoLISP-Code für eine oder mehrere Routinen oder einfach eine Folge von Aufrufen der Funktion **load** enthalten. Letztere Methode ist vorzuziehen, da Änderungen einfacher vorgenommen werden können. Wenn Sie den folgenden Code als *acad.lsp*-Datei speichern, werden die Dateien *meinanw1.lsp*, *bau.lsp* und *zähler.lsp* jedesmal automatisch geladen, wenn Sie AutoCAD starten.

```
(load "meinanw1")
(load "bau")
(load "zähler")
```

AutoCAD sucht in der durch den Bibliothekspfad festgelegten Reihenfolge nach einer *acad.lsp*-Datei. Sie können daher in jedem Zeichnungsverzeichnis eine andere *acad.lsp*-Datei haben. Für bestimmte Typen von Zeichnungen oder Aufgaben können Sie dann spezielle AutoLISP-Routinen festlegen.

Tritt ein AutoLISP-Fehler auf, während Sie die Datei *acad.lsp* laden, wird der Rest der Datei ignoriert und nicht geladen. In der Datei *acad.lsp* aufgeführte Dateien, die nicht existieren oder im AutoCAD-Bibliothekspfad nicht zu finden sind, verursachen generell einen Fehler. Daher ist es ratsam, bei der Funktion **load** das Argument `beiFehler` einzusetzen. Das oben genannte Beispiel für eine *acad.lsp*-Datei kann wie im folgenden dargestellt umgeschrieben werden, damit das Argument `beiFehler` berücksichtigt wird:

```
(princ (load "meinanw1" "\nMEINANW1.LSP-Datei nicht geladen."))
(princ (load "bau" "\nBAU.LSP-Datei nicht geladen."))
(princ (load "zähler" "\nZÄHLER.LSP-Datei nicht geladen."))
(princ)
```

Ist der Aufruf der Funktion **load** erfolgreich, gibt sie den Wert des letzten Ausdrucks der Datei zurück (gewöhnlich der Name der letzten definierten Funktion oder eine Mitteilung zum Gebrauch der Funktion). Ist der Funktionsaufruf nicht erfolgreich, wird der Wert des Arguments `beiFehler` zurückgegeben. Im oben dargestellten Beispiel wird der von der Funktion **load** zurückgegebene Wert an die Funktion **princ** weitergegeben, was dazu führt, daß der Wert in der Befehlszeile angezeigt wird. Tritt zum Beispiel ein Fehler auf, während AutoCAD die Datei *meinanw1.lsp* lädt, zeigt die Funktion **princ** die folgende Meldung an, während AutoCAD die beiden übrigen Dateien lädt.

MEINANW1.LSP-Datei nicht geladen.

Anmerkung Wenn Sie die Funktion **command** in einer *acad.lsp*- oder *.mnl*-Datei benutzen, sollte sie nur von einer **defun**-Anweisung aus aufgerufen werden. Verwenden Sie die Funktion **S::STARTUP**, um Befehle zu definieren, die direkt zu Anfang der Zeichnungssitzung aktiviert werden müssen.

6.3.2.2 Die Datei .mnl - Automatisches Laden

Der andere Dateityp, den AutoCAD automatisch lädt, begleitet Ihre aktuelle Menüdatei und hat die Dateinamenerweiterung *.mnl*. Wenn AutoCAD eine Menüdatei lädt, sucht das Programm nach einer *.mnl*-Datei mit einem passenden Dateinamen. Wird die Datei gefunden, wird sie in den Speicher geladen.

Diese Funktion gewährleistet, daß AutoCAD die für ein fehlerfreies Funktionieren eines Menüs nötigen AutoLISP-Funktionen lädt. So verläßt sich zum Beispiel das Standardmenü von AutoCAD, die Datei *acad.mnu*, darauf, daß die Datei *acad.mnl* korrekt geladen wird. Diese Datei definiert zahlreiche, vom Menü benötigte AutoLISP-Funktionen. Die *.mnl*-Datei wird nach der Datei *acad.lsp* geladen.

Anmerkung Wird eine Menüdatei mit Hilfe der AutoLISP-Funktion **command** geladen - die Syntax gleicht der von `(command "Menü" "Neues_menü")` -, wird die zugehörige *.mnl*-Datei so lange nicht geladen, bis die gesamte AutoLISP-Routine ausgeführt worden ist.

Wenn Sie zum Beispiel ein benutzerdefiniertes Menü mit dem Namen *Neues_menü.mnu* erstellen, und Sie drei AutoLISP-Dateien (*Neu1.lsp*, *Neu2.lsp* und *Neu3.lsp*) laden müssen, damit das Menü korrekt arbeitet, sollten Sie eine ASCII-Textdatei mit dem Namen *Neues_menü.mnl* mit folgendem Inhalt erstellen:

```
(load "Neu1")
(load "Neu2")
(load "Neu3")
(princ "\nNeues_menü Dienstprogramme... geladen.")
(princ)
```

In diesem Beispiel können die Aufrufe der Funktion **princ** verwendet werden, um Statusmeldungen anzuzeigen. Der erste Aufruf der Funktion **princ** führt zu folgender Anzeige in der Befehlszeile:

Neues_menü Dienstprogramme... geladen.

Der zweite Aufruf der Funktion **princ** führt zum stillen Abbruch der AutoLISP-Funktion. Ohne diesen zweiten Aufruf von **princ** würde die Meldung zweimal angezeigt werden. Wie weiter oben erwähnt, können Sie das Argument *beiFehler* als zusätzliche Vorsichtsmaßnahme zu den Aufrufen der Funktion **load** hinzufügen.

6.3.2.3 Automatisches Laden von Befehlen

Wenn Sie mit Hilfe der zuvor beschriebenen Vorgehensweisen einen Befehl automatisch laden, nimmt die Befehlsdefinition einen Teil des zur Verfügung stehenden Speicherplatzes ein. Es spielt dabei keine Rolle, ob Sie den Befehl wirklich benutzen. Die AutoLISP-Funktion **autoload** macht einen Befehl verfügbar, ohne die gesamte Routine in den Speicher zu laden. Wenn Sie Ihrer Datei *acad.lsp* den folgenden Code hinzufügen, werden die Befehle BEF1, BEF2 UND BEF3 von der Datei *bef.lsp* sowie der Befehl NEUBEF von der Datei *neubef.lsp* automatisch geladen.

```
(autoload "BEF" '("BEF1" "BEF2" "BEF3"))
(autoload "NEUBEF" '("NEUBEF"))
```

Wenn Sie das erste Mal einen automatisch geladenen Befehl in der Befehlszeile eingeben, lädt AutoLISP die gesamte Befehlsdefinition aus der entsprechenden Datei. Weitere Informationen zu der Funktion **autoload** finden Sie unter **"autoload"** in Kapitel 13. AutoLISP stellt außerdem Funktionen zum automatischen Laden für ARX-Anwendungen zur Verfügung (siehe **"autoxload"** und **"autoarxload"** in Kapitel 13).

6.4 ARX - AutoCAD Runtime Extension

ARX (AutoCAD Runtime Extension) ist eine neue Kompiliersprachenprogrammumgebung für die Entwicklung von AutoCAD-Anwendungen. Die ARX-Umgebung unterstützt derzeit die ADS-Bibliothek mit ADSRX und wird in Zukunft auch viele andere Bibliotheken unterstützen.

ARX-Anwendungen arbeiten zwecks Leistungsoptimierung im gleichen Prozeß- und Speicherbereich wie AutoCAD. APIs können von der ARX-Umgebung effizienter exportiert werden als von der ADS-Programmumgebung.

6.4.1 Verwenden von ARX-Anwendungen

Verwenden Sie die Option Load des Befehls ARX, um eine ARX-Anwendung zu laden. Nach dem Laden stehen alle durch diese Anwendung definierten Befehle über die Eingabeaufforderung zur Verfügung.

ARX-Anwendungen belegen Speicherplatz. Verwenden Sie die Option Unload des Befehls ARX, wenn Sie eine Anwendung beendet haben und sie aus dem Speicher entfernen möchten.

Sie können eine ARX-Anwendung auch mit der AutoLISP-Funktion **arxload** laden. Die Syntax für die Funktion **arxload** ist fast identisch mit der Syntax der Funktion **load** für AutoLISP-Dateien und mit derjenigen der Funktion **xload** für ADS-Anwendungen. Hat die Funktion **arxload** das ARX-Programm erfolgreich geladen, gibt sie den Programmnamen zurück. Die Syntax für die Funktion **arxload** lautet folgendermaßen:

(arxload *Dateiname [beiFehler]*)

Die beiden Argumente für die Funktion **arxload** lauten *Dateiname* und *beiFehler*. Wie bei der Funktion **load** ist das Argument *Dateiname* obligatorisch und muß den kompletten Pfadnamen der zu ladenden ARX-Programmdatei enthalten. Das Argument *beiFehler* ist fakultativ und wird normalerweise nicht verwendet, wenn Sie ein ARX-Programm von der Befehlszeile aus laden. Im folgenden Beispiel wird die ARX-Anwendung *meinanw.arx* geladen.

```
(arxload "meinanw")
```

Wie bei AutoLISP-Dateien durchsucht AutoCAD den Bibliothekspfad nach der angegebenen Datei. Müssen Sie eine Datei laden, die nicht im Bibliothekspfad enthalten ist, müssen Sie den kompletten Pfadnamen der Datei angeben. Beachten Sie, daß Sie entweder einen einzelnen Schrägstrich (/) oder aber zwei umgekehrte Schrägstriche (\\ ) als Verzeichnistrennzeichen verwenden müssen, wenn Sie einen kompletten Pfadnamen angeben.

Wenn Sie versuchen, eine bereits geladene Anwendung zu laden, wird ein Fehler erzeugt. Vor dem Aufruf von **arxload** sollten Sie mit der Funktion **arx** überprüfen, welche Anwendungen aktuell geladen sind.

Wenn Sie eine Anwendung mit AutoLISP beenden möchten, verwenden Sie die Funktion **arxunload**. Im folgenden Beispiel wird die Anwendung *meinanw* beendet.

```
(arxunload "meinanw")
```

Wenn Sie die Funktion **arxunload** verwenden, wird nicht nur die Anwendung aus dem Speicher entfernt, sondern auch die zu dieser Anwendung gehörenden Befehlsdefinitionen.

6.4.2 Automatisches Laden von ARX-Anwendungen

Die Datei *acad.rx* enthält eine Liste der ARX-Programmdateien, die beim Starten von AutoCAD automatisch geladen werden. Sie können diese Datei mit einem Texteditor- oder Textverarbeitungsprogramm editieren, das Dateien im ASCII-Textformat erstellen kann. Sie können diese Datei an Ihre Erfordernisse anpassen, indem Sie Inhalt hinzufügen oder löschen und die gewünschten ARX-Programme verfügbar machen. AutoCAD liest die Datei auf eine ähnliche Art und Weise wie die Datei *acad*.rx.

Sie können die Funktion des automatischen Ladens mit den ARX-definierten AutoCAD-Befehlen verwenden. (Siehe "Automatisches Laden von Befehlen" und **"autoarxload"** in Kapitel 13.)

Mit der Funktion **arxload** können Sie ARX-Anwendungen aus einer *.mnl*-Datei laden. So wird gewährleistet, daß eine ARX-Anwendung, die für das fehlerfreie Funktionieren eines Menüs benötigt wird, beim Laden der Menüdatei ebenfalls geladen wird.

6.4.3 ADSRX

ADSRX ist eine Untergruppe der ARX-Funktionalität und entspricht dem früheren ADS. ADSRX ist mit den Bibliotheken und Header-Dateien verfügbar, die mit AutoCAD geliefert werden. Mit ADSRX können Sie ADS-Anwendungen problemlos in ARX-Anwendungen übertragen.

Da die Hauptaufgabe von ADSRX darin besteht, ADS-Anwendungen nach ARX zu portieren, wird es in künftigen Releases nicht mehr unterstützt.

6.5 ADS

ADS ist eine veraltete, auf der Programmiersprache C basierende Programmierumgebung zur Entwicklung von AutoCAD-Anwendungen. Die in diesem Abschnitt enthaltenen Informationen sind für Anwender gedacht, die noch ADS-Anwendungen einsetzen. Viele ADS-Anwendungen werden in absehbarer Zukunft noch mit AutoCAD einsatzfähig sein. Da das ADS-API jedoch nicht mehr unterstützt wird, sollten Sie auf ARX-Anwendungen umsteigen.

6.5.1 Arbeiten mit ADS-Anwendungen

Sie laden eine ADS-Anwendung ähnlich wie eine AutoLISP-Datei. Um eine ADS-Anwendung von der AutoCAD-Befehlszeile aus zu laden, benutzen Sie die AutoLISP-Funktion **xload**. Die Syntax der Funktion **xload** ist fast mit der Funktion **load** identisch, die für AutoLISP-Dateien eingesetzt wird. Hat die Funktion **xload** das ADS-Programm erfolgreich geladen, gibt sie den Programmnamen zurück. Die Syntax für die Funktion **xload** lautet folgendermaßen:

(xload *Dateiname [beiFehler]*)

Die beiden Argumente der Funktion **xload** sind *Dateiname* und *beiFehler*. Wie bei der Funktion **load** ist das Argument *Dateiname* obligatorisch und muß aus der kompletten Pfadnamenbeschreibung der zu ladenden ADS-Programmdatei bestehen. Das Argument *beiFehler* ist fakultativ und wird normalerweise nicht verwendet, wenn Sie ein ADS-Programm von der Befehlszeile aus laden. Die folgende Beispielbefehlszeile lädt die ADS-Anwendung *meinanw.exe*.

Befehl: **(xload "meinanw")**

Wie bei AutoLISP-Dateien durchsucht AutoCAD den Bibliothekspfad nach der angegebenen Datei. Müssen Sie eine Datei laden, die nicht im Bibliothekspfad enthalten ist, müssen Sie die komplette Pfadnamenbeschreibung der Datei eingeben. Beachten Sie, daß Sie entweder einen einzelnen Schrägstrich (/) oder aber zwei umgekehrte Schrägstriche (\ ) als Verzeichnistrennzeichen verwenden müssen, wenn Sie einen kompletten Pfadnamen angeben.

Versuchen Sie, ein Programm zu laden, das bereits geladen ist, so wird die folgende Meldung angezeigt (*xxx* ist dabei der Anwendungsname).

Anwendung "*xxx*" ist schon geladen

ADS-Anwendungen benötigen Speicherplatz. Wenn Sie eine Anwendung beendet haben und sie aus dem Speicher entfernen möchten, können Sie die AutoLISP- Funktion **xunload** verwenden. Im folgenden Beispiel wird die Anwendung *meinanw* beendet.

Befehl: **(xunload "meinanw")**

Wenn Sie die Funktion **xunload** verwenden, wird nicht nur die Anwendung aus dem Speicher entfernt, sondern auch die zu dieser Anwendung gehörenden Befehlsdefinitionen.

6.5.2 Automatisches Laden von ADS-Anwendungen

Die Datei *acad.ads* enthält eine Liste der ADS-Programmdateien, die beim Starten von AutoCAD automatisch geladen werden. Sie können diese Datei mit einem Texteditor- oder Textverarbeitungsprogramm editieren, das Dateien im ASCII-Textformat erstellen kann. Sie können diese Datei an Ihre Erfordernisse anpassen, indem Sie Inhalt hinzufügen oder löschen und die gewünschten ADS-Programme verfügbar machen.

AutoCAD sucht in der durch den Bibliothekspfad festgelegten Reihenfolge nach der *acad.ads*-Datei. Sie können daher in jedem Zeichnungsverzeichnis eine andere *acad.ads*-Datei haben. Auf diese Weise stehen Ihnen für bestimmte Typen von Zeichnungen besondere ADS-Programme zur Verfügung. Angenommen, Sie fassen dreidimensionale Modellzeichnungen in einem Verzeichnis mit dem Namen *AcadProj/3d_zeich* zusammen. Wird dieses Verzeichnis als das aktuelle festgelegt, können Sie die Datei *acad.ads* in dieses Verzeichnis kopieren und sie folgendermaßen modifizieren:

```
meinanw1
anderanw
```

Wenn Sie diese neue Datei *acad.ads* in das Verzeichnis *Acad-Proj/3d_zeich* aufnehmen und dann AutoCAD mit diesem als aktuelles Verzeichnis starten, so werden die neuen ADS-Programme geladen und sind von der Eingabeaufforderungszeile von AutoCAD aus verfügbar. Da die ursprüngliche Datei *acad.ads* sich immer noch mit den AutoCAD-Programmdateien in einem Verzeichnis befindet, wird die vorgegebene Datei *acad.ads* geladen, wenn Sie AutoCAD von einem anderen Verzeichnis aus laden, das keine *acad.ads*-Datei enthält.

Mit der Funktion **`xload`** können Sie ADS-Anwendungen von einer *.mnl*-Datei aus laden. Auf diese Weise wird sichergestellt, daß ein ADS-Programm, das für ordnungsgemäße Menüoperationen erforderlich ist, zusammen mit einer Menüdatei geladen wird.

Sie können die Funktion des automatischen Ladens benutzen, die von vielen ADS-definierten AutoCAD-Befehlen verwendet wird. (Siehe "Automatisches Laden von Befehlen" und **`"autoxload"`** in Kapitel 13).

7 AutoLISP-Grundlagen

Dieses Kapitel stellt die Grundlagen der Programmiersprache AutoLISP vor. Da der AutoLISP-Code nicht kompiliert ist, können Sie den Code in der Befehlszeile eingeben und die Ergebnisse sofort sehen. Um Einsteigern das Arbeiten zu erleichtern, sind die Beispiele in diesem Kapitel so formatiert, daß sie ohne Probleme in der Befehlszeile eingegeben werden können.

7.1 AutoLISP-Ausdrücke

Wenn Sie Text an der Eingabeaufforderung eingeben, interpretiert AutoCAD die eingegebenen Zeichen durch Vergleich mit einer internen Liste gültiger Befehlsnamen. Bei einer Übereinstimmung wertet AutoCAD die Definition des entsprechenden Befehls aus. AutoCAD übergibt eingegebene AutoLISP-Codes an das AutoLISP-Interpretierprogramm. Das Kernstück des AutoLISP-Interpretierers ist das Auswertungsprogramm. Es liest eine Codezeile, wertet sie aus und gibt das Ergebnis zurück. Der Code muß in Form eines AutoLISP-Ausdrucks angegeben werden und kann entweder aus einer Datei oder als Benutzereingabe in der AutoCAD-Befehlszeile gelesen werden.

Alle AutoLISP-Ausdrücke besitzen das folgende Format:

```
(Funktion Argumente)
```

Jeder Ausdruck beginnt mit einer linken runden Klammer, besteht aus einem Funktionsnamen und aus optionalen Argumenten für diese Funktion, die selbst Ausdrücke sein können. Der Ausdruck endet mit einer rechten runden Klammer. Jeder Ausdruck gibt einen Wert zurück, der von dem ihn umschließenden Ausdruck verwendet wird; wenn er nicht von einem anderen Ausdruck umschlossen ist, wird der Wert in der AutoCAD-Befehlszeile ausgegeben. Der folgende Code verwendet beispielsweise drei Funktionen.

```
(Funkt1 (Funkt2 Argumente)(Funkt3 Argumente))
```

Die erste Funktion, **Funkt1**, besitzt zwei Argumente und die beiden anderen Funktionen, **Funkt2** und **Funkt3**, besitzen jeweils ein Argument. Die Funktionen **Funkt2** und **Funkt3** sind von der Funktion **Funkt1** eingeschlossen, ihre Werte werden also als Argumente an **Funkt1** zurückgegeben. Funktion **Funkt1** wertet die beiden Argumente aus und gibt das Ergebnis an die Befehlszeile zurück.

Wenn Sie einen AutoLISP-Ausdruck an der AutoCAD-Eingabeaufforderung eingeben, berechnet AutoLISP den Ausdruck, zeigt das Ergebnis an und zeigt anschließend wieder die Eingabeaufforderung an. Das folgende Beispiel zeigt, wie Sie die Funktion * (Multiplikation) verwenden können, die als Argumente eine oder mehrere reelle Zahlen benötigt.

```
Befehl: (* 2 27)
54
```

Weil dieser Beispielcode nicht in einen anderen Ausdruck eingeschlossen ist, gibt er das Ergebnis in der Befehlszeile aus.

In andere Ausdrücke verschachtelte Ausdrücke geben ihr Ergebnis an den umschließenden Ausdruck zurück. Im folgenden Beispiel wird das Resultat der Funktion **+** (Addition) als eines der Argumente für die Funktion ***** (Multiplikation) verwendet.

```
Befehl: (* 2 (+ 5 10) )
30
```

Wenn Sie eine falsche Anzahl schließender (rechter) Klammern eingeben, zeigt AutoLISP die folgende Eingabeaufforderung an:

```
n>
```

Hierbei ist *n* eine ganze Zahl und angibt an, wie viele Klammerebenen nicht geschlossen sind. Sollten Sie diese Eingabeaufforderung auf dem Bildschirm sehen, müssen Sie *n* rechte Klammern eingeben, damit der Ausdruck berechnet werden kann.

```
Befehl: (* 2 (+ 5 10
2> ) )
30
```

Häufig wird das abschließende Anführungszeichen (") in einer Textzeichenfolge vergessen; in diesem Fall werden die rechten Klammern als Teil der Zeichenfolge interpretiert und haben keine Auswirkung auf den Wert *n*.. Zur Behebung dieses Fehlers müssen Sie die Funktion durch Drücken von ESC abbrechen und nochmals korrekt eingeben.

7.1.1 AutoLISP-Datentypen

Das AutoLISP-Auswertungsprogramm verarbeitet die Ausdrücke in der Reihenfolge und entsprechend dem Datentyp des in den runden Klammern stehenden Codes. Damit Sie AutoLISP umfassend nutzen können, müssen Sie zuerst die Unterschiede zwischen den verschiedenen Datentypen kennen und verstehen, wie sie verwendet werden.

7.1.1.1 Subrs und extern definierte Subrs

Die meisten der in Kapitel 13, "AutoLISP Funktionskatalog", beschriebenen AutoLISP-Funktionen sind integrierte Unterprogramme, sogenannte *Subrs*. Die in Kapitel 14, "Zugriff auf extern definierte Befehle und Systemvariablen", beschriebenen Funktionen werden durch externe ADSRX- und ARX-Anwendungen defininiert, die als *externe Subrs* bezeichnet werden. Bei den Namen der Subrs und externen Subrs wird nicht nach Groß- und Kleinschreibung unterschieden.

Die Funktion **`princ`** ist ein Unterprogramm. Die Funktion **`acad_strlsort`** ist ein externes Unterprogramm.

7.1.1.2 Ganzzahlen

Ganzzahlen enthalten keinen Dezimalpunkt. AutoLISP-Ganzzahlen sind mit Vorzeichen versehene 32-Bit-Zahlen mit einem Wertebereich von +2,147,483,648 bis -2,147,483,647. AutoLISP verwendet zwar intern 32-Bit-Werte, aber die zwischen AutoLISP und AutoCAD übertragenen Ganzzahlwerte sind auf 16 Bit beschränkt. Sie können daher keine Werte an AutoCAD übergeben, die größer als +32767 oder kleiner als -32768 sind. Wenn Sie eine Ganzzahl explizit in einem AutoLISP-Ausdruck verwenden, nennt man diesen Wert *Konstante*. Die Zahlen 2, -56 und 1,200,196 sind Beispiele für gültige AutoLISP-Ganzzahlen.

7.1.1.3 Reelle Zahlen

Reelle Zahlen enthalten einen Dezimalpunkt. Zahlen zwischen -1 und 1 muß eine Null vorangestellt werden. Reelle Zahlen werden im Gleitkommaformat mit doppelter Präzision gespeichert, das mindestens 14 signifikante Stellen ermöglicht, auch wenn in der AutoCAD-Befehlszeile nur 6 Stellen angezeigt werden. Reelle Zahlen können in wissenschaftlicher Schreibweise mit einem optionalen e oder E, gefolgt vom Exponenten der Zahl, angegeben werden (zum Beispiel ist 0.0000041 identisch mit 4.1e-6). Wenn Sie eine reelle Zahl explizit in einem AutoLISP-Ausdruck verwenden, nennt man diesen Wert Konstante. Die Zahlen 3.1, 0.23, -56.123 und 21,000,000.0 sind beispielsweise gültige reelle Zahlen in AutoLISP.

7.1.1.4 Zeichenketten

Eine Zeichenkette besteht aus einer Gruppe von in Anführungszeichen eingeschlossenen Zeichen. Sie können mit Hilfe des umgekehrten Schrägstrichs (\) Steuerzeichen (oder Escapecodes) in eine Zeichenkette einfügen. Eine einzelne Zeichenkette darf nicht mehr als 132 Zeichen enhalten. Wenn Sie eine in Anführungszeichen gesetzte Zeichenkette explizit in einem AutoLISP-Ausdruck verwenden, nennt man diesen Wert *literale Zeichenkette* oder *Zeichenkettenkonstante*.

Beispiele für gültige Zeichenketten sind "Zeichenkette 1" und "\nErsten Punkt eingeben:".

7.1.1.5 Listen

Eine AutoLISP-Liste besteht aus verknüpften Werten, die durch Leerzeichen getrennt und in runde Klammern eingeschlossen sind. Mit Hilfe von Listen können Sie mehrere verknüpfte Werte effizient speichern. AutoCAD beschreibt 3D-Punkte als Listen mit drei reellen Zahlen.

Beispiele für Listen sind (1.0 1.0 0.0), ("dies" "das" "das andere") und (1 "EINS").

7.1.1.6 Auswahlsätze

Auswahlsätze sind Gruppen mit einem oder mehreren Objekten. Sie können mit bestimmten AutoLISP-Routinen interaktiv Objekte zu den Auswahlsätzen hinzufügen oder aus ihnen entfernen.

Im folgenden Beispiel liefert die Funktion **`ssget`** einen Auswahlsatz mit allen Objekten in einer Zeichnung zurück.

Befehl: **(ssget "X")**

<Auswahlsatz: 1>

7.1.1.7 Elementnamen

Ein Elementname ist eine numerische Bezeichnung, die den Objekten in einer Zeichnung zugewiesen wird. Eigentlich ist er ein Zeiger in eine von AutoCAD verwaltete Datei, in der AutoLISP den Datensatz und die Vektoren (wenn sie angezeigt werden) des Objekts sucht. Auf diese Bezeichnung kann von AutoLISP-Funktionen verwiesen werden, um Objekte für verschiedene Verarbeitungen auszuwählen. Intern verweist AutoCAD auf Objekte als Elemente.

Im folgenden Beispiel liefert die Funktion **`entlast`** den Namen des letzten in die Zeichnung eingegebenen Objekts.

Befehl: **(entlast)**

<Elementname: 60000016>

7.1.1.8 Dateideskriptoren

Dateideskriptoren sind alphanumerische Bezeichnungen, die den von AutoLISP geöffneten Dateien zugewiesen werden. Beim Einlesen oder Beschreiben einer Datei müssen die AutoLISP-Funktionen auf die Bezeichnung der Datei verweisen.

Das folgende Beispiel öffnet die Datei *meininfo.dat* zum Lesen.

Befehl: **(open "meininfo.dat" "r")**

<Datei: #34614>

7.1.1.9 Symbole und Variablen

AutoLISP verwendet Symbole, um sich auf Daten zu beziehen. Sie können aus beliebigen alphanumerischen Zeichen und allen Satzzeichen außer den folgenden bestehen: **`( ) . ' " ;`** . Ein Symbolname darf nicht ausschließlich numerische Zeichen enthalten.

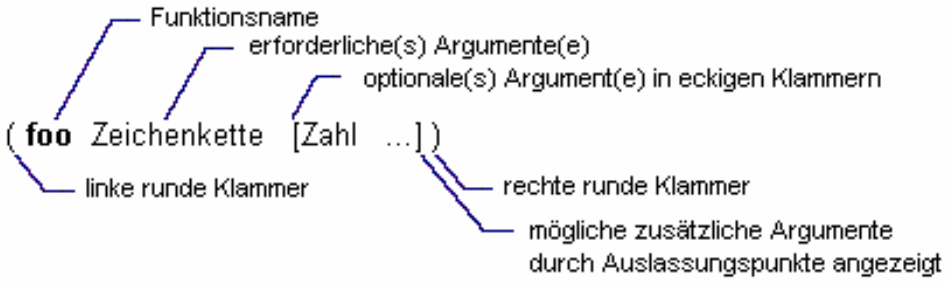
Technisch gesehen bestehen AutoLISP-Anwendungen aus Symbolen oder konstanten Werten, wie beispielsweise Zeichenketten, reellen Zahlen und Ganzzahlen. Aus Gründen der Klarheit wird in diesem Handbuch der Begriff *Symbol* für einen Symbolnamen verwendet, der statische Daten wie beispielsweise Standard- und benutzerdefinierte Funktionen speichert. Der Begriff *Variable* verweist auf einen Symbolnamen, der Programmdaten speichert. Das folgende Beispiel verwendet die Funktion **`setq`**, um der Variablen `str1` den Zeichenkettenwert "`dies ist eine Zeichenkette`" zuzuweisen:

Befehl: **(setq str1 "dies ist eine Zeichenkette")**

"dies ist eine Zeichenkette"

7.1.2 AutoLISP-Funktionssyntax

In diesem Handbuch beschreiben die folgenden Konventionen die Syntax für AutoLISP-Funktionen.



Im vorherigen Beispiel enthält die Funktion **`foo`** ein erforderliches Argument, *`Zeichenkette`*, und ein fakultatives Argument, *`Zahl`*. Es können auch noch weitere *`Zahl`*-Argumente angegeben werden. Oftmals weist der Name des Arguments auf den erwarteten Datentyp hin. Die Beispiele in der folgenden Tabelle zeigen Ihnen sowohl gültige als auch ungültige Aufrufe der Funktion **`foo`**.

Gültige und ungültige Aufrufe der Funktion foo

Gültige Aufrufe	Ungültige Aufrufe
`(foo "catch")`	`(foo 44 13)`
`(foo "catch" 22)`	`(foo "fi" "foe" 44 13)`
`(foo "catch" 22 31)`	`(foo)`

7.2 AutoLISP-Programmdateien

Sie können zwar AutoLISP-Code an der Eingabeaufforderung eingeben, das Testen und die Fehlersuche gestalten sich aber bedeutend einfacher, wenn Sie den AutoLISP-Code aus einer Datei laden, anstatt ihn bei jeder Änderung neu einzugeben. AutoLISP-Code wird üblicherweise in ASCII-Textdateien mit dem Dateierweiterung *.lsp* oder *.mnl* gespeichert Sie können jedoch AutoLISP-Code aus einer beliebigen ASCII-Textdatei laden, wenn Sie der Funktion **`load`** den vollständigen Dateinamen übergeben. Die Syntax der AutoLISP-Ausdrücke ist bei der Eingabe in Dateien oder an der Eingabeaufforderung grundsätzlich die gleiche.

7.2.1 Kommentare

Kommentartext ist häufig in AutoLISP-Programmdateien enthalten. Er ist sowohl für den Programmierer als auch für spätere Benutzer nützlich, die das Programm überarbeiten müssen. Kommentartext kann für folgendes verwendet werden

- Protokollieren von Titel, Autor und Erstellungsdatum
- Einfügen von Anweisungen zur Benutzung einer Routine
- Einfügen von erläuternden Hinweisen in den Rumpf einer Routine
- Während der Fehlersuche Notizen machen
- Eingeben von Zeichen zur optischen Strukturierung

Kommentare beginnen mit einem Semikolon (`;`) und erstrecken sich bis zum Ende der Zeile.

```
; Diese ganze Zeile ist ein Kommentar.
(setq area (* pi r r))  ; Rechenbereich des Kreises
```

Text innerhalb von `;| ... |;` wird ingoriert; Kommentare können deshalb innerhalb einer Programmzeile stehen oder sich über mehrere Zeilen erstrecken. Diese Art von Kommentartext heißt *eingebundener* Kommentar.

```
(setq tmode ;|Hinweistext|; (getvar "tilemode"))
```

Das folgende Beispiel zeigt Kommentartext, der sich über mehrere Zeilen erstreckt:

```
(setvar "orthomode" 1) ;|Kommentartext beginnt hier
wird auf dieser Zeile fortgesetzt
und endet hier unten|; (princ "\nORTHOMODE aktiviert.")
```

Es wird empfohlen, daß Sie beim Schreiben von AutoLISP-Programmen viel Kommentartext hinzufügen. Einige der mit AutoCAD gelieferten *.lsp* -Dateien (aber nicht alle) enthalten anschauliche Beispiele, wie Sie Programmdateien kommentieren können. Die Datei *ai_utils.lsp* enthält sehr viel Kommentartext sowie einige interessante und nützliche AutoLISP-Routinen. Beachten Sie die folgenden Punkte, wenn Sie Ihren Kommentierungsstil entwickeln:

- Der Beschreibungsteil im Header sollte den Gebrauch der Datei so genau beschreiben, daß sie ohne weitere Dokumentation verwendet werden kann.
- Die Anzahl der Strichpunkte vor einem Kommentartext und die Position eines Kommentars in einer Datei kann dazu dienen, den Inhalt oder die Wichtigkeit des Kommentars anzuzeigen.
- Viele Monitore können nur 80 Textspalten anzeigen; wenn einer Ihrer Kommentare in der 81. Spalte beginnt, wird er möglicherweise nicht angezeigt.

Es ist wichtiger, daß Kommentartext überhaupt vorhanden ist, als daß er bestimmten Layout-Regeln entspricht. Dies ist unabhängig von der Art, nach der Sie Kommentartext erstellen.

7.2.2 Einrückung und Ausrichtung

In AutoLISP werden sehr viele runde Klammern verwendet, was das Lesen der Programme erschweren kann. Traditionell wird der Code durch Einrückungen übersichtlicher gemacht. Die Einrückung ermöglicht es Ihnen, den Programmcode so zu formatieren, daß eine Codezeile um so weiter rechts beginnt, je tiefer sie verschachtelt ist. Die

Verschachtelung wird dabei so tief, daß Sie pro Ebene nur um wenige Leerzeichen nach rechts einrücken sollten, im Gegensatz zum normalen C-Sprachstandard, bei dem der Code pro Ebene um vier Leerzeichen nach rechts versetzt wird.

In AutoLISP entsprechen mehrere Leerzeichen zwischen Variablennamen, Konstanten und Funktionsnamen einem einzigen Leerzeichen. Das Zeilenende wird auch wie ein Leerzeichen behandelt.

Die folgenden beiden Ausdrücke führen zum gleichen Ergebnis.

```
(setq test1 123 test2 456)
(setq
    test1 123
    test2 456
)
```

Anmerkung Tabulatoren werden in AutoLISP zwar normalerweise wie Leerzeichen behandelt, sollten aber trotzdem nicht verwendet werden. Manche Plattformen interpretieren sie möglicherweise anders, was zu unvorhersehbaren Resultaten führen könnte.

Die Einrückungsregeln sind schwierig zu erklären, lassen sich aber an Beispielen einfach zeigen. Sie können das Schreiben übersichtlicher Programme am einfachsten erlernen, indem Sie den Code anderer Programmautoren lesen und bestimmte Vorgehensweisen übernehmen. Zahlreiche LISP-Bearbeitungssysteme enthalten Programme, sogenannte "Formatierungsdrucker", die beliebig formatierten LISP-Code einlesen und dann die entsprechenden Einrückungen vornehmen.

7.3 AutoLISP-Variablen

Eine AutoLISP-Variable nimmt den Datentyp des Werts an, der ihr zugewiesen wird. Sie behält ihren ursprünglichen Wert so lange, bis ihr ein neuer zugewiesen wird. Mit der AutoLISP-Funktion **`setq`** können Sie einer Variablen einen Wert zuweisen.

```
(setq Variablenname1 Wert1 [Variablenname2 Wert2...])
```

Die Funktion **`setq`** weist dem gegebenen Variablennamen den angegebenen Wert zu. Sie gibt den Wert als Funktionsergebnis zurück. Wenn Sie an der Eingabeaufforderung **`setq`** eingeben, legt AutoCAD die Variable fest und zeigt den Wert an.

Befehl: **(setq val 3 abc 3.875)**
3.875
Befehl: **(setq layr "AUSSENWÄNDE")**
"AUSSENWÄNDE"
Befehl:

Jede Variable benötigt Speicherplatz, deshalb sollten Sie Variablennamen bei der Programmierung wiederverwenden oder, wenn Sie sie nicht mehr benötigen, auf `nil` setzen. Wenn eine Variable auf `nil` gesetzt wird, wird der für den Wert der Variablen verwendete Speicherplatz freigegeben. Wenn Sie beispielsweise die Variable `val` nicht mehr brauchen, können Sie ihren Wert mit dem folgenden Ausdruck aus dem Speicher freigeben:

Befehl: **(setq val nil)**
nil

7.3.1 Verwendung von Variablen in der Befehlszeile

Nachdem Sie einer Variablen einen Wert zugewiesen haben, können Sie sie als Antwort auf AutoCAD-Eingabeaufforderungen verwenden. Sie können dadurch komplexe Namen oder Zahlen speichern und später wieder benutzen. Wenn Sie den Wert einer Variablen auf diese Weise verwenden wollen, müssen Sie vor dem Variablennamen ein Ausrufezeichen (!) eingeben. Um zum Beispiel den Wert der Variablen `abc` als Antwort auf eine Eingabeaufforderung zu verwenden, die als Wert eine reelle Zahl erwartet, geben Sie **!abc** ein.

Spaltenabstand: **!abc**

Wenn in Ihrer Zeichnung beispielsweise ein Layer mit dem Namen AUSSENWÄNDE vorhanden ist, den Sie mit dem Befehl LAYER aktuell machen wollen, könnten Sie den Variablennamen anstelle des vollständigen Layer-Namens eingeben.

Befehl: **layer**

?/Mach/SEtzen/Neu/EIn/Aus/FArbe/Ltyp/FRieren/Tauen/SPerren/ENtsperren: **se**

Neuer aktueller Layer<0>: **!layr**

?/Mach/SEtzen/Neu/EIn/Aus/FArbe/Ltyp/FRieren/Tauen/SPerren/ENtsperren:

Befehl:

Sie können auf AutoCAD-Eingabeaufforderungen auch mit einem AutoLISP-Ausdruck antworten. Wenn Sie den gleichen Wert bei mehreren Eingabeaufforderungen verwenden wollen, gehen Sie folgendermaßen vor:

Winkel: **(setq win 47.338)**

Diese Eingabe bewirkt folgendes: Sie weist der Variablen `win` den Wert 47.338 zu und gibt den Wert an die Eingabeaufforderung Winkel zurück. Wenn Sie das nächste Mal diesen Wert verwenden wollen, brauchen Sie nur **!win** einzugeben.

Da normale Zeichenketten mit den Zeichen "!" oder "(" beginnen können, müssen Sie die Systemvariable TEX TEVAL auf 1 setzen, damit die Zeichenkette der Variablen oder des Ausdrucks an die Befehle TEXT und ATTDEF übergeben werden. Sie können keine Variablen oder Ausdrücke verwenden, um Zeichenfolgen an den Befehl DTEXT oder in Dialogfeldern zu übergeben. Ebenso ist auch die Aktivierung eines AutoCAD-Befehls durch eine Variable nicht möglich. Wenn Sie zum Beispiel die Variable `x` auf die Zeichenfolge `"Linie"` setzen und dann als Antwort auf die AutoCAD-Eingabeaufforderung **!x** eingeben, zeigt AutoCAD den Wert `"Linie"` an; der Befehl LINIE wird nicht ausgeführt. Mit der Funktion **`command`** können Sie AutoCAD-Befehle von AutoLISP-Ausdrücken aus ausführen. Siehe **`"command"`** in chapter 13.

7.3.2 Vordefinierte Variablen

AutoCAD stellt Ihnen drei vordefinierte Variablen zur Verfügung, die Sie in AutoLISP-Anwendungen verwenden können.

`PAUSE`

Definiert als Zeichenkette, die aus einem einzelnen umgekehrten Schrägstrich (\) besteht. Diese Variable wird zusammen mit der Funktion **`command`** verwendet, um auf eine Benutzereingabe zu warten.

`PI`

Definiert als Konstante π (pi). Sie besitzt etwa den Wert 3.1415926.

`T`

Definiert als Konstante `T`. Sie wird als Wert ungleich `nil` verwendet.

Anmerkung Sie können die Werte dieser Variablen mit der Funktion **`setq`** ändern. Andere Anwendungen könnten jedoch darauf angewiesen sein, daß ihre Werte konstant bleiben, Sie sollten diese Variablen deshalb nicht ändern.

7.4 Zahlenbearbeitung

AutoLISP verfügt über die folgenden Funktionen, mit denen Sie Ganzzahlen und reelle Zahlen bearbeiten können:

+ (Addition)	abs	gcd	minusp
- (Subtraktion)	atan	log	rem
* (Multiplikation)	cos	logand	sin
/ (Division)	exp	logior	sqrt
~ (bitweises NOT)	expt	lsh	zerop
1+ (Inkrementieren)	fix	max	
1- (Dekrementieren)	float	min	

In Kapitel 13, sind alle Funktionen zur Bearbeitung von Zahlen einzeln beschrieben.

Sie können mit den Funktionen zur Bearbeitung von Zahlen nicht nur komplexe mathematische Berechnungen in Anwendungen durchführen, sondern sie auch bei Ihrer täglichen Arbeit mit AutoCAD verwenden. Wenn Sie zum Beispiel eine Detailzeichnung von einer Stahlverbindung mit einer 2.5-Zoll-Schraube anfertigen, die einen Durchmesser von 0.5 Zoll hat, wieviele Windungen hat diese Schraube dann insgesamt, wenn sie pro Zoll 13 Windungen aufweist? Verwenden Sie die Funktion **`*`** (Multiplikation) an der Eingabeaufforderung.

Befehl: **(* 2.5 13)**

32.5

Die arithmetischen Funktion mit einer *Zahl* als Argument (im Gegensatz zu denen mit beispielsweise *num* oder *Winkel*) geben, je nachdem, ob Sie Ganzzahlen oder reelle Zahlen als Argumente übergeben, unterschiedliche Werte zurück. Sind alle Argumente Ganzzahlen, ist auch der Rückgabewert eine Ganzzahl. Wenn jedoch mindestens ein Argument eine reelle Zahl ist, wird eine reelle Zahl zurückgegeben. Um sicherzustellen, daß Ihre Anwendung reelle Werte übergibt, müssen Sie mindestens ein Argument als reelle Zahl angeben.

Befehl: **(/ 12 5)**

2

Befehl: **(/ 12.0 5)**

2.4

AutoLISP verwendet zwar intern 32-Bit-Werte, aber die zwischen AutoLISP und AutoCAD übertragenen Ganzzahlwerte sind auf 16 Bit beschränkt (daher können Sie keinen Wert an AutoCAD übergeben, der größer als +32767 oder kleiner als -32768 ist). Wenn Sie Werte verwenden müssen, die über diesen Bereich hinausgehen, konvertieren Sie sie mit der Funktion **float** in eine reelle Zahl, weil diese als 32-Bit-Werte übergeben werden.

Befehl: **(setq 32bit (float (* 3 30000)))**

90000.0

7.5 Zeichenkettenbearbeitung

AutoLISP verfügt über Funktionen für die Arbeit mit Zeichenketten. Es gibt folgende Funktionen zur Bearbeitung von Zeichenketten:

strcase	strlen	wcmatch
strcat	substr	

Mit den folgenden Funktionen können Sie Zeichenketten in numerische Werte und numerische Werte in Zeichenketten konvertieren. Sie werden in "Konvertierungen" erläutert.

Angtof	atof	distof
angtos	atoi	itoa
ascii	chr	rtos

Die folgenden Funktionen zeigen die Werte von Zeichenketten in der Befehlszeile an. Sie werden in "Anzeigesteuerung" erläutert.

prin1	print
(princ)	prompt

Die Funktion **strcase** wandelt alle alphabetischen Zeichen in einer Zeichenkette in Groß- oder in Kleinbuchstaben um. Sie nimmt zwei Argumente: eine Zeichenkette und ein optionales Argument, das festlegt, ob die Zeichen groß- oder kleingeschrieben zurückgegeben werden. Wenn das optionale zweite Argument nicht angegeben wird, erhält es den Wert `nil`, und **strcase** liefert die Zeichen in Großschreibung zurück.

Befehl: **(strcase "Dies ist ein TEST.")**

"DIES IST EIN TEST."

Wenn das zweite Argument angegeben wird und einen Wert ungleich `nil` besitzt, werden die Zeichen in Kleinschreibung ausgegeben. AutoLISP stellt Ihnen für solche Situationen, in denen ein Wert ungleich `nil` als eine Art Wahr/Falsch-Umschalter verwendet wird, die vordefinierte Variable `T` zur Verfügung. Sie müssen `T` hier jedoch nicht verwenden; wenn dies für Ihre Anwendung besser geeignet ist, können sie einen beliebigen Wert ungleich `nil` verwenden.

Befehl: **(strcase "Dies ist ein TEST." T)**

"dies ist ein test."

Die Funktion **strcat** fügt mehrere Zeichenketten zu einer einzigen Zeichenkette zusammen. Sie ist nützlich, wenn Sie eine variable Zeichenkette in eine konstante Zeichenkette einfügen möchten. Der folgende Code weist einer Variablen eine Zeichenkette zu und fügt diese dann mit Hilfe der Funktion **strcat** in die Mitte einer anderen Zeichenkette ein.

Befehl: **(setq str "GROSSER") (setq bigstr (strcat "Dies ist ein " str " Test."))**
"Dies ist ein GROSSER Test."

Eine einzelne Zeichenkette darf nicht mehr als 132 Zeichen enthalten. Sie können aber Zeichenketten beliebiger Länge erstellen, indem Sie mehrere Zeichenketten mit der Funktion **strcat** zusammenfügen.

Wenn die Variable `bigstr` auf die vorhergehende Zeichenketten gesetzt wird, können Sie mit Hilfe der Funktion **strlen** die Anzahl der Zeichen (einschließlich Leerzeichen) in dieser Zeichenkette ermitteln.

Befehl: **(strlen bigstr)**
19

Die Funktion **substr** liefert eine Teilkette einer Zeichenkette. Sie besitzt zwei Argumente, die angegeben werden müssen, sowie ein optionales Argument. Das erste Argument ist die Zeichenkette. Das zweite Argument ist eine Ganzzahl, die das erste Zeichen der Zeichenkette bestimmt, das in die Teilkette aufgenommen wird. Wenn ein drittes Argument angegeben wird, legt es die Anzahl der Zeichen in der Teilkette fest. Ohne das dritte Argument gibt **substr** alle Zeichen, beginnend mit dem angegebenen ersten Zeichen, bis zum Ende der Zeichenkette zurück.

Sie können zum Beispiel die Funktion **substr** verwenden, um aus einem Dateinamen die aus drei Buchstaben bestehende Erweiterung zu entfernen. Weisen Sie zuerst einer Variablen einen Dateinamen zu. (Je nach Ihrer Anwendung würden Sie dazu normalerweise eine Systemvariable abfragen oder eine Benutzereingabe anfordern.)

Befehl: **(setq Dateinam "bigfile.txt")**
"bigfile.txt"

Sie benötigen eine Zeichenkette, die alle Zeichen außer den vier letzten (den Punkt und die Dateinamenerweiterung) enthält. Ermitteln Sie zuerst mit Hilfe der Funktion **strlen** die Länge der Zeichenkette und ziehen von diesem Wert vier ab. Legen Sie dann mit der Funktion **substr** das erste Zeichen und die Länge der Teilkette fest.

Befehl: **(setq newlen (- (strlen Dateinam) 4))**
7
Befehl: **(substr Dateinam 1 newlen)**
"bigdat"

Wenn Sie in Ihrer Anwendung den Wert der Variablen `newlen` danach nicht mehr benötigen, können Sie die beiden Codezeilen auch zu einer einzigen zusammenfassen.

Befehl: **(substr Dateinam 1 (- (strlen Dateinam) 4))**
"bigdat"

7.5.1 Steuerzeichen in Zeichenketten

Innerhalb von Zeichenketten in Anführungszeichen können Sie mit Hilfe des umgekehrten Schrägstrichs (`\`) Steuerzeichen (oder Escape-Codes) in eine Zeichenkette einfügen. In der folgenden Tabelle sind die derzeit zulässigen Steuerzeichen aufgeführt.

Steuerzeichen

Code	Beschreibung
`\\`	\-Zeichen
`\"`	"-Zeichen
`\e`	Escape-Zeichen
`\n`	Zeilenvorschubzeichen
`\r`	Wagenrücklauf-Zeichen
`\t`	Tab-Zeichen
`\`*nnn*	Zeichen, dessen Oktalcode *nnn* entspricht
`\U+`*xxxx*	Unicode-Zeichenfolge

`\M+nxxxx` Multibyte-Zeichenfolge

Die Funktion **prompt** erweitert die Steuerzeichen und zeigt sie in der Befehlszeile an. Die Funktion **prompt** akzeptiert eine Zeichenkette als einziges Argument, gibt sie in der Befehlszeile aus und gibt `nil` zurück. Alle Steuerzeichen in der Zeichenkette werden erweitert. Siehe **"prompt"** in Kapitel 13.

Wenn Sie den umgekehrten Schrägstrich (`\`) oder das Anführungszeichen (`"`) in einer in Anführungszeichen gesetzten Zeichenkette verwenden wollen, müssen Sie einen umgekehrten Schrägstrich (`\`) voranstellen.

Befehl: **(prompt "Der \"Dateiname\" ist: D:\\ACAD\\TEST.TXT. ")**

Der "Dateiname" ist: D:\ACAD\TEST.TXT. nil

Mit Hilfe des Steuerzeichens (`\e`) können Sie Escape-Sequenzen an das Betriebssystem senden, um die Grafikanzeige zu ändern, die Cursorbewegungen zu steuern und die Tasten neu zuzuordnen. In der Dokumentation Ihres Betriebssystems finden Sie Informationen darüber, welche Escape-Sequenzen Sie verwenden können. Bei DOS-Systemen werden Escape-Sequenzen oft benutzt, um die Tastenzuordnungen zu ändern. Der folgende Code verwendet die Funktion **prompt**, um der Taste F12 die Zeichenfolge **qsave** RETURN zuzuordnen.

Befehl: **(prompt "\e[0;134;'qsave';13p")**

nil

Die Zeichenkette im vorhergehenden Beispiel besteht im wesentlichen aus vier Komponenten. Mit **\e[** wird DOS mitgeteilt, daß eine Escape-Sequenz folgt, und der Code **0;134;** gibt die Taste F12 an. Wenn die Taste F12 gedrückt wird, entspricht der Abschnitt **'qsave'**; der an die Eingabeaufforderung übergebenen Zeichenkette, und **13p** gibt an, daß übergeben wird und der Zeichenkette folgt. Wenn die Zeichenkette, die der Eingabeaufforderung zugeführt wird, in Hochkomma gesetzt und mit einem Strichpunkt abgeschlossen wird, können Sie anstelle von **qsave** jede gültige Befehlsfolge verwenden.

Mit dem Zeilenvorschubzeichen (`\n`) können Sie an einer bestimmten Stelle in einer Zeichenkette einen Zeilenumbruch durchführen.

Befehl: **(prompt "Ein Beispiel für das \nZeilenvorschubzeichen. ")**

Ein Beispiel für das

Zeilenvorschubzeichen. nil

Mit Hilfe des Wagenrücklaufzeichens (`\r`) wird die Ausgabe am Anfang der aktuellen Zeile fortgesetzt. Dies ist besonders für die Anzeige inkrementeller Informationen (wie beispielsweise die Anzahl der bereits verarbeiteten Objekte) nützlich.

AutoLISP gibt nach der Auswertung eines Ausdrucks den Wert des letzten Ausdrucks zurück. Weil die Funktion **prompt** den Rückgabewert `nil` liefert, geben die vorhergehenden Beispiele nach den erweiterten Zeichenketten den Wert `nil` aus. Die Funktion **princ** zeigt auch eine Zeichenkette in der Befehlszeile an, gibt aber statt `nil` den Ausdruck (in diesem Fall eine Zeichenkette) zurück. Die Funktion **princ** unterscheidet sich auch dadurch von **prompt**, daß ihr Argument ein anderer Datentyp als eine Zeichenkette sein kann und optional ist. Wenn Ihr AutoLISP-Ausdruck also mit einem Aufruf von **princ** ohne Argumente endet, sehen Sie kein abschließendes `nil` auf dem Bildschirm (weil er nichts zurückgeben kann). Diesen Vorgang nennt man *Beenden ohne Ausgabe*.

Mit dem Tabulatorzeichen (\t) können Sie Text in Zeichenketten einrücken oder in Spalten anordnen. Beachten Sie, wie in diesem Beispiel durch den Aufruf der Funktion **princ** das abschließende `nil` nil unterdrückt wird.

Befehl: **(prompt "\nName\tBüro\n- - - - -\t- - - - -**

1> **\nGabi\t101\nDirk\t102\nTheo\t103\n") (princ)**

Name	Büro
Gabi	101
Dirk	102
Theo	103

Sie können auch Zeichen verwenden, die nicht auf der Standardtastatur vorhanden sind. Geben Sie dazu das Steuerzeichen *nnn* an, wobei *nnn* der oktale Code des anzuzeigenden Zeichens ist. Eine Liste der ASCII-Standardzeichen (mit den dezimalen Codes 0 bis 127), zusammen mit ihren dezimalen, oktalen und hexadezimalen Codes, finden Sie in Anhang A, "ASCII-Codes",. Die Codes der erweiterten Zeichen (mit den dezimalen Codes 128 bis 256) sind von Ihrem installierten Zeichensatz abhängig. Die erweiterten Zeichen variieren von Platform zu Platform und auch zwischen verschiedenen Anwendungen und Geräten auf der gleichen Platform. Unter "ASCII-Code-Konvertierung" finden Sie ein Anwendungsbeispiel, das alle Zeichen Ihres Systems in einem AutoCAD-Textfenster darstellt und diese in eine Datei schreibt.

Wenn Sie in Ihrem Grafikfenster die Schrift Courier verwenden, können Sie mit folgendem Code das Symbol "±" anzeigen:

Befehl: **(prompt "Der Wert ist \261 7001. ") (princ)**

Der Wert ist ± 7001.

7.5.2 Platzhaltersuche

Mit Hilfe der Funktion **`wcmatch`** können Sie in Ihrer Anwendung eine Zeichenkette mit einem Platzhaltermuster vergleichen. Sie können sie verwenden, wenn Sie einen Auswahlsatz erstellen (mit **`ssget`**) und wenn Sie erweiterte Objektdaten durch Anwendungsnamen abrufen **`entget`**).

Die Funktion **`wcmatch`** vergleicht eine einzelne Zeichenkette mit einem Muster. Sie gibt `T` zurück, wenn die Zeichenkette mit dem Muster übereinstimmt, und `nil` wenn das nicht der Fall ist. Platzhaltermuster gleichen regulären Ausdrücken, die von vielen Systemen und Anwendungsprogrammen verwendet werden. In dem Muster werden Buchstaben und Ziffern als solche behandelt; eckige Klammern können verwendet werden, um optionale Zeichen anzugeben oder Bereiche von Buchstaben bzw. Ziffern festzulegen; ein Fragezeichen (?) entspricht einem einzelnen Zeichen, ein Sternchen (`*`) mehreren aufeinanderfolgenden Zeichen; weitere spezielle Zeichen besitzen innerhalb des Musters eine besondere Bedeutung. Wenn Sie das Zeichen `*` am Anfang und am Ende des Suchmusters angeben, können sich die gesuchten Zeichen überall in der Zeichenkette befinden. Weitere Information finden Sie unter **`"wcmatch"`** in chapter 13.

In den folgenden Beispielen wird zuerst die Zeichenkette `matchme` deklariert und initialisiert:

Befehl: **(setq matchme "dies ist eine Zeichenkette - Test1 Test2 Ende")**

"dies ist eine Zeichenkette - Test1 Test2 Ende"

Mit dem folgenden Code wird geprüft, ob `matchme` mit den vier Zeichen `"dies"` beginnt:

Befehl: **(wcmatch matchme "dies*")**

T

Das folgende Beispiel zeigt Ihnen, wie Sie Klammern in einem Suchmuster verwenden können. In diesem Fall gibt **`wcmatch`** `T` zurück, wenn `matchme` `"Test4"`, `"Test5"`, `"Test6"` oder `"Test9"` enthält (beachten Sie die Verwendung des Zeichens `*`):

Befehl: **(wcmatch matchme "*Test[4-69]*")**

nil

Aber

Befehl: **(wcmatch matchme "*Test[4-61]*")**

T

weil die Zeichenkette `"Test1"` enthält.

Sie können auch mehrere Suchmuster, durch Kommas getrennt, angeben. Der folgende Code gibt `T` zurück, wenn `matchme` `"ABC"` entspricht oder mit `"XYZ"` *beginnt* oder mit `"end"` *endet*.

Befehl: **(wcmatch matchme "ABC,XYZ*,*end")**
T

7.6 Gleichheit und Bedingung

AutoLISP verfügt über die folgenden Funktionen, die eine Überprüfung auf Gleichheit, sowie bedingte Verzweigungen und Schleifen ermöglichen:

= (ist gleich)	and	or
/= (ungleich)	Boole	repeat
< (kleiner als)	cond	while
<= (kleiner oder gleich)	eq	
> (größer als)	equal	
>= (größer oder gleich)	if	

Die einzelnen Gleichheits- und Bedingungsfunktionen werden in Kapitel 13, "AutoLISP Funktionskatalog" beschrieben.

7.7 Listenbearbeitung

AutoLISP verfügt über Funktionen für die Arbeit mit Listen. Es gibt folgende Funktionen zur Listenbearbeitung:

append	foreach	listp	reverse
assoc	last	mapcar	subst
car und cdr	length	member	
cons	list	nth	

Der vorliegende Abschnitt enthält Beispiele für die Funktionen **`append`**, **`assoc`**, **`car`**, **`cons`**, **`list`**, **`nth`** und **`subst`**. In Kapitel 13, "AutoLISP Funktionskatalog". sind die einzelnen Funktionen zur Bearbeitung von Listen beschrieben.

Mit Hilfe von Listen können Sie mehrere verknüpfte Werte effizient speichern. Verschiedene AutoLISP-Funktionen dienen als Grundlage für die Programmierung zwei- und dreidimensionaler Grafikanwendungen. Diese Funktionen geben Punktwerte in Form von Listen zurück.

Die Funktion **`list`** stellt eine einfache Methode zur Gruppierung verbundener Objekte dar. Diese Objekte können verschiedene Datentypen haben. Der folgende Code faßt drei verbundene Objekte als Liste zusammen.

Befehl: **(setq lst1 (list 1.0 "Eins" 1))**
(1.0 "Eins" 1)

Sie können ein bestimmtes Objekt aus dieser Liste mit der Funktion **`nth`** aufrufen. Die Funktion **`nth`** akzeptiert zwei Argumente. Das erste Argument ist eine Ganzzahl, die festlegt, welches Objekt zurückgegeben wird. Eine 0 legt das erste Objekt in einer Liste fest, eine 1 das zweite Objekt und so weiter. Das zweite Argument ist die Liste selbst. Der folgende Code gibt das zweite Objekt in der Liste `lst1` zurück.

Befehl: **(nth 1 lst1)**
"Eins")

Die Funktionen **`car`** und **`cdr`** stellen eine andere Methode dar, Objekte aus einer Liste aufzurufen. Beispiele für die Verwendung von **`car`** und **`cdr`** finden Sie im folgenden Abschnitt, "Punktlisten"

Es gibt drei Funktionen, mit denen Sie eine existierende Liste modifizieren können. Die Funktion **`append`** gibt eine Liste mit neuen Objekten zurück, die an das Ende der existierenden Liste angefügt wurden, und die Funktion **`cons`** gibt eine Liste mit neuen Objekten zurück, die an den Anfang der existierenden Liste angefügt wurden. Die Funktion **`subst`** gibt eine Liste zurück, in der ein neues Objekt jedes erscheinende alte Objekt ersetzt. Diese Funktionen modifizieren die ursprüngliche Liste nicht, sie geben eine geänderte Liste zurück. Wenn Sie die ursprüngliche Liste modifizieren möchten, müssen Sie die alte Liste explizit durch die neue Liste ersetzen.

Die Funktion **`append`** faßt eine beliebige Anzahl von Listen in einer Liste zusammen. Aus diesem Grund müssen alle Argumente für diese Funktion Listen sein. Der folgende Code fügt ein weiteres `"Eins"` an die Liste `lst1` an. Beachten

Sie, daß die Verwendung der Funktion **quote** (oder **'**) eine einfache Methode ist, die Zeichenkette "Eins" zu einer Liste zu machen.

Befehl: **(setq lst2 (append lst1 '("Eins")))**
(1.0 "Eins" 1 "Eins")

Die Funktion **cons** kombiniert ein einzelnes Element mit einer Liste. Sie können mit der Funktion **cons** eine weitere Zeichenkette `"Eins"` an den Anfang dieser neuen Liste, `lst2`, anfügen.

Befehl: **(setq lst3 (cons "Eins" lst2))**
("Eins" 1.0 "Eins" 1 "Eins")

Mit der Funktion **substr** können Sie jedes in einer Liste erscheinende Objekt durch ein neues Objekt ersetzen. Der folgende Code ersetzt alle Zeichenketten `"Eins"` durch die Zeichenkette `"eins"`.

Befehl: **(setq lst4 (subst "eins" "Eins" lst3))**
("eins" 1.0 "eins" 1 "eins")

7.7.1 Punktlisten

AutoLISP hält sich bei der Behandlung von Grafikkoordinaten an die folgenden Konventionen. Punkte werden als *Listen* von zwei oder drei in Klammern eingeschlossene Zahlen ausgedrückt.

2D-Punkte

Ausgedrückt als Listen mit zwei reellen Zahlen (*X* bzw. *Y*), wie bei

```
(3.4 7.52)
```

3D-Punkte

Ausgedrückt als Listen mit drei reellen Zahlen (*X Y* und Z), wie bei

```
(3.4 7.52 1.0)
```

Punktvariablen enthalten *X*, *Y* und (gegebenenfalls) *Z*-Komponenten. Das erste Element in der Liste ist die *X*-Komponente des Punkts, das zweite seine Y-Komponente und das dritte (falls vorhanden) seine *Z*-Komponente. Sie können derartige Listen mit der Funktion **list** erstellen.

Befehl: **(list 3.875 1.23)**
(3.875 1.23)
Befehl: **(list 88.0 14.77 3.14)**
(88.0 14.77 3.14)

Wenn Sie einer Punktvariablen bestimmte Koordinaten zuweisen wollen, können Sie einen der folgenden Ausdrücke verwenden:

Befehl: **(setq pt1 (list 3.875 1.23)**
1> **pt2 (list 88.0 14.77 3.14)**
1> **abc 3.45**
1> **pt3 (list abc 1.23))**
(3.45 1.23)

Der letzte Ausdruck verwendet den Wert der Variablen `abc` als *X*-Komponente des Punkts.

Wenn alle Elemente einer Liste konstante Werte sind, können Sie statt **list** die Funktion **quote** verwenden, um die Liste explizit zu definieren. Die Funktion **quote** gibt einen nicht ausgewerteten Ausdruck zurück.

Befehl: **(setq pt1 (quote (4.5 7.5)))**
(4.7 7.5)

Als Abkürzung für die Funktion quote können Sie ein Hochkomma (') angeben. Der folgende Code führt zum gleichen Ergebnis wie das vorhergehende Beispiel.

Befehl: **(setq pt1 '(4.5 7.5))**
(4.7 7.5)

Sie können auch mit den drei Standardfunktionen **car**, **cadr** und **caddr** einzeln auf die Komponenten *X, Y* und *Z* eines Punkts zugreifen. Die folgenden Beispiele zeigen, wie die Koordinaten *X*, *Y* und *Z* aus einer 3D-Punktliste extrahiert werden. Die Variable `pt` wird auf den Punkt (1.5 3.2 2.0) gesetzt.

Befehl: **(setq pt '(1.5 3.2 2.0))**
(1.5 3.2 2.0)

Die Funktion **car** liefert das erste Element einer Liste. In diesem Beispiel wird der Variablen `x_val` der *X*-Wert des Punkts `pt` zurückgegeben.

Befehl: **(setq x_val (car pt))**
1.5

Die Funktion **cadr** liefert das zweite Element einer Liste. In diesem Beispiel wird der Variablen `y_val` der *Y*-Wert des Punkts `pt` zurückgegeben.

Befehl: **(setq y_val (cadr pt))**
3.2

Die Funktion **caddr** liefert das dritte Element einer Liste. In diesem Beispiel wird der Variablen `z_val` der *Z*-Wert des Punkts `pt` zurückgegeben.

Befehl: **(setq z_val (caddr pt))**
2.0

Sie können mit folgendem Code die untere linke und obere rechte Ecke (`pt1` und `pt2`) eines Rechtecks definieren.

Befehl: **(setq pt1 '(1.0 2.0) pt2 ' (3.0 4.0))**
(3.0 4.0)

Sie können dann mit Hilfe der Funktionen **car** und **cadr** die Variable `pt3` auf die obere linke Ecke des Rechtecks setzen, indem Sie die X-Komponente von `pt1` und die Y-Komponente von `pt2` extrahieren.

Befehl: **(setq pt3 (list (car pt1) (cadr pt2)))**
(1.0 4.0)

Der vorhergehende Ausdruck setzt `pt3` dem Punkt (1.0,4.0) gleich.

7.7.2 Punktierte Paare

Eine andere Methode, die AutoCAD verwendet, um Daten zu organisieren, erfolgt mit einer speziellen Art von Liste, *punktiertes Paar* genannt. Ein punktiertes Paar ist eine Liste, die immer zwei Teile enthalten muß. Wenn AutoLISP ein punktiertes Paar anzeigt, fügt es einen Punkt zwischen den einzelnen Teilen der Liste ein. Die meisten Funktionen zur Bearbeitung von Listen akzeptieren keine punktierten Paare als Argument, aus diesem Grund sollten Sie sich versichern, daß Sie einer Funktion die richtige Liste zuführen.

Die Funktion **cons** kann nicht nur ein Objekt an den Anfang einer Liste anfügen, sie ist auch in der Lage, ein punktiertes Paar zu erstellen. Wenn das zweite Argument der Funktion **cons** ein Atom ist (ein konstanter oder quotierter Wert), erstellt die Funktion ein punktiertes Paar.

Befehl: **(setq d1 (cons 1 "Eins"))**
(1 . "Eins")

Die Funktionen **car**, **cdr** und **assoc** sind zur Bearbeitung von punktierten Paaren nützlich. Der folgende Code erstellt eine *Assoziationsliste*. Eine Assoziationsliste ist eine Liste von Listen sowie die Methode, die AutoCAD zur Aufbewahrung von Elementdefinitionsdaten verwendet (Elementdefinitionsdaten werden in Kapitel 9, "Auswahlsatz-, Objekt- und Symboltabellen funktionen" erläutert). Der folgende Code erstellt eine Assoziationsliste von punktierten Paaren.

Befehl: **(setq d2 (list d1 (cons 2 "Zwei")(cons 3 "Drei")))**
((1 . "Eins") (2 . "Zwei") (3 . "Drei"))

Die Funktion **assoc** gibt eine festgelegte Liste aus einer Assoziationsliste zurück, unabhängig von der Position der festgelegten Liste in der Assoziationsliste. Die Funktion **assoc** sucht in den Listen nach einem bestimmten Schlüsselelement.

Befehl: **(assoc 2 d2)**

(2 . "Zwei")

7.8 Symbol- und Funktionsbearbeitung

AutoLISP stellt Ihnen die Funktionen für die Bearbeitung von Symbolen und Variablen zur Verfügung. Es gibt folgende Funktionen zur Symbolbearbeitung:

atom	not	quote	type
atoms-family	null	set	
boundp	numberp	setq	

In Kapitel 13, "AutoLISP Funktionskatalog".werden alle Funktionen zur Bearbeitung von Symbolen einzeln beschrieben.

AutoLISP stellt Funktionen für die Bearbeitung einer oder mehrerer Funktionsgruppen zur Verfügung. Es gibt folgende Funktionen zur Funktionsbearbeitung:

apply	eval	progn	untrace
defun	lambda	trace	

Dieser Abschnitt zeigt Bespiele für die Funktion **`defun`**. In Kapitel 13, werden alle Funktionen zur Bearbeitung von Funktionen einzeln beschrieben.

Mit AutoLISP können Sie Ihre eigenen Funktionen definieren. Danach können Sie diese benutzerdefinierten Funktionen genauso wie Standardfunktionen in der Befehlszeile oder in anderen AutoLISP-Ausdrücken verwenden. Das gleiche ist mit AutoCAD-Befehlen möglich, weil Befehle nur eine spezielle Art von Funktionen sind.

Die Funktion **`defun`** (siehe **`"defun"`** in Kapitel 13,) erstellt aus einer Gruppe von Ausdrücken eine Funktion oder einen Befehl. Diese Funktion benötigt mindestens drei Argumente, das erste ist der Name der Funktion (Symbolname), die definiert werden soll. Das zweite ist die Argumentliste (eine Liste von Argumenten und lokalen Symbolen, die von der Funktion verwendet werden). Die Argumentliste kann `nil` oder eine leere Liste `( )` sein. Eine nähere Erläuterung von Argumentlisten finden Sie in "Funktionen mit Argumenten." Lokale Symbole werden vom Argument durch einen Schrägstrich (/) abgetrennt. Sie werden in "Lokale Symbole in Funktionen" erläutert. Nach diesen Argumenten folgen die Ausdrücke, aus denen die Funktion besteht; eine Funktionsdefinition muß mindestens einen Ausdruck enthalten.

```
(defun Symbolname ( Argumente / lokale_Symbole )
  Ausdrücke
)
```

Der folgende Code definiert eine einfache Funktion, die keine Argumente benötigt und eine Meldung in der Befehlszeile ausgibt. Beachten Sie, daß die Argumentliste als leere Liste (**()**) übergeben wird.

```
Befehl: (defun FERTIG ( ) (prompt "\nWiedersehen! "))
FERTIG
```

Die Funktion **`FERTIG`** ist jetzt definiert, und Sie können sie wie jede andere Funktion verwenden. Weil sie die Meldung Wiedersehen! in einer neuen Zeile in der Befehlszeile anzeigt, könnten Sie die Funktion **`FERTIG`** folgendermaßen verwenden:

```
Befehl: (prompt "Der Wert ist \127 ")(FERTIG)(princ)
Der Wert ist 127.
Wiedersehen!
```

Funktionen ohne Argumente erscheinen auf den ersten Blick nutzlos. Sie können mit ihnen aber den Status bestimmter Systemvariablen oder Bedingungen abfragen und einen Wert zurückgeben, aus dem der aktuelle Status ersichtlich ist.

Anmerkung Damit keine Konflikte mit nicht verbundenen Funktionen entstehen, sollten Sie das Funktionsnamenpräfix **`S::`**: : als ‚reserviert" ansehen. Verwenden Sie es nur für die spezielle Funktion **`S::STARTUP`**: Siehe "S::STARTUP-Funktion - Automatische Ausführung."

Sie können AutoCAD so konfigurieren, daß bei jeder neuen Sitzung Ihre selbstdefinierten Funktionen automatisch geladen werden. (Siehe "Automatisches Laden und Ausführen.")

7.8.1 C:XXX-Funktionen

Wenn eine Funktion mit dem Namen der Form **C:*XXX*** definiert ist, kann sie in der Befehlszeile auf die gleiche Weise wie ein integrierter AutoCAD-Befehl aufgerufen werden. Diese Eigenschaft können Sie verwenden, um neue Befehle zu AutoCAD hinzuzufügen oder um existierende Befehle neu zu definieren.

Damit Funktionen als AutoCAD-Befehle verwendet werden können, müssen sie folgenden Richtlinien entsprechen:

- Der Funktionsname muß in der Form **C:*XXX*** (Groß- oder Kleinbuchstaben) angegeben werden. Die Namenskomponente **C:** muß immer vorhanden sein, ***XXX*** kann ein beliebiger Befehlsname sein. **C:*XXX***-Funktionen können als Ersatz für AutoCAD-Standardbefehle verwendet werden. (Siehe "Umdefinieren von AutoCAD-Befehlen.")

Anmerkung In diesem Fall ist **C:** kein Verweis auf ein Laufwerk. Es handelt sich um spezielles Präfix, das eine Befehlszeilenfunktion bezeichnet.

- Die Funktion muß ohne Argumente definiert werden. (Lokale Symbole sind erlaubt.)

Eine auf diese Weise definierte Funktion kann transparent von jeder Eingabeaufforderung aller AutoCAD-Standardbefehle aus aktiviert werden, wenn sie nicht die Funktion **command** aufruft. Wenn Sie einen mit **C:*XXX***definierten Befehl transparent aufrufen wollen, müssen Sie der Namenskomponente ***XXX*** ein Hochkomma (') voranstellen.

Sie können einen Standardbefehl transparent aufrufen, während ein **C:*XXX*** Befehl aktiv ist, indem Sie ihm, wie allen transparent zu aktivierenden Befehlen, ein Hochkomma (') **voranstellen. Sie können jedoch keinen C:*XXX*** Befehl transparent aufrufen, wenn ein anderer **C:*XXX*** Befehl aktiv ist. Wenn Sie eine als Befehl definierte Funktion von einer anderen AutoLISP-Funktion aus aufrufen, müssen Sie den ganzen Namen angeben-beispielsweise **(C:HELLO)**.

Wenn Sie nach der Funktionsdefinition die Funktion **setfunhelp** aufrufen, können Sie eine Hilfedatei und ein Hilfethema mit einem benutzerdefinierten Befehl verknüpfen. Ruft ein Benutzer die Hilfe transparent (**'help**) an einer Eingabeaufforderung innerhalb des benutzerdefinierten Befehls auf, wird das Thema, das mit der Funktion setfunhelp festgelegt wurde, angezeigt.

Sie können normalerweise keinen AutoLISP-Ausdruck als Antwort auf Eingabeaufforderungen eines AutoLISP-implementierten Befehls verwenden. Wenn Sie aber in Ihrer AutoLISP-Routine die Funktion **initget** verwenden, können Sie beliebige Tastatureingaben zusammen mit bestimmten Funktionen verwenden. Dadurch kann ein in AutoLISP-implementierter Befehl einen AutoLISP-Ausdruck als Antwort akzeptieren. Ebenso können die von einem DIESEL-Ausdruck zurückgelieferten Werte Auswertungen der aktuellen Zeichnung durchführen und das Ergebnis an AutoLISP zurückgeben. Informationen über die DIESEL-Makroprogrammiersprache finden Sie in Kapitel 5, 3

7.8.1.1 Hinzufügen von Befehlen

Wenn Sie **C:*XXX verwenden, können Sie beispielsweise einen Befehl definieren, der eine einfache Meldung ausgibt.***

Befehl: **(defun C:HALLO () (prompt "Hallo allerseits. \n") (princ))**
C:HALLO

HALLO ist jetzt als Befehl definiert.

Befehl: **hallo**
Hallo allerseits.

Dieser neue Befehl kann transparent angezeigt werden, weil er die Funktion **command** nicht aufruft.

Befehl: **linie**
Von Punkt: **'hallo**
Hallo allerseits.
Von Punkt:

7.8.1.2 Umdefinieren von AutoCAD-Befehlen

Mit AutoLISP, externen Befehlen und der Alias-Funktion können Sie eigene AutoCAD-Befehle definieren. Mit dem Befehl BFLÖSCH können Sie einen AutoCAD-Standardbefehl in einen gleichnamigen benutzerdefinierten Befehl umdefinieren. Später können Sie mit dem Befehl BFRÜCK die Standarddefinition wiederherstellen. Der Befehl BFLÖSCH ist nur für die aktuelle Bearbeitungssitzung wirksam.

Sie können einen entdefinierten Befehl jederzeit aktivieren, indem Sie seinen richtigen Namen (Befehlsname mit vorangestelltem Punkt) angeben. Wenn Sie zum Beispiel QUIT entdefinieren, können Sie durch Eingabe von **.quit** immer noch auf den Befehl zugreifen.

Sehen Sie sich das folgende Beispiel an. Angenommen, Sie wollen, daß AutoCAD Sie jedesmal an den Befehl PLINIE erinnert, wenn Sie den Befehl LINIE verwenden. Um die AutoLISP-Funktion **`C:LINIE`** so zu definieren, daß sie den normalen Befehl LINIE ersetzt, gehen Sie wie folgt vor:

```
Befehl: (defun C:LINIE ( )
1> (princ "Sollte ich nicht PLINIE verwenden?\n")
1> (command ".LINIE") (princ) )
C:LINIE
```

In diesem Beispiel gibt die Funktion **`C:LINIE`** ihre Meldung aus und führt dann den normalen LINIE-Befehl aus (unter Verwendung seines richtigen Namens, .LINIE. Damit AutoCAD die neue Definition für den Befehl LINIE benutzt, müssen Sie den Standardbefehl LINIE umdefinieren. (Die Reihenfolge, in der AutoCAD Befehlsnamen erkennt, ist in "Suchverfahren für Befehle" beschrieben.)

```
Befehl: bflösch
Befehlsname: linie
```

Wenn Sie jetzt an der Eingabeaufforderung **linie** eingeben, benutzt AutoCAD die AutoLISP-Funktion **`C:LINE`**:

```
Befehl: linie
Sollte ich nicht PLINIE verwenden?
.LINIE Von Punkt:
Von Punkt:
```

Das Codebeispiel nimmt an, daß die CMDECHO Systemvariable auf 1 (Ein) gesetzt ist. Wenn CMDECHO auf 0 (Aus) gesetzt ist, zeigt AutoCAD keine Eingabeaufforderungen an, die das Ergebnis eines **`command`** Funktionsaufrufs sind. Der folgende Code verwendet die Systemvariable CMDECHO, um die Anzeige von Eingabeaufforderungen zu kontrollieren.

```
Befehl: (defun C:LINIE ( )
1> (setvar "cmdecho" 0)
1> (princ "Sollte ich nicht PLINIE verwenden?\n")
1> (command ".LINIE") (setvar "cmdecho" 1) (princ) )
C:LINIE
```

Sie könnten diese Eigenschaft zum Beispiel in einem Zeichnungsverwaltungssystem verwenden. Sie könnten die Befehle NEU, ÖFFNEN, QUIT und ENDE so umdefinieren, daß Rechnungsinformationen in eine Protokolldatei geschrieben werden, bevor Sie die Arbeitssitzung beenden.

Anmerkung Sie sollten Ihre Menüs, Skripts und AutoLISP-Programme schützen, indem Sie Befehle mit vorangestelltem Punkt verwenden. Dadurch wird sichergestellt, daß Ihre Anwendungen die integrierten und nicht die umdefinierten Befehle verwenden.

7.8.2 Lokale Symbole in Funktionen

AutoLISP ermöglicht Ihnen, eine Liste von Symbolen (Variablen) zu definieren, die nur für Ihre Funktion verfügbar sind. Diese Symbole werden als *lokale Symbole* bezeichnet. Diese lokalen Symbole stellen sicher, daß die Variablen in Ihren Funktionen nicht von anderen Teilen der Anwendung geändert werden können und daß sie nicht mehr verfügbar sind, nachdem die aufrufende Funktion ihre Aufgabe ausgeführt hat.

Viele benutzerdefinierte Funktionen werden in größeren Anwendungen als Hilfsfunktionen verwendet. Benutzerdefinierte Funktionen enthalten normalerweise auch Variablen, deren Werte und Verwendung nur für diese Funktionen von Bedeutung sind.

Das folgende Beispiel zeigt, wie lokale Symbole in einer benutzerdefinierten Funktion verwendet werden können (vergewissern Sie sich, daß sich mindestens ein Leerschritt zwischen dem Schrägstrich und den lokalen Symbolen befindet).

```
Befehl: (defun LOKAL ( / aaa bbb)
1> (setq aaa "A" bbb "B")
```

1> **(princ (strcat " \naaa hat den Wert " aaa))**

1> **(princ (strcat " \nbbb hat den Wert " bbb))**

1> **(princ)**

LOKAL

Bevor Sie die neue Funktion testen, weisen Sie den Variablen `aaa` und `bbb` andere Werte als die in der Funktion **`LOKAL`** verwendeten zu.

Befehl: **(setq aaa 1 bbb 2)**

2

Sie können mit Hilfe des Ausrufezeichens (!) überprüfen, ob die Variablen `aaa` und `bbb` tatsächlich auf diese Werte gesetzt sind.

Befehl: **!aaa**

1

Befehl: **!bbb**

2

Prüfen Sie jetzt die Funktion **`LOKAL`**.

Befehl: **(lokal)**

aaa hat den Wert A

bbb hat den Wert B

Sie sehen, daß die Funktion die lokalen Werte der Variablen `aaa` und `bbb` ausgegeben hat. Sie können auch das Ausrufezeichen (**!**) verwenden, um anzuzeigen, daß die aktuellen Werte von `aaa` und `bbb` immer noch die nicht lokalen sind.

Befehl: **!aaa**

1

Befehl: **!bbb**

2

Sie können durch dieses Vorgehen nicht nur sicherstellen, daß die Variablen in einer bestimmten Funktion lokal sind, sondern auch, daß der für die Variablen verwendete Speicher anschließend wieder freigegeben wird.

7.8.3 Funktionen mit Argumenten

Sie können mit AutoLISP Funktionen definieren, die Argumente akzeptieren. Im Gegensatz zu vielen AutoLISP-Standardfunktionen sind bei benutzerdefinierten Funktionen aber keine fakultativen Argumente möglich. Wenn Sie eine eigene Funktion mit Argumenten definieren, müssen Sie für jedes Argument einen Wert übergeben.

Die Symbole, die als Argumente verwendet werden sollen, werden in der Argumentliste vor den lokalen Symbolen festgelegt. Argumente werden als spezielle Art lokaler Symbole behandelt; Argumentvariablen sind außerhalb der Funktion nicht verfügbar. Sie dürfen in einer Funktionsdefinition nicht mehreren Argumenten den gleichen Namen geben.

Der folgende Code definiert eine Funktion, die als Argumente zwei Zeichenketten nimmt, verbindet sie mit einer weiteren Zeichenkette und gibt das Ergebnis zurück.

Befehl: **(defun ARGTEST (arg1 arg2 / ccc)**

1> **(setq ccc "Konstante Zeichenkette")**

1> **(strcat ccc ", " arg1 ", " arg2))**

ARGTEST

Diese Art von Funktion kann mehrmals in einer Anwendung verwendet werden, um zwei variable und eine konstante Zeichenkette in einer bestimmten Reihenfolge zusammenzufügen. Sie können den Rückgabewert in einer Variable speichern und danach in der Anwendung benutzen.

Befehl: **(setq newstr (ARGTEST "Zeichenkette 1" "Zeichenkette 2"))**

"Konstante Zeichenkette, Zeichenkette 1, Zeichenkette 2"

Die Variable `newstr` hat jetzt den Wert der drei zusammengefügten Zeichenketten.

7.9 Fehlerbehebung

AutoLISP stellt Funktionen zur Fehlerbehebung zur Verfügung. Es gibt folgende Funktionen zur Fehlerbehebung:

alert *error* exit quit

In Kapitel 13, "AutoLISP Funktionskatalog". wird jede Fehlerbehebungsfunktion einzeln beschrieben.

AutoLISP bietet Ihnen die Möglichkeit, Benutzer- oder Programmfehler zu behandeln. Die Funktion **`*error*`** stellt sicher, daß AutoCAD nach einem Fehler in einen bestimmten Status zurückkehrt. Mit Hilfe dieser benutzerdefinierten Funktion können Sie die Fehlerbedingung einschätzen und eine entsprechende Meldung an den Benutzer zurückgeben.

Wenn AutoLISP bei der Auswertung eines Ausdrucks auf einen Fehler trifft, gibt es eine Meldung im folgenden Format aus:

Fehler: Text

In dieser Meldung beschreibt *Text* den Fehler. Wenn die Funktion **`*error*`** definiert ist (ungleich `-nil`), zeigt AutoLISP keine Fehlermeldung an, sondern führt diese Funktion aus (und übergibt *Text* als einziges Argument). Wenn **`*error*`** nicht definiert oder `nil` ist, beendet AutoLISP die Auswertung und zeigt eine Rückverfolgung der aufrufenden Funktionen bis zu einer Tiefe von 100 Ebenen an. Es zahlt sich aus, diese Fehlerfunktion eingeschaltet zu lassen, während Sie Ihr Programm auf Fehler untersuchen.

In der AutoCAD-Systemvariablen ERRNO wird für den letzten Fehler ein Code gespeichert, den Sie mit der Funktion **`getvar`** abrufen können. Siehe auch Fehlermeldungen und Fehlercodes in Kapitel 16, "AutoLISP-Fehlercodes und Fehlermeldungen". Siehe **`"*error*"`** in Kapitel 13,.

Speichern Sie den aktuellen Inhalt von **`*error*`**, bevor Sie Ihre eigene Funktion **`*error*`** definieren, damit die vorhergehende Fehlerfunktion beim Beenden wiederhergestellt werden kann. Bei einem Fehler ruft AutoCAD die aktuell definierte Funktion **`*error*`** auf und übergibt ihr als einziges Argument eine Zeichenkette, die den Fehler beschreibt. Sie sollten Ihre Funktion **`*error*`** so gestalten, daß sie nach einem ESC (Abbrechen) oder einem Aufruf der Funktion **`exit`** ohne eine Ausgabe beendet. Nehmen Sie dazu die folgenden Ausdrücke in Ihre Fehlerroutine auf.

```
(if
  or
    (/= msg "Funktion abgebrochen")
    (= msg "beenden / schließen abbrechen")
  )
  (princ)
  (princ (strcat "\nFehler: " msg))
)
```

Dieser Code überprüft die übergebene Fehlermeldung und sorgt dafür, daß der Benutzer über den Ursprung des Fehlers informiert wird. Wenn der Benutzer die Routine abbricht, während sie läuft, wird nichts zurückgegeben. Es erfolgt auch keine Ausgabe, wenn eine Fehlerbedingung in Ihrem Code programmiert ist und die Funktion **`exit`** aufgerufen wird. Dabei wird vorausgesetzt, daß Sie bereits den Grund des Fehlers mit Hilfe von Ausgabeausdrücken beschrieben haben. Sie dürfen auch nicht vergessen, als letztes einen Aufruf der Funktion **`princ`** hinzuzufügen, wenn am Ende der Fehlerroutine kein Rückgabewert angezeigt werden soll.

Wenn Sie durch ZURÜCK erlauben, daß Ihre Routine in einem Schritt rückgängig gemacht werden kann, müssen Sie auch die entsprechenden ZURÜCK-Aufrufe angeben, die bei einem normalen Beenden der Routine aufgerufen würden. Sie können in AutoCAD wirklich alles rückgängig machen. Programme könen mit Hilfe der Systemvariablen UNDOCTL und UNDOMARKS festlegen, wie (oder ob) ZURÜCK aufgerufen werden soll.

Das größte Problem bei Fehlerroutinen ist, daß sie normale AutoLISP-Funktionen sind, die vom Benutzer abgebrochen werden können. Halten Sie sie so kurz und schnell wie möglich. Dies erhöht die Wahrscheinlichkeit, daß die ganze Routine ausgeführt wird, wenn sie aufgerufen wird.

Viele ADS- und ARX-definierten Befehle besitzen ihre eigenen Funktionen oder Systemvariablen, die Fehlerinformationen liefern, wie in Kapitel 14, "Zugriff auf extern definierte Befehle und Systemvariablen"., beschrieben.

Sie können Benutzer auch von einer Fehlerbedingung in Kenntnis setzen, indem Sie ein Warnfenster, ein kleines Dialogfeld mit einer Meldung, anzeigen. Rufen Sie dazu die Funktion **alert** auf.

Der folgende Aufruf von **alert** zeigt ein Warnfenster an.

```
(alert "Datei nicht gefunden")
```

Weitere Information finden Sie unter **"alert"** in Kapitel 13.

7.10 Anwendungsbehandlung

AutoLISP stellt Ihnen Funktionen für die Handhabung von Anwendungen und extern definierten Funktionen zur Verfügung: Es gibt folgende Funktionen zur Handhabung von Anwendungen:

ads	arxunload	autoxload	xload
arx	autoarxload	load	xunload
arxload	autoload	startapp	

Dieser Abschnitt enthält Beispiele für die Funktionen **load**, **arx** und **arxload**. Ein Beispiel für die Funktion **autoload** ist in "Automatisches Laden von Befehlen" enthalten. In "AutoLISP Funktionskatalog". in Kapitel 13, werden die einzelnen Funktionen zur Behandlung von Anwendungen beschrieben.

7.10.1 Laden von AutoLISP-Anwendungen

Sie können Funktionsdefinitionen in Dateien mit der Erweiterung *.lsp* speichern, mit der AutoLISP-Funktion **load** laden oder sie in eine *acad.lsp*-Datei oder eine Datei mit der Erweiterung *.mnl* aufnehmen, damit sie beim Starten von AutoCAD geladen werden. Beim Laden einer *.lsp*-Datei werden ihre Ausdrücke ausgewertet. Im allgemeinen verwendet eine *.lsp*-Datei die Funktion **defun** zum Speichern von Funktionsgruppen im Speicher des Computers, damit sie zu einem späteren Zeitpunkt ausgeführt werden können.

7.10.1.1 S::STARTUP-Funktion - Automatische Ausführung

Wenn sich die benutzerdefinierte Funktion **S::STARTUP** in der Datei *acad.lsp* oder in einer *.mnl*-Datei befindet, wird sie aufgerufen, wenn Sie eine neue Zeichnung eingeben oder eine vorhandene Zeichnung öffnen. Sie können also eine Funktion **defun** von **S::STARTUP** in Ihre Datei *acad.lsp* aufnehmen, damit alle notwendigen Konfigurationen durchgeführt werden.

Wenn sie die Standard-AutoCAD-Befehle QUIT und ENDE mit eigenen Versionen überschreiben wollen, benutzen Sie eine *acad.lsp*-Datei mit folgendem Inhalt:

```
(defun C:QUIT ( )
  ... Ihre Definition ...
)
(defun C:ENDE ( )
  ... Ihre Definition ...
)
(defun S::STARTUP ( )
  (command "bflösch" "quit")
  (command "bflösch" "ende")
)
```

Bevor die Zeichnung initialisiert wird, werden mit der Funktion **defun** neue Definitionen der Befehle QUIT und END festgelegt. Nach der Initialisierung wird die Funktion **S::STARTUP** aufgerufen, und die Standarddefinitionen von QUIT und END werden entdefiniert.

Da die Funktion **S::STARTUP** in vielen Dateien definiert werden kann (in einer *acad.lsp*-Datei, einer *.mnl*-Datei oder in einer anderen von diesen beiden Dateien geladenen AutoLISP-Datei), können Sie eine vorher definierte **S::STARTUP**-Funktion überschreiben. Das folgende Beispiel zeigt Ihnen eine Methode, wie Sie sicherstellen können, daß Ihre Startfunktion zusammen mit anderen Funktionen richtig funktioniert.

```
(defun MEINSTART ( )
  ... Ihre Startfunktion...
)
(setq S::STARTUP (append S::STARTUP MEINSTARTUP))
```

Der Code fügt Ihre Startfunktion an eine existierende **S::STARTUP**-Funktion an und definiert die **S::STARTUP**-Funktion so, daß sie Ihren Startcode enthält. Dies funktioniert unabhängig von dem früheren Vorhandensein einer **S::STARTUP**-Funktion.

7.10.1.2 Ladetechniken

AutoLISP-Programme liegen normalerweise als *.lsp*-Dateien vor, die mit der Funktion **load** geladen werden können. Die Funktion **load** erstellt nicht nur Funktionen, die von den Benutzern aufgerufen werden können, sondern ermöglicht es auch dem Programmierer, für das Programm Benutzerhinweise auszugeben. Weil AutoLISP die Ausdrücke in der Datei auswertet, können Sie durch einen Aufruf der Funktion **princ** (oder **print**, prin1**, o. ä.) beliebige Informationen in der Befehlszeile anzeigen. Sie können auch die normale Ausgabe von AutoLISP unterdrücken, indem Sie (princ)** in die letzte Zeile der Datei schreiben.

Das Laden einer Datei bietet noch andere Möglichkeiten für den Aufruf von AutoLISP-Funktionen. Jeder Code in einer *lsp*-Datei, der nicht zu einem **defun**-Ausdruck gehört, wird beim Laden der Datei automatisch ausgeführt. Dadurch können Sie zusätzlich zu der vorher beschriebenen Textausgabe bestimmte Parameter konfigurieren oder andere Initialisierungen durchführen.

7.10.2 Laden von ADS- und ARX-Anwendungen

Bevor Sie einen mit ADS- oder ARX-definierten Befehl oder eine Funktion aus einer AutoLISP-Anwendung aufrufen, sollten Sie überprüfen, ob der Befehl oder die Funktion vorhanden ist. Die Funktionen **ads** und **arx** geben eine Liste der aktuell geladenen ADS- und ARX-Anwendungen zurück.

Der folgende Beispielcode definiert die Funktion **INLIST**, die zwei Argumente akzeptiert. Das erste Argument, *lst*, ist eine Liste von Zeichenketten und das zweite Argument, *Zfolge*, ist eine Zeichenkette. Die Funktion **INLIST** gibt T zurück, wenn *Zfolge* einer der Zeichenketten in *lst* untergeordnet ist. Sie können diese Funktionsdefinition in Ihre Anwendung aufnehmen, damit Sie leichter überprüfen können, ob eine ADS- oder ARX-Anwendung geladen ist.

```
(defun INLIST ( lst Zkette / ct tmpstr ret)
  (setq ct 0 str (strcase str))
  (repeat (length lst)
    (setq tmpstr (strcase (nth ct lst)))
    (if (wcmatch tmpstr (strcat "*" Zkette "*"))
      setq ret T
    )
    (setq ct (1+ ct))
  )
  ret
)
```

Wenn die Funktion **INLIST** festgelegt ist, prüft der folgende Code, ob die ARX-Anwendung Render geladen ist. Ist Render nicht geladen, lädt der Code die Datei *render.arx*. Der Beispielcode enthält ebenfalls ein Argument *beiFehler* (ähnlich dem von der Funktion **load** verwendeten Argument, das weiter oben erläutert wurde), um den Benutzer zu informieren, falls die Datei *render.arx* nicht gefunden wird. Dieser Code verwendet die Liste von Zeichenketten, die von der Funktion **ads** (Pfad und Dateiname) als Argument *lst* erzeugt wurde. Die Zeichenkette, die als Argument *Zfolge* aufgerufen wird, unterscheidet zwischen Groß- und Kleinschreibung und muß der Zeichenkette entsprechen, die Sie auf Ihrer Plattform verwenden. Sie können überprüfen, ob Ihre Plattform zwischen Groß- und Kleinschreibung unterscheidet, indem Sie die Funktion **arx** oder **ads** in der Befehlszeile verwenden.

```
(if (not (INLIST (arx) "RENDER") )
  (setq err (arxload "render" nil)))
(if (not err) (princ "\nFehler: RENDER ARX nicht geladen."))
```

8 Allgemeine Dienstprogramm-funktionen

AutoLISP stellt Ihnen verschiedene Funktionen zur Untersuchung der aktuell geladenen Zeichnung zur Verfügung. Dieses Kapitel stellt Ihnen diese Funktionen vor und beschreibt ihre Verwendung in Verbindung mit anderen Funktionen.

8.1 Abfrage- und Befehlsfunktionen

Die in diesem Abschnitt beschriebenen Abfrage- und Befehlsfunktionen ermöglichen direkten Zugriff auf AutoCAD-Befehle und -Zeichenfunktionen. Ihre Eigenschaften hängen vom aktuellen Status des AutoCAD-Systems und der Umgebungsvariablen sowie von der aktuell geladenen Zeichnung ab. Es gibt folgende Abfrage- und Befehlsfunktionen:

acad_colordlg	getcfg	getvar	setvar
Befehl	getenv	setcfg	ver

Der vorliegende Abschnitt enhält Beispiele zu den Funktionen **command**, **getcfg**, **getvar**, **setcfg**, und **getvar**. Die Funktionen **help**, **getenv** und **ver** werden in Kapitel 13, "AutoLISP Funktionskatalog" beschrieben. Der Zugang zur AutoCAD-Hilfe wird durch die Funktion **help** implementiert.

8.1.1 Befehlsübergabe

Die Funktion **command** übergibt einen AutoCAD-Befehl direkt an die AutoCAD-Eingabeaufforderung. Die Funktion **command** besitzt eine Argumentliste von variabler Länge. Diese Argumente müssen den Typen und Werten entsprechen, die von den Eingabeaufforderungen dieses Befehls erwartet werden; sie können Zeichenketten, reelle Zahlen, Ganzzahlen, Punkte, Elementnamen oder Auswahlsatznamen sein. Daten wie Winkel, Abstände und Punkte können entweder als Zeichenketten oder als Werte (als Ganzzahlen, reelle Zahlen oder als Liste von Punkten) übergeben werden. Eine leere Zeichenkette (`""`) entspricht der Eingabe eines Leerzeichens oder von RETURN über die Tastatur.

Die Befehle, die Sie mit der Funktion **command** aufrufen können, unterliegen einigen Beschränkungen. Siehe **"command"** in Kapitel 13.

Das folgende Codefragment zeigt typische Anforderungen von **command**.

```
(command "Kreis" "0,0" "3,3")
(command "Objekthöhe" 1)
(setq p1. '(1.0 1.0 3.0))
(setq rad 4.5)
(command "Kreis" p1 rad)
```

Befindet sich AutoCAD beim Aufruf dieser Funktionen an der Eingabeaufforderung, werden folgende Schritte durchgeführt:

1. Der erste Aufruf von **command** übergibt Punkte als Zeichenketten an den Befehl KREIS (ein Kreis mit dem Mittelpunkt bei (0.0,0.0), der durch (3.0,3.0) verläuft, wird gezeichnet).
2. Der zweite Aufruf übergibt eine Ganzzahl an den Befehl OBJEKTHÖHE. Die aktuelle Objekthöhe wird auf 1.0 geändert.
3. Der dritte Aufruf verwendet einen 3D-Punkt und einen reellen(Gleitkomma) Wert, die beide als Variablen gespeichert und als Referenz an den Befehl KREIS übergeben werden (ein weiterer [extrudierter] Kreis mit dem Zentrum (1.0,1.0,3.0) und dem Radius 4.5 wird gezeichnet).

8.1.1.1 Unterstützung von Fremdsprachen

Wenn Sie AutoLISP-Programme entwickeln, die möglicherweise mit einer fremdsprachigen Version von AutoCAD verwendet werden, werden die Standardbefehle und Schlüsselwörter von AutoCAD automatisch übersetzt, wenn Sie jedem Befehl oder Schlüsselwort einen Unterstrich (_) voranstellen.

```
(command "_line" pt1 pt2 pt3 "_c")
```

Wenn Sie als Präfix einen Punkt verwenden (um die Verwendung von umdefinierten Befehlen zu vermeiden), können Sie Punkt und Unterstrich in beliebiger Reihenfolge verwenden; Sowohl `"._line"` als auch "_.line" sind gültig.

8.1.1.2 Warten auf Benutzereingabe

Wird bei der Ausführung eines AutoCAD-Befehls das vordefinierte Symbol PAUSE als Argument für **command** angetroffen, wird die Ausführung des Befehls unterbrochen und eine direkte Benutzereingabe erwartet (normalerweise

durch die Auswahl eines Punkts oder Ziehen). Dies entspricht dem Pausenmechanismus für Menüs (umgekehrter Schrägstrich).

Führen Sie einen transparenten Befehl aus, während die Funktion **command** unterbrochen ist, bleibt sie weiterhin unterbrochen. So können Benutzer die Befehle 'ZOOM und 'PAN während einer Pause von **command** ausführen. Die Pause wird erst dann beendet, wenn AutoCAD gültige Eingaben erhält und kein transparenter Befehl mehr ausgeführt wird. Der folgende Code beginnt beispielsweise mit dem Befehl KREIS, legt den Mittelpunkt bei (5,5) fest und erwartet dann vom Benutzer, den Radius eines Kreises auf dem Bildschirm zu ziehen. Wenn der Benutzer den gewünschten Punkt festlegt (oder den gewünschten Radius eingibt), wird die Funktion mit dem Zeichnen einer Linie von (5,5) nach (7,5) fortgesetzt.

```
(command "Kreis" "5,5" pause "Linie" "5,5" "7,5" "")
```

Menüeingaben werden von einer AutoLISP-Pause nicht unterbrochen. Ist eine Menüoption aktiv, während die Funktion **command** auf eine Benutzereingabe wartet, kann diese Eingabe auch über das Menü erfolgen. Wollen Sie auch die Menüoption unterbrechen, müssen Sie sie mit einem umgekehrten Schrägstrich (\) markieren. Wird eine gültige Eingabe vorgenommen, wird sowohl die Funktion **command** als auch die Menüoption fortgesetzt.

8.1.1.3 Übergeben von Auswahlpunkten an AutoCAD-Befehle

Einige AutoCAD-Befehle (wie beispielsweise STUTZEN, DEHNEN und ABRUNDEN) erwarten vom Benutzer neben der Angabe des Objekts selbst die eines *Auswahlpunkts*. Sie müssen diese Gruppen von Objekt- und Punktdaten zunächst als Variablen speichern, um sie ohne die Verwendung von PAUSE mit Hilfe der Funktion **command** übergeben zu können. Punkte können als Zeichenketten innerhalb der Funktion **command** übergeben werden, sie können aber auch außerhalb der Funktion definiert und als Variablen übergeben werden, wie im folgenden Beispiel gezeigt. In diesem Codefragment wird eine Möglichkeit genutzt, einen Elementnamen und einen Auswahlpunkt an die Funktion **command** zu übergeben.

```
(command "Kreis" "5,5" "2")          Zeichnet einen Kreis

(command "Linie" "3,5" "7,5" "")     Zeichnet eine Linie
(setq el  (entlast))                 Ermittelt den letzten Elementnamen
(setq pt '(5 7))                     Setzt den Punkt pt
(command "stutzen" el "" pt "")      Führt das Stutzen aus
```

Befindet sich AutoCAD beim Aufruf dieser Funktionen an der Eingabeaufforderung, werden folgende Schritte durchgeführt:

1. Ein Kreis mit dem Mittelpunkt (5,5) und dem Radius 2 wird gezeichnet.
2. Eine Linie von (3,5) bis (7,5) wird gezeichnet.
3. Die Variable `el` wird mit dem Namen des Objekts erstellt, das der Datenbank zuletzt hinzugefügt wurde. (Weitere Erläuterungen zu Objekten und Objektbearbeitungsfunktionen siehe *Kapitel 9,* "Auswahlsatz-, Objekt- und Symboltabellen funktionen",.)
4. Die Variable `pt` wird erstellt, die einen Punkt auf dem Kreis angibt. (Dieser Punkt wählt den Abschnitt des Kreises aus, der gestutzt werden soll.)
5. Der Befehl STUTZEN wird ausgeführt, indem das Objekt `el` und der durch `pt` angegebene Punkt ausgewählt werden.

8.1.2 System- und Umgebungsvariablen

Die Funktionen **getvar** und **setvar** ermöglichen AutoLISP-Anwendungen, Werte von AutoCAD-Systemvariablen zu überprüfen und zu verändern. Diese Funktionen verwenden eine Zeichenkette zur Bestimmung des Variablennamens. Die Funktion **setvar** gibt einen Wert des Typs an, den die Systemvariable erwartet. Es gibt verschiedene Typen von AutoCAD-Systemvariablen: Ganzzahlen, reelle Zahlen, Zeichenketten, 2D-Punkte und 3D-Punkte. Werte, die **setvar** als Argumente übergeben werden, müssen den richtigen Datentyp besitzen. Wird ein ungültiger Typ verwendet, zeigt AutoLISP eine Fehlermeldung an. Eine Liste der Systemvariablen und ihrer Datentypen finden Sie unter "System- und Umgebungsvariablen" in der AutoCAD Command Reference. AutoLISP-Funktionen:getvar, AutoLISP-Funktionen: setvar

Das folgende Codefragment stellt sicher, daß alle folgenden ABRUNDEN-Befehle einen Radius von mindestens 1 verwenden:

```
(if (< (getvar "filletrad") 1)
  (setvar "filletrad" 1)
)
```

Mit der Zusatzfunktion **getenv** können AutoLISP-Routinen auf die aktuell definierten Umgebungsvariablen des Betriebssystems zugreifen.

8.1.3 Konfigurationssteuerung

AutoCAD benutzt die Datei *acad14.cfg*, um Konfigurationsdaten zu speichern. Der Abschnitt `App-Data` dieser Datei erlaubt es Benutzern und Entwicklern, für ihre Anwendungen relevante Konfigurationsinformationen zu speichern. Mit den Funktionen **getcfg** und **setcfg** können AutoLISP-Anwendungen die Parameterwerte im Abschnitt `AppData` überprüfen und ändern.

8.2 Anzeigesteuerung

AutoLISP enthält Funktionen zur Steuerung der AutoCAD-Anzeige, einschließlich der Text- und Grafikfenster. Einige dieser Funktionen benötigen Eingaben vom AutoCAD-Benutzer. Folgende Funktionen stehen zur Anzeigesteuerung zur Verfügung:

graphscr	menucmd	print	textpage
grdraw	menugroup	prompt	textscr
grtext	prin1	redraw	vports
grvecs	princ	terpri	

Die Funktionen **terpri** und **vports** werden in Kapitel 13 beschrieben.

8.2.1 Anzeigen von Meldungen in der Befehlszeile

Die grundlegenden Ausgabefunktionen sind **prompt**, **princ**, **prin1** und **print**. Die Funktion **prompt** zeigt eine Meldung (eine Zeichenkette) in der AutoCAD-Befehlszeile an und gibt `nil` zurück. Die Funktionen **princ**, **prin1** und **print** zeigen einen Ausdruck (nicht notwendigerweise eine Zeichenkette) in der Befehlszeile an, geben den Ausdruck zurück und können wahlweise eine Ausgabe an eine Datei senden. Es bestehen folgende Unterschiede zwischen den Funktionen: **princ** zeigt Zeichenketten ohne die umschließenden Anführungszeichen an, **prin1** und **print** zeigen Zeichenketten in Anführungszeichen an; **print** fügt eine Leerzeile vor und ein Leerzeichen nach dem Ausdruck ein.

Auch die folgenden Funktionen zur Dateibearbeitung können im Befehlszeilenbereich Ausgaben anzeigen. (Weitere Informationen zu diesen Funktionen finden Sie im Abschnitt "Dateibearbeitung.")

read-char	read-line	write-char	write-line

Die Länge einer mit **prompt** angezeigten Zeichenkette sollte nicht über die der Befehlszeile im Grafikfenster hinausgehen. Diese umfaßt normalerweise nicht mehr als 80 Zeichen.

Das folgende Beispiel verdeutlicht die Unterschiede zwischen den vier grundlegenden Ausgabefunktionen und zeigt, wie sie die gleiche Zeichenkette verarbeiten. AutoLISP bietet zahlreiche Steuerzeichen, die in Zeichenketten verwendet werden können. Eine vollständige Liste finden Sie in "Steuerzeichen in Zeichenketten."

```
(setq str "Die \"zulässige\" Toleranz beträgt \261 \274\"")

(prompt str)    gibt aus    Die "zulässige" Toleranz beträgt ± 1/4"
                und gibt zurück     nil

(princ str)     gibt aus    Die "zulässige" Toleranz beträgt ± 1/4"
                und gibt zurück    "Die "zulässige" Toleranz beträgt ± 1/4""

(prin1 str)     gibt aus    Die "zulässige" Toleranz beträgt ± 1/4"
                und gibt zurück    "Die "zulässige" Toleranz beträgt ± 1/4""

(print str)     gibt aus    <Leerzeile>
                            "Die "zulässige" Toleranz beträgt ± 1/4"" <Leerzeichen>
                und gibt zurück    "Die "zulässige" Toleranz beträgt ± 1/4""
```

8.2.2 Menüsteuerung

Die Funktion **menucmd** steuert die Anzeige der Grafikfenster-Menüs. Sie akzeptiert ein Zeichenkettenargument, welches ein Untermenü des aktuellen Menüs bezeichnet, sowie die Operation, die mit dem Untermenü durchgeführt werden soll. Diese Operation kann Anzeigen, Ändern oder Abfragen des Untermenüs umfassen.

Die Funktion **menucmd** nimmt ein Zeichenkettenargument mit zwei durch ein Gleichheitszeichen getrennten Feldern an:

"*menu_area*=*action*"

Mit dieser Syntax kann ein Untermenü in einen festgelegten Menübereich geladen werden oder eine Menüoption oder ein aktuell geladener Menübereich bearbeitet werden. Das Feld *Menübereich* legt fest, auf welchen Teil des Menüs sich die Operation bezieht. Das Feld kann einen Menübereich wie beispielsweise `P0` (für das Cursormenü) oder `S` (für das Bildschirmmenü) oder eine spezielle Menüoption festlegen. Das Feld *Operation* legt die Operation fest, die auf den Menübereich oder die Menüoption ausgeführt werden soll, oder bestimmt ein Untermenü, das in den Menübereich geladen werden soll. Die Menübereiche, die über diese Syntax bearbeitet werden können, entsprechen denen, die in Untermenüreferenzen in Menüdateien verwendet werden.

Jeder Menübereich hat ein aktuell geladenes Untermenü. Vorgabemäßig ist das erste Untermenü, das einer Menüabschnittsbezeichnung folgt, in diesen Menübereich geladen.

Wenn *Menübereich* ein Pull-Down-Menü oder ein Bilddateimenü festlegt, kann *Operation* ein Sternchen (*) sein. Dadurch wird das Menü angezeigt (Pull-Down-Menüs und Bilddateimenüs werden nicht automatisch angezeigt, wenn sie aufgerufen werden). In Windows werden nur das P0-(Cursor-)Menü und Bilddateimenüs mit dem Sternchen (*) angezeigt. Zusätzliche Informationen über Menüdateien finden Sie in Kapitel 4, "Benutzerspezifische Menüs"."Statuszeilen- konfiguration und Makroprogrammiersprache DIESEL".

Anmerkung Nehmen Sie das Dollarzeichen, das vor gleichen Befehlen in einer Menüdatei steht, *nicht* in das Zeichenkettenargument auf. Ebenso dürfen Sternchen, die in der Menüdatei zur Kennzeichnung von Untermenübezeichnungen dienen, *nicht* in das Feld *Operation* des Arguments übernommen werden.

Der folgende Aufruf der Funktion **menucmd** bewirkt, daß das in der aktuellen Menüdatei definierte Untermenü **OFANG angezeigt wird (vorausgesetzt, das Bildschirmmenü ist aktiviert).

```
(menucmd "S=OFANG")
```

In Windows können Sie auf die Menügruppe verweisen. Das kann nützlich sein, wenn mehrere Menüs geladen sind, die den gleichen Untermenünamen haben. Der folgende Code zeigt das Bildschirmuntermenü **OFANG in der AutoCAD-Menügruppe an.

```
(menucmd "S=ACAD.OFANG")
```

Die Funktion **menucmd** kann Untermenüs in die Menübereiche BUTTONS und AUX laden. Wenn Sie möchten, daß Ihre Digitalisierschaltfläche, je nachdem, ob der Tablett-Modus ein- oder ausgeschaltet ist, unterschiedlich funktionieren, können Sie zwei Untermenüs im Abschnitt ***BUTTONS1, **DIGBUTTONS- und **TABBUTTONS- definieren und mit dem folgenden Code dazwischen hin- und herschalten.

`(menucmd "B1=DIG-BUTTONS")`	*Aktiviert das DIG-BUTTONS-Untermenü*
`(menucmd "B1=TAB-BUTTONS")`	*Aktiviert das TAB-BUTTONS-Untermenü*

Der folgende Code lädt das ***POP0-Menü in den P0-(Cursor-)Menübereich und zeigt es an.

`(menucmd "P0=POP0")`	*Lädt das* `***POP0`*-Menü in den P0-Menübereich*
`(menucmd "P0=*")`	*Zeigt es an*

Wenn Sie sicher sind, daß das richtige Menü in einen bestimmten Menübereich geladen wurde, brauchen Sie es nicht mehr jedes Mal speziell zu laden, wenn Sie es anzeigen möchten.

Der folgende Aufruf zeigt das Pull-Down-Menü an, das aktuell an der Position `P1` (erstes Pull-Down-Menü) geladen ist.

```
(menucmd "P1=*")
```

Wird `"P1=*"` ohne vorheriges Laden des Menüs verwendet, kann dies zu unerwarteten Ergebnissen führen. Obwohl Sie praktisch jedes Menü an einer Pull-Down- oder Cursormenüposition laden können, empfiehlt es sich jedoch, nur Menüs zu verwenden, die speziell für diesen Menübereich vorgesehen sind. Sie können beispielsweise das Untermenü MEHRDATEN mit folgendem Code an der Position `P1` laden:

`(menucmd "P1=MEHRDATEN")`	*Lädt das Menü* `**MEHRDATEN` *an der Menüposition P1*
`(menucmd "P1=*")`	*Zeigt es an*

Das Menü bleibt an dieser Position, bis Sie es durch Laden eines anderen Menüs ersetzen, wie im folgenden gezeigt:

```
(menucmd "P1=POP1")
```

Verwendet Ihr Menü Deaktivierungs- (Abblend-) und Markierungsfunktionen, können Sie mit der Funktion **menucmd** den Status einer Menübeschriftung abrufen und ändern. Der folgende Aufruf ermittelt den aktuellen Status der vierten Beschriftung im Pull-Down-Menü `P2`.

`(menucmd "P2.4=#?")` *Wenn deaktiviert, wird* `"P2.4=~" zurückgegeben.`

Diese Funktionsaufrufe aktivieren und deaktivieren dieselbe Beschriftung:

`(menucmd "P2.4=")` *Aktiviert die Beschriftung*
`(menucmd "P2.4=~")` *Deaktiviert die Beschriftung*

Links neben den Menübeschriftungen können Sie auch Markierungen einfügen oder entfernen.

Die vorstehend beschriebene Methode zur Behandlung von Menüoptionen funktioniert relativ gut bei einzelnen statischen Menüs. Sie wird jedoch unzuverlässig, wenn sich die Positionen der Menüoptionen beim Laden von Teilen einer Menüdatei ändern. Sie können die Eigenschaften Menügruppe und Namenfeld verwenden, um Menüoptionen zurückzuverfolgen. Sie legen eine Menüoption also nicht durch ihre Position in der Menüdatei fest, sondern legen die Menügruppe und das Namenfeld der Menüoption fest.

Wenn Sie die Menügruppe verwenden, um Menübeschriftungen zu aktivieren, zu deaktivieren oder zu markieren, müssen Sie dem Gruppennamen ein G voranstellen, wie in den folgenden Beispielen gezeigt.

`(menucmd "Gacad.ID_New=~")` *Deaktiviert die Beschriftung*
`(menucmd "Gacad.ID_New=")` *Deaktiviert die Beschriftung*

AutoLISP-Funktionen können nicht nur Menübeschriftungen aktivieren und deaktivieren, sie können auch den in der Beschriftung angezeigten Text ändern, indem sie einen DIESEL-Zeichenkettenausdruck in die Beschriftung einfügen. Da DIESEL nur Zeichenketten als Eingabe akzeptiert, können Sie mit einer USERS1-5-Systemvariablen Informationen an den DIESEL-Ausdruck übergeben, wenn die Variable zuvor mit dem Rückgabewert der Funktion initialisiert wurde.

Sie können die Funktion **menucmd** auch verwenden, um DIESEL-Zeichenkettenausdrücke innerhalb einer AutoLISP-Funktion auszuwerten. Die folgende Routine gibt die aktuelle Zeit aus:

```
(defun C:CTIME ( / ctim)
  (setq ctim
    (menucmd "M=$(edtime,$(getvar,date),H:MMam/pm)"))
  (princ (strcat "\nDie aktuelle Zeit ist " ctim))
  (princ)
)
```

Weitere Informationen zur Verwendung von DIESEL-Ausdrücken mit AutoLISP und eine Liste der DIESEL-Funktionen finden Sie in Kapitel 5, "Statuszeilen- konfiguration und Makroprogrammiersprache DIESEL"."Statuszeilen-konfiguration und Makroprogrammiersprache DIESEL".

8.2.3 Steuern von Grafik- und Textfenstern

Sie können die Anzeige der Grafik- und Textfenster von einer AutoLISP-Anwendung aus steuern. Bei AutoCAD-Installationen mit nur einem Bildschirm zeigt ein Aufruf von **graphscr** das Grafikfenster und ein Aufruf von **textscr** das Textfenster an. Die Verwendung dieser Funktion entspricht dem Umschalten mit der Funktionstaste zur Bildschirmumschaltung. Die Funktion **textpage** ist mit **textscr** vergleichbar, **textpage** löscht jedoch den Inhalt des Textfensters, bevor es angezeigt wird (wie bei den AutoCAD-Befehlen STATUS und LISTE).

Die Funktion **redraw** ähnelt dem AutoCAD-Befehl NEUZEICH, bietet jedoch mehr Möglichkeiten, die Anzeige zu steuern: sie zeichnet nicht nur den gesamten Grafikbereich neu, sondern kann auch angeben, daß ein einzelnes Objekt neu gezeichnet oder rückgängig gemacht wird. Bei einem komplexen Objekt, wie bei einer Polylinie oder einem Block, kann **redraw** das gesamte Objekt zeichnen (bzw. die Zeichnung rückgängig machen) oder seinen Header. Die Funktion **redraw** kann angegebene Objekte auch markieren oder deren Markierung aufheben.

8.2.4 Steuern von systemnahen Grafikfunktionen

AutoLISP stellt Funktionen zur Verfügung, die systemnahe Grafikfunktionen steuern und direkten Zugriff auf den AutoCAD-Grafikbildschirm und die Eingabegeräte ermöglichen.

Die Funktion **grtext** zeigt Text direkt in den Status- oder Menübereichen an, markiert oder nicht markiert. Die Funktion **grdraw** zeichnet einen Vektor in das aktuelle Ansichtsfenster, wobei Farbe und Markierung gesteuert werden können. Die Funktion **grvecs** zeichnet mehrere Vektoren.

Anmerkung Da diese Funktionen auf AutoCAD-Codes basieren, kann sich deren Funktionsweise von Release zu Release ändern. Die Aufwärtskompatibilität von Anwendungen, die auf diese Funktionen zugreifen, kann nicht garantiert werden. Die Arbeitsweise der Funktionen hängt außerdem von der aktuellen Hardware-Konfiguration ab: insbesondere die Funktionsweise von Anwendungen, die auf **grtext** zurückgreifen, ist wahrscheinlich nicht bei allen Konfigurationen gleich, wenn der Entwickler diese Funktionen nicht mit Bedacht implementiert (siehe Kapitel 4, "Benutzerspezifische Menüs") und den Bezug auf spezifische Eigenschaften der Hardware vermeidet. Da es sich bei diesen Funktionen um systemnahe Funktionen handelt, geben sie nahezu keine Fehlermeldungen aus und können die Anzeige des Grafikbildschirms unerwartet ändern (das folgende Beispiel zeigt einen Weg, dies zu vermeiden).

Die folgende Sequenz stellt die durch fehlerhafte Aufrufe von **grtext**, **grdraw** oder **grvecs** hervorgerufene Vorgabeanzeige des Grafikbildschirms wieder her:

```
(grtext)          Stellt Standardtext wieder her
(redraw)
```

8.3 Anfordern von Benutzereingaben

Mit den folgenden Funktionen zur Eingabe durch den Benutzer kann der Benutzer in einer AutoLISP-Anwendung zur Dateneingabe aufgefordert werden. Es gibt folgende Funktionen zur Eingabe durch den Benutzer:

entsel	getfiled	getpoint	nentsel
getangle	getint	getreal	nentselp
getcorner	getkword	getstring	
getdist	getorient	initget	

8.3.1 Die getxxx-Funktionen

Jede Benutzereingabefunktion **getxxx**wartet auf eine Dateneingabe des richtigen Typs und gibt den eingegebenen Wert zurück. Die Anwendung kann eine optionale Eingabeaufforderung festlegen, die erscheint, bevor die Funktion angehalten wird. In der folgenden Tabelle finden Sie die **getxxx**-Funktionen mit den zur Benutzereingabe erforderlichen Datentypen.

Gültige Eingaben für die Benutzereingabefunktionen getxxx

Funktionsname	Datentyp der Benutzereingabe
`getint`	Ein Ganzzahlwert in der Befehlszeile
`getreal`	Ein reeller Wert oder Ganzzahlwert in der Befehlszeile
`getstring`	Eine Zeichenkette in der Befehlszeile
`getpoint`	Ein Punktwert in der Befehlszeile oder auf dem Bildschirm ausgewählt
`getcorner`	Ein Punktwert (die gegenüberliegende Ecke eines Rechtecks) in der Befehlszeile oder auf dem Bildschirm ausgewählt
`getdist`	Ein reeller Wert oder Ganzzahlwert (eines Abstands) in der Befehlszeile oder durch Auswahl von Punkten auf dem Bildschirm festgelegt
`getangle`	Ein Winkelwert (im aktuellen Winkelformat) in der Befehlszeile oder durch Auswahl von Punkten auf dem Bildschirm festgelegt
`getorient`	Ein Winkelwert (im aktuellen Winkelformat) in der Befehlszeile oder durch Auswahl von Punkten auf dem Bildschirm festgelegt
`getkword`	Ein vordefiniertes Schlüsselwort oder seine Abkürzung in der Befehlszeile

Anmerkung Obwohl die Funktionen **getvar**, **getcfg** und **getenv** mit den Buchstaben *g, e* und *t* beginnen, handelt es sich bei ihnen *nicht* um Funktionen, die Benutzereingaben entgegennehmen. Sie werden in "Abfrage- und Befehlsfunktionen" erläutert.

Die Funktionen **getint**, **getreal** und **getstring** erwarten eine Benutzereingabe in der AutoCAD-Eingabeaufforderungszeile. Sie können nur einen Wert des Typs zurückgeben, der auch angefordert wurde.

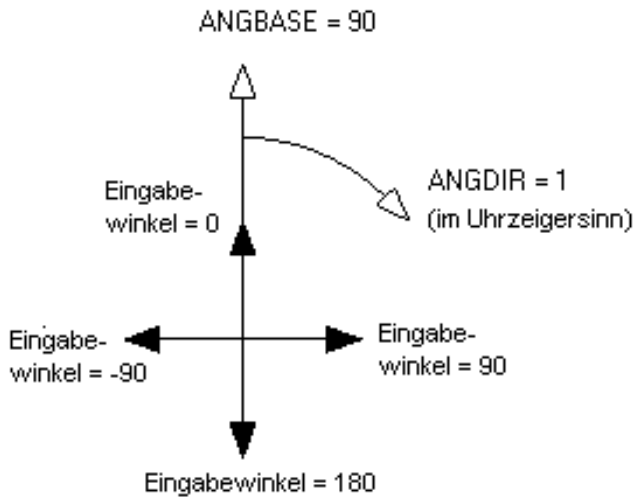
Die Funktionen **getpoint**, **getcorner** und **getdist** warten auf eine Benutzereingabe in der Eingabeaufforderungszeile oder die Auswahl von Punkten auf dem Grafikbildschirm. Die Funktionen **getpoint** und **get-corner** geben 3D-Werte für die Punkte, **getdist** gibt einen reellen Wert zurück.

Sowohl **getangle** als auch **getorient** erwarten vom Benutzer die Eingabe des Werts für einen Winkel in der Eingabeaufforderungszeile oder durch Auswahl von definierten Punkten auf dem Grafikbildschirm. Für die Funktion **getorient** befindet sich der Winkel Null immer auf der rechten Seite: "Ost" oder "3 Uhr". Für **getangle** legt der Wert von ANGBASE den Winkel Null fest, der auf einen beliebigen Winkel gesetzt werden kann. Sowohl **getangle** als auch **getorient** geben den Wert eines Winkels (reell) im Bogenmaß zurück, der gegen den Uhrzeigersinn von einem Ausgangspunkt (Winkel Null) aus angegeben wird, und zwar für **getangle** gleich dem ANGBASE-Wert und für **getorient** auf der rechten Seite.

Nehmen wir beispielsweise an, daß ANGBASE auf 90 Grad (Nord) eingestellt ist und ANGDIR auf 1 gesetzt wird (Winkel werden im Uhrzeigersinn größer). Die folgende Tabelle zeigt, welche Werte (im Bogenmaß) **getangle** und **getorient** für typische Eingabewerte (in Grad) zurückgeben.

Mögliche Rückgabewerte für getangle und getorient

Eingabe (Grad)	getangle	getorient
0	0.0	1.5708
-90	1.5708	3.14159
180	3.14159	4.71239
90	4.71239	0.0



Die Funktion **getangle** berücksichtigt bei der Verarbeitung von Eingaben die Einstellungen von ANGDIR und ANGBASE. Sie können **getangle** zur Ermittlung des Drehwinkels für eine Blockeinfügung verwenden, da eine Eingabe von 0 Grad immer einen Winkel 0 im Bogenmaß zurückgibt. Die Funktion **getorient** berücksichtigt nur ANGDIR. Verwenden Sie **getorient**, um Winkel zu ermitteln, wie beispielsweise den Winkel der Basislinie eines Textobjekts. Beispielsweise gibt **getorient** für eine mit dem Winkel 0 erstellte Textzeile den Winkelwert 90 zurück, wenn ANGBASE und ANGDIR wie oben beschrieben eingestellt sind.

Die Funktionen zur Anwendereingabe profitieren von den Fehlerprüfungen von AutoCAD. Einfache Fehler werden von AutoCAD erkannt und von den Funktionen zur Anwendereingabe nicht zurückgegeben. Ein vorhergehender Aufruf von **initget** bietet weitere Filtermöglichkeiten und verringert so die Notwendigkeit zur Fehlerprüfung.

Die Funktion **getkword** unterbricht die Ausführung und erwartet die Eingabe eines Schlüsselworts oder seiner Abkürzung. Schlüsselwörter müssen mit der Funktion **initget** definiert werden, bevor **getkword** aufgerufen wird. Alle Benutzereingabefunktionen (außer **getstring**) akzeptieren Schlüsselwortwerte zusätzlich zu den normalerweise zurückzugebenden Werten, vorausgesetzt, **initget** wurde aufgerufen, um die Schlüsselwörter zu definieren.

Alle Benutzereingabefunktionen ermöglichen die Angabe eines optionalen Arguments *Anfrage*. Die Verwendung dieses Arguments ist einem vorherigen Aufruf der Funktionen **prompt** oder **princ** vorzuziehen. Wird das Argument *Anfrage* beim Aufruf der Benutzereingabefunktion angegeben, wird die Eingabeaufforderung bei einer ungültigen Benutzereingabe erneut aktiviert. Gibt der Benutzer solche Informationen ein, wenn kein Argument *Anfrage* vorhanden ist, erscheint in der Eingabeaufforderungszeile von AutoCAD die folgende Meldung:

Nochmaliger Versuch:

Dies kann verwirrend sein, wenn die ursprüngliche Eingabeaufforderung nicht mehr im Befehlsbereich zu sehen ist.

Anmerkung Der AutoCAD-Benutzer kann normalerweise *nicht* mit der Eingabe eines AutoLISP-Ausdrucks auf eine Benutzereingabefunktion antworten. Verwendet Ihre AutoLISP-Routine die Funktion **init-get**, sind beliebige Tastatureingaben bei bestimmten Funktionen zulässig, die einen AutoLISP-Ausdruck als Antwort auf einen in AutoLISP implementierten Befehl erlauben. Dies wird in "Beliebige Tastatureingaben" erläutert.

8.3.2 Steuern der Bedingungen von Benutzereingabefunktionen

Die Funktion **initget** ermöglicht die Steuerung des nächsten Aufrufs einer Benutzereingabefunktion. Die Funktion **initget** richtet verschiedene Optionen ein, die von der nächsten **entsel**, **nentsel**, **nentselp** oder **getxxx**-Funktion (außer **getstring**, **getvar** und **getenv**) verwendet werden können. Diese Funktion akzeptiert die Argumente *Bits* und *Z_Kette*, die beide fakultativ sind. Das Argument *Bits* bestimmt ein oder mehrere Steuerbits, die bestimmte Eingabewerte für den nächsten Aufruf einer Benutzereingabefunktion aktivieren oder deaktivieren. Mit dem Argument *Z_Kette* können Schlüsselwörter festgelegt werden, die von der folgenden Benutzereingabefunktion erkannt werden.

Die von **initget** festgelegten Steuerbits und Schlüsselwörter sind nur für den nächsten Aufruf einer Benutzereingabefunktion gültig. Danach werden sie verworfen. Die Anwendung muß **initget** nicht ein zweites Mal aufrufen, um bestimmte Bedingungen aufzuheben.

8.3.2.1 Eingabeoptionen für Benutzereingabefunktionen

Der Wert des Arguments *Bits* beschränkt für den folgenden Aufruf einer Benutzereingabefunktion die Typen der Benutzereingabe. Dies reduziert die Fehlerprüfung. Es folgen einige der verfügbaren Bitwerte: 1 schließt eine leere Eingabe aus, 2 schließt die Eingabe von 0 (Null) aus, und 4 schließt die Eingabe eines negativen Werts aus. Werden diese Werte für den der Funktion **getint** folgenden Aufruf verwendet, muß der Benutzer einen Ganzzahlwert größer als 0 eingeben.

Um mehrere Bedingungen gleichzeitig festzusetzen, addieren Sie die Werte (in beliebiger Kombination) und erhalten so einen *Bits*-Wert zwischen 0 und 255. Wenn *Bits* nicht aufgenommen oder auf 0 gesetzt wird, ist keine der Steuerbedingungen für den nächsten Aufruf einer Benutzereingabefunktion gültig. (Eine vollständige Liste der **initget**-Biteinstellungen finden Sie in **"initget"**, Kapitel 13,.)

```
(initget (+ 1 2 4))
(getint "\nWie alt sind Sie? ")
```

Diese Sequenz fragt den Benutzer nach seinem Alter. Versucht der Benutzer, einen negativen Wert oder Null einzugeben, nur RETURN zu drücken oder eine Zeichenkette einzugeben, gibt AutoCAD eine Fehlermeldung aus. Die Funktion **getint** läßt nur die Eingabe von Ganzzahlwerten zu.

8.3.2.2 Schlüsselwortoptionen

Das fakultative Argument *Z_Kette* legt eine Liste von Schlüsselwörtern fest, die vom nächsten Aufruf einer Benutzereingabefunktion erkannt werden.

Die Funktion **initget** ermöglicht, daß auch die Abkürzungen von Schlüsselwörtern erkannt werden. Die Benutzereingabefunktion gibt ein vordefiniertes Schlüsselwort zurück, wenn die Benutzereingabe der Schreibweise eines Schlüsselworts entspricht (keine Unterscheidung von Groß- und Kleinschreibung) oder der Benutzer die Abkürzung eines Schlüsselworts eingibt. Es gibt zwei Methoden, Schlüsselwörter abzukürzen; sie werden in "Spezifikationen für Schlüsselwörter" erläutert.

Die folgende benutzerdefinierte Funktion enthält einen Aufruf von **getreal**, dem der Aufruf von **initget** zur Definition von zwei Schlüsselwörtern vorangeht. Die Anwendung sucht nach diesen Schlüsselwörtern und setzt den Eingabewert entsprechend.

```
(defun C:GETNUM (/ num)
  (initget 1 "Pi Zwei-Pi")
  (setq num (getreal "Pi/Zwei-Pi/<Zahl>: "))
  (cond
    ((eq num "Pi") Pi)
    ((eq num "Zwei-Pi") (* 2.0 Pi))
    (T num)
  )
)
```

Dieser Aufruf von **initget** verhindert die Eingabe eines leeren Werts (*Bits* = 1) und erzeugt eine Liste der beiden Schlüsselwörter "Pi" und "Zwei-Pi". Anschließend fordert die Funktion **getreal** mit der folgenden Eingabeaufforderung eine reelle Zahl an:

Pi/Zwei-Pi/<Zahl>:

Das Ergebnis wird in der lokalen Variablen num abgelegt. Gibt der Benutzer eine Zahl ein, wird diese von **C:GETNUM** zurückgegeben. Wenn der Benutzer aber das Schlüsselwort Pi (oder einfach P) eingibt, gibt **getreal** das Schlüsselwort Pi zurück. Die Funktion **cond** erkennt dies und gibt in diesem Fall den Wert von p zurück. Das Schlüsselwort Zwei-Pi wird analog verarbeitet.

Anmerkung Sie können **initget** auch verwenden, um die Eingabe von Schlüsselwörtern bei **entsel**, **nentsel** und **nentselp** zu ermöglichen. Weitere Informationen zu diesen Funktionen finden Sie unter "Objektbearbeitung" und **"entsel", "nentsel",** sowie **"nentselp"** in Kapitel 13,.

8.3.2.3 Beliebige Tastatureingaben

Mit der Funktion **initget** sind beliebige Tastatureingaben in die meisten **getxxx**-Funktionen möglich. Diese Eingaben werden der Anwendung als Zeichenkette zurückgegeben. Um dem Benutzer den Aufruf einer AutoLISP-Funktion an der Eingabeaufforderung einer **getxxx**-Funktion zu ermöglichen, kann eine Anwendung geschrieben werden.

Die folgenden Funktionen zeigen eine Methode, die AutoLISP ermöglicht, auf einen **getxxx** Funktionsaufruf zu antworten:

```
(defun C:ARBENTRY ( / pt1)
  (initget 128)                      ; Setzt ein beliebiges Eingabebit
  (setq pt1 (getpoint "\nPunkt: "))  ; Nimmt einen Wert vom Benutzer entgegen.
  (if (= 'STR (type pt1))            ; Wenn es sich um eine Zeichenkette
    (setq pt1 (eval (read pt1)))     ; handelt, wird sie in ein Symbol
                                             ; konvertiert und,falls möglich,
    pt1                              ; wird einfach der Wert zurückgegeben.
  )
)
(defun REF ( )
  (setvar "LASTPOINT" (getpoint "\nBezugspunkt: "))
  (getpoint "\nNächster Punkt: " (getvar "LASTPOINT"))
)
```

Wenn sowohl die Funktion **C:ARBENTRY** als auch die Funktion **REF** in die Zeichnung geladen ist, ist die folgende Befehlssequenz gültig.

Befehl: **arbentry**

Punkt: **(ref)**

Bezugspunkt: *Wählen Sie einen Punkt*

Nächster Punkt: **@1,1,0**

8.3.2.4 Eingabeprüfung

Sie sollten Ihren Code gegen unbeabsichtigte Benutzerfehler absichern. Die **getxxx**-Funktionen zur Benutzereingabe von AutoLISP übernehmen das zu einem großen Teil für Sie. Sie sollten jedoch keinesfalls vergessen, die Erfüllung anderer Bedingungen zu überprüfen, die das Programm stellt und die nicht von den **getxxx**-Funktionen kontrolliert werden. Die Programmintegrität kann erheblich gestört werden, wenn Sie die Gültigkeitsprüfung von Eingaben vernachlässigen.

8.4 Geometrische Funktionen

Eine Gruppe von Funktionen ermöglicht es Anwendungen, rein geometrische Informationen und geometrische Daten der Zeichnung zu ermitteln. Die Funktionen der geometrischen Dienstprogramme sind:

angle	inters	polar
Abstand	osnap	textbox

Die Funktion **angle** berechnet den Winkel im Bogenmaß zwischen einer Linie und der *X*-Achse (des aktuellen BKS), **distance** berechnet den Abstand zwischen zwei Punkten, und **polar** ermittelt einen Punkt mit Hilfe von Polarkoordinaten (bezogen auf einen Anfangspunkt). Die Funktion **inters** ermittelt den Schnittpunkt zweier Linien. Die Funktionen **osnap** und **textbox** werden getrennt beschrieben.

Das folgende Codefragment zeigt Aufrufe von Funktionen der geometrischen Dienstprogramme:

```
(setq pt1 '(3.0 6.0 0.0))
(setq pt2 '(5.0 2.0 0.0))
(setq basis '(1.0 7.0 0.0))
(setq rads  (angle pt1 pt2))          Winkel in der XY-Ebene des aktuellen BKS
                                      (Wert wird im Bogenmaß zurückgegeben)
(setq len  (distance pt1 pt2))        Abstand im 3D-Raum

(setq endpt  (polar basis bogenm länge))
```

Der Aufruf von **polar** legt `endpt` auf einen Punkt, der den gleichen Abstand von (1,7) hat wie `pt1` von `pt2` und der im gleichen Winkel zur *X*-Achse liegt wie der Winkel zwischen `pt1` und `pt2`.

8.4.1 Objektfang

Die Funktion **osnap** kann einen Punkt mit Hilfe eines der AutoCAD-Objektfangmodi ermitteln. Die Fangmodi werden in einem Zeichenkettenargument festgelegt.

Der folgende Aufruf von **osnap** sucht den Mittelpunkt eines Objekts in der Nähe von `pt1`:

```
(setq pt2 (osnap pt1 "mittp"))
```

Der folgende Aufruf sucht den Mittelpunkt, den Endpunkt oder das Zentrum des Objekts, das `pt1` am nächsten liegt:

```
(setq pt2 (osnap pt1 "mittp,endp,Zentrum"))
```

In beiden Beispielen wird `pt2` auf den Fangpunkt gesetzt, wenn ein Punkt gefunden wird, der die Objektfangbedingungen erfüllt. Falls mehr als ein Fangpunkt diese Bedingungen erfüllt, wird ein Punkt auf der Basis der Einstellungen der Systemvariable SORTENTS ausgewählt. Andernfalls wird `pt2` auf `nil` gesetzt.

Anmerkung Die Systemvariable APERTURE bestimmt die Ausdehnung des Objektfensters und damit den zulässigen Abstand eines ausgewählten Punkts von einem Objekt.

8.4.2 Textabmessungen

Die Funktion **textbox** gibt die diagonalen Koordinaten eines Rechtecks zurück, das ein Textobjekt einschließt. Sie benötigt als einziges Argument eine Elementdefinitionsliste, wie sie von **entget** zurückgegeben wird (eine Liste der zugeordneten Gruppencodes und -werte). Diese Liste kann eine vollständige Beschreibung der Zuordnungsliste des Textobjekts umfassen oder nur die Zeichenfolge beschreiben.

Die von **textbox** zurückgegebenen Punkte beschreiben den Begrenzungsrahmen des Textobjekts (ein imaginäres Rechteck, welches das Textobjekt einschließt), als ob sich sein Einfügepunkt bei (0,0,0) befände und sein Drehwinkel 0 wäre. Die erste zurückgegebene Liste ist der Punkt (0.0 0.0 0.0), falls das Textobjekt nicht schräg oder vertikal liegt oder Buchstaben mit Unterlängen (wie g oder p) enthält. Der Wert der ersten Punktliste gibt den Abstand zwischen dem Texteinfügepunkt und der unteren linken Ecke des kleinsten, den Text umschließenden Rechtecks an. Die zweite Punktliste bezeichnet die obere rechte Ecke dieses Rechtecks. Die zurückgegebenen Punktlisten beschreiben immer die untere linke und die obere rechte Ecke dieses Begrenzungsrahmens, ohne Berücksichtigung der Ausrichtung des zu messenden Texts.

Das folgende Beispiel zeigt die minimal zulässige Elementdefinitionsliste, die **textbox** akzeptiert. Da keine weiteren Informationen angegeben werden, verwendet **textbox** die aktuellen Vorgaben für Textstil und -höhe.

Befehl: **(textbox '((1 . "Hallo allerseits")))**

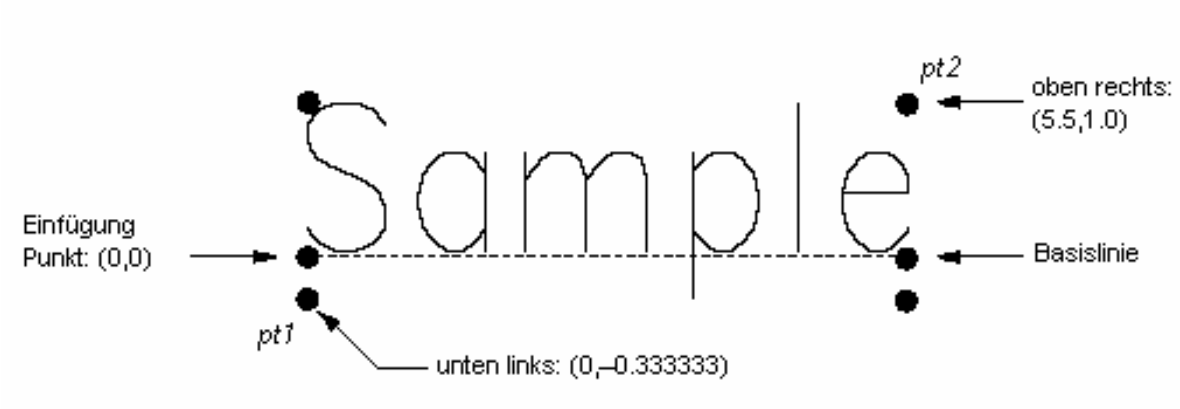
((0.0 0.0 0.0) (2.80952 1.0 0.0))

Die tatsächlichen Werte, die von **textbox** zurückgegeben werden, sind je nach dem aktuellen Textstil unterschiedlich.

Das folgende Beispiel zeigt eine Möglichkeit, der Funktion **textbox** eine Elementdefinitionsliste zur Verfügung zu stellen.

Befehl: **dtext**
Position/Stil/<Startpunkt>: **1,1**
Höhe <1.0000>: RETURN
Drehwinkel <0>: RETURN
Text: **test**
Text: RETURN
Befehl: **(setq e (entget (entlast)))**
((-1 . <Elementname: 6000001c>) (0 . "TEXT") (8 . "0")
(10 1.0 1.0 0.0) (40 . 1.0) (1 . "Test") (50 . 0.0)
(41 . 1.0)(51 . 0.0) (7 . "STANDARD") (71 . 0) (72 . 0)
(11 0.0 0.0 0.0) (210 0.0 0.0 1.0) (73 . 0))
Befehl: **(textbox e)**
((0.0 0.0 0.0) (0.8 0.2 0.0))

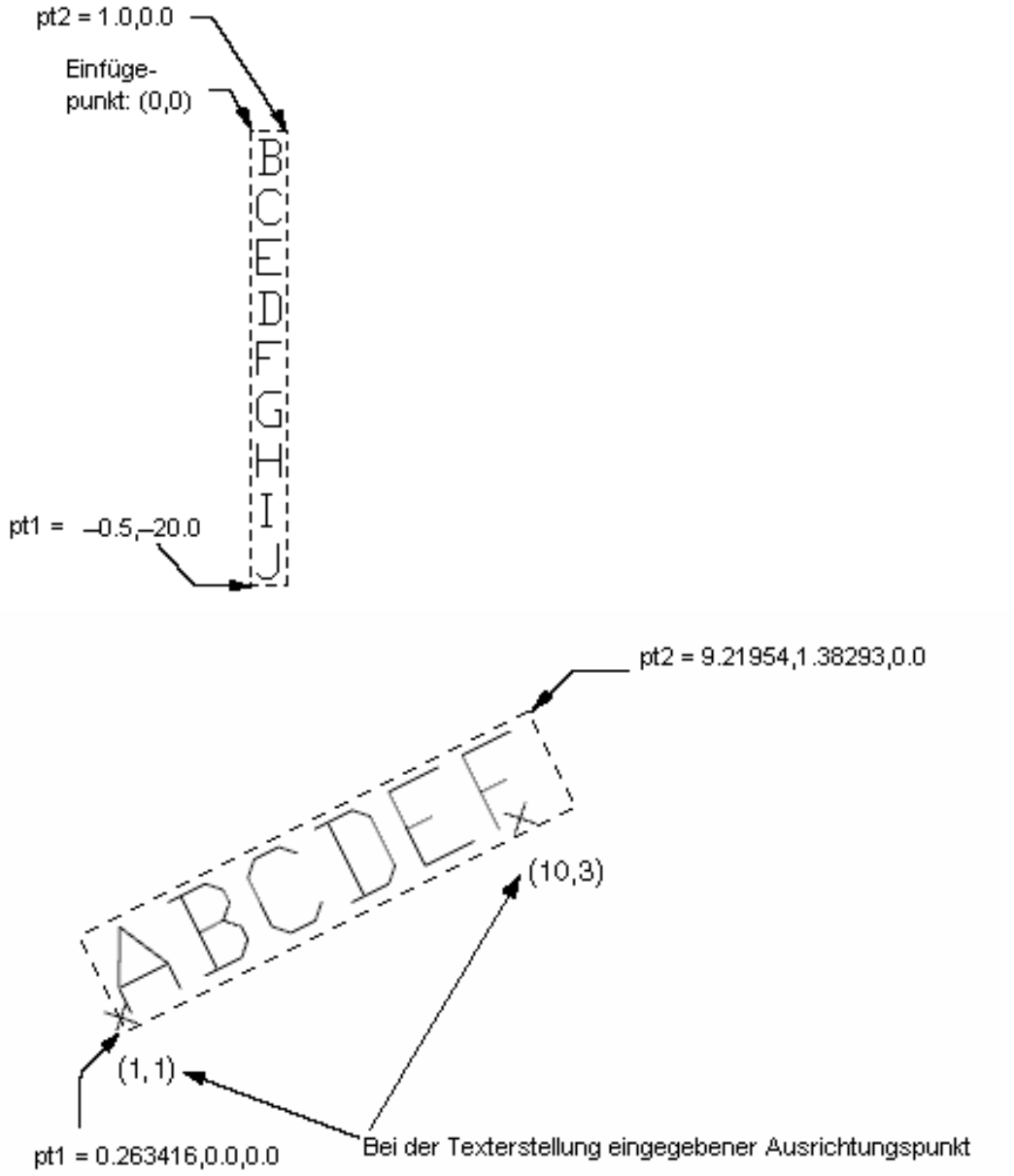
In der folgenden Abbildung sehen Sie die Ergebnisse der Anwendung von **textbox** auf ein Textobjekt mit der Höhe 1.0. Ebenfalls können Sie die Basislinie und den Einfügepunkt des Texts erkennen.



Von textbox zurückgegebene Punkte

Ist der Text vertikal oder gedreht, ist `pt1` weiterhin die untere linke und `pt2` weiterhin die obere rechte Ecke; der untere linke Punkt hat möglicherweise negative Koordinaten.

Die folgende Abbildung zeigt die Punktwerte (`pt1` und `pt2`), die **textbox** für Beispiele von vertikalem und ausgerichtetem Text zurückgibt. In beiden Beispielen ist die Höhe der Buchstaben 1.0. (Bei ausgerichtetem Text wird die Buchstabenhöhe den Ausrichtungspunkten angepaßt.)



Vertikaler und ausgerichteter Text

Auch bei der Verwendung von vertikalen Schriftstilen werden die Punkte wie bei horizontalen Stilen in der Reihenfolge von links nach rechts und von unten nach oben zurückgegeben, so daß die erste Punktliste negative Koordinatenwerte enthält.

Unabhängig von Textausrichtung und -stil werden die von **textbox** zurückgegebenen Punkte so konvertiert, daß sich der Texteinfügepunkt (Gruppencode 10) auf den Ursprung des Objektkoordinatensystems (OKS) für das verknüpfte Textobjekt bezieht. Auf diesen Punkt können Sie sich beziehen, indem Sie die von **textbox** zurückgegebenen Koordinaten in Punkte konvertieren, welche die tatsächlichen Grenzen des Texts definieren. Die folgenden Beispielroutinen verwenden beide **textbox**, um ein Rechteck unabhängig von der Ausrichtung um ausgewählten Text zu plazieren.

Die erste Routine verwendet die Funktion **textbox**, um ein Rechteck um das gewählte Textobjekt zu zeichnen.

```
(defun C:TBOX ( / textent tb ll ur ul lr)
  (setq textent (car (entsel "\nText wählen: ")))
  (command "bks" "Objekt" textent)
  (setq tb (textbox (list  (cons -1 textent)))
        ll (car tb)
        ur  (cadr tb)
        ul  (list (car ll) (cadr ur))
        lr (list (car ur) (cadr ll))
  )
```

```
    (command "plinie" ll lr ur ul "Schließen")
    (command "bks" "p")
    (princ)
  )
```

Die zweite Routine, die folgt, führt dieselbe Aufgabe wie die erste Routine durch, indem sie die geometrischen Berechnungen mit den AutoLISP-Funktionen **sin** und **cos** durchführt. Das Ergebnis ist nur dann korrekt, wenn das aktuelle BKS parallel zur Ebene des Textobjekts ist.

```
  (defun C:TBOX2 ( / textel ang sinrot cosrot
                     t1 t2 p0 p1 p2 p3 p4)
    (setq textel (entget (car (entsel "\nText wählen: "))))
    (setq p0 (cdr (assoc 10 textel))
          ang (cdr (assoc 50 textel))
          sinrot (sin ang)
          cosrot (cos ang)
          t1 (car (textbox textel))
          t2 (cadr (textbox textel))
          p1 (list
            (+ (car p0)
              (- (* (car t1) cosrot)(* (cadr t1) sinrot))
            )
            (+ (cadr p0)
              (+ (* (car t1) sinrot)(* (cadr t1) cosrot))
            )
        )
        p2 (list
          (+ (car p0)
            (- (* (car t2) cosrot)(* (cadr t1) sinrot))
          )
          (+ (cadr p0)
            (+ (* (car t2) sinrot)(* (cadr t1) cosrot))
          )
        )
        p3 (list
          (+ (car p0)
            (- (* (car t2) cosrot)(* (cadr t2) sinrot))
          )
          (+ (cadr p0)
            (+ (* (car t2) sinrot)(* (cadr t2) cosrot))
          )
        )
        p4 (list
          (+ (car p0)
            (- (* (car t1) cosrot)(* (cadr t2) sinrot))
          )
          (+ (cadr p0)
            (+ (* (car t1) sinrot)(* (cadr t2) cosrot))
          )
        )
    )
    (command "plinie" p1 p2 p3 p4 "c")
    (princ)
  )
```

8.5 Konvertierungen

Bei den in diesem Abschnitt beschriebenen Funktionen handelt es sich um Dienstprogramme zur Konvertierung von Datentypen und Einheiten. Es gibt folgende Konvertierungsfunktionen:

angtof	atof	cvunit	rtos
angtos	atoi	distof	trans
ascii	chr	itoa	

Die Konvertierungsfunktionen werden auch in Kapitel 13 beschrieben.

8.5.1 Zeichenkettenkonvertierung

Die Funktionen **rtos** (reelle Zahl in Zeichenkette) und **angtos** (Winkel in Zeichenkette) konvertieren in AutoCAD verwendete numerische Werte in Zeichenkettenwerte, die als Ausgabe oder als Textdaten verwendet werden können. Die Funktion **rtos** konvertiert einen reellen Wert, die Funktion **angtos** konvertiert einen Winkel. Das Format der Ergebniszeichenfolge wird durch den Wert der AutoCAD-Systemvariablen gesteuert: Einheiten und Genauigkeit werden für reelle (lineare) Werte durch LUNITS und LUPREC und für Winkelwerte durch AUNITS und AUPREC festgelegt. Die Bemaßungsvariable DIMZIN steuert für beide Funktionen, wie vorangestellte und nachgestellte Nullen in die Ergebniszeichenkette geschrieben werden.

Die folgenden Codefragmente zeigen Aufrufe von **rtos** und die zurückgegebenen Werte (unter der Annahme, daß die Variable DIMZIN gleich 0 ist). Die Genauigkeit (das dritte Argument von **rtos**) wird auf 4 Stellen im ersten Aufruf und auf 2 Stellen in den anderen Aufrufen gesetzt.

```
(setq x 17.5)
(setq str "\nWert formatiert als ")

(setq fmtval  (rtos x 1 4))          Modus 1 = wissenschaftlich
(princ (strcat str fmtval))          zeigt an    Wert formatiert als 1.7500E+01

(setq fmtval  (rtos x 2 2))          Modus 2 = dezimal
(princ (strcat str fmtval))          zeigt an    Wert formatiert als 17.50

(setq fmtval (rtos x 3 2))           Modus 3 = engineering
(princ (strcat str fmtval))          zeigt an  Wert formatiert als 1'-5.50"

(setq fmtval (rtos x 4 2))           Modus 4 = architectural
(princ (strcat str fmtval))          zeigt an  Wert formatiert als 1'-5 1/2"

(setq fmtval (rtos x 5 2))           Modus 5 = bruch
(princ (strcat str fmtval))          zeigt an  Wert formatiert als 17 1/2
```

Ist die Systemvariable UNITMODE auf 1 gesetzt, um Einheiten so anzuzeigen, wie sie eingegeben werden, unterscheidet sich die von **rtos** zurückgegebene Zeichenkette bei engineering (*Modus* ist gleich 3), architectural (*Modus* ist gleich 4) und gebrochenen (*Modus* ist gleich 5) Einheiten. So würden die ersten beiden Zeilen der obigen Beispielausgabe unverändert bleiben, die letzten drei Zeilen jedoch würden folgendermaßen aussehen:

Wert formatiert als 1'5.50"
Wert formatiert als 1'5-1/2"
Wert formatiert als 17-1/2

Da die Funktion **angtos** die Systemvariable ANGBASE berücksichtigt, gibt der folgende Code immer "0" zurück.

```
(angtos (getvar "angbase"))
```

Es gibt keine AutoLISP-Funktion, welche die Drehung von ANGBASE bezogen auf Null (Ost) als Zeichenkettenversion im aktuellen Modus und mit der aktuellen Genauigkeit zurückgibt.

Wollen Sie das Maß der Drehung von ANGBASE bezogen auf AutoCAD-Null (Ost) oder die Größe eines beliebigen Winkels ermitteln, können Sie eine der folgenden Methoden wählen:

- Addieren Sie den gewünschten Winkel zu dem aktuellen Wert von ANGBASE, und überprüfen Sie, ob der Betrag des Ergebnisses größer als 2p (2 * Pi) ist. Ist dies der Fall, subtrahieren Sie 2p; ist das Ergebnis negativ, addieren Sie p. Benutzen Sie die Funktion **angtos** mit diesem Ergebnis als Argument.

- Speichern Sie den Wert von ANGBASE in einer temporären Variablen, setzen Sie ANGBASE auf 0 und werten Sie die Funktion **angtos** aus. Setzen Sie anschließend ANGBASE wieder auf den ursprünglichen Wert.

Auch das Subtrahieren des Ergebnisses von `(atof (angtos 0))` von 360 Grad (2p im Bogenmaß oder 400 Neugrad) ergibt die Drehung von ANGBASE von 0 ausgehend.

Die Funktion **distof** (Abstand zum Gleitkomma) ist die Ergänzungsfunktion zu **rtos**, so daß die folgenden Aufrufe, welche die im vorhergehenden Beispiel erstellten Zeichenketten verwenden, alle denselben Wert zurückgeben: 17.5. (Beachten Sie die Verwendung des umgekehrten Schrägstrichs (\) in den Modi 3 und 4.)

`(distof "1.7500E+01" 1)`	*Modus 1 = wissenschaftlich*
`(distof "17.50" 2)`	*Modus 2 = dezimal*
`(distof "1'-5.50\"" 3)`	*Modus 3 = engineering*
`(distof "1'-5 1/2\"" 4)`	*Modus 4 = architectural*
`(distof "17 1/2" 5)`	*Modus 5 = bruch*

Die folgenden Codefragmente zeigen ähnliche Aufrufe von **angtos** und die zurückgegebenen Werte (weiterhin soll DIMZIN gleich 0 sein). Die Genauigkeit (das dritte Argument von **angtos**) wird im ersten Aufruf auf 0 gesetzt, auf 4 Stellen in den nächsten drei Aufrufen und auf 2 Stellen im letzten Aufruf.

```
(setq ang 3.14159 kette2 "\nWinkel formatiert als ")
```

`(setq fmtval (angtos ang 0 0))`	*Modus 0 = Grad*
`(princ (strcat str2 fmtval))`	*zeigt an* `Winkel formatiert als 180`
`(setq fmtval (angtos ang 1 4))`	*Modus 1 = Altgrad*
`(princ (strcat str2 fmtval))`	*zeigt an* `Winkel formatiert als 180d0'0"`
`(setq fmtval (angtos ang 2 4))`	*Modus 2 = Neugrad*
`(princ (strcat str2 fmtval))`	*zeigt an* `Winkel formatiert als 200.000g`
`(setq fmtval (angtos ang 3 4))`	*Modus 3 = Bogenmaß*
`(princ (strcat str2 fmtval))`	*zeigt an* `Winkel formatiert als 3.1416r`
`(setq fmtval (angtos ang 4 2))`	*Modus 4 = Feldmaß*
`(princ (strcat str2 fmtval))`	*zeigt an* `Winkel formatiert als W`

Die Systemvariable UNITMODE wirkt sich auch auf die von **angtos** zurückgegebenen Zeichenketten aus, wenn eine Zeichenkette im Feldmaß zurückgegeben wird (*Modus* ist gleich 4). Ist UNITMODE gleich 0, kann die zurückgegebene Zeichenkette Leerstellen enthalten (beispielsweise `"N 45d E"`); ist UNITMODE gleich 1, enthält die Zeichenkette keine Leerstellen (beispielsweise `"N45dE"`).

Die Funktion **angtof** ist die Ergänzungsfunktion zu **angtos**, so daß alle folgenden Aufrufe denselben Wert zurückgeben: 3.14159.

`(angtof "180" 0)`	*Modus 0 = Grad*
`(angtof "180d0'0\"" 1)`	*Modus 1 = Altgrad*
`(angtof "200.0000g" 2)`	*Modus 2 = Neugrad*
`(angtof "3.14159r" 3)`	*Modus 3 = Bogenmaß*
`(angtof "W" 4)`	*Modus 4 = Feldmaß*

Geben Sie eine Zeichenkette ein, die einen Abstand in Fuß und Zoll oder einen Winkel in Altgrad angibt, müssen Sie den Anführungszeichen einen umgekehrten Schrägstrich voranstellen (\"), damit es nicht wie das Ende einer Zeichenkette wirkt. Dieses Verfahren wird in den vorangehenden Beispielen von **angtof** und **distof** gezeigt.

8.5.2 Winkelkonvertierung

Wenn Ihre Anwendung Winkelwerte von Bogenmaß in Grad konvertieren soll, können Sie die Funktion **angtos** verwenden. Diese Funktion gibt eine Zeichenkette zurück, die dann mit der Funktion **atof** in einen Gleitkommawert konvertiert wird.

```
(setq pt1 '(1 1) pt2 '(1 2))
(setq rad (angle pt1 pt2))
setq deg (atof (angtos rad 0 2)))
```
gibt zurück `90.0`

Nun könnte es vorteilhaft sein, eine Funktion **DTR** (Grad in Bogenmaß) in Ihre Anwendung aufzunehmen. Der folgende Code stellt dies dar.

```
; Konvertiere Winkel von Bogenmaß in Grad
(defun RTD (r)
  (* 180.0 (/ r pi))
)
```

Wenn diese Funktion festgelegt ist, können Sie die Funktion **DTR** durch Ihre gesamte Anwendung hindurch verwenden, wie bei

```
(setq deg (DTR rad))
```
gibt zurück `90.0`

Sie können auch von Bogenmaß nach Grad konvertieren. Der folgende Code stellt dies dar.

```
; Konvertiere Winkel von Grad nach Bogenmaß
(defun DTR (d)
  (* pi (/ d 180.0))
)
```

8.5.3 ASCII-Code-Konvertierung

AutoLISP stellt zur Bearbeitung von dezimalen ASCII-Codes die Funktionen **ascii** und **chr** zur Verfügung. Die Funktion **ascii** gibt den dezimalen ASCII-Wert zurück, der mit einer Zeichenkette verknüpft ist, und **chr** gibt das Zeichen zurück, das mit einem dezimalen ASCII-Wert verknüpft ist.

Um die Zeichen Ihres Systems mit ihren Codes in dezimaler, oktaler und hexadezimaler Form sehen zu können, speichern Sie den folgenden AutoLISP-Code in eine Datei *ascii.lsp*. Laden Sie dann diese Datei und geben Sie den neuen ASCII-Befehl an der Eingabeaufforderung ein. Der Befehl gibt die ASCII-Codes auf dem Bildschirm aus und schreibt sie in die Datei *ascii.txt*. **C:ASCII** verwendet eine andere Funktion **BASE**. Dieses Konvertierungsprogramm ist bei anderen Anwendungen nützlich.

```
; BASE konvertiert von einer dezimalen Ganzzahl in eine Zeichenkette in einer
anderen Basis
(defun BASE ( bas int /ret yyy zot )
  (defun zot (i1 i2 / xxx)
    (if (> (setq xxx (rem i2 i1)) 9)
      (chr (+ 55 xxx))
      (itoa xxx)
    )
  )
  (setq ret (zot bas int) yyy (/ int bas))
  (while (>= yyy bas)
    (setq ret (strcat (zot bas yyy) ret))
    (setq yyy (/ yyy bas))
  )
  (strcat (zot bas yyy) ret)
)

(defun C:ASCII ( / prüf out ct Code dez okt hex)
  (initget "Ja")
  (setq prüf (getstring "\n Schreibe in ASCII.TXT, fortfahren? <Y>: "))
  (if (or (= prüf "Ja")(= prüf ""))
    (progn
      (setq out (open "ascii.txt" "w") chk 1 Code 0 ct 0)
      (princ "\n \n CHAR  DEZ  OCT  HEX \n")
      (princ "\n \n CHAR  DEZ  OCT  HEX \n" out)
      (while prüf
        (setq dez (strcat "  " (itoa Code))
          oct (base 8 Code) hex (base 16 Code))
        (setq dez (substr dez (- (strlen dez) 2) 3))
        (if (< (strlen oct) 3)(setq oct (strcat "0" oct)))
        (princ (strcat "\n" (chr Code) "     " dez "  "
```

```
          oct "   " hex ) )
        (princ (strcat "\n" (chr Code) "      " dez "   "
          oct "   " hex ) out)
        (cond
          ((= code 255)(setq prüf nil))
          ((= ct 20)
            (setq xxx (getstring
               "\n \nDrücken Sie 'X' zum Verlassen oder eine beliebige "))
            Taste, um fortzufahren: (if (= (strcase xxx) "X")
              (setq chk nil)
              (progn
                (setq ct 0)
                (princ "\n \n CHAR  DEZ  OCT  HEX \n")
              )
            )
          )
        )
        (setq ct (1+ ct) Code (1+Code))
      )
      (close out)
      (setq out nil))
    )
  )
  (princ)
)
```

8.5.4 Konvertierung von Einheiten

Die Datei *acad.unt* definiert verschiedene Konvertierungen von Einheiten, zum Beispiel für die Umwandlung von Meilen in Kilometer, Fahrenheit in Celsius usw. Die Funktion **cvunit** wandelt einen in einem Einheitensystem ausgedrückten Wert in den äquivalenten Wert eines anderen Systems um. Die zwei Einheitensysteme werden von Zeichenketten bestimmt, welche die Ausdrücke der in *acad.unt* definierten Einheiten enthalten.

Die Funktion **cvunit** konvertiert *keine* inkompatiblen Dimensionen. Beispielsweise konvertiert sie Zoll nicht in Gramm.

Konvertiert **cvunit** eine Einheit während einer Arbeitssitzung mit dem Zeicheneditor zum ersten Mal, muß die Zeichenkette, welche die Einheit bestimmt, in *acad.unt* gesucht werden. Sind in Ihrer Anwendung viele Werte zu konvertieren, ist es effektiver, zunächst den Wert 1.0 durch einen einzelnen Aufruf von **cvunit** zu konvertieren und den zurückgegebenen Wert dann als Skalierfaktor für die folgenden Konvertierungen zu verwenden. Dieses Vorgehen eignet sich für alle Einheiten, die in *acad.unt* definiert sind, außer für Temperaturskalen, da diese sowohl eine Nullpunktverschiebung als auch einen Skalierfaktor benötigen.

8.5.4.1 Konvertieren von Zoll in Meter

Sind die Einheiten der aktuellen Zeichnung engineering oder architectural (Fuß und Zoll), konvertiert die folgende Routine einen benutzerdefinierten Abstand von Zoll in Meter:

```
(defun C:I2M ( / eng_len metric_len eng metric)
  (princ "\nKonvertiere Zoll in Meter. ")
  (setq engl_länge
    (getdist "\nGeben Sie eine Entfernung in Zoll ein: "))
  (setq metrische_länge  (cvunit engl_länge "Zoll" "Meter"))
  (setq engl (rtos engl_länge 2 4)
        metrisch (rtos metrische_länge 2 4))
  (princ)
    (strcat "\n\t" engl " Zoll = " metrisch " Meter."))
  (princ)
)
```

8.5.4.2 Definitionsdatei für Maßeinheiten

Mit der Definitionsdatei für Maßeinheiten *acad.unt* können Sie Faktoren zur Konvertierung von Daten von einem Einheitensystem in ein anderes definieren. Die Definitionen in *acad.unt* sind im Format ASCII und werden von der Funktion **cvunit** zur Konvertierung von Einheiten verwendet.

Mit Hilfe eines Texteditors können Sie die Definitionen neuer Einheiten in *acad.unt* einfügen. Eine Definition besteht aus zwei Zeilen in der Datei--dem *Einheitennamen* und der *Einheitendefinition*. In der ersten Zeile muß ein Sternchen (*) in der ersten Spalte dem Namen der Einheit vorangehen. Dem Namen einer Einheit können Sie, getrennt durch Kommas, mehrere Abkürzungen oder verschiedene Schreibweisen zuordnen. Tritt der Name einer Einheit im Singular und Plural auf, können Sie dies mit folgendem Format festlegen:

```
*[ [Stamm] [ ( [singular.] plural) ] ]...
```

Sie können mehrere Ausdrücke für Singular und Plural bestimmen. Diese müssen sich nicht am Ende des Worts befinden, und eine Pluralform ist nicht erforderlich.

```
*inch(es)
*milleni(um.a)
*f(oot.eet) oder (foot.feet)
```

In der Zeile unter **Einheit_name* wird die Einheit als *Basiseinheit* oder *abgeleitete Einheit* definiert.

Basiseinheiten

Eine Basiseinheit ist ein aus Konstanten bestehender Ausdruck. Beginnt die Zeile unter **Einheit_ name* mit einem anderen Zeichen als dem Gleichheitszeichen (=), definiert sie Basiseinheiten. Sie besteht aus fünf Ganzzahlen und zwei reellen Zahlen in folgender Form:

c, e, h, k, m, r1, r2

Die fünf Ganzzahlen entsprechen den Exponenten dieser fünf Konstanten:

c Lichtgeschwindigkeit im Vakuum

e Elektronenladung

h Plancksches Wirkungsquantum

k Boltzmann-Konstante

m Ruhemasse des Elektrons

Diese Exponenten definieren als Gruppe die Dimension einer Einheit: Länge, Masse, Zeit, Volumen usw.

Die erste reelle Zahl (r1) ist ein Faktor, die zweite (r2) stellt eine zusätzliche Nullpunktverschiebung dar (nur für Temperaturkonvertierungen). Die Definition einer Basiseinheit ermöglicht verschiedene Schreibweisen der Einheit (z.B. *meter* und *metre*); es wird keine Unterscheidung von Groß- und Kleinschreibung vorgenommen. Nachfolgend finden Sie ein Beispiel für die Definition einer Basiseinheit.

```
*meter(s),metre(s),m
-1,0,1,0,-1,4.1214856408e11,0
```

In diesem Beispiel definieren folgende Konstanten einen Meter:

$$\left(\frac{1}{c} \times h \times \frac{1}{m}\right) \times (4.1214856 \times 10^{11})$$

Abgeleitete Einheiten

Eine abgeleitete Einheit wird durch andere Einheiten ausgedrückt. Beginnt die auf **Einheit name* folgende Zeile mit einem Gleichheitszeichen (=), definiert sie abgeleitete Einheiten. Gültige Operatoren in diesen Definitionen sind * (Multiplikation), / (Division), + (Addition), − (Subtraktion) und ^ (Potenzierung). Sie können eine vordefinierte Einheit bestimmen, indem Sie sie benennen und Abkürzungen verwenden, falls vorhanden. Die Elemente einer Formel werden multipliziert, bis ein anderer arithmetischer Operator angegeben wird. Beispielsweise definiert die Einheitendatenbank die dimensionslosen Faktoren, so daß Sie eine Einheit wie "micro-inches" durch Eingabe von **`micro inch`** festlegen können. Nachfolgend finden Sie Beispiele für Definitionen von abgeleiteten Einheiten.

```
; Flächenmaße

*township(s)
=93239571.456 meter^2
```

Ein Township ist als 93,239,571.456 Quadratmeter definiert.

```
; Elektromagnetische Einheiten
```

```
*volt(s),v
=watt/ampere
```

In diesem Beispiel wird Volt als Watt geteilt durch Ampere definiert. In *acad.unt* sind Watt und Ampere als Basiseinheiten definiert.

Benutzerkommentare

Um Kommentare einzufügen, beginnen Sie die Zeile mit einem Semikolon. Der Kommentar setzt sich bis zum Ende der Zeile fort.

```
; Diese ganze Zeile ist ein Kommentar.
```

Rufen Sie die Datei *acad.unt* auf, wenn Sie weitere Informationen und Beispiele wünschen.

8.5.5 Konvertierung von Koordinatensystemen

Die Funktion **trans** konvertiert einen Punkt oder eine Verschiebung von einem Koordinatensystem in ein anderes. Sie benötigt einen Punkt als Argument *pt*, welches entweder als dreidimensionaler (3D-)Punkt oder als 3D-Verschiebungsvektor interpretiert werden kann, entsprechend dem Wert eines Verschiebungsarguments namens *versch*. Das Argument *versch* muß ungleich Null sein, wenn *pt* als Verschiebungsvektor behandelt werden soll, andernfalls wird *pt* als Punkt interpretiert. Ein *von*-Argument bezeichnet das Koordinatensystem, in dem *pt* angegeben wird, ein *nach*-Argument das gewünschte Koordinatensystem. Es folgt die Syntax für die Funktion **trans**.

(trans *pt von nach [Anz]*)

Sie können die folgenden AutoCAD-Koordinatensysteme durch Argumente *von* und *nach* bezeichnen.

WKS

Weltkoordinatensystem: das "Bezugs-" Koordinatensystem. Alle anderen Koordinatensysteme werden relativ zum WKS definiert, das sich niemals ändert. Werte, die im WKS angegeben werden, bleiben bei Änderungen anderer Koordinatensysteme unverändert.

BKS

Benutzerkoordinatensystem: das "Arbeits"- Koordinatensystem. Der Benutzer definiert ein solches Koordinatensystem, um Zeichenaufgaben einfacher durchführen zu können. Alle an AutoCAD-Befehle übergebenen Punkte einschließlich derer, die von AutoLISP-Routinen und externen Funktionen zurückgegeben werden, beziehen sich auf das aktuelle BKS (falls der Benutzer sie nicht mit einem Sternchen in der Befehlszeile kennzeichnet). Soll Ihre Anwendung Koordinaten des WKS, OKS oder AKS an AutoCAD-Befehle übergeben, müssen Sie sie zunächst mit der Funktion **trans** in Werte des BKS konvertieren.

OKS

Objektkoordinatensystem: von **entget** zurückgegebene Punktwerte werden in diesem Koordinatensystem bezogen auf das Objekt selbst ausgedrückt. Diese Punkte werden für gewöhnlich je nach beabsichtigter Verwendung des Objekts in das WKS, das aktuelle BKS oder das AKS konvertiert. Umgekehrt müssen Punkte in OKS-Koordinaten umgewandelt werden, bevor sie mit **entmod** oder **entmake** in die Datenbank geschrieben werden. Dies ist auch als *Elementkoordinatensystem* bekannt.

AKS

Anzeigekoordinatensystem: das Koordinatensystem, in das Objekte konvertiert werden, bevor sie auf dem Bildschirm angezeigt werden. Der Ursprung des AKS ist der in der AutoCAD-Systemvariablen TARGET gespeicherte Punkt, seine *Z*-Achse ist die Blickrichtung. Anders ausgedrückt ist ein Ansichtsfenster immer eine Draufsicht seines AKS. Diese Koordinaten können verwendet werden, um festzulegen, an welcher Position die Anzeige für den AutoCAD-Benutzer erfolgt.

Sind die Ganzzahlcodes *von* und *nach* 2 und 3, in beliebiger Reihenfolge, gibt 2 das AKS für das aktuelle Ansichtsfenster des Modellbereichs an und 3 das AKS für den Papierbereich (PBAKS). Wird der Code 2 mit einem anderen Ganzzahlwert als 3 verwendet (oder einer anderen Angabe zur Festlegung des Koordinatensystems), wird angenommen, daß das AKS des aktuellen Raums, Papierbereich oder Modellbereich, angezeigt werden soll. Das andere Argument wird als Angabe eines Koordinatensystems im aktuellen Raum interpretiert.

PSDCS

Papierbereichs-AKS: dieses Koordinatensystem kann *nur* in das oder von dem aktuell aktiven AKS im Modellbereich des Ansichtsfensters konvertiert werden. Hierbei handelt es sich im wesentlichen um eine 2D-Transformation, wobei die *X* und *Y*-Koordinaten immer skaliert und verschoben werden, wenn das Argument

`versch` den Wert 0 hat. Die *Z*-Koordinate *wird* skaliert, jedoch niemals einer Parallelverschiebung unterzogen; sie kann deshalb zur Ermittlung des Skalierfaktors zwischen den beiden Koordinatensystemen verwendet werden. Das PBAKS (Ganzzahlcode 2) kann nur in den aktuellen Modellbereich des Ansichtsfensters konvertiert werden: ist das Argument `von` gleich 3, muß das Argument `nach` gleich 2 sein und umgekehrt.

Die Argumente `von` und `nach` können ein Koordinatensystem auf eine der folgenden Arten festlegen:

- Als ein Ganzzahlcode, der das WKS, das aktuelle BKS oder das aktuelle AKS (des aktuellen Ansichtsfensters oder Papierbereichs) bezeichnet.
- Als ein Elementname, der von einer der in Kapitel 9, "Auswahlsatz-, Objekt- und Symboltabellen funktionen"."Statuszeilen- konfiguration und Makroprogrammiersprache DIESEL". beschriebenen Elementnamen- oder Auswahlsatzfunktionen zurückgegeben wird. Dieser bestimmt das OKS des genannten Objekts. Bei planaren Objekten kann das OKS vom WKS abweichen, wie im *AutoCAD Benutzerhandbuch* beschrieben. Besteht keine Abweichung, entspricht die Konvertierung zwischen OKS und WKS einer Abbildung des OKS auf sich selbst.
- Als ein 3D-Hochzugsvektor. Hochzugsvektoren werden immer in Weltkoordinaten ausgedrückt; ein Hochzugsvektor von (0,0,1) bezeichnet das WKS selbst.

In der folgenden Tabelle finden Sie die gültigen Ganzzahlcodes, die als Argumente `von` und `nach` verwendet werden können.

Codes der Koordinatensysteme

Code	Koordinatensystem
0	Welt (WKS)
1	Benutzer (aktuelles BKS)
2	Anzeige; mit Code 0 oder 1 AKS des aktuellen Ansichtsfensters, mit Code 3 AKS des aktuellen Modellbereich-Ansichtsfensters
3	Papierbereich AKS, PBAKS (nur bei Angabe von Code 2)

Im folgenden Beispiel wird ein Punkt vom WKS in das aktuelle BKS konvertiert.

```
(setq pt '(1.0 2.0 3.0))
(setq cs_von 0)                    WKS
(setq cs_nach 1)                   BKS
(trans pt cs_von cs_nach 0)          disp = 0 gibt an, daß pt ein Punkt ist
```

Wird das aktuelle BKS um 90 Grad gegen den Uhrzeigersinn um die *Z*-Achse des WKS gedreht, gibt der Aufruf von **`trans`** den Punkt (2.0,-1.0,3.0) zurück. Vertauschen Sie jedoch die `von`- und `nach` -Werte, erreichen Sie ein anderes Ergebnis, wie im folgenden Code gezeigt.

```
(trans pt cs_nach cs_von 0) ; das Ergebnis ist  (-2.0,1.0,3.0)
```

8.5.5.1 Punktkonvertierungen

Führen Sie mit der Funktion **`trans`** Punktkonvertierungen durch und wollen diesen Programmteil beschleunigen, können Sie Ihre eigene Konvertierungsmatrix auf der AutoLISP-Seite erstellen, indem Sie **`trans`** einmal verwenden, um die "Basisvektoren" 0 0 0), (1 0 0), (0 1 0) und (0 0 1) zu konvertieren. Das Schreiben von Funktionen zur Matrizenmultiplikation in AutoLISP kann schwierig sein, so daß sich der Aufwand nur lohnt, wenn Ihr Programm viele Konvertierungen durchführt.

8.6 Dateibearbeitung

AutoLISP stellt folgende Funktionen zur Dateibearbeitung und zur Dateneingabe und -ausgabe zur Verfügung:

close	help	read-line	write-line
findfile	open	setfunhelp	
getfiled	read-char	write-char	

Beispiele für die Funktionen **`findfile`**, **`getfiled`** und **`help`** finden Sie in diesem Abschnitt. Die anderen Funktionen zur Dateibearbeitung werden in Kapitel 13 erläutert.

8.6.1 Dateisuche

Eine Anwendung kann zur Suche nach einem bestimmten Dateinamen die Funktion **`findfile`** verwenden. Die Anwendung kann das zu durchsuchende Verzeichnis festlegen oder den aktuellen AutoCAD-Bibliothekspfad benutzen.

Im folgenden Codefragment durchsucht **`findfile`** den AutoCAD-Bibliothekspfad nach dem gewünschten Dateinamen:

```
(setq datname "refc.dwg")
(setq Dat (findfile datname))
(if Dat
  (setq datname Dat)
  (princ (strcat "\nDatei "datname" nicht gefunden: " ))
)
```

Ist der Aufruf von **`findfile`** erfolgreich, erhält die Variable `datname` als Wert die Zeichenkette mit dem vollständigen Pfadnamen, wie beispielsweise

```
"/heim/arbeit/dat/refc.dwg"
```

Anmerkung Wenn Sie einen DOS-Pfadnamen angeben, müssen Sie diesen für AutoLISP erkennbar machen, indem Sie dem umgekehrten Schrägstrich (`\`) einen weiteren umgekehrten Schrägstrich voranstellen. Alternativ können Sie den Schrägstrich (`/`) als Verzeichnistrennzeichen verwenden.

Die Funktion **`getfiled`** zeigt ein Dialogfeld mit Dateien des angegebenen Dateityps im festgelegten Verzeichnis an. Auf diese Weise können AutoLISP-Routinen auf das Dialogfeld get file von AutoCAD zugreifen.

Ein Aufruf von **`getfiled`** benötigt vier Argumente, um das Aussehen und die Funktionsweise des Dialogfelds zu bestimmen. Die Anwendung muß die folgenden Zeichenkettenwerte festlegen, von denen jeder den Wert `nil` haben kann: einen Titel, der oben im Dialogfeld positioniert wird; einen Standarddateinamen, der im Eingabefeld unten im Dialogfeld angezeigt wird; und einen Dateityp, der angibt, welche Dateien anfangs im Listenfeld zur Auswahl angezeigt werden. Das letzte Argument ist ein Ganzzahlwert, der angibt, wie die Auswahl einer Datei vom Dialogfeld verarbeitet wird.

Diese einfache Routine verwendet **`getfiled`**, um Ihnen Ihre Verzeichnisstruktur anzuzeigen und Sie eine Datei auswählen zu lassen:

```
(defun C:DDIR ( )
  (setq ddat (getfiled "Verzeichnisliste" "" "" 2))
  (princ (strcat "\nVariable 'ddat' auf ausgewählte Datei gesetzt " ddat))
  (princ)
)
```

Dies ist ein sehr nützlicher Befehl. Die Variable `ddat` erhält als Wert die ausgewählte Datei und kann dann von anderen AutoLISP-Funktionen oder als Antwort auf eine Eingabeaufforderung verwendet werden, die einen Dateinamen erwartet. Geben Sie **!ddat** ein, um die Variable als Antwort auf eine Eingabeaufforderung zu verwenden.

Anmerkung **!ddat** *kann nicht* in einem Dialogfeld verwendet werden. Eine solche Angabe ist nur in einer Befehlszeile gültig.

Weitere Information finden Sie unter **"`getfiled`"** in Kapitel 13.

8.6.2 Zugriff auf Hilfedateien

Die Funktion **help** ermöglicht den Zugriff sowohl auf die AutoCAD-Hilfedateien (*.ahp*) als auch auf die Windows-Hilfedateien (*.hlp*). Je nach Erweiterung der Hilfedatei ruft die Funktion **help** die AutoCAD- oder Windows-Hilfeanwendung mit der angegebenen Datei auf. Mit dieser Funktion ermöglichen Sie in Ihren Anwendungen den Zugriff auf Hilfeinformationen. Das folgende Codefragment ruft die Standard-AutoCAD-Hilfedatei auf und gibt Informationen zum Befehl LINIE an.

```
(help "" "linie")
```

Sie können eine Hilfedatei erzeugen, die Informationen zu Ihren Anwendungen oder zu den in Ihrer beruflichen Praxis verwendeten Prozeduren enthält. Der folgende benutzerdefinierte Befehl zeigt Ihre Hilfedatei *MehrHilf.ahp* an.

```
(defun C:MYHELP ( )
  (help "MehrHilf.ahp")
  (princ)
)
```

Informationen zur Erstellung und Modifizierung von Hilfedateien finden Sie unter "Anpassen der Online-Dokumentation."

Die Funktion **setfunhelp** bietet Hilfe bei benutzerdefinierten Befehlen. Wenn Sie nach der Definition Ihres neuen Befehls **setfunhelp** aufrufen, wird eine spezielle Hilfedatei und ein spezielles Hilfethema mit diesem Befehl verknüpft. Das folgende Beispiel ordnet das Hilfethema Meinbefehl in der Datei *morehelp.ahp* dem benutzerdefinierten Befehl MEINBEFEHL zu.

```
(defun C:MEINBEFEHL ( )
  .
  . Befehlsdefinition
  .
)
(setfunhelp c:meinbefehl "MehrHilf.ahp" "MeinBefehl")
```

8.7 Zugriff auf Geräte und Steuerung

AutoLISP stellt Funktionen zum Zugriff auf die Daten der verschiedenen Eingabegeräte zur Verfügung. Es stehen die folgenden Funktionen für den Zugriff auf und die Steuerung von Geräten zur Verfügung:

grread tablet

Die folgenden Funktionen zur Dateibearbeitung können auch Eingaben aus dem Tastaturpuffer lesen. (Weitere Informationen zu diesen Funktionen finden Sie im Abschnitt "Dateibearbeitung.")

read-char read-line

8.7.1 Zugriff auf Benutzereingaben

Die Funktion **grread** gibt "unbearbeitete" Benutzereingaben zurück, unabhängig davon, ob sie über die Tastatur oder über ein Zeigegerät (Maus oder Digitalisiergerät) erfolgten; aktiviert der Aufruf von **grread** Tracking, gibt die Funktion eine digitalisierte Koordinate zurück, die beispielsweise zum Ziehen benutzt werden kann.

Anmerkung Es gibt keine Garantie dafür, daß Anwendungen, die **grread** aufrufen, aufwärtskompatibel sind. Da sie von der aktuellen Hardware-Konfiguration abhängig sind, arbeiten Anwendungen, die **grread** aufrufen, wahrscheinlich nicht in allen Konfigurationen identisch.

8.7.2 Kalibrieren von Tabletts

AutoCAD-Benutzer können ein Digitalisiertablett mit dem Befehl TABLETT kalibrieren. Siehe "TABLETT" in der *AutoCAD Befehlsreferenz*. Die Funktion **tablet** ermöglicht Anwendungen, die Kalibrierung durchzuführen, indem die Kalibrierung direkt vorgenommen wird und die Einstellungen für zukünftige Verwendung gespeichert werden.

Das erste Argument der Funktion **tablet** ist ein Ganzzahlen-`Code`. Ist `Code` gleich Null, gibt die Funktion die aktuelle Kalibrierung zurück. Ist `Code` gleich 1, wird die Kalibrierung entsprechend der übrigen Argumente eingestellt. Kalibrierungen werden als vier 3D-Punkte ausgedrückt (zusätzlich zu `Code`). Die ersten drei Punkte--`Reihe1`, `Reihe2` und `Reihe3`--sind die ersten drei Zeilen der Transformationsmatrix des Tabletts. Der vierte Punkt, `Richtung`, ist ein

Vektor, der senkrecht zu der Ebene steht, in der die Tablettoberfläche angenommen wird (ausgedrückt im WKS, dem Weltkoordinatensystem). Wird mit dem Befehl TABLETT kalibriert, wird vorausgesetzt, daß die Oberfläche des Tabletts in der *XY*-Ebene des aktuellen BKS liegt.

Anmerkung Die Systemvariable TABMODE prüft, ob der Tablettmodus ein- (1) oder ausgeschaltet (0) ist. Sie können ihn mit der Funktion **`setvar`** steuern.

Die folgende Beispielroutine ruft die aktuelle Tablettkalibrierung ab und speichert sie in der Variablen `tkal`.

```
(defun C:TABGET ( )
  (setq tkal tablet 0))
  (if tkal
    (princ)
      (strcat "\nKonfiguration gesichert, "
        "verwenden Sie TABSET, um Kalibrierung aufzurufen.")
    )
    (princ "\nKalibrierung nicht möglich ")
  )
  (princ)
)
```

War die Routine erfolgreich, enthält das Symbol `tkal` jetzt die von der Funktion tablet zurückgegebene Liste. Diese Liste sieht beispielsweise aus wie folgt:

```
(1 (0.00561717 -0.000978942 -7.5171)
  (0.000978942 0.00561717 -9.17308)
  (0.0 0.0 1.0)
  (0.0 0.0 1.0)
)
```

Sie können die Routine **`C:TABSET`** verwenden, um die Kalibrierung auf die von der vorstehenden Routine ermittelten Werte zurückzusetzen.

```
(defun C:TABSET ( )
  (if (not apply 'tablet tkal))
    (princ "\nKalibrierung kann nicht zurückgesetzt werden. ")
    (progn
      (princ "\nTablettkalibrierung zurückgesetzt. ")
      (setvar "tabmode" 1)
      (if (= (getvar "tabmode") 0)
        (princ "\nTablettmodus kann nicht eingeschaltet werden ")
      )
    )
  )
  (princ)
)
```

Die Transformationsmatrix der Größe 3x3, die einen zweidimensionalen Punkt konvertieren soll, wird übergeben als *`Reihe1`*, *`Reihe2`* und *`Reihe3`*. Der 2D-Punkt wird als Spaltenvektor in *homogenen Koordinaten* ausgedrückt (1.0 wird als drittes Element angefügt), so daß die Transformation wie folgt aussieht:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} M_{00} & M_{01} & M_{02} & M_{03} \\ M_{10} & M_{11} & M_{12} & M_{13} \\ M_{20} & M_{21} & M_{22} & M_{23} \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1.0 \end{bmatrix}$$

Die Berechnung eines Punkts verläuft ähnlich wie im dreidimensionalen Raum. AutoCAD transformiert den Punkt mit den folgenden Formeln:

$$X' = M_{00}X + M_{01}Y + M_{02}$$
$$Y' = M_{10}X + M_{11}Y + M_{12}$$
$$D' = M_{20}X + M_{21}Y + 1.0$$

Um den Ergebnisvektor wieder in einen 2D-Punkt zu überführen, werden die ersten beiden Komponenten durch die dritte dividiert (den Skalierfaktor), so daß sich der Punkt aus ergibt.

Bei projizierenden Transformationen, die am häufigsten auftreten, übernimmt **`tablet`** die komplette Berechnung. Allerdings sind bei Affin- und Orthogonaltransformationen sowohl M20 als auch M21 0, so daß D' den Wert 1,0 annimmt. Die Berechnung von und die Division werden ausgelassen; der 2D-Punkt ergibt sich einfach zu (X',Y').

Wie im vorstehenden Abschnitt bereits angesprochen, ist eine Affintransformation eine spezielle Form einer projizierenden Transformation. Eine orthogonale Transformation ist wiederum eine Sonderform einer affinen Transformation: Sowohl M20 und M21 sind Null, während M00 = M11 und M10 = -M01 sind.

Anmerkung Wenn Sie eine Kalibrierung vornehmen, ist die zurückgegebene Liste *nicht* gleich der Liste, die man erhält, wenn `Richtung` nicht normalisiert ist; AutoCAD normalisiert den Richtungsvektor, bevor er zurückgegeben wird. Außerdem gewährleistet AutoCAD, daß das dritte Element in der dritten Spalte (`Reihe3[Z]`) gleich 1 ist. Diese Situation tritt allerdings nur dann auf, wenn die Kalibrierung nicht unter Verwendung von Werten erfolgt, die AutoCAD mit **`tablet`** ermittelt. Die Situation kann dann entstehen, wenn Ihr Programm die Transformation selbst berechnet.

8.8 ASE-Schnittstelle von AutoLISP

AutoLISP bietet nicht nur direkten Zugriff auf ASE-Befehle in AutoLISP-Anwendungen (siehe "Zugriff auf extern definierte Befehle und Systemvariablen" Kapitel 14,), sondern auch internen Zugriff auf einen Großteil der Funktionalität der ASE-Befehle. AutoLISP-Anwendungen verfügen somit über eine Schnittstelle zum ausführbaren ASE-Modul. Die ASE-Programmschnittstelle von AutoLISP bietet AutoLISP-Anwendungen die folgenden Möglichkeiten. Eine Beschreibung der AutoLISP-Schnittstelle zu ASE finden Sie im *ASE Developer's Guide*.

9 Auswahlsatz-, Objekt- und Symboltabellen funktionen

Die meisten AutoLISP-Funktionen, die Auswahlsätze und Objekte bearbeiten, identifizieren einen Satz oder ein Objekt anhand des *Elementnamens*. Bevor eine AutoLISP-Anwendung einen Auswahlsatz oder ein Objekt bearbeiten kann, muß sie zunächst durch Aufruf einer der Funktionen, die den Namen eines Auswahlsatzes oder einen Elementnamen zurückgeben, den aktuellen Namen ermitteln.

Bei Auswahlsätzen, die nur für die aktuelle Sitzung gelten, ergeben sich aus der Vergabe von Namen keine Probleme; anders ist dies bei Objekten, da diese in der Datenbank der Zeichnung gespeichert werden. Eine Anwendung, die zu verschiedenen Zeiten auf dieselben Objekte in derselben Zeichnung (oder in denselben Zeichnungen) verweisen muß, kann die *Referenzen* des Objekts benutzen.

AutoLISP benutzt Symboltabellen, um Listen mit grafischen und nichtgrafischen Daten wie Layers, Linientypen und Blockdefinitionen zu verwalten, die sich auf eine Zeichnung beziehen. Jeder Eintrag in einer Symboltabelle ist mit einem Elementnamen und einer Referenz verknüpft und kann in ähnlicher Weise wie andere AutoCAD-Elemente behandelt werden.

9.1 Bearbeitung von Auswahlsätzen

AutoLISP bietet Funktionen zur Bearbeitung von Auswahlsätzen. Folgende Funktionen stehen hierfür zur Verfügung:

ssadd	ssget	ssmemb
ssdel	sslength	ssname

Die Funktion **`ssget`** stellt das allgemeinste Mittel zur Erstellung eines Auswahlsatzes dar. Sie kann einen Auswahlsatz auf eine der folgenden Weisen erstellen:

- Durch explizite Angabe der auszuwählenden Objekte mit Hilfe der Optionen Letztes, Vorher, Fenster, Impliziert, FPolygon, Kreuzen, KPolygon oder Zaun.
- Durch Angabe eines einzelnen Punktes.
- Durch Auswahl der gesamten Datenbank.
- Durch Aufforderung des Benutzers zur Auswahl von Objekten.

Bei allen Optionen können Sie das Verfahren des *Filterns* benutzen, um eine Liste von Attributen und Bedingungen anzugeben, denen die ausgewählten Objekte entsprechen müssen.

Anmerkung Die Namen von Auswahlsätzen und Elementen sind temporär; das heißt, sie gelten nur für die aktuelle Arbeitssitzung.

Das erste Argument von **`ssget`** ist eine Zeichenkette, die beschreibt, welche Auswahloption benutzt werden soll. Die beiden nächsten Argumente, *`pt1`* und *`pt2`*, spezifizieren Punktewerte für die betreffenden Optionen (sie sollten weggelassen werden, wenn sie im konkreten Fall nicht zutreffen). Eine Punkteliste, *`pt-- list`*, muß als Argument bei den Auswahlmethoden angegeben werden, die eine Auswahl nach Polygonen ermöglichen (das heißt, Zaun, Kreuzenpolygon und Fensterpolygon). Das letzte Argument, *`Filterliste`*, ist optional. Wenn *`Filterliste`* angegeben wird, wird dadurch die Liste der Elementfeldwerte spezifiziert, die für das Filtern benutzt werden. Auswahlfilter sind in "Auswahlsatz-Filterlisten" ausführlicher beschrieben. Die nachfolgende Tabelle faßt alle gültigen Moduswerte und deren Argumente zusammen. (Eine Filterliste kann bei allen aufgeführten Auswahlmethoden als zusätzliches Argument benutzt werden.)

Auswahloptionen für ssget

Modus	Auswahlmethode	Prototypen
keiner	Auswahl durch Benutzer oder Auswahl eines einzelnen Punktes (wenn pt1 *`angegeben wird)`*	**`ssget`**- Filter: **`(ssget pt1)`**
`"L"`	Zuletzt erstelltes Objekt, das auf dem Bildschirm zu sehen war	**`(ssget "L")`**

Anmerkung Wenn der *Modus* "X" angegeben wird und keine *Filterliste* hinzugefügt wird, wählt **ssget** *alle* Elemente der Datenbank aus, einschließlich der Elemente auf Layers, die ausgeschaltet, gefroren und außerhalb des sichtbaren Bildschirmbereichs sind.

Das folgende Codefragment zeigt typische Aufrufe von **ssget**.

```
(setq pt1 '(0.0 0.0 0.0)
      pt2 '(5.0 5.0 0.0)
      pt3 '(4.0 1.0 0.0)
      pt4 '(2.0 6.0 0.0)
)
(ssget filter-list)
```

Definiert pt1, pt2, pt3, und pt4 mitPunktwerten

Fordert den Benutzer zu einer allgemeinen Objektauswahl auf und übernimmt diese Objekte in eine Auswahlliste

```
(ssget "P")
```

Erzeugt einen Auswahlsatz mit den zuletzt gewählten Objekten

```
(ssget "P")
```

Erzeugt einen Auswahlsatz der Datenbank, die auf dem Bildschirm angezeigt wird.

```
(setq ss1 (ssget pt2))
```

Erzeugt einen Auswahlsatz eines Objekts, das den Punkt (5,5) schneidet.

```
(setq ss1 (ssget "W" pt1 pt2))
```

Erzeugt einen Auswahlsatz, der Objekte innerhalb des Fensters von (0,0) bis (5,5).

```
(setq ss1 (ssget "W" pt1 pt2))
```

Erzeugt einen Auswahlsatz der Objekte, die den Zaun mit den Endpunkten (5,5), (4,1) und (2,6) schneiden.

```
(setq ss1 (ssget "WP" (list pt1 pt2 pt3)))
```

Erzeugt einenAuswahlsatz der Objekte innerhalb des des Polygons mit den Endpunkten (0,0), (5,5) und (4,1)

```
(setq ss1  (ssget "X"))
```

Erzeugt einen Auswahlsatz der Objekte in der Datenbank.

Es ist wichtig, daß Auswahlsätze, die von einer Anwendung nicht mehr benötigt werden, aus dem Speicher entfernt werden. Dies erreichen Sie, indem Sie den Auswahlsatz auf `nil` setzen.

```
(setq ss1 nil)
```

Anmerkung Wir raten davon ab, eine große Anzahl von Auswahlsätzen *gleichzeitig zu verwalten. Eine AutoLISP-Anwendung kann nicht mehr als 128 Auswahlsätze gleichzeitig geöffnet haben. (Diese Grenze kann bei Ihrem System niedriger sein.) Wenn diese Grenze erreicht wird, weigert sich AutoCAD, weitere Auswahlsätze zu erstellen. Halten Sie immer so wenig Auswahlsätze wie möglich gleichzeitig geöffnet, und setzen Sie nicht mehr benötigte Auswahlsätze so bald wie möglich auf nil. Wenn die Höchstzahl der Auswahlsätze erreicht ist, müssen Sie* **gc** aufrufen, bevor **ssget** wieder funktioniert. Informationen über die Abfallsammlung (garbage collection) finden Sie unter "Knotenraum."

9.1.1 Auswahlsatz-Filterlisten

Eine Elementfilterliste ist eine Zuordnungsliste, die DXF-Gruppencodes im selben Format benutzt wie eine Liste, die von **entget** zurückgegeben wird. (Eine Liste von Gruppencodes finden Sie in Anhang C, "DXF-Gruppencodes",.) Die Funktion **ssget** erkennt alle Gruppencodes außer Elementnamen (Gruppe -1), Referenzen (Gruppe 5) und Xdata-Codes (Gruppen größer als 1000). Wenn in einer *Filterliste* ein unzulässiger Gruppencode benutzt wird, wird er von **ssget** ignoriert. Um nach Objekten mit Xdata zu suchen, benutzen Sie den Code -3, wie unter "Suchen nach erweiterten Daten (Extended Data)" beschrieben.

Wenn eine *Filterliste* als letztes Argument von **ssget** angegeben wird, überprüft die Funktion die ausgewählten Objekte und erstellt einen Auswahlsatz, der die Namen aller Elemente enthält, die den angegebenen Kriterien entsprechen. Sie können zum Beispiel einen Auswahlsatz erhalten, der alle Objekte eines bestimmten Typs, auf einem bestimmten Layer oder mit einer bestimmten Farbe umfaßt.

Die *Filterliste* legt fest, welche Eigenschaft (oder Eigenschaften) der Elemente geprüft werden sollen und welche Werte eine Entsprechung darstellen.

Im folgenden Beispiel wird gezeigt, wie Sie *Filterlisten* mit den verschiedenen Auswahloptionen der *Modi* verwenden.

```
(setq ss1 (ssget                          Fordert den Benutzer zu einer allgemeinen
  '((0 . "TEXT")))                        Objektauswahl auf aber fügt dem
)                                         Auswahlsatz nur Textobjekte hinzu
(setq ss1 (ssget "P"                      Erzeugt einen Auswahlsatz mit den
'((0 . "LINE")))                          zuletzt gewählten Objekten, die auch
)                                         Objekte vom Typ Linie sind.
(setq ss1 (ssget "W" pt1 pt2))            Erzeugt einen Auswahlsatz der Objekte
'((8 . "STOCK9")))                        innerhalb  des Fensters, die zum Layer
)                                         STOCK9 gehören.
(setq ss1  (ssget "X"))                   Erzeugt einen Auswahlsatz der Objekte in der
'((0 . "CIRCLE")))                        Datenbank die vom Typ Kreis sind.
)
```

Wenn der Code und der gewünschte Wert bekannt sind, kann die Liste, wie vorher gezeigt, aufgeführt werden. Wenn eines von beiden durch eine Variable spezifiziert wird, muß die Liste konstruiert werden (mit Hilfe der Funktionen **list** und **cons)**.

```
(setq lay_name "STOCK3")
(setq ss1
  (ssget "X")                             Erzeugt einen Auswahlsatz aller Objekte in der
    (list (cons 8 lay_name))              Datenbank, die zum Layer STOCK3 gehören
  )
)
```

Wenn die *Filterliste* mehr als eine Eigenschaft angibt, wird ein Element nur dann in den Auswahlsatz aufgenommen, wenn es *allen* angegebenen Bedingungen entspricht, wie im folgenden Beispiel gezeigt wird:

```
(ssget "X"  (list  (cons 0 "CIRCLE")(cons 8 lay_name)(cons 62 1)))
```

Mit diesem Aufruf werden nur Kreisobjekte auf dem Layer STOCK3 ausgewählt, welche die Farbe Rot haben. Diese Art von Test führt eine Boolesche AND-Operation aus. Weitere Tests für Objekteigenschaften werden unter "Logische Gruppierung von Filtertests" beschrieben.

Die Funktion **ssget** filtert eine Zeichnung, indem sie die ausgewählten Elemente durchsieht und die Felder jedes Hauptelements mit der angegebenen Filterliste vergleicht. Wenn die Eigenschaften eines Elements mit *allen* in der Filterliste angegebenen Feldern übereinstimmen, wird es in den zurückgegebenen Auswahlsatz aufgenommen. Andernfalls wird das Element nicht in die Auswahlliste aufgenommen. Die Funktion **ssget** gibt den Wert **nil** als Rückgabewert, wenn keines der ausgewählten Elemente den Filterkriterien entspricht.

Anmerkung Die Bedeutung bestimmter Gruppencodes kann von Element zu Element variieren, und nicht alle Gruppencodes sind in allen Elementen enthalten. Wenn ein bestimmter Gruppencode in einem Filter angegeben wird, werden Elemente, die diesen Gruppencode nicht enthalten, aus dem von **ssget** zurückgegebenen Auswahlsatz ausgeschlossen.

Wenn **ssget** eine Zeichnung filtert, kann der gefundene Auswahlsatz Elemente aus dem Papierbereich und dem Modellbereich enthalten. Wenn allerdings der Auswahlsatz an einen AutoCAD-Befehl übergeben wird, werden nur Elemente aus dem Bereich benutzt, der gerade wirksam ist. (Der Bereich, zu dem ein Element gehört, wird, wie in Anhang C unter "DXF-Gruppencodes" beschrieben, durch den Wert seiner Gruppe 67 angegeben.

9.1.1.1 Platzhaltermuster in Filterlisten

Symbolnamen, die in Filterlisten angegeben werden, können Platzhaltermuster enthalten. Die Platzhaltermuster, die von **ssget** erkannt werden, sind dieselben, die auch von der Funktion **wcmatch** erkannt werden. Sie werden unter"Platzhaltersuche" und unter **"wcmatch"** in Kapitel 13 beschrieben.

Anmerkung Wenn Sie nach unbenannten Blöcken suchen, müssen Sie das Zeichen `*` mit einem umgekehrten einfachen Anführungszeichen (` ) einleiten, weil das Zeichen `*` von **ssget** als Platzhalter angesehen wird. Sie können zum Beispiel einen unbenannten Block namens `*U2` mit folgender Funktion ansprechen:

```
(ssget "X" '((2 . "`*U2")))
```

9.1.1.2 Suchen nach erweiterten Daten (Extended Data)

Mit der *Filterliste* der Funktion **ssget** können Sie alle Objekte wählen, die erweiterte Daten für eine bestimmte Anwendung enthalten (siehe "Erweiterte Daten (Extended Data) - Xdata"). Hierzu verwenden Sie den Gruppencode -3 wie im folgenden Beispiel:

```
(ssget "X" '((0 . "CIRCLE") (-3 ("APPNAME"))))
```

Dieser Code wählt alle Kreise aus, die erweiterte Daten für die Anwendung "ANWNAME" enthalten. Wenn in der Liste der Gruppe -3 mehr als ein Anwendungsname enthalten ist, wird eine AND-Operation impliziert und das Element muß erweiterte Daten für alle angegebenen Anwendungen enthalten. Der folgende Ausdruck würde demnach alle Kreise mit erweiterten Daten für die Anwendungen" ANW1" und " ANW2" auswählen.

```
(ssget "X" '((0 . "CIRCLE") (-3("ANW1")("ANW2"))))
```

Die Verwendung von Platzhaltern ist zulässig; jeder der beiden folgenden Ausdrücke würde daher alle Kreise mit erweiterten Daten für eine dieser Anwendungen oder für beide auswählen.

```
(ssget "X" '((0 . "CIRCLE") (-3 (""))))
(ssget "X" '((0 . "CIRCLE") (-3("ANW1")("ANW2"))))
```

9.1.1.3 Relationale Tests

Soweit nicht anders definiert, wird für jedes Element der Filterliste *filter-list* ein relationaler Test angenommen. Für numerische Gruppen (Ganzzahlen, reelle Zahlen, Punkte und Vektoren) können Sie andere Relationen definieren, indem Sie einen Gruppencode 4 einbeziehen, der einen relationalen Operator definiert. Der Wert einer Gruppe -4 ist eine Zeichenkette, die den Testoperator angibt, der auf die nächste Gruppe in der Filterliste angewandt werden soll. Weitere Information finden Sie unter "Relationale Tests."

Im folgenden Beispiel werden alle Kreise mit einem Radius (Gruppencode 40) größer oder gleich 2.0 ausgewählt:

```
(ssget "X" '((0 . "CIRCLE") (-4 . ">=") (40 . 2.0)))
```

9.1.1.4 Logische Gruppierung von Filtertests

Sie können Gruppen auch testen, indem Sie verschachtelte Boolesche Ausdrücke erstellen, welche die unter "Logische Gruppierung von Filtertests" beschriebenen Gruppierungsoperatoren einsetzen. Die Gruppierungsoperatoren werden, wie die relationalen Operatoren, durch Gruppen -4 spezifiziert. Sie sind zu Paaren zusammengefaßt und müssen in der Filterliste im richtigen Verhältnis stehen, oder der Aufruf der Funktion **ssget** schlägt fehl. Das folgende Beispiel zeigt eine mögliche Gruppierung von Operatoren in einer Filterliste:

```
(ssget "X"

 '(
    (-4 . "<OR")
      (-4 . "<AND
        (0 . "CIRCLE")
        (40 . 1.0)
      (-4 . "AND>"
      (-4 . "<AND
        (0 . "LINE")
        (8 . "ABC")
      (-4 . "AND>"
    (-4 . "OR>"
  )
)
```

Mit diesem Code werden alle Kreise mit einem Radius von 1.0 und alle Linien auf Layer

"ABC" ausgewählt. Die Gruppierungsoperatoren unterscheiden nicht nach Groß- und Kleinschreibung.

Gruppierungsoperatoren sind nicht innerhalb der Gruppe -3 selbst erlaubt. Bei Angabe mehrerer Anwendungsnamen in einer Gruppe -3 wird ein impliziter Operator AND verwendet. Wenn Sie nach erweiterten Daten mit anderen Gruppierungsoperatoren suchen wollen, geben Sie getrennte Gruppen -3 an, und gruppieren Sie sie nach Wunsch. Mit folgendem Ausdruck können Sie zum Beispiel alle Kreise auswählen, die erweiterte Daten für "ANW1" oder "ANW2" aber nicht für beide enthalten:

```
(ssget "X"
 '((0 . "CIRCLE")
    (-4 . "<XOR"
      (-3 ("ANW1"))
      (-3 ("ANW2"))
    (-4 . "XOR>"
  )
)
```

Sie können die Codierung häufig gebrauchter Gruppierungsoperatoren vereinfachen, indem Sie ihnen ein Symbol zuordnen. Das vorangehende Beispiel könnte folgendermaßen umformuliert werden (beachten Sie, daß Sie in diesem Beispiel jede Liste explizit aufführen müssen):

```
(setq <xor '(-4 . "<XOR")
       xor> '(-4 . "XOR>") )
(ssget "X"
  (list
    '(0 . "CIRCLE")
    <xor
      '(-3 ("ANW1"))
      '(-3 ("ANW2"))
    "xor>"
  )
)
```

Wie Sie sehen, ist diese Methode möglicherweise für kurze Anweisungen nicht sinnvoll, kann jedoch in größeren Anwendungen von Nutzen sein.

9.1.1.5 Bearbeitung von Auswahlsätzen

Wenn ein Auswahlsatz einmal erstellt wurde, können Sie mit den Funktionen **ssadd** und **ssdel** Elemente hinzufügen oder entfernen. Mit Hilfe der Funktion **ssadd** können Sie einen neuen Auswahlsatz erstellen, wie im folgenden Beispiel gezeigt wird. Der folgende Teil eines Ausdrucks erstellt einen Auswahlsatz, der das erste und das letzte Element der aktuellen Zeichnung enthält (**entnext** und **entlast** werden später in diesem Kapitel beschrieben).

```
(setq ename (entnext))                                    Wählt das erste
Element der Zeichnung.
(setq lname (entlast))                                    Wählt das letzte
Element der Zeichnung.
(if (not ename)
  (princ "\nKeine Elemente in der Zeichnung. ") ")
  (progn
    (setq asatz (ssadd ename))
     Erzeugt einen Auswahlsatz des ersten Elements.
    (ssadd lname asatz)
     Fügt das letzte Element zum Auswahlsatz hinzu.
  )
)
```

Das Beispiel wird korrekt ausgeführt, auch wenn die Datenbank nur ein Element enthält (in diesem Fall werden die Argumente von **entnext** und **entlast** auf denselben Elementnamen gesetzt). Wenn **ssadd** der Name eines Elements übergeben wird, das bereits im Auswahlsatz enthalten ist, wird dies ignoriert und *keine* Fehlermeldung ausgegeben. Mit der folgenden Funktion wird das erste Element aus dem im vorherigen Beispiel erstellten Auswahlsatz entfernt:

```
(ssdel ename asatz)
```

Wenn die Zeichnung mehr als ein Element enthält (das heißt, wenn `ename` und `lname` nicht gleich sind), enthält der Auswahlsatz `asatz` nur noch `lname`, das letzte Element der Zeichnung.

Die Funktion **sslength** gibt die Anzahl der in einem Auswahlsatz enthaltenen Elemente zurück; **ssmemb**prüft, ob ein bestimmtes Element zu einem Auswahlsatz gehört. Schließlich gibt die Funktion **ssname** anhand eines Indexes für den Auswahlsatz den Namen eines bestimmten Elements eines Auswahlsatzes zurück (die Elemente eines Auswahlsatzes sind von 0 an durchnumeriert).

Das folgende Codefragment zeigt typische Aufrufe von **ssname**.

```
(setq sset (ssget))                                       Fordert den
Anwender zur Erstellung eines                             Auswahlsatzes auf
(setq ent1 (ssname asatz 0))
  Ermittelt den Namen des ersten Elementes in asatz
(setq ent1 (ssname asatz 3))
  Ermittelt den Namen des ersten Elementes in asatz
(if (not ent4)
  (princ "\nEs müssen mindestens vier Elemente ausgewählt werden. ") ")
)
(setq ilast (sslength asatz))                                          Sucht
Index des letzten Elements in asatz

  Ermittelt den Namen des letzten Elements in asatz
(setq letztel (ssname asatz (1- il)))
```

Unabhängig von der Art, in der Elemente einem Auswahlsatz hinzugefügt wurden, enthält der Satz *nie* doppelte Elemente. Wenn ein Element bereits im Auswahlsatz enthalten ist, werden erneute Eintragungen des Elements ignoriert. Daher gibt **sslength** exakt die Anzahl *verschiedener* Elemente im angegebenen Auswahlsatz zurück.

9.1.2 Übergabe von Auswahlsätzen zwischen AutoLISP und ADSRX-Anwendungen

Bei der Übergabe von Auswahlsätzen zwischen AutoLISP- und ADS/ARX-Anwendungen sollten folgende Punkte beachtet werden:

Wird ein Auswahlsatz in AutoLISP erstellt, in einer AutoLISP-Variable gespeichert und anschließend durch einen Wert überschrieben, der von einer ADSRX-Anwendung wiedergegeben wird, ist der ursprüngliche Auswahlsatz für die Abfallsammlung auswählbar. (Er wird bei der nächsten automatischen oder expliziten Abfallsammlung freigegeben.)

Dies gilt auch dann, wenn der von der ADSRX-Anwendung wiedergegebene Wert dem ursprünglichen Auswahlsatz entspricht. Gibt die ADSRX-Funktion **adsfunc** im folgenden Beispiel denselben Auswahlsatz wieder, der als Argument eingegeben wurde, ist dieser Auswahlsatz für die Abfallsammlung auswählbar, selbst wenn er weiterhin derselben Variable zugeordnet ist.

```
(setq var1 (ssget))
(setq var1 (adsfunc var1))
```

Soll der ursprüngliche Auswahlsatz von der Abfallsammlung verschont bleiben, darf der Wiedergabewert der ADSRX-Anwendung nicht der AutoLIST-Variable, die bereits auf den Auswahlsatz verweist, zugewiesen werden. Durch Ändern des vorherigen Beispiels wie im folgenden angegeben wird der Auswahlsatz, der auf `var 1` verweist, nicht für die Abfallsammlung auswählbar.

```
(setq var1 (ssget))
(setq var2 (adsfunc var1))
```

9.2 Objektbearbeitung

AutoLISP stellt Funktionen zur Bearbeitung von Objekten zur Verfügung. Hierbei handelt es sich um folgende Funktionen:

entdel	entlast	entmod	entupd
entget	entmake	entnext	handent

Die Funktionen zur Objektbearbeitung sind in zwei Kategorien unterteilt: Funktionen, die den Elementnamen eines bestimmten Objekts ermitteln, und Funktionen, die Elementdaten auffinden oder modifizieren.

9.2.1 Elementnamen-Funktionen

Um ein Objekt bearbeiten zu können, muß eine AutoLISP-Anwendung den entsprechenden Elementnamen ermitteln, um ihn für Zugriffe auf die Elementdaten oder Auswahlsatzfunktionen benutzen zu können. Zwei der in diesem Abschnitt beschriebenen Funktionen, **entsel** und **nentsel, geben nicht nur den Namen des Elements zurück sondern auch zusätzliche Informationen zur Benutzung der Anwendung.**

Bei beiden Funktionen muß der AutoCAD-Benutzer ein Objekt interaktiv durch Auswahl eines Punktes auf dem Grafikbildschirm markieren. Mit allen anderen Elementnamen-Funktionen kann ein Element auch ausgewählt werden, wenn es auf dem Bildschirm nicht zu sehen ist oder wenn es sich auf einem gefrorenen Layer befindet. Die Funktion **entsel** fordert den Benutzer auf, ein Objekt durch Auswahl eines Punktes auf dem Grafikbildschirm zu wählen, und **entsel** gibt sowohl den Elementnamen als auch den Wert des ausgewählten Punktes zurück. Bei einigen Elementoperationen muß der Punkt, an dem das Objekt ausgewählt wurde, bekannt sein. Beispiele unter den vorhandenen AutoCAD-Befehlen sind BRUCH, STUTZEN, und DEHNEN. Die Funktion **nentsel** wird unter "Elementkontext- und Koordinatensystem-Konvertierungsdaten" genauer beschrieben. Diese Funktionen akzeptieren Schlüsselwörter, wenn ihnen ein Aufruf von **initget** vorausgeht.

Die Funktion **entnext** ermittelt Elementnamen nacheinander. Wird die Funktion **entnext** ohne Angabe von Argumenten aufgerufen, gibt sie den Namen des ersten Elements in der Datenbank zurück. Wenn das Argument der Name eines Elements der aktuellen Zeichnung ist, gibt **entnext** den Namen des nachfolgenden Elements zurück.

Der folgende Teil eines Codes zeigt, wie **ssadd** in Verbindung mit **entnext** benutzt werden kann, um Auswahlsätze zu erstellen und Teile zu bestehenden Sätzen hinzuzufügen.

```
(setq e1  (entnext))
(if  (not e1)                 Setzt e1 auf den Namen des ersten Elements in der Zeichnung
  (princ "\nKeine Elemente in der Zeichnung. ") ")
  (progn
    (setq ss (ssadd))         Setzt ss auf einen leeren Auswahlsatz.
    (ssadd e1 ss)             Gibt den Auswahlsatz ss mit hinzugefügtem e1 zurück.
    (setq e2 (entnext e1))    Ermittelt das auf e1 folgende Element
    (ssadd e2 ss)             Fügt e2 dem Auswahlsatzss hinzu.
  )
)
```

Die Funktion **entlast** ermittelt den Namen des letzten Elements in der Datenbank. **entlast** kann also aufgerufen werden, um den Namen eines Elements zu ermitteln, das soeben mit einem Aufruf von **command** erstellt wurde.

Sie können den von **entnext** zurückgegebenen Elementnamen auf denselben Variablennamen setzen, der dieser Funktion übergeben wurde. Auf diese Weise "wandert" eine einzige Elementnamen-Variable durch die Datenbank, wie im folgenden Beispiel gezeigt wird:

```
(setq ein_el (entnext))          Ermittelt den Namen des ersten Elements

 (while ein_el
 .
 .                                Verarbeitet ein neues Element
 .
  (setq ein_el (entnext ein_el))
)
 Wert von ein_el ist nun nil
```

9.2.1.1 Elementreferenzen und ihre Verwendung

Die Funktion **handent** ermittelt den Namen eines Elements mit einer bestimmten *Referenz*. Genau wie Elementnamen können auch Referenzen innerhalb einer Zeichnung nur einmal vorkommen. Allerdings ist die Referenz eines Elements immer dieselbe, solange dieses Element existiert. AutoLISP-Anwendungen, die eine bestimmte Datenbank bearbeiten, können mit Hilfe von **handent** den aktuellen Namen des Elements ermitteln, das sie benutzen müssen. Sie können den Befehl DDMODIFY verwenden, um die Referenz eines ausgewählten Objekts zu erhalten.

Das folgende Codefragment benutzt **handent**, um einen Elementnamen zu ermitteln und anzuzeigen.

```
(if  (not (setq e1  (handent "5a2")))
  (princ "\nKein Element mit dieser Referenz vorhanden. ") ")
  (princ e1)
)
```

In einer bestimmten Arbeitssitzung kann dieses Codefragment etwa folgende Ausgabe erzeugen:

<Elementname: 60004722>

In einer anderen Arbeitssitzung mit derselben Zeichnung kann dieses Fragment eine völlig andere Zahl anzeigen. Die Funktion würde jedoch in beiden Fällen auf dasselbe Element zugreifen.

Die Funktion **handent** hat noch eine weitere Verwendung. Elemente, die aus der Datenbank entfernt wurden (mit der Funktion **entdel**, die im folgenden Abschnitt beschrieben wird), werden erst gelöscht, wenn die aktuelle Zeichnung beendet wird. Das bedeutet, daß **handent** die Namen der gelöschten Elemente wiederherstellen kann, die dann in der Zeichnung durch einen zweiten Aufruf von **entdel** wieder erstellt werden können.

Anmerkung Referenzen stehen für Blockdefinitionen, einschließlich Subelemente, zur Verfügung.

Elemente in Zeichnungen, auf die mit Hilfe von XREF Zuordnen verwiesen wird, sind nicht wirklich Teil der aktuellen Zeichnung; ihre Referenzen bleiben unverändert, aber **handent** kann nicht auf sie zugreifen. Wenn allerdings Zeichnungen mit Hilfe von EINFÜGE, EINFÜGE *, XREF Binden (((XBINDEN) oder partiellem DXFIN verbunden werden, gehen die Referenzen der Elemente in der hinzukommenden Zeichnung verloren und den hinzukommenden Elementen werden neue Referenzen zugewiesen, um sicherzustellen, daß jede Referenz in der aktuellen Zeichnung eindeutig bleibt.

9.2.1.2 Elementkontext- und Koordinatensystem-Konvertierungsdaten

Die Befehle **nentsel** und **nentselp** arbeiten ähnlich wie **entsel**, sie geben jedoch zwei weitere Werte zurück, um Elemente zu verarbeiten, die in Blockreferenzen verschachtelt sind.

Ein weiterer Unterschied zwischen diesen Funktionen zeigt sich, wenn der Benutzer auf einen Aufruf von **nentsel** mit der Auswahl eines komplexen Elements reagiert oder wenn ein komplexes Element mit Hilfe von **nentselp** ausgewählt wird. Dann geben diese Funktionen, nämlich den Elementnamen des ausgewählten Subelements zurück und *nicht*, wie **entsel**, den Header des komplexen Elements.

Wenn der Benutzer zum Beispiel eine Polylinie auswählt, gibt **nentsel** einen Kontrollpunkt als Subelement zurück und nicht den Header der Polylinie. Um den Polylinien-Header zu erhalten, muß die Anwendung mit Hilfe von **entnext** zu dem Seqend-Subelement gehen und dann den Namen des Headers aus der Gruppe -2 des Seqend-Subelements ermitteln. Dasselbe gilt, wenn der Benutzer Attribute in einer verschachtelten Blockreferenz auswählt. Die Funktion **nentselp** wird **nentsel** vorgezogen, weil sie eine Konvertierungsmatrix in einem Format zurückgibt, das dem von **grvecs** zurückgegebenen Format entspricht.

Das erste der von **nentsel** zusätzlich zurückgegebenen Elemente ist die Modell-Welt-Konvertierungsmatrix. Dies ist eine Liste, die aus vier Sublisten besteht, von denen jede einen Satz von Koordinaten enthält. Mit Hilfe dieser Matrix können die Elementdefinitionsdatenpunkte von einem internen Koordinatensystem (dem Modellkoordinatensystem MKS) in das WKS konvertiert werden. Der Einfügepunkt des Blocks (dies bezieht sich auch auf externe Referenzen (XRefs), der das ausgewählte Objekt enthält, definiert den Ursprung des MKS. Die Ausrichtung des BKS bei der Erstellung des Blocks bestimmt die Ausrichtung der Achsen des MKS.

Das zweite zusätzliche Element ist eine Liste, die den Elementnamen des Blocks enthält, zu dem das ausgewählte Objekt gehört. Wenn sich das ausgewählte Objekt in einem verschachtelten Block befindet (ein Block innerhalb eines anderen Blocks), enthält die Liste außerdem die Elementnamen aller Blöcke, in denen sich das ausgewählte Element befindet; die Liste beginnt mit dem innersten Block und endet mit dem Namen des äußersten Blocks, der in die Zeichnung eingefügt wurde.

```
(<Elementname: >ename1                  Name des Elements
  (Px Py Pz)                            Auswahlpunkt
  ( (X0 Y0 Z0)                          Modell-Welt-Transformationsmatrix
    (X1 Y1 Z1)                          Konvertierungsmatrix
    (X2 Y2 Z2)
    (X3 Y3 Z3)
  )
  (<Elementname: >ename2                Name des am tiefsten verschachtelten Blocks der
      .                                 ausgewähltes Objekt enthält
      .
      .
  <Elementname: >enamen                 Name des äußersten Blocks der
)                                       ausgewähltes Objekt enthält
```

Erstellen Sie im folgenden Beispiel einen Block, der mit der Funktion **nentsel** benutzt werden soll.

Befehl: **linie**
Von Punkt: **1,1**
Nach Punkt: **3,1**
Nach Punkt: **3,3**
Nach Punkt: **1,3**
Nach Punkt: **d**
Befehl: **block**
Blockname (oder ?): **quadrat**
Einfügebasispunkt: **2,2**
Objekte wählen: *Wählen Sie die vier gerade gezeichneten Linien aus.*
Objekte wählen : RETURN

Fügen Sie den Block in ein BKS ein, dessen *Z-Achse* um 45 Grad gedreht ist.

Befehl: **BKS**
Ursprung/zAChse/3Punkt/Objekt/ANsicht/X/Y/Z/Vorher/Holen/Sichern/Löschen/?
<Welt>: **z**
Drehwinkel um die Z-Achse <0>: **45**
Befehl: einfügen
Blockname (oder ?): **quadrat**
Einfügepunkt: **7,0**
X-Faktor <1> / Ecke / XYZ: : RETURN
Y-Faktor (Vorgabe=X): RETURN
Drehwinkel: RETURN

Wählen Sie mit **nentsel** die untere linke Seite des Quadrats aus.

```
(setq ndaten (nentsel))
```

Mit diesem Ausdruck wird `ndata` einer Liste gleichgesetzt, die etwa folgendermaßen aussieht:

```
<Elementname: 400000a0>          Elementname
 (6.46616 -1.0606 0.0)           Auswahlpunkt
 ((0.707107 0.707107 0.0)        Modell-Welt-
  (-0.707107 0.707107 0.0)       Konvertierungsmatrix
  (0.0 -0.0 1.0)
  (4.94975 4.94975 0.0)
 )
 (<Elementname: 6000001c>)       Name des Blocks mit dem ausgewählten Objekt.
)
```

Wenn Sie den Elementnamen und die Modell-Welt-Konvertierungsmatrix einmal erhalten haben, können Sie die Elementdefinitionsdatenpunkte vom MKS in das WKS konvertieren. Wenden Sie `entget` und **`assoc`** auf den Elementnamen an, um die Definitionspunkte ausgedrückt in MKS-Koordinaten zu erhalten. Übergeben Sie dann die Punkte und die Daten der Modell-Welt-Konvertierungsmatrix (die Sie im ersten Aufruf von **`nentsel`** erhalten haben) den nachfolgenden Formeln.

Wenn das ausgewählte Element kein verschachteltes Element ist, wird die Konvertierungsmatrix mit der Identitätsmatrix gleichgesetzt.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Die folgenden Gleichungen zeigen, wie ein Punkt oder Vektor konvertiert wird.

$$X' = XM_{00} + YM_{10} + ZM_{20} + M_{30}$$
$$Y' = XM_{01} + YM_{11} + ZM_{21} + M_{31}$$
$$Z' = XM_{02} + YM_{12} + ZM_{22} + M_{32}$$

Die Mij*, wobei 0* i, *j 2 ist, sind die Koordinaten der Modell-Welt-Transformationsmatrix;* X, Y*,* Z *ist der Elementdefinitionsdatenpunkt ausgedrückt in MKS-Koordinaten, und* X', Y', Z' ist der sich ergebende Elementdefinitionsdatenpunkt, ausgedrückt in WKS-Koordinaten.

Anmerkung Um einen *Vektor* (statt eines Punktes) zu konvertieren, fügen Sie den Verschiebungsvektor [M30 M31 M32] (aus der vierten' Spalte der Konvertierungsmatrix) nicht hinzu.

Das folgende Beispiel zeigt, wie Sie den MKS-Startpunkt einer Linie erhalten (Gruppencode 10), die in einer Blockdefinition enthalten ist. Der Ausdruck speichert die Elementdaten (unter Benutzung des vorher mit **`nentsel`** erhaltenen Elementnamens) in dem Symbol `edaten` und gibt folgendes zurück: wird:

```
(setq edaten  (assoc 10  (entget  (car ndata))))
(10 -1.0 1.0 0.0)
```

Der folgende Ausdruck speichert die Subliste der Modell-Welt-Konvertierungsmatrix in dem Symbol `matrix`.

```
(setq matrix (caddr ndata))
```

Folgendes wird zurückgegeben:

```
((0.707107 0.707107 0.0)                X-Konvertierung
  (-0.707107 0.707107 0.0)              Y-Konvertierung
  (0.0 -0.0 1.0)                        Z-Konvertierung
  (4.94975 4.94975 0.0)                 Verschiebung vom WKS-Ursprung
)
```

Wenden Sie die Konvertierungsformel für X' an, um die X-Koordinate des Startpunkts der Linie von einer MKS-Koordinate in eine WKS-Koordinate zu konvertieren. Speichern Sie die Ergebnisse in dem Symbol `antwort`:

```
(setq antwort
  (+
          hinzufügen:
    (* (car (nth 0 matrix))(cadr edaten))
          M00 * X
    (* (car (nth 1 matrix))(caddr edaten))
         M10 * Y
    (* (car (nth 2 matrix))(cadddr edaten))
         M20 * Z
    (car (nth 3 matrix))
          M30
  )
)
```

Dieser Ausdruck gibt den Wert 3.53553 zurück, die WKS-*X*-Koordinate des Startpunkts der ausgewählten Linie.

9.2.1.3 Elementzugriffsfunktionen

Die Elementzugriffsfunktionen sind relativ langsam. Es ist sinnvoller, den Inhalt eines bestimmten Elements (oder eines Symboltabelleneintrags) einmal aufzurufen und diese Daten dann im Speicher zu halten, als AutoCAD immer wieder nach denselben Daten zu fragen. Achten Sie darauf, daß die Daten gültig bleiben; wenn der Benutzer die Möglichkeit hat, das Element oder den Symboltabelleneintrag zu verändern, sollten Sie die Elementzugriffsfunktion erneut aufrufen, um die Gültigkeit der Daten zu gewährleisten.

9.2.2 Elementdatenfunktionen

Die in diesem Abschnitt beschriebenen Funktionen bearbeiten Elementdaten und können benutzt werden, um die aktuelle Zeichnungsdatenbank zu verändern.

Die Funktion **`entdel`** löscht ein angegebenes Element. Das Element wird erst am Ende der aktuellen Arbeitssitzung endgültig gelöscht; wenn also die Anwendung **`entdel`** ein zweites Mal während dieser Sitzung aufruft und dasselbe Element angibt, wird der Löschvorgang rückgängig gemacht.

Anmerkung Attribute und Polylinien-Kontrollpunkte *können nicht* unabhängig von ihren übergeordneten Elementen gelöscht werden. Die Funktion **`entdel`** bearbeitet nur Hauptelemente. Wenn Sie ein Attribut oder einen Kontrollpunkt löschen müssen, können Sie mit Hilfe von **`command`** die AutoCAD-Befehle ATTEDIT oder PEDIT aufrufen.

Die Funktion **`entget`** gibt die Definitionsdaten eines angegebenen Elements zurück. Die Daten werden in Form einer Liste bereitgestellt. Jeder Punkt in der Liste wird durch einen DXF-Gruppencode spezifiziert. Der erste Punkt der Liste enthält den aktuellen Namen des Elements.

In diesem Beispiel gelten die aufgeführten (Vorgabe-) Bedingungen für die aktuelle Zeichnung:

- Name des Layers 0.
- Name des Linientyps CONTINUOUS.
- Die Erhebung ist 0.

Der Benutzer hat mit der folgenden Befehlsfolge eine Linie gezeichnet.

Befehl: **linie**
Von Punkt: **1,2**
Nach Punkt: **6,6**
Nach Punkt: RETURN

Eine AutoLISP-Anwendung kann mit Hilfe der folgenden AutoLISP-Funktion die Definitionsdaten der Linie aufrufen und drucken:

```
(defun C:PRINTDXF ( )
  (setq ent (entlast))          Setzt ent auf den Namen des letzten Elements
  (setq entl (entget ent))      Setzt entl auf Assoziationsliste des letzten Elements
  (setq ct 0)                   Setzt ct (einen Zähler) auf 0
  (textpage)                    Wechselt zum Textbildschirm
  (princ "\nentget des letzten Elements:")
  (repeat (length entl)         Wiederholt dies für alle Einträge der Liste
    (print (nth ct entl))       Gibt ein Zeilenvorschubzeichen gefolgt von den Listeneinträgen aus.
    (setq ct (1+ ct))           Erhöht den Zähler um eins
  )
  (princ)                       Beendet das Programm
)
```

Diese Anweisung ergibt folgende Druckausgabe:

```
Ergebnisse von entget des letzten Elements:
(-1 . <Elementname: 60000014>)
(0 . "LINE")
(8 . "0")
(5 . "2D")
(10 1.0 2.0 0.0)
(11 6.0 6.0 0.0)
(210 0.0 0.0 1.0)
```

Die erste Zeile der Liste (-1) enthält den Namen des Elements, das durch diese Liste dargestellt wird. Die in diesem Abschnitt beschriebene Funktion **entmod** kann damit das zu modifizierende Element identifizieren.

Die Codes für die Komponenten des Elements sind die, die von DXF benutzt werden und in Anhang C, "DXF-Gruppencodes", dokumentiert sind. Wie bei DXF werden die Element-Header-Einträge (Farbe, Linientyp, Objekthöhe, Attribut-Folgen-Flag und Elementreferenz) nur dann exportiert, wenn sie von den Vorgabewerten abweichen. Im Unterschied zu DXF werden fakultative Elementdefinitionsfelder immer zurückgegeben, gleichgültig ob sie mit den Vorgaben übereinstimmen oder nicht. Zugehörige *X*-, *Y*- und *Z*-Koordinaten werden als Einzelpunktvariable und nicht als getrennte Gruppen *X* (10), *Y* (20) und *Z* (30) zurückgegeben.

Die Funktion **assoc** durchsucht eine Liste nach einer Gruppe eines bestimmten Typs. Der folgende Code gibt den Objekttyp "LINIE" (0) aus der Liste entl zurück:

```
(cdr (assoc 0 entl))
```

Wenn die angegebene DXF-Gruppe in der Liste nicht vorhanden ist (oder wenn es keine gültige DXF-Gruppe ist), gibt **assoc** nil zurück.

Die Funktion **entmod** modifiziert ein Element. Sie übergibt eine Liste, die dasselbe Format hat, wie eine von **entget** zurückgegebene Liste, wobei jedoch (vermutlich) einige Gruppenwerte des Elements durch die Anwendung modifiziert wurden. Diese Funktion ergänzt **entget**. Das von AutoLISP am häufigsten benutzte Verfahren zur Aktualisierung der Datenbank besteht darin, ein Element mit **entget** aufzurufen, seine Elementliste zu modifizieren und dann die Liste mit **entmod** wieder an die Datenbank zurückzugeben.

Das folgende Codefragment ruft die Definitionsdaten des ersten Elements der Zeichnung auf und ändert seine Layereigenschaft in MEINLAYER.

```
(setq en (entnext))             Setzt en auf den Namen des ersten Elementes in der Zeichnung
(setq ed (entget en))           Setzt ed auf die Elementdaten des Elementes en
(setq ed
  (subst (cons 8 "MEINLAYER")
    (assoc 8 ed)                Ändert die Layergruppe in ed
    ed                          zum Layer MEINLAYER
  )
)
(entmod ed)                     Ändert den Layer des Elementes en in der Zeichnung
```

Es gibt Einschränkungen hinsichtlich der Änderungen in der Datenbank, die **entmod** vornehmen kann. Die Funktion **entmod** kann folgende Daten *nicht* ändern:

- Den Typ oder die Referenz des Elements.
- Interne Felder. (Interne Felder sind die Werte, die AutoCAD bestimmten Gruppencodes zuweist: -2, Elementnamensreferenz; -1, Elementname; 5, Elementreferenz.) Jeder Versuch, ein internes Feld zu ändern - zum Beispiel den Hauptelementnamen in einem Seqend-Subelement (Gruppe -2) - wird ignoriert.
- Ansichtsfensterelemente. Ein Versuch, ein Ansichtsfensterelement zu ändern, verursacht einen Fehler.

Andere Einschränkungen gelten für das Modifizieren von Bemaßungen und Schraffurmustern.

AutoCAD muß alle Objekte (außer Layer), auf welche die Liste verweist, erkennen. Der Name jedes Textstils, Linientyps, Symbols oder Blocks, der in einer Elementliste vorkommt, muß in der aktuellen Zeichnung definiert worden sein, *bevor* die Elementliste **entmod** zugeführt wird. Es gibt eine Ausnahme: **entmod** akzeptiert neue Layernamen.

Wenn die Elementliste auf einen Layernamen verweist, der in der aktuellen Zeichnung noch nicht definiert worden ist, erzeugt **entmod** einen neuen Layer. Die Attribute des neuen Layers sind die Standardvorgabewerte, die von der Option Neu des AutoCAD-Befehls LAYER benutzt werden.

Die Funktion **entmod** *kann* Unterelemente wie Polylinienkontrollpunkte und Blockattribute modifizieren.

Anmerkung Wenn Sie mit **entmod** ein Element in einer Blockdefinition modifizieren, wirkt sich dies auf alle EINFÜGE oder XREF-Referenzen auf diesen Block aus. Außerdem können Elemente in Blockdefinitionen von **entdel** *nicht* gelöscht werden.

Eine Anwendung kann durch Aufruf der Funktion **entmake** der Datenbank auch ein Element hinzufügen. Wie bei **entmod** ist auch das Argument von **entmake** eine Liste, deren Format der von **entget** zurückgegebenen Liste ähnelt. Das neue Element, das durch die Liste beschrieben wird, wird an die Datenbank angefügt (es wird zum letzten Element der Zeichnung). Wenn es sich um ein komplexes Element (eine Polylinie oder einen Block) handelt, wird es erst in die Datenbank aufgenommen, wenn es vollständig ist.

Der folgende Ausdruck erstellt einen Kreis auf dem Layer `MEIN LAYER`.

```
(entmake '((0 . "CIRCLE")          Objekttyp
           (8 . "MEINLAYER")       Layer
           (10 5.0 7.0 0.0)        Mittelpunkt
           (40 . 1.0)              Radius
) )
```

Die Einschränkungen bei **entmake** ähneln denen von **entmod**:

- An erster oder zweiter Stelle in der Liste *muß* der Objekttyp spezifiziert werden. Der Typ muß ein gültiger DXF-Gruppencode sein. Wenn an der ersten Stelle nicht der Typ angegeben wird, kann dort *nur* der Name des Elements angegeben werden: Gruppe -1 (der Name wird nicht in der Datenbank gespeichert).
- AutoCAD *muß* alle Objekte (außer Layer), auf welche die Liste verweist, erkennen. Es gibt eine Ausnahme: **entmake** akzeptiert neue Layernamen.
- An **entmake** übergebene interne Felder werden ignoriert.
- **entmake** *kann keine* Ansichtsfensterelemente erstellen.

entmod und **entmake** führen dieselben Konsistenzprüfungen an den ihnen übergebenen Elementdaten durch wie der Befehl DXFIN beim Lesen von DXF-Dateien. Sie scheitern, wenn sie keine gültigen Zeichnungselemente erstellen können.

9.2.2.1 Arbeiten mit Blöcken

In einer Anwendung ist keine direkte Methode zur Überprüfung vorhanden, ob auf einen Block, der in der Tabelle BLOCK aufgelistet ist, tatsächlich durch ein in die Zeichnung eingefügtes Objekt verwiesen wird. Mit dem folgenden Code können Sie die Zeichnung dahingehend überprüfen, ob Verweise auf Blöcke vorliegen:

```
(ssget "X" '((2 . "BLOCKNAME")) )
```

Sie müssen ebenfalls alle Blockdefinitionen auf das Vorhandensein von verschachtelten Blöcken überprüfen.

9.2.2.2 Unbenannte Blöcke

Die Blockdefinitionstabelle ((BLOCK) einer Zeichnung kann unbenannte Blöcke enthalten, die AutoCAD erstellt, um Schraffurmuster und Assoziativbemaßung zu unterstützen. Die Funktion **entmake** kann neben `*Dnnn` (Bemaßungen) und `*Xnnn` (Schraffurmuster) noch andere unbenannte Blöcke erstellen. Unbenannte Blöcke ohne Referenz werden zu Beginn jeder Arbeitssitzung aus der BLOCK-Tabelle gelöscht. Unbenannte Blöcke mit Referenz (Blöcke, die eingefügt wurden) werden *nicht* gelöscht. Sie können mit **entmake** eine Blockreferenz (ein Objekt) auf einen unbenannten Block erstellen (einfügen). (Sie können jedoch *keinen* unbenannten Block an den Befehl EINFÜGE übergeben.) Sie können den Block mit **entmake** auch neu definieren. Sie können die Elemente eines Blocks (aber nicht das Blockobjekt selbst) mit **entmod** modifizieren.

Der Name (Gruppe 2) eines von AutoLISP oder ADS erstellten Blocks hat die Form `*Unnn`, wobei `nnn` eine von AutoCAD erzeugte Zahl ist. Außerdem wird das niederwertige Bit des *Blocktyp-Flags* (Gruppe 70) eines unbekannten Blocks auf 1 gesetzt. Wenn **entmake** einen Block erzeugt, dessen Name mit einem Sternchen (`*`) anfängt und dessen "Unbenannt-Bit" gesetzt ist, behandelt AutoCAD diesen Block als unbenannten Block und weist ihm einen Namen zu. Zeichen hinter dem `* in der an` **entmake** übergebenen Namenszeichenkette werden ignoriert.

Anmerkung Namen unbenannter Blöcke bleiben *nicht* konstant. Obwohl ein unbenannter Block mit Referenz permanent wird, kann sich der aus Zahlen bestehende Teil seines Namens von Sitzung zu Sitzung ändern.

9.2.2.3 Erstellung komplexer Elemente

Um ein komplexes Element (eine Polylinie oder einen Block) zu erstellen, rufen Sie **entmake** für jedes Subelement erneut auf. Wenn **entmake** eine Anfangskomponente eines komplexen Elements erhält, erstellt die Funktion eine temporäre Datei, in der die Definitionsdaten gesammelt werden (und die erweiterten Daten, sofern vorhanden. Siehe "Erweiterte Daten (Extended Data) - Xdata" Bei jedem nachfolgenden Aufruf von **entmake** prüft die Funktion, ob die temporäre Datei existiert. Wenn ja, wird das neue Subelement an die Datei angefügt. Ist die Definition des komplexen Elements abgeschlossen (d. h. wenn **entmake** ein entsprechendes Seqend- oder Endblk-Elements), wird das Element auf Konsistenz überprüft. Ist es gültig wird es der Zeichnung hinzugefügt. Die Datei wird gelöscht, wenn das komplexe Element vollständig ist oder seine Erstellung abgebrochen wurde.

Anmerkung Das Element erscheint erst in der Zeichnungsdatenbank, wenn das abschließende Seqend- oder Endblk-Subelement **entmake** zugeführt wurde. Insbesondere kann mit Hilfe von **entlast** *nicht* auf das zuletzt erstellte Subelement eines noch nicht abgeschlossenen komplexen Elements zugegriffen werden.

Wie in den vorhergehenden Ausführungen bereits angedeutet, kann **entmake** nur jeweils ein komplexes Element auf einmal erstellen. Wenn **entmake** während der Erstellung eines komplexen Elements ungültige Daten oder ein Element erhält, das kein passendes Subelement ist, werden sowohl das ungültige Element als auch das gesamte komplexe Element verworfen. Um die Erzeugung eines komplexen Elements abzubrechen, rufen Sie **entmake** ohne Angabe von Argumenten auf.

Das folgende Beispiel enthält fünf Aufrufe von **entmake**, die ein einzelnes komplexes Element, eine Polylinie, erstellen. Die Polylinie hat den Linientyp "Strich" (DASHED) und die Farbe BLAU. Sie hat drei Kontrollpunkte an den Koordinaten (1,1,0), (4,6,0) und (3,2,0). Alle anderen fakultativen Definitionsdaten nehmen die Vorgabewerte an. (Damit dieses Beispiel korrekt abläuft, muß der Linientyp DASHED geladen sein.)

```
(entmake '((0 . "POLYLINE")                    Objekttyp
           (62 . 5)                            Farbe
           (6 . "dashed")                      Linientyp
           (66 . 1)                            Kontrollpunkte
folgen
) )
(entmake '((0 . "VERTEX")                      Objekttyp
           (10 1.0 1.0 0.0)                    Startpunkt
) )
(entmake '((0 . "VERTEX")                      Objektart
           (10 4.0 6.0 0.0)                    Zweiter Punkt
) )
(entmake '((0 . "VERTEX")                      Objektart
           (10 3.0 2.0 0.0)                    Dritter Punkt
) )
(entmake '((0 . "SEQEND")))                    Sequenzende
```

Anmerkung Bei der Definition von punktierten Paaren wie im obigen Beispiel *muß* der Punkt von Leerzeichen umgeben sein. Ist das nicht der Fall, erhalten Sie die Fehlermeldung "Ungültiges punktiertes Paar!".

Blockdefinitionen beginnen mit einem Blockelement und enden mit einem Subelement "endblk". Blockdefinitionen können nicht verschachtelt werden und nicht auf sich selbst verweisen. Eine Blockdefinition *kann* Verweise auf andere Blockdefinitionen enthalten.

Anmerkung Bevor Sie einen neuen Block mit **entmake** erstellen, sollten Sie mit Hilfe von **tblsearch** sicherstellen, daß der Name des neuen Blocks eindeutig ist. Die Funktion **entmake** prüft nicht, ob ein Name in der Blockdefinitionstabelle bereits vorhanden ist und so kann sie bestehende Blöcke neu definieren.

Blockreferenzen können das Flag "Attribute folgen" (Gruppe 66) enthalten. Ist dieses vorhanden und gleich 1, wird davon ausgegangen, daß dem Einfügeobjekt eine Reihe von Attributelementen (attrib) folgt. Die Folge von Attributen wird durch ein Sequenzende-Subelement (Seqend) abgeschlossen.

Polylinienelemente enthalten immer das Flag "Kontrollpunkte folgen" (ebenfalls Gruppe 66). Der Wert dieses Flags muß 1 sein, und dem Flag muß eine Folge von Kontrollpunktelementen folgen, abgeschlossen durch ein Seqend-Subelement.

Komplexe Elemente können sowohl im Modellbereich als auch im Papierbereich vorkommen, aber nicht in beiden. Wenn Sie den aktuellen Bereich durch Aufruf von MBEREICH oder PBEREICH (mit **command**) während der Erstellung eines komplexen Elements gewechselt haben, wird das komplexe Element durch einen nachfolgenden Aufruf von **entmake** abgebrochen. Dies kann auch geschehen, wenn das Subelement eine Gruppe 67 enthält, deren Wert nicht mit dem der Gruppe 67 des Element-Headers übereinstimmt.

9.2.3 Elementdatenfunktionen und der Grafikbildschirm

Änderungen in der Zeichnung, die mit den Elementdatenfunktionen vorgenommen wurden, werden auf dem Grafikbildschirm angezeigt, vorausgesetzt das Element, das gelöscht, dessen Löschung rückgängig gemacht, das modifiziert oder erstellt wurde, befindet sich in einem Bereich und auf einem Layer, der gerade auf dem Bildschirm sichtbar ist. Hiervon gibt es eine Ausnahme: wenn mit **entmod** ein Subelement modifiziert wird, wird nicht das Aussehen des gesamten (komplexen) Elements aktualisiert. Wenn zum Beispiel eine Anwendung 100 Kontrollpunkte einer komplexen Polylinie mit 100 Aufrufen von **entmod** modifiziert, dauert das Neuberechnen und Neuanzeigen der gesamten Polylinie viel zu lang. Statt dessen kann die Anwendung eine Folge von Änderungen an Subelementen vornehmen und dann das gesamte Element mit einem einzigen Aufruf der Funktion **entupd** neu anzeigen.

Betrachten wir folgendes Beispiel: Wenn das erste Element in der aktuellen Zeichnung eine Polylinie mit mehreren Kontrollpunkten ist, modifiziert die folgende Anweisung den zweiten Kontrollpunkt der Polylinie und regeneriert die Bildschirmanzeige.

```
(setq e1 (entnext))                    Setzt e1 auf den
Elementnamen der Polylinie
(setq e2 (entnext e1))                 Setzt e2 auf ihren ersten
Kontrollpunkt
(setq v2 (entnext v1))                 Setzt v2 auf ihren zweiten
Kontrollpunkt
(setq ed (entget e2))                  Setzt ed auf die
Kontrollpunktdaten
(setq v2d
  subst
    '(10 1.0 2.0 0.0)
     (assoc 10 v2d)                    Ändert die Position des
Kontrollpunktes in v2d
      ed                               auf den Punkt (1,2,0)
  )
)
(entmod ed)                            Verschiebt den Kontrollpunkt
in der Zeichnung
(entupd e1)                            Regeneriert das
Polylinienelement e1
```

Mit dem Argument von **entupd** kann entweder ein Hauptelement oder ein Subelement angegeben werden. In jedem Fall regeneriert **entupd** das *gesamte* Element. Zwar wird **entupd** hauptsächlich für komplexe Elemente benutzt, es kann damit aber jedes beliebige Element der aktuellen Zeichnung regeneriert werden.

Anmerkung Um sicherzustellen, daß alle Instanzen der Blockreferenzen aktualisiert werden, müssen Sie die Zeichnung durch den Aufruf des AutoCAD-Befehls REGEN (mit **command**) regenerieren. Die Funktion **entupd** reicht nicht aus, wenn das modifizierte Element zu einer Blockdefinition gehört.

9.2.4 Polylinien und Lwpolylinien

Das Lwpolylinien-Element oder die "optimierte Polylinie" ist neu in Release 14. Eine Lwpolylinie wird in einer Datenbank als ein einzelnes grafisches Element definiert. Darin unterscheidet sie sich von einer normalen Polylinie, die als eine Gruppe von Subelementen definiert ist. Lwpolylinien werden schneller dargestellt und benötigen weniger Festplattenspeicher und RAM.

In Release 14 werden 3D-Polylinien immer als standardmäßige Polylinienelemente erstellt. 2D-Polylinien werden als Lwpolylinienelemente erstellt, wenn sie nicht mit dem Befehl PEDIT gekrümmt oder angepaßt wurden. Wenn eine Zeichnung aus einem älteren Release in Release 14 geöffnet wird, werden alle 2D-Polylinien automatisch in Lwpolylinien konvertiert, wenn sie nicht gekrümmt oder angepaßt wurden oder Xdaten enthalten.

9.2.4.1 Bearbeitung von kurven-angeglichenen und spline-angeglichenen Polylinien

Wenn eine AutoLISP-Anwendung mit Hilfe von **entnext** von einem Kontrollpunkt einer Polylinie zum nächsten geht, stößt sie eventuell auf Kontrollpunkte, die nicht explizit erstellt wurden. Hilfskontrollpunkte werden automatisch durch die Optionen Kurve angleichen und Kurvenlinie des Befehls PEDIT eingefügt. Sie können diese Hilfskontrollpunkte ohne Bedenken ignorieren, weil Änderungen daran bei der nächsten Verwendung von PEDIT durch den Benutzer zum Angleichen der Polylinie überholt werden.

Die Gruppe-70-Flags des Polylinienelements zeigen an, ob die Polylinie an eine Kurve (Bitwert 2) oder ein Spline (Bitwert 4) angeglichen wurde. Wenn keines dieser Bits gesetzt wurde, handelt es sich bei allen Kontrollpunkten der Polylinie um reguläre, vom Benutzer definierte Kontrollpunkte. Wenn jedoch das Bit für Kurvenangleichung (2) gesetzt wurde, ist bei alternierenden Kontrollpunkten der Polylinie in der Gruppe 70 der Bitwert 1 gesetzt; Wenn Sie mit Hilfe von **entmod** die Kontrollpunkte einer solchen Polylinie verschieben wollen, um die Kurve mit Hilfe von PEDIT neu anzugleichen, ignorieren Sie diese Kontrollpunkte.

Wenn das Flagbit des Polylinienelements für Spline-Angleichung (Bit 4) gesetzt ist, sind verschiedenartige Kontrollpunkte - einige mit Flagbit 1 (durch Kurvenangleichung eingefügt, wenn die Systemvariable SPLINESEGS negativ war), einige mit Bitwert 8 (eingefügt durch Spline-Angleichung) und alle anderen mit Bitwert 16 (Splinerahmen-Kontrollpunkt). Wenn Sie mit Hilfe von **entmod** die Kontrollpunkte verschieben und das Spline anschließend neu angleichen wollen, verschieben Sie nur die regulären Kontrollpunkte.

9.2.5 Bearbeitung von nichtgrafischen Objekten

AutoCAD verwendet zwei Typen von nichtgrafischen Objekten, Wörterbuchobjekte und Symboltabellenobjekte. Obwohl es Ähnlichkeiten zwischen diesen beiden Objekttypen gibt, werden sie unterschiedlich bearbeitet.

Alle Objekttypen werden von den Funktionen **entget**, **entmod**, **entdel** und **entmake** unterstützt, auch wenn einige Objekttypen die Ausführung dieser Funktionen individuell festlegen (und dokumentieren) und deshalb die Bearbeitung teilweise oder ganz verweigern. Für in AutoCAD integrierte Objekte gelten folgende Richtlinien (alle Richtlinien und Beschränkungen gelten sowohl für grafische als auch für nichtgrafische Objekte): Nichtgrafische Objekte können der Funktion **entupd** *nicht* zugeführt werden.

Der Objekttyp legt bei der Verwendung von **entmake** fest, wo das Objekt abgelegt wird. Wenn beispielsweise ein Layerobjekt **entmake** zugeführt wird, geht es automatisch zur Layersymboltabelle. Wenn **entmake** ein grafisches Objekt zugeführt wird, wird es im "aktuellen Bereich" (Modell oder Papier) abgelegt.

9.2.5.1 Symboltabellenobjekte

Folgende Regeln gelten für Symboltabellen:

- Symboltabelleneinträge dürfen mit wenigen Einschränkungen mit **entmake** erstellt werden, müssen jedoch gültige Datensatzdarstellungen sein. Namenkonflikte können nur in der Tabelle VPORT auftreten. *ACTIVE-Einträge können nicht erstellt werden.
- Mit der Funktion **entget** kann auf den Status von Symboltabelle und Symboltabelleneintrag zugegriffen werden. Die Funktion **tblobjname** kann verwendet werden, um den Elementnamen eines Symboltabelleneintrags aufzurufen.
- Symboltabellen selbst können nicht mit **entmake** erstellt werden; Symboltabelleneinträge können mit **entmake** erstellt werden.
- Referenzgruppen (5, 105) dürfen weder in **entmod** geändert, noch in **entmake** festgelegt werden.

- Viele Felder der Einträge in Symboltabellen dürfen mit **entmod** modifiziert sein. Eine Liste von Datensätzen der Symboltabellen muß den Elementnamen enthalten, damit sie **entmod** zugeführt werden kann. Diesen kann die Funktion **entget** liefern; die Funktionen **tblsearch** und **tblnext** sind hierzu jedoch nicht in der Lage. Die Gruppe 70 der Symboltabelleneinträge wird bei Operationen mit **entmod** und **entmake** ignoriert.

  Die Änderung der Namen von Symboltabelleneinträgen in bereits vorhandene Namen wird nur für die Symboltabelle VPORT akzeptiert. Die nachfolgenden Einträge dürfen nicht modifiziert oder umbenannt werden. Eine Ausnahme bilden die meisten LAYER-Einträge, die umbenannt werden können, sowie externe Daten, die für alle Symboltabelleneinträge modifiziert werden können.

Einträge, die nicht geändert oder umbenannt werden können

Tabelle	Eintragsname
VPORT	*ACTIVE
LINETYPE	CONTINUOUS
LAYER	Mit Ausnahme der externen Daten dürfen Einträge nicht umbenannt und modifiziert werden

Folgende Einträge dürfen nicht umbenannt werden, sind aber mit Einschränkungen modifizierbar

Einträge, die nicht umbenannt werden können

Tabelle	Eintragsname
LAYER	0
STYLe	STANDARD
DIMSTYLE	STANDARD
BLOCKS	*MODEL_SPACE
BLOCKS	*PAPER_SPACE
APPID	Einträge dürfen nicht umbenannt werden

9.2.5.2 Wörterbuchobjekte

Folgende Regeln gelten für Wörterbuchobjekte:

- Wörterbuchobjekte können mit **entget** untersucht und ihre externen Daten mit **entmod** modifiziert werden. Ihre Einträge können mit **entmod** nicht verändert werden. Alle Zugriffe auf Einträge werden mit den Funktionen **dictsearch** und **dictnext** durchgeführt.
- Der Inhalt der Wörterbucheinträge kann mit **entmod** *nicht* modifiziert werden; dies gilt nicht für externe Daten, die verändert werden können.
- Wörterbucheinträge, die mit ACAD* beginnen, können *nicht* umbenannt werden.

9.3 Erweiterte Daten (Extended Data) - Xdata

Es stehen verschiedene AutoLISP-Funktionen zur Verarbeitung *erweiterter Daten* zur Verfügung, die von Anwendungen erstellt werden, welche mit ARX oder AutoLISP geschrieben wurden. Wenn ein Element erweiterte Daten enthält, folgen diese den regulär definierten Daten des Elements.

Es stehen folgende Funktionen zur Bearbeitung erweiterter Daten zur Verfügung:

regapp xdroom xdsize

Sie können die erweiterten Daten eines Elements durch Aufruf von **entget** bereitstellen. Die Funktion **entget** ruft die regulär definierten Daten eines Elements und die erweiterten Daten für die im Aufruf von **entget** angegebenen Anwendungen auf.

Wenn erweiterte Daten mit **entget** bereitgestellt werden, wird der Anfang der erweiterten Daten durch einen Code -3 angezeigt. Der Code -3 ist in einer Liste enthalten, die der ersten Gruppe 1001 vorausgeht. Die Gruppe 1001 enthält den

Namen der zuerst aufgerufenen Anwendung, wie es in der Tabelle gezeigt und in den folgenden Abschnitten beschrieben wird.

Gruppencode	Feld	
(-1, -2	Elementname)	Normale Element-
(0-239	Reguläre Definitionsdatenfelder)	definitionsdaten
	.	
	.	
)	.	
(-3	Satzmarke für erweiterte Daten	Erweiterte Daten
(1001	Registrierter Anwendungsname 1)	
(1000, 1002-1071	XDATA-Felder)	
	.	
	.	
	.	
(1001	Registrierter Anwendungsname 2)	
(1000, 1002-1071	XDATA-Felder)	
	.	
	.	
	.	
(1001	Registrierter Anwendungsname 3)	
	.	
	.	

9.3.1 Organisation von erweiterten Daten

Erweiterte Daten bestehen aus einer oder mehreren Gruppen 1001, wobei jede mit einem eindeutigen Anwendungsnamen beginnt.

Die von **`entget`** zurückgegebenen Gruppen erweiterter Daten folgen den Definitionsdaten in der Reihenfolge, in der sie in der Datenbank gespeichert sind. Dies wird schematisch in der Abbildung dargestellt.

Innerhalb der einzelnen Anwendungsgruppen werden Inhalt, Bedeutung und Organisation der Daten von der Anwendung selbst definiert. AutoCAD behält diese Informationen bei, verwendet sie aber nicht. Die Tabelle zeigt auch, daß die Gruppencodes für erweiterte Daten im Bereich 1000-1071 liegen. Viele dieser Gruppencodes stehen für bekannte Datentypen:

Zeichenkette

1000. Zeichenketten in erweiterten Daten können bis zu 255 Byte lang sein (das 256. Byteist für das Zeichen NULL reserviert).

Anwendungs-name

1001 (ebenfalls eine Zeichenkette). Anwendungsnamen können maximal 31 Byte lang sein (das 32. Byte ist für das Zeichen Null reserviert) und müssen den Regeln für Symboltabellennamen (wie Layernamen) folgen. Ein Anwendungsname kann Buchstaben, Ziffern und die Sonderzeichen Dollarzeichen (`$`), Bindestrich (-) und Unterstrich (_) enthalten. Er darf *keine* Leerzeichen enthalten. Buchstaben im Namen werden in Großbuchstaben umgewandelt.

Layername

1003. Name eines mit zu den erweiterten Daten verknüpften Layers.

Datenbankreferenz

1005. Referenz eines Elements in der Zeichnungsdatenbank.

3D-Punkt

1010. Drei Realzahlenwerte, die in einem Punkt enthalten sind.

Reelle Zahl

1040. Ein Realzahlenwertt.

Ganzzahl

1070. Eine 16-Bit-Ganzzahl (mit oder ohne Vorzeichen).

Lang

1071. Eine 32-Bit-Ganzzahl (`Lang`) mit Vorzeichen. Wenn der in einer Gruppe 1071 erscheinende Wert eine `kurze` Ganzzahl oder ein Realzahlenwert ist, wird er in eine `lange` Ganzzahl umgewandelt; wenn der Wert ungültig ist (zum Beispiel eine Zeichenkette), wird er in eine lange Null (`0L`) umgewandelt.

Anmerkung AutoLISP verwaltet Gruppen 1071 als Realzahlwerte. Wenn Sie mit **`entget`** die Definitionsliste eines Elements aufrufen, die eine Gruppe 1071 enthält, wird der Wert als reelle Zahl zurückgegeben, wie im folgenden Beispiel zu sehen ist:

(1071 . 12.0)

Wenn Sie eine Gruppe 1071 in einem Element mit **`entmake`** oder **`entmod`** erstellen wollen, können Sie entweder einen reellen Wert oder einen Ganzzahlwert benutzen, wie im folgenden Beispiel gezeigt wird:

```
(entmake '((..... (1071 . 65537) .... ))) 12) .... )))
(entmake '((..... (1071 . 65537) .... ))) 12.0) .... )))
(entmake '((..... (1071 . 65537) .... ))) 65537.0) .... )))
 (entmake '((..... (1071 . 65537) .... ))) 65537) .... )))
```

AutoLISP gibt auf trotzdem den Gruppenwert als reelle Zahl zurück:

```
(entmake '((..... (1071 . 65537) .... ))) 65537) .... )))
```

Der vorhergehende Ausdruck gibt folgendes Ergebnis zurück:

(1071 . 65537.0)

ARX verwaltet Gruppen 1071 immer als `lange` Ganzzahlen.

Verschiedene andere Gruppen erweiterter Daten haben in diesem Zusammenhang eine besondere Bedeutung (wenn die Anwendung sie benutzen will):

Steuerzeichen-kette

1002. Eine Steuerzeichenkette in erweiterten Daten kann entweder `"{"` oder `"}"` sein. Diese Klammern ermöglichen der Anwendung, ihre Daten in Listen zu unterteilen. Die öffnende Klammer leitet eine Liste ein, die schließende Klammer beendet sie. Listen können verschachtelt sein.

Anmerkung Wenn eine Gruppe 1001 in einer Liste erscheint, wird sie als Zeichenkette behandelt und *nicht* als Beginn einer neuen Anwendungsgruppe.

Binäre Daten

1004. Binäre Daten, die in *Gruppen* mit variabler Länge angeordnet werden, welche in ADS mit der Struktur `ads_binary` verarbeitet werden können. Die maximale Länge einer solchen Gruppe ist 127 Byte.

Anmerkung AutoLISP kann solche binären Gruppen *nicht* direkt verarbeiten; es gelten also für binäre Gruppen dieselben Vorkehrungen wie für `lange` Gruppen (1071).

Weltbereichs-position

1011. Im Unterschied zu einem einfachen 3D-Punkt werden die WKS-Koordinaten zusammen mit dem übergeordneten Element verschoben, skaliert, gedreht und gespiegelt, zu dem die erweiterten Daten gehören. Wenn der Befehl STRECKEN auf das übergeordnete Element angewandt wird, wird die WKS-Position ebenfalls gestreckt, wenn dieser Punkt innerhalb des Auswahlfensters liegt.

Weltbereichs-verschiebung

1012. Ein 3D-Punkt, der zusammen mit dem übergeordneten Element skaliert, gedreht oder gespiegelt, jedoch *nicht* gestreckt oder verschoben wird.

Weltrichtung

1013. Ein 3D-Punkt, der zusammen mit dem übergeordneten Element skaliert, gedreht oder gespiegelt, jedoch *nicht* gestreckt oder verschoben wird. Die WKS-Richtung ist eine normalisierte Verschiebung, die immer eine einheitliche Länge aufweist.

Abstand

1041. Ein reeller Wert, der zusammen mit dem übergeordneten Element skaliert wird.

Skalierfaktor

1042. Auch ein reeller Wert, der zusammen mit dem übergeordneten skaliert wird.

Die DXF-Gruppencodes für erweiterte Daten werden auch in Anhang C, "DXF-Gruppencodes" beschrieben.

9.3.2 Registrierung einer Anwendung

Um von AutoCAD erkannt zu werden, müssen der oder die von der Anwendung benutzten Namen registriert werden. Anwendungsnamen werden unter den erweiterten Daten jedes Elements gespeichert, das sie benutzt, sowie in der Tabelle APPID. Die Registrierung erfolgt mit der Funktion **regapp**. Die Funktion **regapp** spezifiziert eine Zeichenkette, die als Anwendungsname benutzt werden soll. Wenn der Name erfolgreich in APPID aufgenommen wird, gibt die Funktion den Namen der Anwendung zurück, andernfalls gibt sie `nil` zurück. Das Ergebnis `nil` zeigt an, daß der Name in der Symboltabelle bereits vorhanden ist. Dies ist kein wirklicher Fehler, sondern ein erwarteter Rückgabewert, weil der Anwendungsname nur einmal pro Zeichnung registriert werden muß.

Um registriert zu werden, sollte eine Anwendung zunächst prüfen, ob ihr Name nicht bereits in der Tabelle APPID vorhanden ist. Ist der Name nicht vorhanden, muß die Anwendung ihn registrieren. Andernfalls kann sie einfach fortfahren und die Daten so benutzen, wie es später in diesem Abschnitt beschrieben wird.

Die folgende Anweisung zeigt die typische Verwendung von **regapp**. Die Funktion **tblsearch** wird in "Zugriff auf Symboltabellen und Wörterbücher" beschrieben.

```
(setq anwname "MEINEAPP_2356")
    Eindeutiger Anwendungsname
(if  (tblsearch "appid" anwname)
     Prüft, ob bereits registriert
  (princ (strcat
    "\n" anwname " bereits registriert. )) "))
  (if (=  (regapp appname) nil)
     Andere Probleme
    (princ (strcat
      "\nKann XDATA für " anwname nicht registrieren". )) "))
  )
)
```

Anmerkung Die Funktion **regapp** stellt eine Sicherheitsmaßnahme dar, kann aber nicht garantieren, daß zwei getrennte Anwendungen nicht ein und denselben Namen gewählt haben. Eine Möglichkeit, dies sicherzustellen, besteht in der Einführung eines Namensgebungsschemas, das den Namen des Unternehmens oder des Produkts mit einer eindeutigen Zahl verbindet (wie eine Telefonnummer oder aktuelles Datum und Uhrzeit).

9.3.3 Auffinden von erweiterten Daten

Mit Hilfe des Aufrufs von **entget** kann eine Anwendung die von ihr registrierten erweiterten Daten erhalten. Die Funktion **entget** kann sowohl die Definitionsdaten als auch die erweiterten Daten für die angeforderten Anwendungen zurückgeben. Es ist ein weiteres Argument erforderlich, *Anwendung*, das die Anwendungsnamen angibt. Die an **entget** übergebenen Namen müssen mit den durch einen vorhergehenden Aufruf von **regapp** registrierten Anwendungen übereinstimmen; sie können auch Platzhalterzeichen enthalten.

Assoziativschraffurmuster enthalten als Vorgabe erweiterte Daten. Der folgende Code zeigt die Assoziationsliste dieser erweiterten Daten.

Befehl: **(entget (car (entsel)) '("ACAD"))**

Objekt wählen : *Auswahl einer Assoziativschraffur*

Bei Eingabe dieses Ausdrucks in der Befehlszeile erhalten Sie etwa folgende Liste:

```
((-1 . <Elementname: 600000c0>) (0 . "INSERT") (8 . "0") (2 . "*X0")
(10 0.0 0.0 0.0) (41 . 1.0) (42 . 1.0) (50 . 0.0) (43 . 1.0) (70 . 0) (71 . 0)
(44 . 0.0) (45 . 0.0) (210 0.0 0.0 1.0) (-3 ("ACAD" (1000 . (HATCH)
(1002 . "{") (1070 . 16) (1000 . "LINE") (1040 . 1.0) (1040 . 0.0)
(1002 . "}"))))
```

Das folgende Beispiel zeigt eine typische Anweisungssequenz zum Auffinden erweiterter Daten für zwei angegebene Anwendungen. Beachten Sie, daß das Argument `Anwendung` die Anwendungsnamen in Listenform übergibt.

```
(setq working_elist
  (entget ent_name
    '("EIGENE_ANW" "EINE_ANDERE")      Nur erweiterte Daten von " EIGENE_ANW "
  )                                    und "EINE_ANDERE" werden ermittelt
)
(if working_elist
  (progn
    ...                                Aktualisiert Arbeitselementgruppen
    (entmod working_elist)             Nur erweiterte Daten von registrierten
  )                                    Anwendungen, die noch in der Liste working_elist
                                       sind, werden modifiziert
```

Wie das Beispiel zeigt, können Sie mit **entget** ermittelte erweiterte Daten durch Aufruf von **entmod** modifizieren, genauso wie Sie mit Hilfe von **entmod** normale Definitionsdaten modifizieren. (Sie können erweiterte Daten auch erstellen, indem Sie sie in der an **entmake** zu übergebenden Elementliste definieren.)

Die Rückgabe der erweiterten Daten nur für diejenigen Anwendungen, die speziell angefordert wurden, schützt eine Anwendung davor, die Daten einer anderen Anwendung zu beschädigen. Dies steuert auch den Speicherplatz, den eine Anwendung benötigt, und vereinfacht die Verarbeitung der erweiterten Daten, die von einer Anwendung durchgeführt werden muß.

Anmerkung Da die durch das Argument `Anwendung` übergebenen Zeichenketten Platzhalterzeichen enthalten können, werden durch Angabe des Anwendungsnamens `"*"` von **entget** alle einem Element zugeordneten erweiterten Daten zurückgegeben.

9.3.4 Zuordnung erweiterter Daten zu einem Element

Mit Hilfe erweiterter Daten können Sie Informationen jedes beliebigen Typs speichern. Im folgenden Beispiel werden erweiterte Daten einem Element zugeordnet.

Sie müssen zunächst ein Element zeichnen (eine Linie oder einen Kreis) und dann folgende Anweisung eingeben:

```
(setq lastent (entget (entlast)))      Ruft die Assoziationsliste der  Definitions-
                                       daten für das letzte Element auf
(regapp "NEWDATA")                     Registriert den Anwendungsnamen
(setq exdata                           Setzt die Variable exdata gleich
 '((-3 ("NEWDATA"                      den neuen erweiterten Daten
  (1000 . "Diese Sache ist neu!")      In diesem Fall ein Text
  )))
)
(setq newent
  (append lastent exdata))             Fügt die neue Dateiliste an die Liste des Elements an.
(entmod newent)                        Modifiziert das Element anhand der neuen
                                       Definitionsdaten
```

Um zu prüfen, ob Ihre neuen erweiterten Daten dem Element zugeordnet wurden, geben Sie folgenden Ausdruck ein, und wählen Sie das Objekt aus.

```
(entget (car (entsel)) '("NEUDATEN"))
```

Dieses Beispiel zeigt die grundlegende Methode der Zuordnung von erweiterten Daten zu einem Element.

9.3.5 Verwaltung der Speichernutzung durch erweiterte Daten

Erweiterte Daten sind zur Zeit auf 16 KByte pro Element beschränkt. Da die erweiterten Daten eines Elements von mehreren Anwendungen erstellt und verwaltet werden können, können sich Probleme ergeben, wenn die Größe der erweiterten Daten ihre Grenze erreicht. AutoLISP stellt zwei Funktionen zur Verfügung, **xdsize** und **xdroom**, welche die Verwaltung der Speicherbelegung durch erweiterte Daten unterstützen. Wenn an **xdsize** eine Liste mit erweiterten Daten übergeben wird, gibt die Funktion die Größe des Speicherplatzes (in Byte) zurück, den die Daten belegen werden; wenn **xdroom** der Name eines Elements zugeführt wird, gibt diese Funktion die verbleibende Zahl freier Byte zurück, die von dem Element noch belegt werden können.

Die Funktion **xdsize** liest eine Liste erweiterter Daten, die recht lang sein kann. Diese Funktion ist relativ langsam; wir empfehlen daher, sie nicht allzu häufig zu benutzen. Es empfiehlt sich, sie (in Verbindung mit **xdroom**) im Rahmen der Fehlerbehebung zu benutzen. Wenn ein Aufruf von **entmod** fehlschlägt, können Sie mit Hilfe von **xdsize** und **xdroom** feststellen, ob der Fehler darauf zurückzuführen ist, daß dem Element nicht genügend Platz für erweiterte Daten zur Verfügung stand.

9.3.6 Referenzen in erweiterten Daten

Erweiterte Daten können Referenzen enthalten (Gruppe 1005), um relationale Strukturen innerhalb einer Zeichnung zu speichern. Ein Element kann auf andere verweisen, indem die Referenz des anderen in den eigenen erweiterten Daten gespeichert wird. Die Referenz kann dann später aus den erweiterten Daten aufgerufen und an **handent** zur Ermittlung des anderen Elements übergeben werden. Da eventuell mehr als ein Element auf ein anderes verweist, sind die Referenzen erweiterter Daten nicht unbedingt eindeutig. Der Befehl PRÜFUNG verlangt, daß Referenzen in erweiterten Daten entweder `NULL` oder gültige Elementreferenzen sind (innerhalb der aktuellen Zeichnung). Die beste Möglichkeit sicherzustellen, daß erweiterte Elementreferenzen gültig sind, besteht darin, die Referenz eines Elements, auf das verwiesen wird, direkt mit Hilfe von **entget** aus dessen Definitionsdaten zu holen. Der Referenzwert befindet sich in Gruppe 5.

Wenn Sie auf Elemente in anderen Zeichnungen verweisen (zum Beispiel auf Elemente, die mit XREF zugeordnet werden), können Sie Fehlermeldungen von PRÜFUNG vermeiden, indem Sie erweiterte Elementzeichenketten (Gruppe 1000) anstelle von Referenzen (Gruppe 1005) verwenden, weil die Referenzen von Elementen, auf die extern verwiesen wird, entweder in der aktuellen Zeichnung nicht gültig sind oder ein Konflikt mit gültigen Referenzen vorliegt. Wenn jedoch eine externe Referenz XREF-Zuordnen in XREF-Binden geändert wird oder mit der aktuellen Zeichnung auf andere Weise verbunden wird, muß die Anwendung die Referenzen des Elements entsprechend revidieren.

Wenn Zeichnungen mit Hilfe von EINFÜGE, EINFÜGE*, XREF-Binden (XBIND) oder partiellem DXFIN verbunden werden, werden die Referenzen übertragen, so daß sie in der aktuellen Zeichnung gültig werden. (Wenn die eingehende Zeichnung keine Referenzen benutzt hat, werden neue zugewiesen.) Referenzen erweiterter Elemente, die auf eingehende Elemente verweisen, werden durch Aufruf dieser Befehle ebenfalls übertragen.

Wenn das Element in eine Blockdefinition aufgenommen wird (mit dem Befehl BLOCK), werden dem Element innerhalb des Blocks neue Referenzen zugewiesen. (Wenn das Originalelement mit Hilfe von HOPPLA wiederhergestellt wird, behält es seine ursprünglichen Referenzen.) Die Werte der Referenzen erweiterter Daten bleiben unverändert. Wenn ein Block aufgelöst wird (mit dem Befehl URSPRUNG), *werden* Referenzen erweiterter Daten übertragen, und zwar ähnlich ihrer Übertragung bei einer Verbindung von Zeichnungen. Wenn die Referenz der erweiterten Daten auf ein Element verweist, das nicht in dem Block enthalten ist, bleibt sie unverändert. Wenn jedoch die Referenz der erweiterten Daten auf ein Element verweist, das in dem Block enthalten ist, wird der Datenreferenz der Wert der Referenz des neuen (aufgelösten) Elements zugewiesen.

9.4 Xrecord-Objekte

Xrecord-Objekte werden zum Speichern und Verwalten von beliebigen Daten verwendet. Sie bestehen aus DXF-Gruppencodes mit "Normalobjekt"-Gruppen (d. h. Nicht-Xdata-Gruppencodes), deren Bereich sich von 1 bis 369 für unterstützte Bereiche erstreckt. Dieses Objekt ähnelt in seinem Konzept XDaten, es unterliegt jedoch keinen Einschränkungen hinsichtlich Größe oder Reihenfolge.

Xrecord-Objekte sind so angelegt, daß ihre Funktionsweise die Releases R13c0 bis R13c3 nicht beeinträchtigt. Werden sie jedoch in eine Ebene vor R13c4 eingelesen, werden Xrecord-Objekte entfernt.

Folgende Beispiele bieten Methoden zur Erstellung und Auflistung von Xrecord-Daten.

```
(defun C:MAKEXRECORD( / xrec xname )
  ; Erstellen der Xrecord-Datenliste
  (setq xrec '((0 . "XRECORD")(100 . "AcDbXrecord")
    (1 . (1 . "Dies ist eine Testliste")
    (10 1.0 2.0 0.0) (40 . 3.14159) (50 . 3.14159)
    (62 . 1) (70 . 180))
  )

  ; Entmakex zur Erstellung eines Xrecord ohne Besitzer verwenden
  (setq xname (entmakex xrec))

  ; Neuen Xrecord zum benannten Objekt-Wörterbuch hinzufügen
  (dictadd (namedobjdict) "XRECLIST" xname)

  (princ)
)


(defun C:LISTXRECORD ( / xlist )
  ; Auffinden des Xrecord im benannten Objekt-Wörterbuch
  (setq xlist (dictsearch (namedobjdict) "XRECLIST"))

  ; Drucken Xrecord-Datenliste
  (princ xlist)

  (princ)
)
```

9.5 Zugriff auf Symboltabellen und Wörterbücher

AutoLISP bietet Funktionen für den Zugriff auf Einträge in Symboltabellen und Wörterbüchern. Hierbei handelt es sich um die folgenden Funktionen:

dictsearch	namedobjdict	tblnext	tblsearch
dictnext	snvalid	tblobjname	

Beispiele für die Funktionen **tblnext** und **tblsearch** finden Sie in diesem Abschnitt. Informationen über die anderen Zugriffsfunktionen für Symboltabellen und Wörterbücher entnehmen Sie bitte Kapitel 13. Weitere Informationen über nichtgrafische Objekte finden Sie unter "Bearbeitung von nichtgrafischen Objekten."

9.5.1 Symboltabellen

Symboltabelleneinträge können auch mit folgenden Funktionen bearbeitet werden:

entdel	entmake	handent
entget	entmod	

Die Funktion **tblnext** liest Symboltabelleneinträge nacheinander, die Funktion **tblsearch** sucht nach bestimmten Einträgen. Tabellennamen werden als Zeichenketten angegeben. Gültige Namen sind `"LAYER"`, `"LTYPE"`, `"VIEW"`, `"STYLE"`, `"BLOCK"`, `"UCS"`, `"VPORT"`, `"DIMSTYLE"` und `"APPID"`. Beide Funktionen geben Listen mit DXF-Gruppencodes zurück, die den von **entget** zurückgegebenen Elementdaten ähneln.

Der erste Aufruf von **tblnext** gibt den ersten Eintrag in der angegebenen Tabelle zurück. Weitere Aufrufe, die sich auf dieselbe Tabelle beziehen, geben die nachfolgenden Einträge zurück, wenn das zweite Argument von **tblnext** (*erster*) nicht Null ist; in diesem Fall gibt **tblnext** den ersten Eintrag erneut zurück.

Im folgenden Beispiel ermittelt die Funktion **GETBLOCK** den Symboltabelleneintrag für den ersten Block (sofern vorhanden) in der aktuellen Zeichnung und zeigt ihn in Form einer Liste an.

```
(defun C:GETBLOCK (/ blk ct)
  (setq blk (tblnext "BLOCK" 1))          Ermittelt den ersten BLOCK-Eintrag
  (setq ct 0)                             Setzt ct (einen Zähler) auf 0
  (textpage)                              Schaltet zum Textbildschirm
  (princ "\nErgebnisse von GETBLOCK: ")
  (repeat (length entl)                   Wiederholt für alle Einträge der Liste
    (print (nth ct entl))                 Gibt ein Zeilenvorschubzeichen gefolgt von
                                          den Listeneinträgen aus.
    (setq ct (1+ ct))                     Erhöht den Zähler um eins
  )
  (princ)                                 Beendet das Programm
)
```

Aus der Blocktabelle ermittelte Einträge enthalten eine Gruppe -2, die den Namen des ersten Elements in der Blockdefinition enthält. Wenn der Block leer ist, ist dies der Name des Elements ENDBLK des Blocks, den man bei nicht-leeren Blöcken nie antrifft. In einer Zeichnung mit einem einzigen Block mit Namen BOX wird beim Aufruf von **GETBLOCK** folgendes ausgegeben (der Wert für den Namen variiert von Sitzung zu Sitzung).

```
Ergebnisse von GETBLOCK:
(0 . "BLOCK")
(2 . "BOX")
(70 . 0)
(10 9.0 2.0 0.0)
(-2 . <Elementname: 40000126>)
```

Wie bei **tblnext** ist auch bei **tblsearch** das erste Argument eine Zeichenkette, die eine Tabelle benennt, das zweite Argument jedoch ist eine Zeichenkette, die ein bestimmtes Symbol in der Tabelle bezeichnet. Wenn das Symbol gefunden wird, gibt **tblsearch** seine Daten zurück. Diese Funktion hat noch ein drittes Argument, *Setzfolg*, mit dessen Hilfe Sie Operationen mit **tblnext** koordinieren können. Wenn *Setzfolg* `nil` ist, hat der Aufruf von **tblsearch** keine Auswirkungen auf **tblnext**; wenn jedoch *Setzfolg* nicht `nil` ist, gibt der nächste Aufruf von **tblnext** den Tabelleneintrag zurück, der dem von **tblsearch** gefundenen Eintrag folgt.

Die Option *setnext* ist hilfreich, wenn Sie die Symboltabelle `VPORT` bearbeiten, da alle Ansichtsfenster in einer bestimmten Ansichtsfensterkonfiguration denselben Namen haben (z. B. *ACTIVE).

Wenn auf die Symboltabelle `VPORT` bei deaktiviertem TILEMODE zugegriffen wird, wirken sich Veränderungen erst aus, wenn TILEMODE aktiviert wird. Verwechseln Sie nicht `VPORTS`, wie es durch die Symboltabelle `VPORT` beschrieben wird, mit Ansichtsfensterelementen im Papierbereich.

Mit der folgenden Anweisung werden die einzelnen Ansichtsfenster in der Konfiguration `4VIEW` bearbeitet.

```
(setq v (tblsearch "VPORT" "4VIEW" T))      Sucht den ersten VPORT-Eintrag
(while (and v (= (cdr (assoc 2 v)) "4VIEW"))
  .
  .                                         . . .  Verarbeitet den Eintrag  . . .
  .
  (setq v (tblnext "VPORT"))                Holt den nächsten VPORT-Eintrag
)
```

9.5.2 Wörterbucheinträge

Ein Wörterbuch ist ein Container-Objekt, das den Symboltabellen von der Funktion her ähnelt. Mit den Funktionen **dictsearch** und **dictnext** können Wörterbucheinträge abgefragt werden. Jeder Wörterbucheintrag besteht aus einem Textnamenschlüssel und einer Besitzreferenz für das Eintragsobjekt. Wörterbucheinträge können entfernt werden, indem Eintragsobjektnamen der Funktion **entdel** zugeführt werden. Der Textnamenschlüssel verwendet die gleiche Syntax und die gleichen gültigen Zeichen wie die Namen für Symboltabellen.

9.5.2.1 Zugriff auf AutoCAD-Gruppen

Das folgende Beispiel stellt eine Methode für den Zugriff auf Elemente in einer Gruppe dar. Hierbei wird davon ausgegangen, daß in der aktuellen Zeichnung eine Gruppe mit dem Namen G1 vorhanden ist.

```
(setq objdict (namedobjdict))
(setq grpdict (dictsearch objdict "ACAD_GROUP"))
```

Auf diese Weise wird die Variable grpdict auf die Elementdefinitionsliste des ACAD_GROUP-Wörterbuchs gesetzt und folgendes zurückgegeben:

((-1 . <Elementname: 8dc10468>) (0 . "DICTIONARY") (5 . "D")
(102 . "{ACAD_REACTORS") (330 . <Elementname: 8dc10460>)
(102 . "}") (100 . "AcDbDictionary") (3 . "G1")
(350 . <Elementname: 8dc41240>))

Der folgende Code setzt die Variable `group1` auf die Elementdefinitionsliste der Gruppe G1.

```
(setq group1 (dictsearch (cdar grpdict) "G1"))
```

Folgendes wird zurückgegeben:

((-1 . <Elementname: 8dc10518>) (0 . "GROUP") (5 . "23")
(102 . "{ACAD_REACTORS") (330 . <Elementname: 8dc10468>)
(102 . "}") (100 . "AcDbGroup") (300 . "Linie und Kreis") (70 . 0) (71 . 1)
(340 . <Elementname: 8dc10510> <Elementname 8dc10550>))

Die 340-Gruppencodes entsprechen den Elementen, die zur Gruppe gehören.

10 Arbeiten mit Dialogfeldern

Anwender können in AutoCAD Dialogfelder erstellen und implementieren, die mit ihren Anwendungen zusammenarbeiten. Dateien der Dialogsteuersprache DCL (Dialog Control Language) legen das Aussehen eines Dialogfelds fest. DCL wird im Teil III dieses Handbuchs, "Programmierbare Dialogfelder", erläutert. Die Funktionalität eines Dialogfelds wird von einer AutoLISP- oder einer ADS-Anwendung gesteuert. In diesem Kapitel wird beschrieben, wie Dialogfelder zusammen mit AutoLISP verwendet werden können. Obwohl in diesem Kapitel einige Beispiele für DCL-Dateien vorgestellt werden, ist es unter Umständen sinnvoll, zuerst Teil III zu lesen.

10.1 Öffnen und Schließen von Dialogfeldern

AutoLISP stellt Ihnen folgende Funktionen zum Öffnen und Schließen von Dialogfeldern zur Verfügung

done_dialog	new_dialog	term_dialog
load_dialog	start_dialog	unload_dialog

Bevor Sie Dialogfeldfunktionen verwenden können, müssen Sie eine DCL-Datei erstellen, die das Dialogfeld definiert. Speichern Sie den folgenden DCL-Code in einer mit dem Namen *hallo.dcl*, die sich in einem Verzeichnis Ihres Supportpfads befindet. Diese DCL-Datei definiert einen Dialog mit der Bezeichnung Beispiel-Dialogfeld, der ein Textfeld und eine einzige Schaltfläche OK enthält.

```
Hallo: Dialog {
  Bezeichnung = "Beispiel-Dialogfeld";
  : Text { Bezeichnung = "Hallo, Leute."; }
  ok_only;
}
```

Die Anzeige eines Dialogfelds erfordert einige Schritte. Zuerst müssen Sie mit der Funktion **load_dialog** die DCL-Datei in den Speicher laden und die DCL-Bezeichnungsnummer ermitteln. Danach rufen Sie **new_dialog** auf und übergeben der Funktion als Argumente den Dialognamen und die DCL-Bezeichnungsnummer. Wenn der Rückgabewert von **new_dialog** nicht `nil` ist, muß **start_dialog** aufgerufen werden, um die Kontrolle über den Dialog an AutoCAD und den Benutzer zu übergeben. Da die Dialogfeldkomponente `ok_only` (die Schaltfläche OK) ein vordefiniertes Feld ist, ist seine zugehörige Operation ebenfalls vordefiniert. Normalerweise würden Sie Operationen zuweisen sowie den Status und den Wert einer Dialogfeldkomponente festlegen, bevor Sie **start_dialog** aufrufen (siehe "Operationsausdrücke und Rückmeldungen"). Die der Dialogfeldkomponente `ok_only` zugeordnete Operation lautet **done_dialog**. Wenn Sie die Schaltfläche OK wählen, führt AutoCAD den Aufruf von **done_dialog** der AutoLISP-Anwendung zu und beendet den Dialog. Danach führen Sie die DCL-Bezeichnungsnummer der Funktion **unload_dialog** zu, welche die DCL-Datei aus dem Speicher entfernt. (Eine vollständige Beschreibung der aufeinanderfolgenden Dialogfeldfunktionen finden Sie unter "Funktionsfolgen.")

Wenn Sie die Datei *hallo.dcl* in einem Verzeichnis Ihres Supportpfads sichern, können Sie die folgende AutoLISP-Funktion zur Anzeige verwenden. (Das vorliegende Release besitzt fast keine Fehlerprüfung, deshalb ist die Struktur leichter zu erkennen.)

```
(defun C:HALLO( / dcl_id )
  (setq dcl_id (load_dialog "hallo.dcl"))      Die DCL-Datei laden
  (if (not (new_dialog "hallo" dcl_id))        Das Dialogfeld initialisieren
    (exit)                                     Verlassen, wenn ein Fehler auftritt
  )
  (start_dialog)                               Das Dialogfeld anzeigen
  (unload_dialog dcl_id)                       Die DCL-Datei aus dem Speicher entfernen
  (princ)
)
```

Der Aufruf von **start_dialog** bleibt aktiv, bis der Anwender eine Dialogfeldkomponente auswählt (normalerweise eine Schaltfläche), deren zugehöriger *Operationsausdruck* **done_dialog** aufruft. Der Aufruf von **done_dialog** kann von der Komponente explizit angezeigt werden oder durch das auf `true` gesetzte Attribut `is_cancel` in der ausgewählten Komponente angezeigt werden.

Warnung Theoretisch sollte das Dialogfeld die Steuerung von Eingaben zu dem Zeitpunkt übernehmen, zu dem Sie **start_dialog** aufrufen. Unter Windows und auf anderen Plattformen übernimmt es jedoch die Steuerung, sobald Sie

new_dialog aufrufen. Auf das Schreiben von Programmen hat dies keinen Einfluß. Beim interaktiven Aufruf dieser Funktionen müssen Sie sie jedoch als eine Anweisung in die AutoCAD-Befehlszeile eingeben. Schließen Sie sie in einer Funktion wie **progn** ein. Andernfalls kann beim interaktiven Aufruf von **new_dialog** das System hängenbleiben. Der interaktive Aufruf von **new_dialog** und **start_dialog** kann bei der Fehlersuche hilfreich sein. (Ein Beispiel für die interaktive Verwendung dieser Funktionen finden Sie in "Fehlerbehebung in DCL.")

Während ein Dialogfeld aktiv ist -- das heißt, während des Aufrufs von **start_dialog** -- können Sie die unten aufgeführten AutoLISP-Funktionen *nicht* verwenden. Diese Funktionen beeinflussen entweder die Bildschirmanzeige, die nicht verändert werden darf, während ein Dialogfeld sichtbar ist, oder erfordern eine Anwendereingabe, die nicht vom Dialogfeld verarbeitet wird.

command	getcorner	getstring	nentsel
entdel	getdist	graphscr	osnap
entmake	getint	grdraw	prompt[a]
entmod	getkword	grread	redraw
entsel	getorient	grtext	ssget [b]
entupd	getpoint	grvecs	textpage
getangle	getreal	menucmd	textscr

Funktionen, die Text ausgeben, wie **print, princ**, *und* **prin1,** *sind insbesondere zur Anzeige bei der Fehlersuche in einem Dialogfeld nützlich. Verwenden Sie diese Funktionen nicht für ein fertiges Produkt. Wenn Ihr Dialogfeld den Bereich für die Befehlseingabe überlappt, überschreiben diese Funktionen diesen Teil des Dialogfelds.*

Interaktive Optionen von **ssget** *sind im Gegensatz zu den anderen Optionen nicht erlaubt.*

Ruft Ihr Programm eine dieser Funktionen zwischen den Aufrufen von **start_dialog** und **done_dialog** auf, schließt AutoCAD alle Dialogfelder und zeigt die folgende Fehlermeldung an:

AutoCAD hat diese Funktion zurückgewiesen

Ist der Wert der Systemvariablen CMDACTIVE größer als 7, ist ein Dialogfeld aktiv. Die Systemvariable CMDACTIVE besitzt bitcodierte Werte, welche die Aktivität von Befehlen, Skripts und Dialogfeldern kennzeichnen. Siehe "Systemvariablen" in der *AutoCAD Befehlsreferenz*.

Wie bereits in der zweiten Fußnote der Tabelle erwähnt, sind Aufrufe von **ssget** erlaubt, allerdings nicht zusammen mit interaktiven Optionen.

Soll der Anwender Eingaben auf dem Grafikbildschirm und nicht im Dialogfeld machen (beispielsweise, um einen Punkt zu bestimmen oder ein Objekt auszuwählen), müssen Sie das Dialogfeld *ausblenden*. Dazu rufen Sie **done_dialog** auf, so daß der Grafikbildschirm wieder sichtbar wird, und starten das Dialogfeld erneut, wenn der Benutzer eine Auswahl getroffen hat. Weitere Informationen finden Sie unter "Ausblenden von Dialogfeldern."

Mit der Funktion **term_dialog** werden alle geöffneten Dialogfelder auf einmal geschlossen, als ob der Benutzer sie jeweils einzeln geschlossen hätte. Mit dieser Funktion können Sie eine Reihe verschachtelter Dialogfelder schließen.

10.2 Bearbeiten von Komponenten und Attributen

AutoLISP stellt Ihnen die folgenden Funktionen für die Bearbeitung von Dialogfeldkomponenten und Attributen zur Verfügung:

action_tile	get_tile	set_tile
get_attr	mode_tile	

Beispiele für die Verwendung dieser Funktionen finden Sie in diesem Abschnitt. Weitere Informationen hierzu finden Sie in Kapitel 13, "AutoLISP Funktionskatalog".

10.2.1 Operationsausdrücke und Rückmeldungen

Um zu definieren, welche Operation ausgeführt werden soll, wenn eine bestimmte Komponente eines Dialogfelds aktiviert wird, verknüpfen Sie diese Komponente mit einem AutoLISP-Ausdruck, indem Sie die Funktion

action_tile aufrufen. Dies nennt man einen *Operationsausdruck*. Innerhalb eines Operationsausdrucks müssen Sie oft auf Attribute in der DCL-Datei zugreifen. Dies ermöglichen Ihnen die Funktionen **get_tile** und **get_attr**. Die Funktion **get_attr** ruft die benutzerdefinierten Attribute innerhalb der DCL-Datei auf. Die Funktion **get_tile** ermittelt den aktuellen *Laufzeit*wert einer Komponente auf der Grundlage der Benutzereingabe in dieser Komponente. Operationsausdrücke müssen nach dem Aufruf von **new_dialog** und vor dem Aufruf von **start_dialog** festgelegt werden.

Informationen zu der Art und Weise, nach der der Benutzer eine Komponente ausgewählt oder den Inhalt einer Komponente modifiziert hat, werden dem Operationsausdruck als eine *Rückmeldung* zurückgegeben. In den meisten Fällen kann jede aktive Komponente in einem Dialogfeld eine Rückmeldungsfunktion generieren. Die Antwort eines Operationsausdrucks auf eine Rückmeldung (oft als *Rückmeldungsfunktion* bezeichnet) sollte eine Gültigkeitsprüfung für die verknüpfte Komponente durchführen und Informationen des Dialogfelds aktualisieren, die vom Wert der Komponente abhängen. Das Aktualisieren des Dialogfelds kann das Zurückgeben einer Fehlermeldung, das Deaktivieren anderer Komponenten oder die Anzeige von passendem Text in einem Eingabe- oder Listenfeld einschließen.

Nur die Schaltfläche OK (oder ihre Entsprechung) sollte die Komponentenwerte abfragen, damit die zuletzt gewählten Einstellungen des Anwenders dauerhaft gespeichert werden. Anders ausgedrückt, Sie sollten die mit Komponentenwerten belegten Variablen innerhalb einer Rückmeldung der Schaltfläche OK aktualisieren und nicht die Rückmeldung einer einzelnen Komponente. Wenn eine dauerhafte Variable innerhalb einer Rückmeldungsfunktion einer einzelnen Komponente aktualisiert wird, besteht keine Möglichkeit mehr, die Variable bei der Auswahl der Schaltfläche Abbrechen zurückzusetzen. Entdeckt die Rückmeldung der Schaltfläche OK einen Fehler, sollte eine Fehlermeldung angezeigt und der Fokus auf diejenige Komponente gesetzt werden, die den Fehler verursacht hat. Das Dialogfeld sollte keinesfalls verlassen werden.

Besitzt ein Dialogfeld mehrere Komponenten, die ähnlich behandelt werden können, kann es bequemer sein, diesen Komponenten nur eine Rückmeldungsfunktion zuzuweisen. Der Grundsatz, die Änderungen des Anwenders erst bei der Auswahl der Schaltfläche OK zu übernehmen, gilt weiterhin.

Es gibt neben dem Aufruf von **action_tile** noch zwei weitere Möglichkeiten, Operationen zu definieren. Sie können mit dem Aufruf **new_dialog** eine *Vorgabe*-Operation für das gesamte Dialogfeld definieren oder eine Operation über das Komponentenattribut `Operation` festlegen. Eine Beschreibung der beiden Möglichkeiten, Operationen zu definieren, und der Reihenfolge, in der sie angeordnet werden, finden Sie unter "Standard- und DCL-Operationen."

10.2.1.1 Operationsausdrücke

Ein Operationsausdruck kann auf Variablen der folgenden Tabelle zugreifen, kennzeichnen, welche Komponente ausgewählt war, und den Komponentenstatus zum Zeitpunkt der Operation beschreiben. Die Variablennamen sind reserviert. Ihre Werte sind schreibgeschützt und besitzen nur bei einem Zugriff innerhalb eines Operationsausdrucks eine definierte Bedeutung.

Variablen für Operationsausdrücke

Variable	Beschreibung
`$key`	Das Schlüsselattribut der Komponente, die ausgewählt war. Diese Variable steht bei allen Operationen zur Verfügung.
`$value`	Die Zeichenkette des aktuellen Komponentenwerts (beispielsweise die Zeichenkette eines Eingabefelds oder "1" bzw. "0" eines Schaltfelds). Diese Variable steht bei allen Operationen zur Verfügung. **Hinweis** Handelt es sich bei der Komponente um ein Listenfeld (oder ein Pop-Up-Listenfeld), liefert die Variable `$value nil` zurück, wenn kein Eintrag ausgewählt wurde.
`$data`	Die von der Anwendung verwalteten Daten (sofern vorhanden), die mit **client_data_tile** unmittelbar nach dem Aufruf von **new_dialog** festgelegt wurden. Diese Variable steht Ihnen bei allen Operationen zur Verfügung, `$data` besitzt aber nur definierte Werte, wenn Ihre Anwendung diese mit dem Aufruf (**client_data_tile**) initialisiert hat. Siehe "Anwendungsspezifische Daten."
`$reason`	Der Ursachencode, der anzeigt, welcher Schritt des Anwenders diese Operation ausgelöst hat. Wird bei den Komponenten `edit_box`, `list_box`,

	`image_button` und `slider` verwendet.
	Diese Variable zeigt an, warum die Operation durchgeführt wurde. Ihr Wert wird für jede Operation gesetzt, muß jedoch nur dann berücksichtigt werden, wenn er mit einer der Komponenten `edit_box`, `list_box`, `image_button` oder `slider` verknüpft ist. Einzelheiten finden Sie unter "Rückmeldungsursachen."
`$key`	Das Schlüsselattribut der Komponente, die ausgewählt war.
	Diese Variable steht bei allen Operationen zur Verfügung.

Ist `edit1` ein Textfeld, wird der Operationsausdruck des nachfolgenden Aufrufs von **`action_tile`** ausgewertet, wenn der Anwender das Textfeld verläßt.

```
(action_tile "Bearbeitung1" "(setq ns $value)")
```

Jetzt enthält `$value` die Zeichenkette, die der Anwender eingegeben hat. Der Ausdruck speichert sie in der Variablen `ns`.

Das zweite Beispiel sichert den Namen der ausgewählten Komponente, so daß sich das Programm später darauf beziehen kann.

```
(action_tile "Bearbeitung1" "(setq neukomp $key)")
```

Die Variable `neukomp` wird auf den Schlüsselnamen der ausgewählten Komponente gesetzt: hier auf `"Bearbeitung1"`. Die Variable `$key` ist insbesondere in Funktionen nützlich, die als Operationen für mehrere getrennte Komponenten dienen.

Anmerkung Anmerkung Wenn eine Komponente in mehr als einem Aufruf von **`action_tile`** benannt wird, hat nur der letzte Aufruf (vor **`start_dialog`**) eine Wirkung. (Es ist, als ob einer Variablen mehrere Werte zugeordnet werden sollten.) Die Funktion Programmierbare Dialogfelder (PDB) erlaubt Ihnen nur eine Operation pro Komponente.

10.2.1.2 Rückmeldungsursachen

Die *Rückmeldungsursache*, die in der Variablen `$reason` zurückgegeben wird, gibt den Grund der Operation an. Ihr Wert wird für jede Operation gesetzt, untersucht werden muß dieser Wert jedoch nur dann, wenn er mit einer der Komponenten `edit_box` , `list_box`, `image_button`, oder `slider` verknüpft ist. In der folgenden Tabelle werden die möglichen Werte angegeben. Der Text im Anschluß an die Tabelle beschreibt die Codes 2, 3 und 4 ausführlicher. Code 1 wird vollständig in der Tabelle erläutert.

Codes für Rückmeldungsursachen

Code	Beschreibung
1	Dieser Wert tritt bei den meisten Funktionskomponenten auf. Der Benutzer hat die Komponente ausgewählt (möglichst durch Drücken von RETURN , wenn die Komponente die Vorgabe ist und die Plattform Direktbefehle zuläßt).
2	Eingabefelder: Der Anwender hat das Eingabefeld verlassen, aber noch keine abschließende Auswahl getroffen.
3	Bildlauffelder: Der Anwender hat den Wert des Bildlauffelds geändert, indem er die Bildlaufmarke gezogen hat. Er hat jedoch noch keine abschließende Auswahl getroffen.
4	Listenfelder und Bildschaltflächen: Diese Rückmeldungsursache folgt immer auf Code 1. Damit ist normalerweise die Wiederherstellung der vorherigen Auswahl gemeint. Die vorherige Auswahl sollte *niemals* rückgängig gemacht werden, da dies den Anwender verwirrt.

Code 2 - Eingabefelder

Der Anwender hat das Eingabefeld verlassen, indem er entweder TAB gedrückt oder ein anderes Feld gewählt hat. Er hat aber noch keine abschließende Auswahl getroffen. Ist dies die Ursache für die Rückmeldung eines Eingabefelds, sollte Ihre Anwendung den Wert der zugeordneten Variablen nicht aktualisieren, aber eine Gültigkeitsprüfung für den Wert im Eingabefeld durchführen.

Code 3 - Bildlauffelder

Der Anwender hat den Wert des Bildlauffelds geändert, indem er die Anzeige gezogen oder einen entsprechenden Schritt ausgeführt hat. Er hat jedoch noch keine abschließende Auswahl getroffen. Ist dies die Ursache für die Rückmeldung eines Bildlauffelds, sollte Ihre Anwendung nicht den Wert der zugeordneten Variablen aktualisieren, sondern den Text, der den Status des Bildlauffelds anzeigt. Weitere Informationen finden Sie unter "Bildlauffelder." Codebeispiele finden Sie unter "Bearbeiten von Bildlauffeldern."

Code 4 - Listenfelder

Der Anwender hat auf ein Listenfeld doppelgeklickt. Sie können die Bedeutung eines Doppelklicks in Ihrer Anwendung selbst festlegen. Liegt der Hauptzweck eines Listenfelds darin, einen Listeneintrag auszuwählen, sollte beim Doppelklicken der Eintrag ausgewählt und das Dialogfeld verlassen werden. (In diesem Fall sollte das Attribut is_default der Komponente list_box auf true gesetzt sein.) Falls das Listenfeld nicht die vorrangige Komponente des Dialogfelds ist, sollte ein Doppelklick genauso wie eine Auswahl behandelt werden (Code 1).

Listenfelder, die dem Anwender die Auswahl mehrerer Einträge erlauben (multiple_select = true), können Doppelklicks *nicht* unterstützen.

Code 4 - Bildschaltflächen

Der Anwender hat auf eine Bildschaltfläche doppelgeklickt. Sie können die Bedeutung eines Doppelklicks in Ihrer Anwendung selbst festlegen. In vielen Fällen ist es ausreichend, eine Schaltfläche durch einfaches Klicken anzuwählen. Andererseits kann es besser sein, die Komponente durch einfaches Klicken (oder eine Tastatureingabe) lediglich zu markieren und dann mit RETURN oder einem Doppelklick auszuwählen.

10.2.1.3 Standard- und DCL-Operationen

Die Funktion **`action_tile`** ist nicht die einzige Möglichkeit, eine Operation festzulegen. Die DCL-Beschreibung einer Komponente kann in AutoLISP das Attribut `Operation` beinhalten, und der Aufruf von **`new_dialog`** kann eine Standardoperation für das gesamte Dialogfeld definieren. Eine Komponente kann zu einem bestimmten Zeitpunkt immer nur mit einer Operation verknüpft werden. Wenn die DCL und die Anwendung mehr als eine Operation bestimmen, setzen sich diese nach folgender Rangfolge gegenseitig außer Kraft:

1. Die Standardoperation, die durch den Aufruf von **`new_dialog`** festgelegt worden ist (wird nur verwendet, wenn der Komponente keine Operation explizit zugewiesen worden ist)
2. Die Operation, die durch das Attribut `Operation` in der DCL-Datei festgelegt worden ist
3. Die Operation, die durch den Aufruf von **`action_tile`** zugewiesen worden ist (höchste Prioritätsstufe)

10.2.2 Bearbeiten von Komponenten

Ihr Programm besitzt zum Zeitpunkt der Initialisierung und während der Ausführung der Operation (Rückmeldung) eine gewisse Kontrolle über die Komponenten, die sich im aktuellen Dialogfeld befinden. In diesem Abschnitt werden Funktionen vorgestellt, mit denen diese Kontrolle möglich ist. Außerdem wird gezeigt, wie die Modi und Werte einer Komponente verändert werden können.

10.2.2.1 Initialisieren der Modi und Werte

Die Initialisierung einer Komponente kann folgendes beinhalten:

- Sie erhält anfangs den Tastaturfokus des Dialogfelds.
- Sie wird deaktiviert oder aktiviert.
- Ihre Inhalte werden markiert, falls die Komponente ein Eingabefeld oder ein Bild ist.

Diese Aufgaben werden durch Aufrufe von **`mode_tile`** ausgeführt. Der Wert eines Feld kann mit **`set_tile`** gesetzt werden.

Mit dem folgenden Code können Sie einen Vorgabewert (beispielsweise einen Familiennamen) in einem Eingabefeld anzeigen und den Fokus innerhalb des Dialogfelds zu Anfang auf dieses Feld setzen:

```
(setq name_str "Schwarz")                    Vorgabe
(set_tile "Nachname" name_str)               Feld wird initialisiert
(mode_tile "Nachname" 2)                     2 setzt den Fokus auf die Komponente
```

Mit einem zusätzlichen Aufruf von **mode_tile** können Sie den gesamten Inhalt eines Eingabefelds markieren, so daß der Anwender die Möglichkeit erhält, sofort den Standardinhalt zu überschreiben. Dies sehen Sie im folgenden Beispiel:

```
(mode_tile "Nachname" 3)                     3 wählt den Inhalt des Dialogfelds
```

Anmerkung Auf einigen Plattformen wird ein Eingabefeld beim Erhalt des Fokus automatisch markiert. In diesen Fällen ist der gerade beschriebene Schritt überflüssig, richtet aber auch keinen Schaden an.

10.2.2.2 Ändern der Modi und Werte zum Zeitpunkt der Rückmeldung

Sie können den Wert einer Komponente zum Zeitpunkt der Rückmeldung überprüfen. Wenn die Anwendung den Wert abruft, läßt sich dieser mit einem erneuten Aufruf von **set_tile** ändern. Mit **mode_tile** kann der Status einer Komponente auch während der Rückmeldung geändert werden. In der folgenden Tabelle werden die Werte des Arguments *Modus* für die Operation **mode_tile** dargestellt.

Komponentenmodi bei mode_tile

Wert	Beschreibung
0	Aktiviert die Komponente
1	Deaktiviert die Komponente
2	Übergibt der Komponente den Fokus
3	Wählt den Inhalt eines Eingabefelds aus
4	Schaltet die Bildmarkierung ein oder aus

Wenn Sie eine Komponente, die derzeit den Fokus hat, mit **mode_tile** deaktivieren, müssen Sie **mode_tile** nochmals aufrufen, um den Fokus auf eine andere Komponente zu setzen (in den meisten Fällen auf die Stelle im Dialogfeld, an die der Tabulator als nächstes springen würde). Ansonsten würde der Fokus auf einer deaktivierten Komponente verbleiben, was unlogisch ist und zu Fehlern führen kann.

Ein gutes Beispiel zur Verwendung einer Komponente, die sich "selbst deaktiviert", ist eine Anzahl von "Dialogfeldseiten", die der Anwender mit den Schaltflächen Vorwärts und Rückwärts durchblättern kann. Wenn der Anwender Vorwärts auf der vorletzten Seite wählt, soll die Schaltfläche deaktiviert werden. Das gleiche gilt für das Anwählen der Schaltfläche Rückwärts auf der zweiten Seite. In beiden Fällen muß durch den Code die angewählte Schaltfläche deaktiviert und der Fokus auf ein anderes Feld gesetzt werden.

Angenommen, das Feld `"Gruppe_ein"` sei ein Schalter, der eine Gruppe von Optionen mit dem Namen `"Gruppe"` steuert. Wird der Schalter ausgeschaltet, sind die gruppierten Felder inaktiv und sollten nicht verändert werden. In diesem Fall könnten Sie für den Schalter die folgende Operation definieren. (Beachten Sie den Gebrauch des Steuerzeichens \", das die Verwendung von Anführungszeichen im Argument von **action_tile** ermöglicht.)

```
(action_tile "Gruppe_ein" "(mode_tile \"Gruppe\" (- 1 (atoi $value)))")
```

Die Subtraktion und der Aufruf von **atoi** im Operationsausdruck legen das Argument *mode* der Funktion **mode_tile** fest. Da ein Schalter den Wert 0 hat, wenn er ausgeschaltet ist, und den Wert 1, wenn er eingeschaltet ist, kehrt die Subtraktion seinen Wert um. Damit steuert *mode*, ob die Gruppe aktiviert ist.

Sie können neben dem Wert einer Komponente mit der Funktion **get_attr** noch andere Attribute untersuchen. Sie können beispielsweise die Beschriftung einer Schaltfläche `"drückmich"` ermitteln.

```
(get_attr "drückmich" "beschriftung")
```

Anmerkung Verwenden Sie **get_attr**, um ein Attribut `value` aufzurufen, wird das Attribut `value` verwendet, das in der DCL-Datei gespeichert ist (der Anfangswert der Komponente). Die Funktion **get_tile** bestimmt den aktuellen Laufzeitwert der Komponente. Beide Werte müssen nicht übereinstimmen.

Die Funktion **get_attr** liefert den Wert des ausgewählten Attributs als Zeichenkette zurück.

10.2.2.3 Bearbeiten von gruppierten Auswahlsymbolen

Auswahlsymbole werden immer in Gruppen zusammengefaßt. Jedes Auswahlsymbol besitzt entweder den Wert `"1"` für Ein bzw. `"0"` für Aus. Der Wert einer solchen Gruppe ist das Attribut `key` der derzeit ausgewählten Schaltfläche. Das PDB-Paket (Programmable Dialog Box) verwaltet die Werte der einzelnen Auswahlsymbole und stellt sicher, daß zu einem bestimmten Zeitpunkt nur eine Schaltfläche aktiviert ist. Sie können jedem Auswahlsymbol eine Operation zuweisen. Allerdings ist es einfacher, der ganzen Gruppe eine Operation zuzuweisen und dann den Wert der Gruppe daraufhin zu überprüfen, welches Auswahlsymbol ausgewählt war.

Betrachten Sie das folgende Beispiel: Eine Gruppe von Auswahlsymbolen steuert, welche Ansicht eines dreidimensionalen Objekts nach dem Verlassen des Dialogfelds angezeigt wird. Die Gruppe enthält vier Auswahlsymbole (es könnten aber auch mehr sein).

```
(action_tile "view_sel" "(pick_view $value)")
  .
  .
  .
(defun pick_view (which)
  (cond
    ((= which "vorn") (setq show_which 0))
    ((= which "oben") (setq show_which 1))
    ((= which "links") (setq show_which 2))
    ((= which "rechts") (setq show_which 3))
  )
)
```

In diesen Beispielen ist jedes Auswahlsymbol mit einer einzigen Variablen verbunden, die mehrere Werte annehmen kann. Die Variablen können auch zusätzliche Operationen auslösen, wie zum Beispiel die Deaktivierung der Auswahl in einem Dialogfeld. Bei großen Gruppen können Sie die zugehörigen Werte in einer Tabelle speichern. Wenn Sie eine Tabelle verwenden, sollten Sie diese so strukturieren, daß sie nicht von der Reihenfolge der Schaltflächen innerhalb der Gruppe abhängt. Das PDB-Paket verlangt diese Einschränkung nicht. Die Reihenfolge kann sich mit der DCL-Datei ändern.

10.2.2.4 Bearbeiten von Bildlauffeldern

Bearbeitet Ihre Anwendung Operationen und Rückmeldungen von Bildlauffeldern, sollte sie den Ursachencode überprüfen, den sie mit der Rückmeldung erhält. Dies ist nicht zwingend notwendig, verringert aber den Aufwand. Wie häufig Bildlauffelder Rückmeldungen generieren, hängt von der Plattform ab. Einige Plattformen generieren bei *jeder* Mausbewegung, die das Bildlauffeld erkennt, einen Rückmeldungscode 1.

Die folgende Funktion zeigt das grundlegende Schema für die Behandlung von Bildlauffeldern durch Funktionen. Die Funktion wird von einem Operationsausdruck aufgerufen, der mit dem Bildlauffeld verknüpft ist. Die von der Funktion verwendete Komponente `slider_info` zeigt den aktuellen Wert des Bildlauffelds in dezimaler Form an . Oft ist eine derartige Komponente gleichzeitig ein Eingabefeld, das dem Anwender erlaubt, das Bildlauffeld entweder zu verschieben oder direkt einen Wert einzugeben. Gibt der Anwender einen Wert in `slider_info` ein, sollte die Rückmeldung des Eingabefelds wiederum den Wert des Bildlauffelds aktualisieren.

```
(action_tile
  "meinlauffeld"
  "(slider_action $value $reason)"
)
(action_tile
  "slider_info"
  "(ebox_action $value $reason)"
)
  .
  .
  .
(defun slider_action(val why)
  (if (or (= why 2) (= why 1))
  (set_tile "slider_info" val)  Zwischenergebnis anzeigen
  )
)
```

```
  (defun ebox_action(val why)
    (if (or (= why 2) (= why 1))
    (set_tile "meinlauffeld" val)  Zwischenergebnis anzeigen
    )
)
```

10.2.2.5 Bearbeiten von Eingabefeldern

Operationen und Rückmeldungen, die Eingabefelder behandeln, ähneln denen für Bildlauffelder. Da aber Zeichen in Eingabefeldern bereits sichtbar sind, müssen Zwischenergebnisse nicht gesondert behandelt werden. Eingabefelder geben nur dann einen Rückmeldungscode zurück, wenn der Fokus auf dieses Bild verlorengegangen ist.

Das folgende Codebeispiel überprüft den Wert, zeigt ihn aber nicht erneut an.

```
  (action_tile "eingabefeld" "(edit_action $value $reason)")
    .
    .
    .
  (defun edit_action (val why)
    (if (or (= why 2) (= why 4))
      .                          An dieser Stelle Bereichsprüfung
      .                          des aktuellen Werts durchführen
      .
    )
  )
```

10.2.3 Verschachteln von Dialogfeldern

Sie erzeugen und verwalten verschachtelte Dialogfelder, indem Sie **new_dialog** und **start_dialog** innerhalb eines Operationsausdrucks oder einer Rückmeldungsfunktion aufrufen. So kann beispielsweise durch die Aufnahme des folgenden Ausdrucks in eine Funktion diese das Dialogfeld "Hallo, Leute" anzeigen, wenn der Anwender die Schaltfläche `Schaltfläche_1` auswählt.

```
  (action_tile "Schaltfläche_1" "(c:hallo)")
```

Der Anwender muß das verschachtelte Dialogfeld zuerst verlassen, bevor er wieder im vorigen Dialogfeld arbeiten kann.

Anmerkung AutoCAD erlaubt eine Verschachtelung von bis zu acht Dialogfeldern. Aus Gründen der Übersichtlichkeit sollten allerdings nicht mehr als vier Dialogfelder verschachtelt werden.

10.2.4 Ausblenden von Dialogfeldern

Ein Anwender kann keine interaktive Auswahl treffen, während ein Dialogfeld aktiv ist. Möchten Sie einem Anwender die Auswahl vom Grafikbildschirm aus ermöglichen, müssen Sie das Dialogfeld zuerst *ausblenden* und danach wiederherstellen. Das Ausblenden eines Dialogfelds entspricht dem Beenden mit **done_dialog**, allerdings muß Ihre Rückmeldungsfunktion im Argument *`Status`* von **done_dialog** anzeigen, daß das Dialogfeld lediglich ausgeblendet und nicht beendet oder mit Abbrechen verlassen wurde. Setzen Sie *`Status`* auf einen von der Anwendung definierten Wert. Die Funktion **start_dialog** gibt diesen Wert zurück, wenn das Dialogfeld ausgeblendet wird. Ihr Programm muß dann den von **start_dialog** zurückgelieferten Status untersuchen, um den nächsten Schritt zu bestimmen. Das bedeutet, daß Sie die Rückgabewerte von **start_dialog** abfangen müssen, indem Sie diesen Aufruf in eine Schleife einschließen. Damit können Sie das Dialogfeld nach dem Ausblenden wiederherstellen. Weitere Informationen über vorgabe- und anwendungsdefinierte *`Status`*–Werte finden Sie unter **"done_dialog"** in chapter 13.

Das Beispielprogramm *bmake.lsp* besitzt eine Schaltfläche Select Point, die das Dialogfeld ausblendet, um dem Anwender die Bestimmung eines Punktes auf dem Grafikbildschirm zu ermöglichen. Beim Auswählen dieser Schaltfläche wird das Dialogfeld mit dem speziellen *`Status`* 4 geschlossen.

```
  (action_tile "Ausw_pkt" "(done_dialog 4)")
```

Nach der Rückkehr von **start_dialog** untersucht das Programm den Rückkehrstatus und aktualisiert, falls notwendig, eine Punktvariable.

```
  (setq nächst_Schritt (start_dialog))
  (cond                      ; Nächsten Schritt bestimmen
                             ; Wenn der Basispunkt ausgewählt wurde. . .
    ((= nächst_Schritt 4)
      (setq Ausw_pkt (getpoint "Einfügebasispunkt: "))
      (setq x_pkt (rtos (car Ausw_pkt) 2 4))
```

```
      (setq y_pkt (rtos (cadr Ausw_pkt) 2 4))
      (setq z_pkt (rtos (caddr Ausw_pkt) 2 4))
    )
  )
```

Der Inhalt der Hauptfunktion ist in eine Schleife eingeschlossen, damit die Funktionen **`new_dialog`** und **`start_dialog`** (zusammen mit **`action_tile`**, **`set_tile`**, **`start_list`** und ähnlichen Funktionen) wiederholt aufgerufen werden, bis der Anwender OK oder Abbrechen wählt. Der folgende Beispielcode ist ein Teil der Beispielanwendung *bmake.lsp*.

```
(defun bmake_main ...
                                        ; Dialogfeld laden und globale Initialisierung durchführen
  (while (< 2 nächst_Schritt)           ; Status deutet auf Ausblenden
                                        ; Individuelle Dialogfeldinitialisierung wie Aufrufe von
                                        ; new_dialog, action_tile, set_tile,
                                        ; und start_list
    (setq nächst_Schritt
      (start_dialog)
    )
    (condS
                                        ; Prüfen, ob nächst_Schritt darauf deutet, daß das Feld
                                        ; ausgeblendet ist, und entsprechend handeln
    )
    .
    .
    .
  )
)
```

Ist das Dialogfeld verschachtelt, ist das Vorgehen etwas komplexer, im Prinzip jedoch dasselbe:

```
(defun c:maindlg ( / nächst_Schritt nächst_Schritt1 )
  (setq nächst_Schritt 5)
 (if
    (<
      (setq dcl_id
        (load_dialog "maindlg.dcl")
      )
    0)
    (exit)
  )
  (while (< 1 nächst_Schritt)           ; Innerhalb der Schleife bleiben, bis OK
    (new_dialog "maindlg" dcl_id)       ; oder Abbrechen gedrückt wird
    (action_tile "x" "(subdlg)")        ; Initialisierungen und set_tiles
    (cond
      ((= nächst_Schritt1 3)            ; Bedingung erfüllt, wenn ein Punkt bestimmt werden soll
        (subdlg)                        ; Aufruf SUBDLG-Routine
        (if (/= 3 nächst_Schritt1)      ; Hauptinitialisierung
          (setq nächst_Schritt
            (start_dialog)
          )
        )
      )
      (t (setq nächst_Schritt
        (start_dialog))                 ; Beginn der Ausführung
      )
    )
    (cond
      ((= 2 nächst_Schritt)             ; Punkt bestimmen
        (setq xx
          (getpoint "\nAuswahlpunkt: ")
        )
      )
```

```
      )
    )
  )
  (defun subdlg()
    (new_dialog "subdlg" dcl_id)
    (action_tile "alle_ausb"
      "(done_dialog 3)")            ; Untergeordnetes Dialogfenster ausblenden
    (setq nächst_Schritt1 (start_dialog))
    (if (= 3 nächst_Schritt1)
      (done_dialog 2)               ; Übergeordnetes Dialogfenster ausblenden
    )
  )
```

Warnung Wenn Sie verschachtelte Dialogfelder durch mehrfache Aufrufe von **new_dialog** anzeigen, sollten Sie darauf achten, daß für jeden Aufruf von **new_dialog** auch ein entsprechender Aufruf von **done_dialog** erfolgt (gegebenenfalls durch eine Rückmeldung). Andernfalls kann in Ihrer Anwendung ein Fehler auftreten.

Obwohl die Funktion **term_dialog** alle Dialogfelder sofort verläßt, gibt sie keinen Statuscode zurück. Daher kann Ihre Anwendung mit Hilfe dieser Funktion nicht zwischen einem Ausblenden von Dialogfeldern und einem Abbruch eines Dialogfelds aufgrund einer Fehlerbedingung unterscheiden.

10.2.5 Anfordern eines Paßworts

Im folgenden Beispiel wird mit einem einfachen Dialogfeld vom Benutzer ein Paßwort angefordert. Es verwendet das DCL-Attribut `password_char`, um den vom Benutzer eingegebenen Text zu maskieren.

Die Datei *getpass.dcl* definiert ein Dialogfeld mit dem Namen `passdlg`, das zwei Felder enthält: das Feld `edit_box`, in das der Benutzer das Paßwort eingibt, und das Feld `ok_cancel`.

```
// GETPASS.DCL
//
passdlg : Dialog {
  label = "Paßwortgeschützt";
  : edit_box {
  label   = "Paßwort: ";
    edit_width = 20;
    key = "paßwort";
    password_char = "?"
  }
  ok_cancel;
}
```

Die Datei *getpass.lsp* definiert die Funktion **GETPASS**, die die Datei *getpass.dcl* lädt und das Dialogfeld `passdlg` anzeigt. Bei Eingabe von Text in das Eingabefeld wird dieser vom Zeichen `password_char` maskiert, das von der DCL-Datei definiert wird. Die Funktion, die dem Eingabefeld zugewiesen wird, stellt sicher, daß die vom Benutzer eingegebenen Zeichen der Variablen `pass` zugewiesen werden.

```
;; GETPASS.LSP
;;
(defun GETPASS ( / dcl_id pass )
  (setq dcl_id (load_dialog "getpass.dcl"))
  (if (new_dialog "passdlg" dcl_id)
    (progn
      (action_tile "password" "(setq pass $value)")
      (start_dialog)
      (unload_dialog dcl_id)
    )
    (princ "Fehler: kann GETPASS.DCL nicht laden. ")
  )
  pass
)
```

Die Funktion **GETPASS** gibt die vom Benutzer eingegebene Zeichenfolge wieder.

10.3 Listen- und Pop-Up-Listenfelder

AutoLISP stellt Ihnen die folgenden Funktionen zur Verwendung von Listenfeldern und Pop-Up-Listenfeldern in Dialogfeldern zur Verfügung:

add_list end_list start_list

Sie konfigurieren die in Listen- und Pop-Up-Listenfeldern angezeigten Listen durch eine Folge von Aufrufen dieser drei Funktionen. Nachdem eine Liste erstellt worden ist, können Sie sie überarbeiten. Es gibt drei mögliche Operationen. Jede von ihnen wird im Argument `Operation` der Funktion **`start_list`** bestimmt. (Die Werte des Arguments sind im folgenden in Klammern dargestellt.)

- Erstellen einer neuen Liste (3). Dies ist die Vorgabeoperation.

  Nach dem Aufruf von **`start_list`** können Sie **`add_list`** wiederholt aufrufen. Jeder Aufruf von **`add_list`** fügt einen neuen Eintrag in die Liste ein. Beenden Sie die Bearbeitung der Liste mit **`end_list`**.

- Ändern eines Listeneintrags (1).

  Nach **`start_list`** rufen Sie **`add_list`** *einmal* auf, um denjenigen Eintrag, dessen Index im Aufruf von **`start_list`** angegeben worden ist, zu ändern. (Rufen Sie **`add_list`** mehrmals auf, wird derselbe Eintrag mehrmals geändert.) Beenden Sie die Bearbeitung der Liste mit **`end_list`**. Der erste Eintrag, der in die Liste eingefügt wird, besitzt den Index 0.

- Anhängen eines Listeneintrags (2).

  Rufen Sie nach **`start_list`** **`add_list`** auf, um einen Listeneintrag am Ende der Liste anzuhängen. Bei einem mehrfachen Aufruf von **`add_list`** werden mehrere Einträge an die Liste angehängt. Beenden Sie die Bearbeitung der Liste mit **`end_list`**.

Unabhängig von der auszuführenden Listenoperation müssen die drei Operationen der Reihe nach aufgerufen werden: zuerst **`start_list`**, danach (möglicherweise mehrfach) **`add_list`**, und zum Abschluß **`end_list`**.

Die Funktion **`mapcar`** hilft Ihnen dabei, eine unbearbeitete AutoLISP-Liste in ein anzeigbares Listenfeld zu verwandeln. Im folgenden Beispiel enthält die Liste `Anw_namen` Zeichenfolgen, die Sie in einem Listenfeld `Auswahl` anzeigen wollen. Sie können mit diesem Codefragment die Liste konfigurieren und anzeigen.

```
(start_list "Auswahl")            ;Den Namen des Listenfelds angeben
(mapcar ' add_list Anw_namen)     ;Die AutoLISP-Liste angeben
(end_list)
```

Da 3 (Listenerstellung) die Vorgabe ist, wird in diesem Beispiel für das Argument kein expliziter Wert angegeben.

Der Wert eines Felds `list_box` ist der Index (oder die Indizes) des ausgewählten Eintrags (oder seiner Einträge). Benötigt Ihr Programm den eigentlichen Text, der mit dem Index verbunden ist, muß die Originalliste gespeichert werden. Entsprechend müssen die Änderungen protokolliert werden, wie es in den folgenden Beispielen gezeigt wird.

Das Hinzufügen von Listeneinträgen ähnelt der Erstellung einer neuen Liste. Sie können beispielsweise einer Liste `Anw_namen`, die bereits 12 Einträge besitzt, mit dem folgenden Programmcode eine zweite Liste `neu_Namen` anhängen:

```
(start_list "Auswahl" 2)
(mapcar 'add_list neu_Namen)
(end_list)
```

Da das Anhängen einer Liste nicht die Voreinstellung ist, muß die Listenoperation (Code 2) explizit angegeben werden.

Die Änderung eines einzelnen Listeneintrags erfordert nur einen einzigen Aufruf von **`add_list`**. In diesem Fall wird der Index des zu ändernden Eintrags angegeben:

```
(start_list "Auswahl" 1 5)  ;Den sechsten Listeneintrag ändern
(add_list "ÜBERRASCHUNG!")  ;Achtung! Der erste Eintrag hat den Index 0.
(end_list)
```

Anmerkung Sie können einen Listeneintrag *nicht* hinzufügen oder löschen, ohne die Liste von Grund auf neu zu erstellen.

Da der Wert einer Komponente von `list_box` führende Leerzeichen enthalten kann (besonders, wenn Sie mehrere Einträge abrufen), dürfen Sie einen Wert nicht durch Zeichenkettenvergleich prüfen. Konvertieren Sie den Wert zuerst mit **atoi** in eine Ganzzahl, bevor Sie das Listenfeld bearbeiten. Sie können auch die Funktion **read** verwenden, die ein Argument automatisch in eine Ganzzahl konvertiert. Mit dem folgenden Codefragment können Sie in einer Liste namens `nureins`, die nur eine einfache Auswahl zuläßt, überprüfen, ob der dritte Listeneintrag ausgewählt worden ist. Zuerst muß geprüft werden, ob die Zeichenkette leer ist, da die Funktionen vom Typ **atoi** sowohl für leere Zeichenketten als auch für die Zeichenkette `"0"` 0 zurückliefern.

```
(setq index ( get_tile "nureins"))
(cond
  ((/= index "")
    (= 2 (atoi index))      ; Den dritten Eintrag bearbeiten
    ...
  )
)
```

Anmerkung Werte einer Pop-Up-Liste besitzen niemals führende Leerzeichen. Deshalb müssen Werte einer Pop-Up-Liste auch nicht konvertiert werden. Pop-Up-Listen erlauben zudem keine Mehrfachauswahl.

Unterstützt das Listenfeld eine Mehrfachauswahl, muß Ihr Programm die Werte konvertieren und die einzelnen Werte in der Zeichenkette verarbeiten. Die folgende Definition von **MK_LIST** liefert eine Liste zurück, die nur die Einträge enthält, die der Anwender aus der ursprünglichen Liste `displist` ausgewählt hat. (In diesem Beispiel wird die angezeigte Liste `displist` als globale Variable geführt.) Die Funktion **MK_LIST** soll mit dem aktuellen Wert von `$value` aufgerufen werden.

```
(defun MK_LIST (readlist / count item retlist)
  (setq count 1)
  (while (setq item (read readlist))
    (setq retlist (cons (nth item displist) retlist))
    (while (and (/= " " (substr readlist count 1))
      (/= "" (substr readlist count 1)))
      (setq count (1+ count))
    )
    (setq readlist (substr readlist count))
  )
  (reverse retlist)
)
```

Diese beiden Beispiele sind auch auf Listen anwendbar, die nur eine einfache Auswahl zulassen.

10.4 Bilddateien und Bildschaltflächen

AutoLISP stellt Ihnen die folgenden Funktionen zur Verwendung von Bilddateien und Bildschaltflächen zur Verfügung:

dimx_tile	end_image	slide_image	vector_image
dimy_tile	fill_image	start_image	

Beispiele für die Verwendung dieser Funktionen finden Sie in diesem Abschnitt. Diese Funktionen werden ebenfalls in Kapitel 13 erläutert.

10.4.1 Erstellen von Bildern

Die Aufruffolge zum Erzeugen von Bildern für Bilddateien und Bildschaltflächen ist ähnlich aufgebaut wie bei den Listen. Die Funktion **start_image** startet die Bilderzeugung, **end_image** beendet sie. Die Optionen, die angeben, was gezeichnet werden soll, werden allerdings in getrennten Funktionsaufrufen anstatt in Argumenten bestimmt.

vector_image

Zeichnet im aktuellen Bild einen Vektor (eine einzelne gerade Linie).

fill_image

Zeichnet im aktuellen Bild ein gefülltes Rechteck.

slide_image

Zeichnet im aktuellen Bild ein AutoCAD-Dia.

Vektoren und gefüllte Rechtecke genügen für einfache Bilder, wie beispielsweise Farbflächen (gefüllte Rechtecke), die im AutoCAD-Dialogfeld Farbe wählen verwendet werden, um die Farbauswahl des Anwenders anzuzeigen. Für kompliziertere Bilder sind Dias geeigneter. Allerdings benötigt das Anzeigen von Dias viel Zeit. Deshalb sollten Sie diese einfach gestalten, wenn Sie sie verwenden.

Anmerkung Wenn Sie in Bildkomponenten Dias mit gefüllten Objekten verwenden (beispielsweise große Polylinien, Festkörper und 3D-Flächen), erscheint das Bild als Umriß, außer Sie erstellen die Dias von einem Bild, das mit dem Befehl SHADE erzeugt wurde.

Die Bildzeichenfunktion **`vector_image`** erwartet von Ihnen absolute Koordinaten. Die Funktionen **`fill_image`** und **`slide_image`** erwarten dagegen eine Startkoordinate und relativ dazu Angaben über die Breite und Höhe. Deshalb benötigen Sie die genauen Abmessungen der Bilddatei bzw. der Bildschaltflächen, weil diese Werte normalerweise beim Entwurf eines Dialogfelds zugewiesen werden. Das PDB-Paket stellt Funktionen zur Verfügung, welche die Breite und Höhe einer bestimmten Komponente zurückgeben. AutoLISP besitzt zwei solche Meßfunktionen: **`dimx_tile`** und **`dimy_tile`**. Sie sollten sie aufrufen, bevor Sie mit der Erstellung eines Bilds beginnen. Der Ursprung einer Komponente (0,0) ist immer ihre linke obere Ecke.

Farben können entweder mit der AutoCAD-Farbnummer oder mit einer "logischen" Farbnummer aus der folgenden Tabelle bestimmt werden. (Die Werte und Abkürzungen werden vom ADI (Autodesk Device Interface [ADI®]) definiert).

Symbolische Namen für Farbattribute

Farbnummer	Mnemonischer ADI-Code	Bedeutung
-2	BGLCOLOR	Aktueller Hintergrund des AutoCAD-Grafikbildschirms
-15	DBGLCOLOR	Aktuelle Hintergrundfarbe des Dialogfelds
-16	DFGLCOLOR	Aktuelle Vordergrundfarbe des Dialogfelds (für Text)
-18	LINELCOLOR	Aktuelle Linienfarbe des Dialogfelds

Im folgenden Beispiel ist `"Kur_Farbe"` eine Bilddatei, die vollständig rot gefüllt werden soll.

```
(setq Breite (dimx_tile  "Kur_Farbe")
      Höhe (dimy_tile "Kur_Farbe"))
(start_image "Kur_Farbe")
(fill_image 0 0 Breite Höhe 1)      ; 1 = AutoCAD-Rot
(end_image)
```

Sie können die Funktionen zum Zeichnen eines Bilds beliebig miteinander kombinieren. In diesem Beispiel wird ein Bild gefüllt und anschließend ein vertikaler Streifen darüber gezeichnet.

```
(setq Breite (dimx_tile "Streifen")
        Höhe (dimy_tile "Streifen"))
(start_image "Streifen")
(fill_image 0 0 Breite Höhe 3)  ; 3 = AutoCAD-Grün
(setq x (/ Breite 2))           ; Vektor vertikal zentrieren
(vector_image x 0 x Höhe 4)     ; 4 = AutoCAD-Cyan
(end_image)
```

Die Dias, die Sie mit **`slide_image`** anzeigen, können entweder selbständige Diadateien (*.sld*) oder Teile einer Diabibliotheksdatei (*.slb*) sein. Befindet sich das Dia in einer *.sld*-Datei, so geben Sie seinen Namen ohne die Dateinamenserweiterung *.sld* an (beispielsweise `"Vansicht"`). Befindet sich das Dia in einer Diabibliothek, geben Sie zuerst den Namen der Bibliothek (ohne Erweiterung) an und nachfolgend in Klammern den Namen des Dias (ebenfalls ohne Erweiterung, z. B. `"Allansit(Vansicht)"`). Die Funktion **`slide_image`** sucht das Dia oder die Diabibliotheksdatei, wobei es dem aktuellen AutoCAD-Bibliothekssuchpfad folgt (siehe **`"load_dialog"`** in chapter 13).

Im folgenden Beispiel befindet sich das Dia in einer eigenen Datei *Oansicht.sld*.

```
(setq x (dimx_tile "Ansicht")
```

```
      y (dimy_tile "Ansicht"))
(start_image "Ansicht")
( slide_image 0 0 x y "Oansicht")
(end_image)
```

Vektoren werden in Dias oft weiß gezeichnet (Farbnummer 7), da dies die vorgegebene Hintergrundfarbe eines Bilds ist. Ist eine zum ersten Mal auf dem Bildschirm angezeigte Bildkomponente leer, versuchen Sie, das Attribut `Farbe` der entsprechenden Komponente auf `graphics_background` umzuschalten. (Sie können den Bildhintergrund auch mit dem Aufruf **`slide_image`**, dem ein **`fill_image`** vorangestellt wird, ändern.)

10.4.2 Bearbeiten von Bildschaltflächen

Sie können Bildschaltflächen genau wie eine normale Schaltfläche behandeln. Das bedeutet, Sie können mit Ihr eine einzige Operation auslösen. Sie können das PDB-Paket aber auch zur Definition von Regionen für die Schaltfläche verwenden, so daß die ausgelöste Operation davon abhängt, *welchen Teil* der Bildschaltfläche der Anwender ausgewählt hat. Die Vorgehensweise hierfür ist einfach: die Operation oder die Rückmeldung einer Bildschaltfläche gibt die Position (*X,Y*) zurück, die der Anwender ausgewählt hat. Die Koordinaten liegen in einem bestimmten Bereich der Bildschaltfläche (wie sie von den Meßfunktionen auch zurückgeliefert werden). Ihre Anwendung muß nun der Positionsauswahl eine Bedeutung zuweisen, indem implizit Regionen für das Bild definiert werden. Der Dialog DDVPOINT nutzt diese Eigenschaft gut.

Im folgenden Beispiel besitzt Ihre Bildschaltfläche zwei Farbflächen, die mit **`fill_image`** erstellt wurden. Es soll jeweils nur eine von beiden ausgewählt werden, je nach der Region, die der Anwender wählt. Wenn die Bildschaltfläche horizontal geteilt ist (im oberen Teil dunkel, im unteren Teil hell), genügt es, daß Ihre Operation nur eine Abmessung testet.

```
(action_tile "Bild_fl" "(pick_shade $key $value $y)")
  ...
(defun pick_shade (key val y)
  (setq threshold
    (/ ( dimy_tile  key) 2))  ; Bild ist horizontal geteilt
  (if (> y threshold)         ; Achtung! Ursprung liegt oben links
    (setq result "Hell")
  (setq result "Dunkel") )
)
```

10.5 Anwendungsspezifische Daten

AutoLISP stellt Ihnen die Funktion **`client_data_tile`** zur Behandlung von anwendungsspezifischen Daten zur Verfügung. Die Funktion **`client_data_tile`** weist einer Komponente anwendungsspezifische Daten zu. Zum Zeitpunkt der Rückmeldung sind die Daten als Variable `$data` verfügbar und müssen als Zeichenkette vorliegen. Client-Daten werden in DCL nicht dargestellt, sie sind nur während der Laufzeit Ihrer Anwendung gültig. Die Verwendung von Client-Daten ist mit der Verwendung von anwenderdefinierten Attributen vergleichbar. Der Hauptunterschied liegt darin, daß anwenderdefinierte Attribute schreibgeschützt sind, während Client-Daten während der Laufzeit verändert werden können. Außerdem können Endanwender die anwenderdefinierten Attribute in der entsprechenden DCL-Datei auswerten, Client-Daten sind hingegen für Endanwender nicht sichtbar.

Da Ihr Programm eine Liste verwalten muß, die in einem Listenfeld (oder einem Pop-Up-Listenfeld) angezeigt wird, können diese Informationen mit Client-Daten leicht behandelt werden. Mit den folgenden Änderungen an der Funktionsdefinition von **`MK_LIST`** können Sie die Liste in ein Argument verwandeln. (Die vollständige Funktion finden Sie unter "Listen- und Pop-Up-Listenfelder.")

```
(defun MK_LIST (readlist displist / )
```

Dieser Code macht eine globale Listenvariable überflüssig. Die folgenden Aufrufe im Hauptteil der Dialogfeldsteuerroutine verbinden durch den Aufruf von **`client_data_tile`** eine kurze Liste mit der Komponente und übergeben diese dann mit Hilfe eines Operationsausdrucks an **`MK_LIST`**:

```
(client_data_tile
  "colorsyslist"
  "Rot-Grün-Blau Cyan-Magenta-Gelb Farbton-Sättigung-Wert"
)
(action_tile
  "colorsyslist"
  "(setq usrchoice (mk_list $value $data))"
```

)

10.6 Zusammenfassung der Dialogfeldfunktionen

Dieser Abschnitt gibt einen Überblick und die Zusammenfassung einer üblichen Dialogfeldbearbeitung. Er enthält die Konzepte, die weiter oben in diesem Kapitel erläutert wurden. Dieser Abschnitt beschreibt darüber hinaus eine Beispielanwendung, auf die Sie zurückgreifen können, wenn Sie Ihre eigene Dialogfeldfunktion entwerfen und einsetzen.

10.6.1 Funktionsfolgen

In der folgenden Aufzählung ist die typische Reihenfolge von Funktionsaufrufen dargestellt:

1. Mit dem Aufruf von **load_dialog** laden Sie die DCL-Datei.
2. Mit dem Aufruf von **new_dialog** zeigen Sie ein bestimmtes Dialogfeld an.

   Überprüfen Sie den Wert, der von **new_dialog** zurückgegeben wird. Der Aufruf von **start_dialog** kann zu unvorhersehbaren Ergebnissen führen, wenn der Aufruf von **new_dialog** nicht funktioniert hat.
3. Durch die Konfiguration von Feldwerten, Listen und Bildern initialisieren Sie das Dialogfeld. Beim Konfigurieren von Operationsausdrücken oder Rückmeldungsfunktionen mit **action_tile** verfahren Sie genauso. Zu diesem Zeitpunkt werden normalerweise oft noch andere Funktionen aufgerufen, beispielsweise: **set_tile** und **mode_tile** für allgemeine Komponentenwerte und -zustände, **start_list**, **add_list** und **end_list** für Listenfelder sowie für Bilder die Meßfunktionen mit **start_image**, **vector_image**, **fill_image**, **slide_image** und **end_image**. Außerdem können Sie jetzt **client_data_tile** aufrufen, um anwendungsspezifische Daten mit Ihrem Dialogfeld und dessen Komponenten zu verbinden.
4. Mit dem Aufruf von **start_dialog** schalten Sie die Steuerung des Dialogfelds ein, damit der Anwender Eingaben machen kann.
5. Verarbeiten Sie die Anwendereingaben innerhalb Ihrer Operationen (Rückmeldungen). Sie sollten dies insbesondere dann tun, wenn Sie wahrscheinlich **get_tile**, **get_attr**, **set_tile** und **mode_tile** verwenden werden.
6. Der Anwender drückt die Schaltfläche zum Verlassen des Dialogfelds und löst damit eine Operation aus, die **done_dialog** aufruft. Dies führt bewirkt, daß **start_dialog** einen Wert zurückgibt. Entfernen Sie jetzt mit **unload_dialog** die DCL-Datei aus dem Speicher.

Dieses Schema behandelt zu einem gegebenen Zeitpunkt nur jeweils ein Dialogfeld und eine DCL-Datei. Anwendungen besitzen aber normalerweise viele Dialogfelder. Der einfachste und schnellste Weg, diese Dialogfelder zu behandeln, besteht darin, alle Dialogfelder in einer einzigen DCL-Datei zu speichern. Der Aufruf **load_dialog** lädt in diesem Fall dann alle Dialogfelder zugleich und erlaubt es Ihnen, mit **new_dialog** ein beliebiges Dialogfeld aufzurufen. Wenn nur wenig Speicher verfügbar ist, müssen Sie eventuell jedoch mehrere DCL-Dateien erstellen. Mit **unload_dialog** können Sie dann einen Satz von Dialogfeldern aus dem Speicher entfernen, bevor Sie mit **load_dialog** einen neuen Satz laden.

10.6.2 Beispieldialogfeld zur Blockdefinition

Die Beispielanwendung *bmake.lsp* und die damit verbundene Datei *bmake.dcl* stellen einige hilfreiche Dialogfeldtechniken vor. Diese Dateien befinden sich im Verzeichnis *sample*. Die Anwendung *bmake* stellt im Grunde eine interaktive Schnittstelle zu der AutoLISP-Funktion **entmake** dar. Sie können die Anwendung zur Definition neuer Blöcke oder zur Anzeige der Blocknamen verwenden. Unter anderem werden in *bmake* die folgenden Techniken vorgestellt:

- Das Ausblenden von Dialogfeldern, indem ein spezieller Statuscode für **done_dialog** definiert und an **start_dialog** übergeben wird. Betrachten Sie die Hauptschleife der Funktion **C:BMAKE** (nach den Aufrufen von **load_dialog** und **action_tile**).
- Die Verwendung eines Umschaltfelds, um ein anderes Feld zu aktivieren oder zu deaktivieren. Betrachten Sie die Definition der Funktion **DO_UNNAMED**.
- Der Aufbau einer Liste für ein Listenfeld. Siehe die Funktionen **PAT_MATCH** und **SORT**.
- Das Anzeigen des AutoCAD-Standardhilfe-Dialogfelds. Betrachten Sie die Funktion **DO_HELP**.

Neben dieser Demonstration der Dialogfeldtechniken zeigt *bmake*, wie Programme gut gestaltet werden.

11 AutoLISP-Lernprogramm

Dieses Lernprogramm zeigt Ihnen, wie Sie einen neuen Befehl zu AutoCAD hinzufügen. Es erklärt die Funktionsweise von AutoLISP und zeigt Ihnen, wie Sie die Leistungsfähigkeit des Programms optimal nutzen. Das Lernprogramm verwendet zwar Beispiele aus der Landschaftsarchitektur, aber die vorgestellten Konzepte sind auf alle Anwendungsgebiete übertragbar.

Das Lernprogramm wendet sich an erfahrene AutoCAD-Anwender. Für die Aufgaben benötigen Sie einen Texteditor, der Dateien im ASCII-Format erstellen kann.

11.1 Ziel

Sie entwickeln in dieser Einführung einen neuen Befehl für AutoCAD, der einen Gartenweg zeichnet und diesen mit kreisförmigen Platten füllt. Ihr neuer Befehl soll folgende Eingabeaufforderungen enthalten:

Befehl: **Weg**
Startpunkt des Wegs: *Geben Sie den Startpunkt an.*
Endpunkt des Wegs: *Geben Sie den Endpunkt an.*
Halbe Breite des Wegs: *Geben Sie eine Zahl an.*
Radius der Platten: *Geben Sie eine Zahl an.*
Abstand zwischen den Platten: *Geben Sie eine Zahl an.*

Mit der Eingabe des *Start-* und des *Endpunkts* legen Sie die Mittellinie eines Wegs fest. Danach geben Sie die halbe Breite des Wegs und den Radius der kreisförmigen Platten an. Zuletzt bestimmen Sie noch den Abstand zwischen den Platten. Sie geben anstatt der vollen Breite nur die halbe Breite des Wegs ein, da diese leichter dargestellt werden kann.

11.2 Erste Schritte

Sie entwickeln diese Anwendung "von innen nach außen" (oder "von unten nach oben") und werden dabei häufig Winkel verwenden. AutoLISP definiert Winkel in Bogenmaß. Mit dem Bogenmaß werden Winkel von 0 bis 2 * p (pi) gemessen. Da aber die meisten Anwender sich Winkel in Grad vorstellen, werden Sie eine Funktion definieren, die Grad in Bogenmaß konvertiert. Erzeugen Sie mit Hilfe Ihres Texteditors die Datei *gw.lsp*. Geben Sie folgendes Programm ein:

```
;  Winkel von Grad in Bogenmaß konvertiert

(defun wib (w)
  (* pi (/ w 180.0))
)
```

Führen Sie sich nun vor Augen, was dieses Programm bewirkt. Sie definieren eine Funktion mit Hilfe der Funktion **`defun`**. Die Funktion heißt **`wib`** (Abkürzung für Winkel in Bogenmaß). Es ist ein Argument erforderlich, `w`, der Winkel in Grad. Das Ergebnis ist der folgende Ausdruck:

```
p * (w / 180.0)
```

Die in LISP-Schreibweise ausgedrückte Funktion liest sich als *das Produkt von PI multipliziert mit dem Quotienten aus w geteilt durch 180.0*. p ist von AutoLISP mit dem Wert 3.14159.... vordefiniert. Bei der Zeile, die mit einem Semikolon beginnt, handelt es sich um einen *Kommentar*; AutoLISP ignoriert den gesamten Text einer Zeile nach einem Semikolon.

Speichern Sie die Datei auf einem Laufwerk, starten Sie AutoCAD und öffnen Sie eine neue Zeichnung. (Der Name spielt keine Rolle, da Sie die Zeichnung nicht speichern müssen.) Laden Sie die Funktion, indem Sie an der Eingabeaufforderung folgendes eingeben:

Befehl: **(load "gw")**
WIB

AutoLISP lädt die Funktion und wiederholt ihren Namen WIB (wenn sich *gw.lsp* im AutoCAD-Suchpfad befindet). Wenn Sie im folgenden Starten Sie AutoCad, und laden Sie das Programm lesen, verfahren Sie wie soeben beschrieben.

Testen Sie nun die Funktion, indem Sie sie mit unterschiedlichen Werten ausführen. Gemäß der Definition des Bogenmaßes sollten 0 Grad dem Bogenmaß 0 entsprechen. Geben Sie also folgendes ein:

Befehl: **(wib 0)**

Wenn Sie eine Zeile eingeben, die mit einer linken Klammer beginnt, übergibt AutoCAD den Ausdruck zur Auswertung an AutoLISP. In diesem Fall werten Sie die soeben definierte Funktion **wib** aus und übergeben ihr das Argument 0. Nach der Auswertung der Funktion zeigt AutoCAD das Ergebnis an, so daß die Eingabe folgende Ausgabe erzeugen sollte:

0.0

Versuchen Sie es nun mit 180 Grad. Geben Sie folgendes ein:

Befehl: **(wib 180)**

Sie erhalten das Ergebnis:

3.14159

180 Grad entspricht also p in Bogenmaß. Dies entspricht genau der vorgesehenen Arbeitsweise der Funktion.

Verlassen Sie nun AutoCAD und rufen Sie Ihren Texteditor wieder auf.

11.3 Auswerten von Eingaben

Der Befehl "Gartenweg" verlangt vom Anwender die Eingabe, an welchen Stellen der Weg gezeichnet werden soll, wie breit er sein soll, wie groß die Platten sein sollen und mit welchem Abstand sie gezeichnet werden sollen. Sie werden nun eine Funktion definieren, die den Anwender zur Eingabe dieser Werte auffordert und daraus verschiedene Werte berechnet, die im Verlauf der Befehlsausführung benötigt werden.

Fügen Sie mit Ihrem Texteditor die folgenden Zeilen in die Datei *gw.lsp* ein.

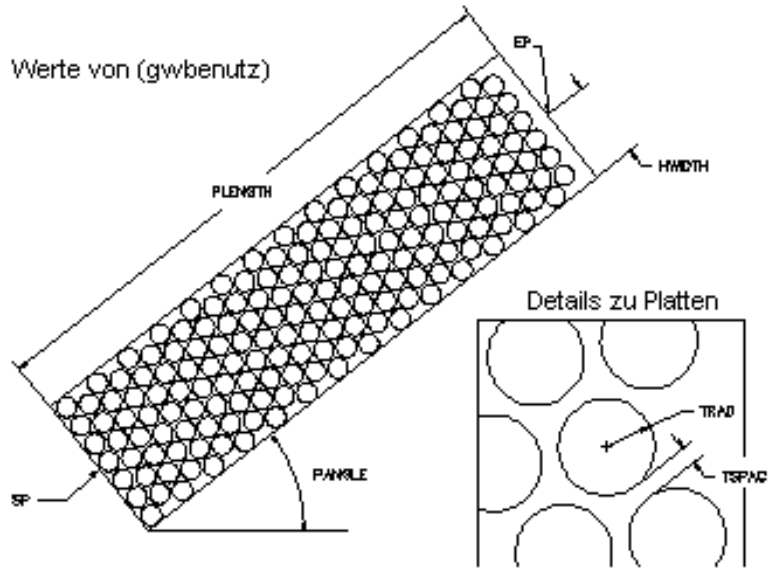
```
; Sammelt Informationen für Gartenweg

(defun gwbenutz ()
  (setq sp (getpoint "\nStartpunkt des Wegs: "))
  (setq ep (getpoint "\nEndpunkt des Wegs: "))
  (setq hbreite (getdist "\nHalbe Breite des Wegs: " sp))
  (setq prad (getdist "\nRadius der Platten: " sp))
  (setq pabst (getdist "\nAbstand zwischen den Platten: " sp))

  (setq pangle (angle sp ep))
  (setq plength (distance sp ep))
  (setq breite (* 2 hbreite))
  (setq angp90 (+ pangle (wib 90))) ; Winkel des Wegs + 90 deg
  (setq angm90 (- pangle (wib 90))) ; Winkel des Wegs - 90 deg
)
```

Es ist nicht erforderlich, die Ausdrücke einzurücken, aus denen Ihre Funktion besteht. Allerdings tragen Einzüge und Zeilenumbrüche zur übersichtlichen Strukturierung von Programmen bei. Eine klare Anordnung der Klammern um größere Ausdrücke verringert die Gefahr, Klammern zu vergessen oder falsch zuzuordnen. Sie sollten anstelle von Tabulatoren Leerzeichen zum Einrücken von Zeilen verwenden. So bleiben die Einrückungen gleich, wenn Sie die Datei in einem Editor oder einem Textverarbeitungsprogramm öffnen.

Hier wurde eine Funktion **gwbenutz** definiert. Sie nimmt keine Argumente entgegen und fordert den Anwender zur Eingabe der gewünschten Parameter auf. Die Funktion **setq** setzt eine AutoLISP-Variable auf einen bestimmten Wert. Der erste Aufruf von **setq** setzt die Variable `sp` (Startpunkt) auf das Ergebnis der Funktion **getpoint**, die den Anwender zur Angabe eines Punkts auffordert. Eine Zeichenkette definiert die Eingabeaufforderung, die AutoCAD beim Einlesen des Punktes verwendet. Mit der Funktion **getdist** ermitteln Sie die halbe Breite des Wegs, den Plattenradius und den Abstand zwischen den Platten. Durch das zweite Argument der Funktion **getdist**, `sp`, wird der *Basispunkt* für den Abstand festgelegt. Dadurch wird der Abstand bei Angabe eines Punkts in AutoCAD relativ zum Startpunkt des Wegs definiert, und der Punkt wird mit einer Gummibandlinie verknüpft.

Die Funktion berechnet verschiedene häufig verwendete Variablen sofort nach Erhalt der Anwendereingaben. Die Variable `pangle` wird auf den Winkel zwischen Anfangs- und Endpunkt des Wegs eingestellt. Die Funktion **angle** gibt diesen Winkel bei Angabe zweier Punkte zurück. Die Variable `plength` wird auf die Länge des Wegs eingestellt. Die Funktion **distance** berechnet den Abstand zweier Punkte. Da nur die halbe Breite des Wegs angegeben wurde, wird die Breite als das Zweifache dieses Werts berechnet. Zuletzt berechnen und speichern Sie in den Variablen `angp90` und `angm90` die Winkelausrichtung plus bzw. minus 90 Grad. (Da Winkel in AutoLISP in Bogenmaß angegeben werden, verwenden Sie vor der Berechnung dieser Werte die Funktion **wib**, um Grad in Bogenmaß umzuwandeln.)

Die folgende Abbildung zeigt Ihnen, wie die Variablen der Funktion **gwbenutz** die Abmessungen des Wegs festlegen.



Speichern Sie das aktualisierte Programm. Starten Sie AutoCAD, und laden Sie die Funktion. Testen Sie nun die Eingabefunktion, um sicherzustellen, daß sie korrekt arbeitet. Aktivieren Sie die Funktion, indem Sie folgendes eingeben:

Befehl: **(gwbenutz)**

Geben Sie bei den Eingabeaufforderungen folgendes ein:

Startpunkt des Wegs: **2,2**
Endpunkt des Weg: **9,8**
Halbe Breite des Wegs: **2**
Radius der Platten: **.2**
Abstand zwischen den Platten: **.1**

Die Funktion **gwbenutz** verwendet Ihre Eingaben zur Berechnung der zusätzlich benötigten Variablen und zeigt anschließend das Ergebnis der letzten Berechnung an (in diesem Fall -0.86217, der Wert der Funktion `angm90` in Bogenmaß). Sie können sich alle Variablen, die von der Funktion **gwbenutz** gesetzt werden, zurückgeben lassen, indem Sie bei der Eingabe dem Variablennamen ein Ausrufezeichen (**!**) voranstellen. Dies veranlaßt AutoCAD, nach der Auswertung einer Variablen das Ergebnis zu drucken. Bei Eingabe der folgenden Befehle sollten Sie die in Klammern angegebenen Ergebnisse erhalten.

Befehl: **!sp**
(2.0 2.0 0.0)
Befehl: **!ep**
(9.0 8.0 0.0)
Befehl: **!hbreite**
2.0
Befehl: **!breite**
4.0
Befehl: **!prad**
0.2
Befehl: **!pabst**
0.1
Befehl: **!pangle**
0.708626
Befehl: **!plength**
9.21954
Befehl: **!angp90**

2.27942

Befehl: **!angm90**

-0.86217

Die Variablen `sp` und `ep` werden als 3D-Punkte (*X*, *Y* und *Z*) zurückgegeben; ignorieren Sie bei dieser Übung die Komponente *Z*.

`pangle`, `angp90` und `angm90` werden ebenfalls in Bogenmaß angegeben. Nach der Überprüfung der Werte verlassen Sie AutoCAD und kehren zur Bearbeitung der Datei *gw.lsp* in den Texteditor zurück.

11.4 Zeichnen des Umrisses

Nachdem der Anwender die nötigen Angaben zur Positionsbestimmung des Wegs gemacht hat, können Sie den Umriß zeichnen. Fügen Sie zur Datei *gw.lsp* die folgenden Zeilen hinzu.

```
; Zeichnet Umriß des Wegs
  (defun zeichum ()
    (command "plinie"
      (setq p (polar sp angm90 hbreite))
      (setq p (polar p pangle plength))
      (setq p (polar p angp90 breite))
      (polar p (+ pangle (wib 180)) plength)
      "schließen"
    )
  )
```

Durch diese Erweiterung wird eine Funktion mit der Bezeichnung **`zeichum`** definiert. Diese Funktion verwendet die mit der Funktion **`gwbenutz`** ermittelten Werte für den Startpunkt, den Winkel und die Länge des Wegs und zeichnet den Umriß als Polylinie. Die Funktion **`zeichum`** verwendet die Funktion **`command`**, um Befehle und Daten an AutoCAD weiterzugeben. Die Funktion **`command`** nimmt eine beliebige Anzahl von Argumenten entgegen und übergibt sie alle an AutoCAD.

In diesem Fall wird der AutoCAD-Befehl PLINIE aufgerufen, und es werden die vier Eckpunkte des Wegs angegeben. Die Funktion positioniert die vier Ecken des Wegs mit der Funktion **`polar`** und speichert diese anschließend in der temporären Variablen `p`. Die Funktion **`polar`** erwartet als erstes Argument einen Punkt, als zweites einen Winkel und als drittes Argument einen Abstand und gibt einen Punkt zurück, der im angegebenen Winkel und Abstand vom eingegebenen Punkt liegt. In diesem Beispiel werden die vier Punkte berechnet, die den Weg ausgehend vom Anfangspunkt geometrisch umschließen. Dies bewirkt, daß PLINIE die vierte Kante des Wegumrisses zeichnet und zur Eingabeaufforderung zurückkehrt. Die Befehlsausführung wird mit der Übergabe der Zeichenkette "Schließen" an den Befehl PLINIE beendet.

Um die Funktion zu testen, speichern Sie die aktualisierte Datei *gw.lsp*, starten Sie AutoCAD mit einer neuen Zeichnung, und laden Sie die AutoLISP-Datei wie zuvor beschrieben. Aktivieren Sie die Anwendereingabefunktion:

Befehl: **(gwbenutz)**

Geben Sie wie im vorherigen Schritt die benötigten Werte ein. Testen Sie nun die neue Funktion **`zeichum`**, indem Sie sie aufrufen.

Befehl: **(zeichum)**

Die Funktion übergibt an AutoCAD die Befehle zum Zeichnen des Wegumrisses. Die Begrenzungslinie wird auf dem Bildschirm angezeigt. Nach dem Testen der Funktion verlassen Sie AutoCAD.

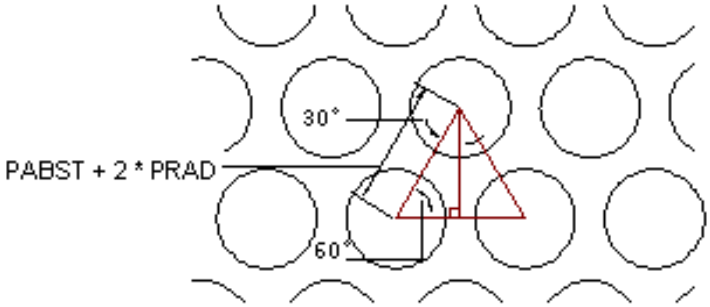
11.5 Zeichnen der Platten

Nachdem Sie nun eine Anwendereingabefunktion und eine Funktion zum Zeichnen des Umrisses eines Wegs definiert und getestet haben, können Sie dazu übergehen, den Weg mit kreisförmigen Platten zu füllen. Dies erfordert geometrisches Verständnis. Wechseln Sie in Ihren Texteditor, und fügen Sie folgenden Code ein:

```
; Zeichnet eine Reihe von Platten mit einem bestimmten Abstand
; entlang des Wegs und versetzt diese Reihe nach Möglichkeit
 (defun zreihe (pd versatz)
  (setq pfirst (polar sp pangle pd))
  (setq pcplat (polar pfirst angp90 versatz))
  (setq plplat pcplat)
```

```
    (while (<(distance pfirst p1plat) (- hbreite prad))
      (command "kreis" p1plat prad)
      (setq p1plat
        (polar p1plat angp90 (+ pabst prad prad)))
    )
    (setq p1plat (polar pcplat angm90 (+ pabst prad prad)))
    (while (<(distance pfirst p1plat) (- hbreite prad))
      (command "kreis" p1plat prad)
      (setq p1plat
        (polar p1plat angm90 (+ pabst prad prad)))
    )
  )

  ; Zeichnet Reihen von Platten
  (defun zeichplat ()
    (setq pdist (+ prad pabst))
    (setq off 0.0)
    (while (<= pdist (- plength prad))
      (zreihe pdist off)
      (setq pdist
        (+ pdist (* (+ pabst prad prad) (sin (wib 60)))))
      (if (= off 0.0)
        (setq off (* (+ pabst prad prad) (cos (wib 60))))
        (setq off 0.0)
      )
    )
  )
```

Um zu verstehen, wie diese Funktionen arbeiten, betrachten Sie die folgende Abbildung. Die Funktion **zreihe** zeichnet eine Reihe von Platten mit einem bestimmten Abstand entlang der Linie, die im ersten Argument bezeichnet worden ist, dabei wird die Reihe um einen im zweiten Argument bezeichneten Abstand lotrecht zur Linie versetzt. Durch diesen Versatz wird eine größere Fläche bedeckt, und gleichzeitig erhält die Pflasterung ein ansprechendes Muster.



Die Funktion **zreihe** verwendet zur Positionsbestimmung der ersten Platte in der Reihe die Funktion **polar**, um dann die im ersten Argument angegebene Bewegung längs des Wegs zu erreichen. Danach dient **polar** dazu, den Versatz lotrecht zum Weg als Abstand festzulegen. Die Funktion **zreihe** zeichnet mit Hilfe der Funktion **while** Kreise, bis der Wegrand erreicht ist. Die Funktion **setq** am Ende der Schleife **while** bewirkt einen Sprung auf die Position der nächsten Platte, indem ein Abstand von zwei Plattenradien und der Fugenbreite eingefügt wird. Eine zweite Schleife **while** zeichnet anschließend die Platten der Reihe in die andere Richtung, bis der andere Wegrand erreicht wird.

Die Funktion **zeichplat** ruft die Funktion **zreihe** wiederholt auf, um alle Plattenreihen zu zeichnen. Ihre Schleife **while** geht den Weg ab und ruft für jede Reihe **zreihe** auf. Platten in benachbarten Reihen bilden gleichseitige Dreiecke (siehe Abbildung). Die Seiten dieser Dreiecke ergeben sich aus dem zweifachen Plattenradius plus dem Plattenabstand. Deshalb ist der Abstand zwischen Reihen entlang des Wegs gleich dem Sinus von 60 Grad multipliziert mit der Dreiecksseitenlänge. Der Abstand ungerader Reihen ist der Cosinus von 60 Grad multipliziert mit dieser Länge.

Die Funktion **if** wird in **zeichplat** verwendet, um jede zweite Reihe zu versetzen. Die Funktion **if** prüft das erste Argument und führt das zweite Argument aus, falls die Bedingung erfüllt ist, ansonsten wird das dritte Argument ausgeführt. In diesem Fall wird, falls `OFF` gleich 0 ist, OFF auf den Abstand zwischen den Plattenmittelpunkten multipliziert mit dem Cosinus von 60 Grad gesetzt (wie gerade erklärt). Wenn `OFF` ungleich 0 ist, wird OFF auf 0 gesetzt. Dadurch kann der Versatz der Reihen beliebig geändert werden.

Bevor Sie das Programm testen, speichern Sie die Datei. Starten Sie AutoCAD, und laden Sie das Programm. Geben Sie dann folgendes ein:

Command: **(gwbenutz)**

Übergeben Sie die Informationen zum Weg wie vorher. Geben Sie dann folgendes ein:

Command: **(zeichum)**

Die Umrißlinie wird wie soeben beschrieben gezeichnet. Geben Sie anschließend folgenden Befehl ein:

Command: **(zeichplat)**

Nun sollten alle Platten innerhalb der Begrenzung gezeichnet werden.

11.6 Erweitern von AutoCAD um einen Befehl

Zum Abschluß können Sie nun die erstellten Funktionen in einen AutoCAD-Befehl einbinden. Wenn Sie in AutoLISP eine Funktion mit dem Namen ***C:XXX*** definieren, wird die Funktion durch Eingabe von ***XXX*** aufgerufen (wenn ***XXX*** kein AutoCAD-Befehl ist). Um die Implementierung des Befehls WEG abzuschließen, definieren Sie eine Funktion mit der Bezeichnung **C:WEG**, mit der Sie nach dem Laden von *gw.lsp* mit dem Befehl WEG zeichnen können.

Bei der Ausführung des Befehls WEG werden die einzelnen an AutoCAD übergebenen Befehle im Befehlszeilenbereich protokolliert und die ausgewählten Punkte werden auf dem Bildschirm durch kleine Kreuze (Markierungspunkte) gekennzeichnet. Nachdem Sie die Fehlersuche in einer Befehlsfunktion abgeschlossen haben, können Sie diese Ausgabe ausschalten. Dann erscheint der mit AutoLISP implementierte Befehl genau wie ein gewöhnlicher AutoCAD-Befehl.

Fügen Sie folgende Zeilen mit Ihrem Texteditor in die Datei *gw.lsp* ein. Starten Sie AutoCAD und laden Sie das Programm.

```
; Führt Befehl aus und ruft dazu folgende Funktion auf
(defun C:WEG ()
  (gwbenutz)
  (setq sblip (getvar "blipmode"))
  (setq scmde (getvar "cmdecho"))
  (setvar "blipmode" 0)
  (setvar "cmdecho" 0)
  (zeichum)
  (zeichplat)
  (setvar "blipmode" sblip)
  (setvar "cmdecho" scmde)
  (princ)
)
```

Durch Hinzufügen der Funktion **C:WEG** fügen Sie den Befehl WEG zu AutoCAD hinzu. Mit Hilfe der Funktion **getvar** erhalten Sie die aktuellen Werte der Systemvariablen BLIPMODE und CMDECHO. Mit set**q** speichern Sie diese Werte in den Variablen `sblip` und `scmde` gespeichert. Mit der Funktion **setvar** setzen Sie die beiden AutoCAD-Variablen auf 0. Damit deaktivieren Sie Markierungspunkte und das Protokollieren von Befehlen. Nachdem Sie die Benutzereingaben mit Hilfe von **gwbenutz** erhalten haben, setzen Sie diese Variablen auf 0. Sie können die Markierungspunkte eingeschaltet lassen, um die Benutzereingabe zu bestätigen.

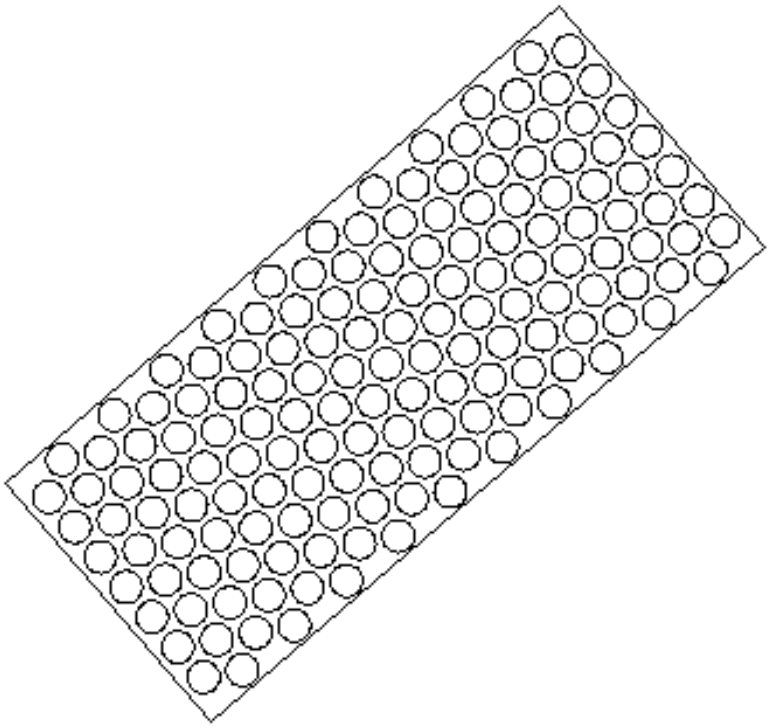
Mit der Funktion **setvar** können Sie nach dem Zeichnen des Wegs beide Variablen auf ihre ursprünglichen Werte zurücksetzen.

Die Funktion **C:WEG** wird ohne weitere Ausgabe beendet, wenn als letzter Befehl die Funktion **princ** aufgerufen wird. Normalerweise geben AutoLISP-Funktionen immer den Wert des letzten Funktionsaufrufs zurück. In diesem Beispiel würde ohne **princ** 1 oder 0 zurückgegeben werden.

Speichern Sie die Datei, starten Sie AutoCAD und geben Sie den Befehl WEG ein. Testen Sie den Befehl durch folgende Eingabe:

Command: **weg**
Startpunkt des Wegs: **2,2**
Endpunkt des Wegs: **9,8**
Halbe Breite des Wegs: **2**
Radius der Platten: **.2**
Abstand zwischen den Platten: **.1**

Dieses Beispiel sollte einen Gartenweg wie in der folgenden Abbildung zeichnen.



Sie können mit dem Befehl WEG experimentieren, dabei können Eingaben sowohl mit dem Zeigegerät als auch über die Tastatur erfolgen.

11.7 Hinzufügen einer Schnittstelle für ein Dialogfeld

Mit Hilfe der Dialogsteuersprache DCL (Dialog Control Language) können Sie benutzerdefinierte Dialogfelder in Ihre AutoLISP-Programme aufnehmen.

Der Befehl WEG nimmt Eingaben bis jetzt nur in der Befehlszeile entgegen. Sie können leicht eine Schnittstelle für ein Dialogfeld zu diesem Befehl hinzufügen, indem Sie die Dialogfeldfunktionen verwenden und damit eine *.dcl* Datei erstellen, die die DCL-Beschreibung des Dialogfelds enthält. DCL wird in Teil III dieses Handbuchs beschrieben, und die Dialogfeldfunktionen von AutoLISP werden in Kapitel 10, "Arbeiten mit Dialogfeldern". erläutert.

Dialogfelder sind insbesondere dann nützlich, wenn der Anwender die Auswahl unter mehreren Möglichkeiten haben soll. Da dieses Programm aber zum jetzigen Zeitpunkt nur sehr wenige Optionen bietet, sollten Sie nun einige hinzufügen.

Mit einer neuen Option, die sie zur Funktion **C:WEG** hinzufügen werden, können Sie die Form der Platten für den Weg festlegen. Außerdem wird eine Funktion zur Fehlerbehebung integriert.

Kopieren Sie zuerst die fertige Version von *gw.lsp* in eine andere Datei, *ddgw.lsp*. (Die meisten mit AutoCAD gelieferten AutoLISP-Programme für Dialogfeldschnittstellen werden nach dem Format *ddxxx.lsp* benannt). Außerdem werden Sie eine neue ASCII-Datei *ddgw.dcl* erstellen, die die DCL-Beschreibung des Dialogfelds enthält.

11.7.1 Die DCL-Datei - *ddgw.dcl*

Das Dialogfeld, das Sie erzeugen, soll zwei runde Optionsfelder für die Form der Platten enthalten (wird ein Feld gewählt, wird das andere deaktiviert): einen Kreis oder ein Polygon. Es soll auch drei Eingabefelder zur Erfassung numerischer Werte enthalten. Radius der Platten, Abstand zwischen den Platten und Anzahl der Seiten (nur verfügbar, wenn das runde Optionsfeld Polygon gewählt wird).

Folgende Schritte sind notwendig, um der Funktion **C:WEG** ein einfaches Dialogfeld hinzuzufügen. Fügen Sie folgenden Code in die Datei *ddgw.dcl* ein.

```
/*  ddgw.dcl - DCL-Datei für DDGW.lsp */
gw_box1 : dialog {
  label = "Eigenschaften der Gartenwegplatten";
  : boxed_radio_row {        // definiert den Bereich für die runden
Optionsfelder
    label = "Plattenform";
    : radio_button {         // definiert das runde Optionsfeld Polygon
```

```
      label = "&Polygon";
      key = "gw_poly";
    }
    : radio_button {         // definiert das runde Optionsfeld Kreis
      label = "&Kreis";
      key = "gw_kreis";
      value = "1";
    }
  }
  : edit_box {      // definiert das Eingabefeld Radius der Platten
    label = "&Radius der Platte";
    key = "gw_prad";
    edit_width = 6;
  }
  : edit_box {      // definiert das Eingabefeld Abstand zwischen den Platten
    label = "&Abstand zwischen den Platten";
    key = "gw_abst";
    edit_width = 6;
  }
  : edit_box {      // definiert das Eingabefeld Anzahl der Seiten
    label = "&Anzahl der Seiten";
    key = "gw_seite";
    edit_width = 4;
  }
  : row {          // definiert die Schaltflächenzeile OK/Abbrechen
    : spacer { breite = 1; }
    : button {    // definiert die Schaltfläche OK
      label = "OK";
      key = "accept";
      breite = 8;
      fixed_width = true;
    }
    : button {    // definiert die Schaltfläche Abbrechen
      label = "cancel";
      is_cancel = true;
      key = "Abbrechen";
      breite = 8;
      ffixed_width = true;
    }
    : spacer { breite = 1;}
  }
}
```

11.7.2 Dialogfeldfunktionen in AutoLISP - *ddgw.lsp*

Nachdem Sie nun die DCL-Datei erzeugt haben, können Sie die benötigten Zeilen in der AutoLISP-Datei *ddgw.lsp* bearbeiten, die Sie aus der Datei *gw.lsp* erstellt haben.

Die Funktionsnamen kennzeichnen die Schritte, welche die einzelnen Funktionen ausführen. Die Funktion **`load_dialog`** lädt ein Dialogfeld. Die Funktion **`set_tile`** setzt den Anfangswert eines bestimmten Felds auf einen Zeichenkettenwert. (Jede Schaltfläche, jedes Eingabefeld usw. wird Feld genannt.) **`action_tile`** bestimmt, welcher Schritt bei der Aktivierung eines Felds stattfindet.

Die neuen Codezeilen werden mit dem Kommentar **`;<-NEU,`** gefolgt von der Codezeile gekennzeichnet. Einige der bereits vorhandenen Zeilen müssen außerdem entfernt oder als Kommentar gekennzeichnet werden. Diese Zeilen werden mit zwei Semikola am Anfang der Zeile markiert (`;;`), am Zeilenende erscheint der Hinweis **`;<-ENTFERNEN`**. Alle neuen und modifizierten Zeilen sind **fett** dargestellt.

```
;;;  DDGW.lsp - der gute alte Gartenweg mit dem neuen Dreh.
; Winkel von Grad in Bogenmaß konvertiert
(defun wib (w)
 (* pi (/ w 180.0))
)
; Sammelt Informationen für Gartenweg
(defun gwbenutz ()
```

```
  (setq sp (getpoint "\nStartpunkt des Wegs: "))
  (setq ep (getpoint "\nEndpunkt des Wegs: "))
  (setq hbreite (getdist "\nHalbe Breite des Wegs: " sp))
;;(setq prad (getdist "\nRadius der Platten: " sp))        ;<-ENTFERNEN
;;(setq pabst (getdist "\nAbstand zwischen den Platten: " sp))        ;<-
ENTFERNEN
(setq pangle (angle sp ep))
  (setq plength (distance sp ep))
  (setq breite (* 2 hbreite))
  (setq angp90 (+ pangle (wib 90)))
  (setq angm90 (- pangle (wib 90))) ;
)
; Zeichnet Umriß des Wegs
(defun zeichum ()
  (command "plinie"
          (setq p (polar sp angm90 hbreite))
          (setq p (polar p pangle plength))
          (setq p (polar p angp90 breite))
          (polar p (+ pangle (wib 180)) plength)
          "schließen"
  )
)
```

Fügen Sie nach der Funktion **zeichum** folgende Codezeilen hinzu.

```
;  Ruft ein Dialogfeld zur Festlegung der Platteneigenschaft auf
(defun gw_dialog ()
  (setq pform "Kreis"
     prad 0.5
     pabst 0.1
     pseiten 8 )
  (setq dcl_id  (load_dialog "ddgw.dcl"))
  (if (not (new_dialog "gw_box1" dcl_id))(exit))
  (set_tile "gw_prad" "0.5")
  (set_tile "gw_abst" "0.1")
  (mode_tile "gw_seite" 1)
  (set_tile "gw_seite" "8")
  (action_tile "gw_kreis"
   "(setq pform \"Kreis\")(mode_tile \"gw_seite\" 1)")
  (action_tile "gw_poly"
   "(setq pform \"Polygon\")(mode_tile \"gw_seite\" 0)")
  (action_tile "cancel" "(done_dialog)(setq gwerr \"\")(exit)")
  (action_tile "accept"
    (strcat
     "(progn (setq prad (atof  (get_tile \"gw_prad\")))"
     "(setq pabst (atof (get_tile \"gw_abst\")))"
     "(setq pseiten (atoi (get_tile \"gw_seite\")))"
     " (done_dialog))"
    )
  )
  (start_dialog)
  (unload_dialog dcl_id)
  (if (= pform "Kreis")
    (defun gw_platte () (command "kreis" p1plat  prad))
    (defun gw_platte () (command "polygon" pseiten  p1plat "" prad))
  )
)
; Definiert Steuerprogramm zur Fehlerkorrektur
(defun gw_err (msg)
  (setq *error* altfehe)
  (if (not gwerr)
    (princ (strcat "\nGartenwegfehler: " msg))
    (princ gwerr)
```

```
  )
  (if sblip (setvar "blipmode" sblip))
  (if scmde (setvar "cmdecho" scmde))
  (princ)
)
```

Der folgende Code sollte bereits in ihrer Datei *ddgw.lsp* enthalten sein. Überarbeiten Sie ihn wie folgt:

```
; Zeichnet eine Reihe von Platten mit einem bestimmten Abstand
; entlang des Wegs und versetzt diese Reihe nach Möglichkeit
(defun zreihe (pd versatz)
  (setvar "snapang" pangle)                                ;<-NEU
  (setq pfirst (polar sp pangle pd))
  (setq pcplat (polar pfirst angp90 versatz))
  (setq p1plat pcplat)
  (while (<(distance pfirst p1plat) (- hbreite prad))
    (gw_platte)                                            ;<-NEU
;;   (command "kreis" p1plat prad)                        ;<-ENTFERNEN
    (setq p1plat (polar p1plat angp90 (+ pabst prad prad)))
  )
  (setq p1plat (polar pcplat angm90 (+ pabst prad prad)))
  (while (<(distance pfirst p1plat) (- hbreite prad))
    (gw_platte)                                            ;<-NEU
;;   (command "kreis" p1plat  prad)                       ;<-ENTFERNEN
    (setq p1plat (polar p1plat angm90 (+ pabst prad prad)))
  )
)
; Zeichnet Reihen von Platten
(defun zeichplat ()
  (setq pdist (+ prad pabst))
  (setq off 0.0)
  (while (<= pdist (- plength prad))
    (zreihe pdist off)
    (setq pdist (+ pdist (* (+ pabst prad prad) (sin (wib 60)))))
    (if (= off 0.0)
      (setq off (* (+ pabst prad prad) (cos (wib 60))))
      (setq off 0.0)
    )
  )
)
; Führt Befehl aus und ruft dazu folgende Funktion auf
(defun C:DDWEG ()                    ; <- ÜBERARBEITEN, WEG durch DDWEG ersetzen
  (setq altfehe *error*                                    ; <- NEU
     *error* gw_err                                      ; <- NEU
     sblip nil                                            ; <- NEU
     scmde nil                                           ; <- NEU
     gwerr nil                                           ; <- NEU
  )                                                        ; <- NEU
  (gwbenutz)
  (setq sblip (getvar "blipmode"))
  (setq scmde (getvar "cmdecho"))
  (setq sang (getvar "snapang"))                           ; <- NEU
  (setvar "blipmode" 0)
  (setvar "cmdecho" 0)
  (zeichum)
  (gw_dialog)                                              ; <- NEU
  (zeichplat)
  (setvar "blipmode" sblip)
  (setvar "cmdecho" scmde)
  (setvar "snapang" sang)                                  ;<-NEU
  (setq *error* altfehe)                                   ; <- NEU
  (princ)
)
```

```
; Druckt Meldung nach dem Laden                          ; <-NEU

(princ "\nDDGW.lsp Loaded. Geben Sie DDWEG ein, um damit zu arbeiten.")     ;
<- NEU
(princ)                                                 ; <- NEU
```

12 Übersicht über AutoLISP-Funktionen

Mit Hilfe der in diesem Kapitel enthaltenen Listen können Sie Funktionen finden, ohne ihren Namen zu kennen. Alle AutoLISP-Funktionen sind kurz erläutert und in folgende funktionelle Gruppen eingeteilt:

- *Grundlegende Funktionen* (arithmetische Funktionen, Funktionen zur Zeichenkettenbearbeitung, relationale Funktionen und Bedingungsfunktionen, Funktionen zur Listenbearbeitung, zur Symbolbearbeitung, zur Funktionsbearbeitung, zur Fehlerbearbeitung und zur Anwendungsbearbeitung)
- *Dienstfunktionen* (Abfrage- und Befehlsfunktionen, Funktionen zur Anzeigesteuerung, Funktionen zur Benutzereingabe, geometrische Funktionen, Konvertierungsfunktionen, Funktionen zur Dateibearbeitung und zum Gerätezugriff)
- *Auswahlsatz-, Objekt- und Symboltabellenfunktionen* (Funktionen zur Bearbeitung von Auswahlsätzen, Objekten, erweiterten Daten und Symboltabellen)
- *Funktionen für programmierbare Dialogfelder* (Funktionen zum Öffnen und Schließen von Dialogfeldern, zur Bearbeitung von Dialogfeldkomponenten und Attributen, zur Bearbeitung von Listenfeldern und Pop-Up-Listen, zur Bearbeitung von Bildkomponenten von Dialogfeldern und zur Bearbeitung anwendungsspezifischer Daten)
- *Speicherverwaltungsfunktionen*

Die Funktionen sind nach Datentypen und den Aktionen angeordnet, die sie ausführen. Die folgenden Informationen sind parallel zu den in Kapitel 7 bis 10 enthaltenen Informationen. Detaillierte Informationen zu den einzelnen AutoLISP Funktionen finden Sie in den alphabetischen Listen in Kapitel 13, "AutoLISP Funktionskatalog".

12.1 Grundlegende Funktionen

Die grundlegenden Funktionen umfassen die arithmetischen Funktionen, die Funktionen zur Zeichenkettenbearbeitung, die relationalen Funktionen und Bedingungsfunktionen, die Funktionen zur Listenbearbeitung, zur Symbolbearbeitung, zur Funktionsbearbeitung, zur Fehlerbearbeitung und zur Anwendungsbearbeitung. Eine ausführlichere Beschreibung findet sich in Kapitel 7, "AutoLISP-Grundlagen".

12.1.1 Arithmetische Funktionen

AutoLISP enthält folgende arithmetische Funktionen:

(+ *[Zahl Zahl]* ...)

Gibt die Summe aller Zahlen zurück.

(- *[Zahl Zahl]* ...)

Subtrahiert die zweite Zahl und folgende Zahlen von der ersten und gibt die Differenz zurück.

(* *[Zahl Zahl]* ...)

Gibt das Produkt aller Zahlen zurück.

(/ *[Zahl Zahl]* ...)

Dividiert die erste Zahl durch das Produkt der verbleibenden Zahlen und gibt den Quotienten zurück.

(~ *Ganzzahl*)

Gibt die bitweise Negation (1er-Komplement) des Arguments zurück.

(1+ *Zahl*)

Gibt das Argument um 1 erhöht zurück.

(1- *Zahl*)

Gibt das Argument um 1 verringert zurück.

(abs *Zahl*)

Gibt den Absolutwert des Argumentes zurück.

(atan *Zahl1 [Zahl2]*)

Gibt den Arkustangens einer Zahl im Bogenmaß zurück.

(cos *Winkel*)

Gibt den Kosinus eines im Bogenmaß ausgedrückten Winkels zurück.

(exp *Zahl*)

Gibt die Konstante e (eine reale Zahl) potenziert auf einen bestimmten Wert zurück (den natürlichen Antilogarithmus).

(expt *Basisexponent*)

Gibt eine Zahl potenziert auf einen bestimmten Wert zurück.

(fix *Zahl*)

Konvertiert eine reale Zahl in den nächstkleineren ganzzahligen Wert.

(float *Zahl*)

Konvertiert eine Zahl in eine reale Zahl.

(gcd *Ganzzahl1 Ganzzahl2*)

Gibt den größten gemeinsamen Nenner zweier Ganzzahlen zurück.

(log *Zahl*)

Gibt den natürlichen Logarithmus einer Zahl als reale Zahl zurück.

(logand *Ganzzahl Ganzzahl* ...)

Gibt das Ergebnis der logischen bitweisen Operation AND einer Liste von Ganzzahlen zurück.

(logior *Ganzzahl Ganzzahl* ...)

Gibt das Ergebnis der logischen bitweisen Operation Inklusiv-OR einer Liste von Ganzzahlen zurück.

(lsh *Ganzzahl Anzahlbits*)

Gibt die logische bitweise Verschiebung einer Ganzzahl um eine bestimmte Bitzahl zurück.

(max *Zahl Zahl* ...)

Gibt die größten der angegebenen Zahlen zurück.

(min *Zahl Zahl* ...)

Gibt die kleinste der angegebenen Zahlen zurück.

(minusp *Zahl*)

Überprüft, ob eine Zahl negativ ist.

(rem *Zahl1 Zahl2* ...)

Dividiert die erste Zahl durch die zweite und gibt den Rest zurück.

(sin *Winkel*)

Gibt den Sinus eines Winkels als reale Zahl in Bogenmaß zurück.

(sqrt *Zahl*)

Gibt die Quadratwurzel einer Zahl als reale Zahl zurück.

(zerop *Zahl*)

Überprüft, ob eine Zahl Null ergibt.

12.1.2 Funktionen zur Zeichenkettenbearbeitung

Es gibt folgende Funktionen zur Bearbeitung von Zeichenketten:

(strcase *Z_kette [wie]*)

Gibt eine Zeichenkette zurück, in der alle alphabetischen Zeichen entweder in Groß- oder Kleinbuchstaben umgewandelt wurden.

(strcat *Z_kette1 [Z_kette2]* ...)

Gibt eine Zeichenkette zurück, welche die Verkettung mehrerer Zeichenketten ist.

(strlen *[Z_kette]* ...)

Gibt eine Ganzzahl zurück, welche der Anzahl der Zeichen in einer Zeichenkette entspricht.

(substr *Z_kette Anfang [Länge]*)

Gibt eine untergeordnete Zeichenkette einer Zeichenkette zurück

(wcmatch *Z_kette Muster*)

Berücksichtigt Platzhalter in einer Zeichenkette.

12.1.3 Relationale Funktionen und Bedingungsfunktionen

Es gibt folgende relationale Funktionen und Bedingungsfunktionen:

(= *Zahl_Z_kette [Zahl_Z_kette]* ...)

Gibt T zurück, wenn alle Argumente numerisch gleich sind, und gibt andernfalls nil zurück.

(/= *Zahl_Z_kette [Zahl_Z_kette]* ...)

Gibt T zurück, wenn die Argumente numerisch nicht gleich sind, und nil, falls Gleichheit herrscht.

(< *Zahl_Z_kette [Zahl_Z_kette]* ...)

Gibt T zurück, wenn das Argument numerisch kleiner ist als das Argument rechts von ihm, und gibt andernfalls nil zurück.

(<= *Zahl_Z_kette [Zahl_Z_kette]* ...)

Gibt T zurück, wenn das Argument numerisch kleiner als oder gleich dem Argument rechts von ihm ist, und gibt andernfalls nil zurück.

(> *Zahl_Z_kette [Zahl_Z_kette]* ...)

Gibt T zurück, wenn das Argument numerisch größer ist als das Argument rechts von ihm, und gibt andernfalls nil zurück.

(>= *Zahl_Z_kette [Zahl_Z_kette]* ...)

Gibt T zurück, wenn das Argument numerisch größer als oder gleich dem Argument rechts von ihm ist, und gibt andernfalls nil zurück.

(and *Ausdruck* ...)

Gibt das logische AND für eine Liste mit Ausdrücken zurück.

(Boole *Funktion Ganzzahl1 Ganzzahl2* ...)

Stellt die allgemeine bitweise Boolesche Funktion dar.

(cond (*Test1 Ergebnis1* ...) ...)

Ist die wichtigste Bedingungsfunktion von AutoLISP.

(eq *Ausdruck1 Ausdruck2*)

Stellt fest, ob zwei Ausdrücke identisch sind.

(equal *Ausdruck1 Ausdruck2 [ungenau]*)

Stellt fest, ob zwei Ausdrücke gleich sind.

(if *Testausdruck dannausdruck [sonstausdruck]*)

Wertet Ausdrücke je nach Bedingung aus.

(or *Ausdruck* ...)

Gibt das logische OR einer Liste von Ausdrücken zurück.

(repeat *Ganzzahl Ausdruck*...)

Wertet jeden Ausdruck so oft wie angegeben aus und gibt den Wert des letzten Ausdrucks zurück.

(while *Testausdruck Ausdruck*...)

Wertet einen Testausdruck aus; ist dieser nicht nil, werden andere Ausdrücke ausgewertet; wiederholt diesen Vorgang, bis der Testausdruck nil ergibt.

12.1.4 Listenbearbeitungsfunktionen

Es gibt folgende Funktionen zur Bearbeitung von Listen:

(acad_strlsort *Liste*)

Sortiert eine Liste von Zeichenketten alphabetisch.

(append *Liste* ...)

Fügt eine beliebige Zahl von Listen zu einer Liste zusammen.

(assoc *Element A_liste*)

Durchsucht eine Assoziationsliste nach einem Element und gibt den entsprechenden Eintrag der Liste zurück.

(car *Liste*) and (cdr *Liste*)

Gibt das erste Element einer Liste zurück oder eine Liste, die alle Elemente bis auf das erste enthält.

(cons *neues-ertes-Element-Liste*)

Die Grundfunktion zur Erstellung von Listen.

(foreach *Name Liste Ausdruck*...)

Berechnet einen Ausdruck für alle Elemente einer Liste.

(last *Liste*)

Gibt das letzte Element einer Liste zurück.

(length *Liste*)

Gibt eine Ganzzahl zurück, welche die Anzahl der Elemente in einer Liste angibt.

(list *Ausdruck* ...)

Nimmt eine beliebige Anzahl von Ausdrücken und faßt sie in einer Liste zusammen.

(listp *Element*)

Überprüft, ob ein Element eine Liste ist.

(mapcar *Funktion Liste1* ... *Listen*)

Gibt eine Liste des Ergebnisses einer durchgeführten Funktion mit den einzelnen Elementen einer oder mehrerer Listen zurück, die der Funktion als Argumente dienen.

(member *Ausdruck Liste*)

Gibt eine Liste des Ergebnisses einer durchgeführten Funktion mit den einzelnen Elementen einer oder mehrerer Listen zurück, die der Funktion als Argumente dienen.

(nth *n Liste*)

Gibt das nte Element einer Liste zurück.

(reverse *Liste*)

Gibt eine Liste mit der Umkehrung ihrer Elemente zurück.

(subst *neu_Element alt_Element Liste*)

Durchsucht eine Liste nach einem alten Element und gibt eine Kopie der Liste zurück, in der das alte Element jedesmal durch ein neues Element ersetzt wird.

12.1.5 Symbolbearbeitungsfunktionen

Es gibt folgende Funktionen zur Symbolbearbeitung:

(atom *Element*)

Überprüft, ob ein Element ein Atom ist.

(atoms-family *Format [Sym_liste]*)

Gibt eine Liste der im Augenblick definierten Symbole zurück.

(boundp *Symbol*)

Überprüft, ob ein Wert an ein Symbol gebunden ist.

(not *Element*)

Überprüft, ob ein Element nil ergibt.

(null *Element*)

Überprüft, ob ein Element an nil gebunden ist.

(numberp *Element*)

Überprüft, ob ein Element eine reale Zahl oder eine Ganzzahl ist.

(quote *Ausdr*)

Gibt einen Ausdruck zurück, ohne ihn ausgewertet zu haben.

(read *[Zeichenkette]*)

Gibt die erste Liste oder das erste Atom zurück, das von einer Zeichenkette übertragen wurde.

(set *Symbol Ausdruck*)

Ordnet den Wert eines in Anführungszeichen angegebenen Symbolnamens einem Ausdruck zu.

(setq *Symbol1 Ausdruck1 [Symbol2 Ausdruck2]* . . .)

Ordnet den Wert eines oder mehrerer Symbole den zugehörigen Ausdrücken zu.

(type *Element*)

Gibt den Typ eines angegebenen Elements zurück.

12.1.6 Funktionsbearbeitungsfunktionen

Es gibt folgende Funktionen zur Funktionsbearbeitung:

(apply *Funktion Liste)*

Gibt eine Argumentliste an eine bestimmte Funktion weiter.

(defun *Symbol Argument Liste Ausdruck* . . .)

Definiert eine Funktion.

(eval *Ausdruck*)

Gibt die Berechnung eines AutoLISP-Ausdrucks als Ergebnis zurück.

(lambda *Argumente Ausdruck* . . .)

Definiert eine unbenannte Funktion.

(progn *[Ausdruck]* . . .)

Wertet jeden Ausdruck der Reihe nach aus und gibt den Wert des letzten Ausdrucks zurück.

(trace *Funktion* ...)

Unterstützt die Fehlersuche in AutoLISP.

(untrace *Funktion* ...)

Löscht den Inhalt des Protokoll-Flags für die definierte Funktion.

12.1.7 Fehlerbearbeitungsfunktionen

Es gibt folgende Funktionen zur Fehlerbearbeitung:

(alert *Z_kette*)

Öffnet ein Warnfenster mit einer Fehler- oder Warnmeldung, die als Zeichenkette übergeben wird.

(*error* *Z_kette*)

Durch den Benutzer zu definierende Funktion zur Fehlerbehandlung.

(exit)

Erzwingt die Beendigung der aktuellen Anwendung.

(quit)

Erzwingt die Beendigung der aktuellen Anwendung.

12.1.8 Anwendungsbearbeitungsfunktionen

Es gibt folgende Funktionen zur Handhabung von Anwendungen:

(ads)

Gibt eine Liste der aktuell geladenen ADS-Anwendungen zurück.

(arx)

Gibt eine Liste der aktuell geladenen ARX-Anwendungen zurück.

(arxload *Anwendung [beiFehler]*)

Lädt eine ARX-Anwendung.

(arxunload *Anwendung [beiFehler]*)

Beendet eine ARX-Anwendung.

(autoarxload *Dateiname Befehlsliste*)

Definiert Befehlsnamen zum Laden einer zugehörigen ARX-Datei.

(autoload *Dateiname Befehlsliste*)

Definiert Befehlsnamen zum Laden einer zugehörigen AutoLISP-Datei.

(autoxload *Dateiname Befehlsliste*)

Definiert Befehlsnamen zum Laden einer zugehörigen ADS-Anwendung.

(load *Dateiname [beiFehler]*)

Wertet die AutoLISP-Ausdrücke in einer Datei aus.

(startapp *Anw_befehl Datei*)

Startet eine Windows-Anwendung.

(xload *Anwendung [beiFehler]*)

Lädt eine ADS- Anwendung.

(xunload ***Anwendung [beiFehler]*****)**

Beendet eine ADS-Anwendung.

12.1.9 Dienstfunktionen

Die Dienstfunktionen umfassen Abfrage- und Befehlsfunktionen, Funktionen zur Anzeigesteuerung, Funktionen zur Benutzereingabe, geometrische Funktionen, Konvertierungsfunktionen, Funktionen zur Dateibearbeitung und zum Gerätezugriff. Eine ausführlichere Beschreibung findet sich in Kapitel 8, "Allgemeine Dienstprogramm- funktionen".

12.1.10 Abfrage- und Befehlsfunktionen

Es gibt folgende Abfrage- und Befehlsfunktionen:

(acad_colordlg ***Farbnum [Flag]*****)**

Öffnet das AutoCAD-Standarddialogfeld für die Farbauswahl.

(acad_helpdlg ***Hilfedatei Thema*****)**

Aktiviert die Hilfefunktion (veraltet).

(command ***[Argumente]*** **...)**

Führt einen AutoCAD-Befehl aus.

(getcfg ***cfg_Name*****)**

Ruft Anwendungsdaten aus dem Teil AppData der Datei acad.cfg ab.

(getcname ***Befehlsname*****)**

Findet den lokalisierten oder englischen Namen eines AutoCAD-Befehls.

(getenv ***Variablenname*****)**

Gibt den Wert einer Umgebungsvariable als Zeichenkette zurück.

(getvar ***Var_name*****)**

Ermittelt den Wert einer AutoCAD-Systemvariablen.

(help ***[Hilfedatei [Thema [Befehl]]]*****)**

Aktiviert die Hilfefunktion.

(setcfg ***cfg_Name cfg_Wert*****)**

Schreibt Anwendungsdaten in den Abschnitt AppData der Datei acad.cfg.

(setfunhelp ***Funktion [Hilfedatei [Thema [Befehl]]]*****)**

Registriert einen benutzerdefinierten Befehl mit der Hilfefunktion, so daß die entsprechenden Hilfedateien und Hilfethemen aufgerufen werden, wenn der Benutzer Hilfe zu diesem Befehl anfordert.

(setvar ***Var_Name Wert*****)**

Setzt eine AutoCAD-Systemvariable auf einen definierten Wert.

(ver)

Gibt eine Zeichenkette zurück, welche die aktuelle AutoLISP-Versionsnummer enthält.

12.1.11 Anzeigesteuerungsfunktionen

Es gibt folgende Funktionen zur Steuerung der Anzeige:

(graphscr)

Zeigt den AutoCAD-Grafikbildschirm an.

(grdraw ***von zu Farbe [Markierung]*****)**

Zeichnet einen Vektor zwischen zwei Punkten im aktuellen Ansichtsfenster.

(grtext ***[Quader Text [Markierung]]*****)**

Schreibt Text in die Statuszeile oder in den Bildschirmmenübereich.

(grvecs *Vliste [Trans]*)

Zeichnet mehrere Vektoren auf dem Grafikbildschirm.

(menucmd *Z_kette*)

Aktiviert Menübefehle oder definiert und ermittelt den Status von Menüoptionen.

(menugroup *Gruppenname*)

Verifies that a menugroup is loaded

(prin1 *[Ausdruck [Dateideskr]]*)

Druckt einen Ausdruck in die Befehlszeile oder schreibt einen Ausdruck in eine geöffnete Datei.

(princ *[Ausdruck [Dateideskr]]*)

Druckt einen Ausdruck in die Befehlszeile oder schreibt einen Ausdruck in eine geöffnete Datei.

(print *[Ausdruck [Dateideskr]]*)

Druckt einen Ausdruck in die Befehlszeile oder schreibt einen Ausdruck in eine geöffnete Datei.

(prompt *Anfrage*)

Zeigt im Eingabeaufforderungsbereich Ihres Bildschirms eine Zeichenkette an.

(redraw *[Elementname [Modus]]*)

Zeichnet ein aktuelles Ansichtsfenster oder ein definiertes Objekt (Element) im aktuellen Ansichtsfenster neu.

(terpri)

Druckt einen Zeilenvorschub in die Befehlszeile.

(textpage)

Wechselt vom Grafik- zum Textbildschirm.

(textscr)

Wechselt vom Grafik- zum Textbildschirm (vergleichbar mit der Funktionstaste zur Bildschirmumschaltung bei AutoCAD).

(vports)

Gibt eine Liste von Ansichtsfensterdeskriptoren für die aktuelle Ansichtsfensterkonfiguration zurück.

12.1.12 Benutzereingabefunktionen

Es gibt folgende Benutzereingabefunktionen:

(entsel *[Anfrage]*)

Fordert den Benutzer zur Auswahl eines einzelnen Objekts (Elements) durch Bestimmung eines Punkts auf.

(getangle *[p] [Anfrage]*)

Erwartet die Benutzereingabe eines Winkels und gibt diesen Winkel im Bogenmaß zurück.

(getcorner *p [msg]*)

Erwartet die Benutzereingabe einer zweiten Ecke für ein Rechteck.

(getdist *[p] [Anfrage]*)

Erwartet die Benutzereingabe einer Entfernung.

(getfiled *Titel Vorgabe Erw Flags*)

Fordert den Benutzer mit dem AutoCAD-Standard-Dateidialogfeld zur Eingabe eines Dateinamens auf und gibt diesen Dateinamen zurück.

(getint *[Anfrage]*)

Erwartet die Benutzereingabe einer Ganzzahl und gibt diese Ganzzahl zurück.

(getkword *[Anfrage]*)

Erwartet die Benutzereingabe eines Schlüsselworts und gibt dieses Schlüsselwort zurück.

(getorient *[p] [Anfrage]*)

Erwartet die Benutzereingabe eines Winkels und gibt diesen Winkel im Bogenmaß zurück.

(getpoint *[p] [Anfrage]*)

Erwartet die Benutzereingabe eines Punkts und gibt diesen Punkt zurück.

(getreal *[Anfrage]*)

Erwartet die Benutzereingabe einer realen Zahl und gibt diese reale Zahl zurück.

(getstring *[cr] [Anfrage]*)

Erwartet die Benutzereingabe einer Zeichenkette und gibt diese Zeichenkette zurück.

(initget *[Bits] [Z_kette]*)

Führt Schlüsselwörter ein, die beim nächsten Aufruf einer Benutzereingabefunktion gebraucht werden.

(nentsel *[Anfrage]*)

Fordert den Benutzer auf, durch Definieren eines Punkts ein Objekt (Element) auszuwählen, und gewährt Zugriff auf die Definitionsdaten, die in einem komplexen Objekt enthalten sind.

(nentselp *[Anfrage] [p]*)

Funktioniert ähnlich wie die Funktion nentsel, benötigt jedoch keine Benutzereingabe.

Geometrische Funktionen

Es gibt folgende geometrische Funktionen:

(angle *p1 p2*)

Gibt den Winkel einer Linie, die durch ihre beiden Endpunkte definiert ist, im Bogenmaß zurück.

(distance *p1 p2*)

Gibt den 3D-Abstand zwischen zwei Punkten zurück.

(inters *p1 p2 p3 p4 [aufseg]*)

Ermittelt den Schnittpunkt zweier Linien.

(osnap *p Modus*)

Gibt als Ergebnis einer Anwendung des Ofang-Modus auf einen festgelegten Punkt einen 3D-Punkt zurück.

(polar *p Winkel Abstand*)

Gibt den 3D-Punkt im BKS in einem festgelegten Winkel und Abstand von einem Punkt zurück.

(textbox *Elem_liste*)

Mißt ein definiertes Textobjekt und gibt die diagonalen Koordinaten eines Feldes zurück, das den Text enthält.

12.1.13 Konvertierungsfunktionen

Es gibt folgende Konvertierungsfunktionen:

(angtof *Z_kette [Modus]*)

Konvertiert eine Zeichenkette, die einen Winkel darstellt, in eine reale (Gleitkomma-) Zahl im Bogenmaß.

(angtos *Winkel [Modus [Genauigkeit]]*)

Konvertiert einen Winkelwert im Bogenmaß in eine Zeichenkette.

(ascii *Z_kette*)

Gibt den ASCII-Wert des ersten Zeichens einer Zeichenkette als Ganzzahl zurück.

(atof *Z_kette*)

Konvertiert eine Zeichenkette in eine reale Zahl.

(atoi *Z_kette*)

Konvertiert eine Zeichenkette in eine Ganzzahl.

(chr *Ganzzahl*)

Gibt das Zeichen als Zeichenkette der Länge 1 zurück, das dem angegebenen ganzzahligen ASCII-Zeichencode entspricht.

(cvunit *Wert von zu*)

Konvertiert einen Wert von einer Maßeinheit in eine andere.

(distof *Z_kette [Modus]*)

Konvertiert eine Zeichenkette, die eine reale (Gleitkomma-) Zahl darstellt, in einen realen Wert.

(itoa *Ganzzahl*)

Gibt die Konvertierung einer Ganzzahl in eine Zeichenkette zurück.

(rtos *Zahl [Modus [Genauigkeit]]*)

Konvertiert eine Zahl in eine Zeichenkette.

(trans *p von zu [Verschiebung]*)

Überträgt einen Punkt (oder eine Verschiebung) von einem Koordinatensystem in ein anderes.

12.1.14 Dateibearbeitungsfunktionen

Es gibt folgende Dateibearbeitungsfunktionen:

(close *Dateideskr*)

Schließt eine geöffnete Datei.

(findfile *Dateiname*)

Durchsucht den AutoCAD-Bibliotheksuchpfad nach der angegebenen Datei.

(open *Dateiname Modus*)

Öffnet eine Datei für den Zugriff durch die AutoLISP-E/A-Funktionen.

(read-char *[Dateideskr]*)

Gibt den dezimalen ASCII-Code zurück, der das Zeichen aus dem Tastatureingabepuffer oder aus einer geöffneten Datei darstellt.

(read-line *[Dateideskr]*)

Liest eine Zeichenkette von der Tastatur oder aus einer geöffneten Datei.

(write-char Zahl *[Dateideskr]*)

Schreibt ein Zeichen auf den Bildschirm oder in eine geöffnete Datei.

(write-line *Z_kette [Dateideskr]*)

Schreibt eine Zeichenkette auf den Bildschirm oder in eine geöffnete Datei.

12.1.15 Gerätezugriffsfunktionen

Es gibt folgende Gerätezugriffsfunktionen:

(grread *[Spur] [alle [Curtyp]]*)

Liest Werte von jedem AutoCAD-Eingabegerät ein.

(tablet *Code [Reihe1 Reihe2 Reihe3 Richtung]*)

Ermittelt und setzt Digitalisier-(Tablett-)kalibrierungen.

12.2 Auswahlsatz-, Objekt- und Symboltabellenfunktionen

Die Auswahlsatz-, Objekt- und Symboltabellenfunktionen umfassen die Funktionen zur Bearbeitung von Auswahlsätzen, Objekten, erweiterten Daten und Symboltabellen. Eine ausführlichere Beschreibung findet sich in Kapitel 9, "Auswahlsatz-, Objekt- und Symboltabellen funktionen"..

12.2.1 Funktionen zur Auswahlsatzbearbeitung

Es gibt folgende Funktionen zur Bearbeitung von Auswahlsätzen:

(ssadd *[Elem_name [Ausw_satz]]*)

Fügt einem Auswahlsatz ein Objekt (Element) hinzu oder erstellt einen neuen Auswahlsatz.

(ssdel *Elem_name Ausw_satz*)

Löscht ein Objekt (Element) aus einem Auswahlsatz.

(ssget *[Modus] [p1 [p2]] [P-liste] [Filterliste]*)

Fordert den Benutzer auf, Objekte (Elemente) zu wählen, und gibt einen Auswahlsatz zurück.

(ssgetfirst)

Legt fest, welche Objekte ausgewählt und mit Griffen versehen werden.

(sslength *Ausw_satz*)

Gibt eine Ganzzahl zurück, welche die Anzahl der Objekte (Elemente) in einem Auswahlsatz enthält.

(ssmemb *Elem_name Ausw_satz*)

Überprüft, ob ein Objekt (Element) in einem Auswahlsatz enthalten ist.

(ssname *Ausw_satz Index*)

Gibt den Objektnamen (Elementnamen) des indizierten Elements eines Auswahlsatzes zurück.

(ssnamex *Ausw_satz Index*)

Findet Informationen zur Art und Weise der Erstellung eines Auswahlsatzes wieder.

(sssetfirst *Griffsatz [Picksatz]*)

Legt fest, welche Objekte ausgewählt und mit Griffen versehen werden.

12.2.2 Objektbearbeitungsfunktionen

Es gibt folgende Funktionenzur Objektbearbeitung:

(entdel *Elem_name*)

Löscht Objekte (Elemente) oder macht das Löschen eines zuvor gelöschten Objekts wieder rückgängig.

(entget *Elem_name [Anw_liste]*)

Ermittelt die Definitionsdaten eines Objekts (Elements).

(entlast)

Gibt den Namen des zuletzt erzeugten, nicht gelöschten Hauptobjekts (Elements) in der Zeichnung zurück.

(entmake *[Elem_liste]*)

Erzeugt ein neues Element (grafisches Objekt) in der Zeichnung.

(entmakex *[Elem_liste]*)

Erstellt ein neues Objekt oder Element, versieht es mit einer Referenz und einem Elementnamen (ohne einen Eigentümer zuzuweisen) und gibt den neuen Namen des Elements zurück.

(entmod *Elem_liste*)

Modifiziert die Definitionsdaten eines Objekts (Elements).

(entnext *[Elem_name]*)

Gibt den Namen des nächsten Objekts (Elements) in der Zeichnung zurück.

(entupd *Elem_name*)

Aktualisiert die Bildschirmdarstellung eines Objekts (Elements).

(handent *Referenz*)

Gibt einen Objektnamen (Elementnamen) auf der Basis seiner Referenz zurück.

12.2.3 Funktionen zur Bearbeitung erweiterter Daten

Es gibt folgende Funktionen zur Bearbeitung erweiterter Daten:

(regapp *Anwendung*)

Registriert einen Anwendungsnamen mit der aktuellen AutoCAD-Zeichnung, um die Verwendung erweiterter Objektdaten vorzubereiten.

(xdroom *Elem_name*)

Gibt die Größe des Bereichs für erweiterte Daten (X-Daten) zurück, der für ein Objekt (Element) zur Verfügung steht.

(xdsize *Liste*)

Gibt die Größe (in Byte), die eine Liste in Anspruch nimmt, wenn Sie in Form von erweiterten Daten mit einem Objekt (Element) verbunden ist.

12.2.4 Funktionen zur Bearbeitung von Symboltabellen und Wörterbüchern

Es gibt folgende Funktionen zur Bearbeitung von Symboltabellen und Wörterbüchern:

(dictadd *Elem_name Symbol neu-Objekt*)

Fügt dem angegebenen Wörterbuch ein nicht-grafisches Objekt hinzu.

(dictdel *Elem_name Symbol*)

Entfernt einen Eintrag aus dem angegebenen Wörterbuch.

(dictnext *Elem_name Symbol [erstes]*)

Findet das nächste Element in einem Wörterbuch.

(dictrename *Elem_name alt_Symbol neu_Symbol*)

Benennt einen Eintrag im Wörterbuch um.

(dictsearch *Elem_name Symbol [Setzfolg]*)

Durchsucht ein Wörterbuch nach einem Element.

(namedobjdict)

Gibt den Elementnamen des in der aktuellen Zeichnung benannten Objektwörterbuchs an, das die Grundlage aller nichtgrafischen Elemente in der Zeichnung bildet.

(setview *Ansichtsbeschr [Ansichtsfen_bez]*)

Erstellt eine Ansicht für ein angegebenes Ansichtsfenster

(snvalid *Symbolname*)

Sucht im Symboltabellennamen nach gültigen Zeichen.

(tblnext *Tabellenname [erstes]*)

Findet das nächste Element in einer Symboltabelle.

(tblobjname ***Tabellenname Symbol*****)**

Gibt den Elementnamen eines festgelegten Symboltabelleneintrags zurück.

(tblsearch ***Tabellenname Symbol [Setzfolg]*****)**

Durchsucht eine Symboltabelle nach einem Symbolnamen.

12.3 Funktionen für programmierbare Dialogfelder

Die Funktionen für programmierbare Dialogfelder umfassen Funktionen zum Öffnen und Schließen von Dialogfeldern, zur Bearbeitung von Dialogfeldkomponenten und Attributen, zur Bearbeitung von Listenfeldern und Pop-Up-Listen, zur Bearbeitung von Bildkomponenten und zur Bearbeitung anwendungsspezifischer Daten. Eine ausführlichere Beschreibung findet sich in Kapitel 10, "Arbeiten mit Dialogfeldern"..

12.3.1 Funktionen zum Öffnen und Schließen von Dialogfeldern

Es gibt folgende Funktionen zum Öffnen und Schließen von Dialogfeldern:

(done_dialog ***[Status]*****)**

Schließt ein Dialogfeld.

(load_dialog ***dcl_Datei*****)**

Lädt eine DCL-Datei.

(new_dialog ***Dialogname dcl_ID [Operation [Bildschirm_P]]*****)**

Eröffnet ein neues Dialogfeld und zeigt es an; kann auch eine Vorgabeoperation angeben.

(start_dialog)

Zeigt ein Dialogfeld an und akzeptiert die Benutzereingabe.

(term_dialog)

Schließt alle aktuellen Dialogfelder, als ob der Benutzer jedes einzelne geschlossen hätte.

(unload_dialog ***dcl_Bez*****)**

Beendet eine DCL-Datei.

12.3.2 Funktionen zur Bearbeitung von Dialogfeldkomponenten und Attributen

Es gibt folgende Funktionen zur Bearbeitung von Dialogfeldkomponenten und Attributen:

(action_tile ***Schlüssel Operationsausdruck*****)**

Weist einer bestimmten Komponente in einem Dialogfeld eine auszuführende Aktion zu.

(get_attr ***Schlüssel Attribu*****)**

Ermittelt den DCL-Wert eines Dialogfeldattributs.

(get_tile ***Schlüssel*****)**

Ermittelt den aktuellen Laufzeitwert einer Dialogfeldkomponente.

(mode_tile ***Schlüssel Modus*****)**

Definiert den Modus einer Dialogfeldkomponente.

(set_tile ***Schlüssel Wert*****)**

Setzt den Wert einer Dialogfeldkomponente fest.

12.3.3 Funktionen zur Bearbeitung von Listenfeldern und Pop-Up-Listen

Es gibt folgende Funktionen zur Bearbeitung von Listenfeldern und Pop-Up-Listen:

(add_list ***Z_kette*****)**

Fügt der aktuellen Dialogfeldliste eine Zeichenkette hinzu oder modifiziert diese.

(end_list)

Beendet die Verarbeitung der aktuellen Dialogfeldliste.

(start_list ***Schlüssel [Operation [Index]]*****)**

Beginnt die Verarbeitung einer Liste im Listenfeld oder in der Dialogfeldkomponente "Pop-Up-Liste".

12.3.4 Funktionen zur Bearbeitung von Bildkomponenten

Es gibt folgende Funktionen zur Bearbeitung von Bildkomponenten:

(dimx_tile ***Schlüssel*****) and (dimy_tile** ***Schlüssel*****)**

Ermittelt die Abmessungen einer bestimmten Dialogfeldkomponente.

(end_image)

Beendet die Erstellung des aktuellen Dialogfeldbilds.

(fill_image x***1 y1 Breite Höhe Farbe*****)**

Zeichnet ein ausgefülltes Rechteck in der aktuellen Dialogfeldbildkomponente.

(slide_image x***1 y1 Breite Höhe Dianame*****)**

Zeigt ein AutoCAD-Dia in der aktuellen Dialogfeldbildkomponente an.

(start_image ***Taste*****)**

Beginnt die Erstellung eines Bilds in der Dialogfeldkomponente.

(vector_image ***x1y1x2y2 Farbe*****)**

Zeichnet einen Vektor in das aktuelle Dialogfeldbild.

12.3.5 Funktionen zur Bearbeitung anwendungsspezifischer Daten

Es gibt die folgende Funktion zur Bearbeitung anwendungsspezifischer Daten:

(client_data_tile ***Schlüssel Clientdaten*****)**

Verbindet Anwendungsdaten mit einer Dialogfeldkomponente.

12.4 Speicherverwaltungsfunktionen

Die folgenden Speicherverwaltungsfunktionen werden in Kapitel 15, "Speicherverwaltung".ausführlicher beschrieben.

(alloc ***Ganzzahl*****)**

Setzt die Segmentgröße auf eine bestimmte Anzahl von Knoten.

(expand ***Zahl*****)**

Weist Knotenraum durch Anforderung einer bestimmten Anzahl von Segmenten zu.

(gc)

Erzwingt eine Abfallsammlung und macht so Knotenraum frei.

(mem)

Zeigt den aktuellen Zustand des AutoLISP-Speichers an.

13 AutoLISP Funktionskatalog

Im folgenden finden Sie eine alphabetische Auflistung aller Standard-AutoLISP-Funktionen, wie sie in AutoCAD definiert sind. Die Funktionen sind alphabetisch aufgeführt.

Jeder Punkt dieses Kapitels enthält eine kurze Beschreibung zur Anwendung der Funktion. Die daran angeschlossene Funktionssyntax listet die von der Funktion benötigten Argumente und deren Abfolge auf. (Spezielle Informationen über Syntax-Anweisungen finden Sie in "AutoLISP-Funktionssyntax.") Sollten Sie den Namen der gesuchten Funktion nicht kennen, verwenden Sie die Auflistung in Kapitel 12, "Übersicht über AutoLISP-Funktionen", die nach funktionellen Gruppen geordnet ist.

Das Argument *Zahl* verlangt nach zusätzlicher Information: Eine Zahl kann eine reale Zahl, eine Ganzzahl oder ein Symbol sein, daß auf einen realen oder ganzzahligen Wert gesetzt ist. Sind alle Argumente Ganzzahlen, ist auch der Rückgabewert eine Ganzzahl. Ist eines der Argumente eine reale Zahl, so werden auch ganze Zahlen in reale Zahlen umgewandelt, und das Ergebnis ist ebenfalls eine reale Zahl.

Die meisten AutoLISP-Funktionen stehen immer zur Verfügung, einige sind jedoch durch eine AutoLISP- oder ARX-Anwendung definiert. Unter der Überschrift "Extern definierte Funktionen" wird die Datei aufgeführt, welche die extern definierten Funktionen enthält. Bevor Sie versuchen, diese Funktionen zu benutzen, sollten Sie ihre Verfügbarkeit überprüfen.

13.1 + (Addition)

Gibt die Summe aller Zahlen zurück.

(+ *[Zahl Zahl]* ...)

Wenn Sie nur ein `Zahl`-Argument angeben, wird diese Zahl zu Null addiert, d. h. die Zahl wird zurückgegeben. Werden keine Argumente angegeben, wird 0 zurückgegeben.

`(+ 1 2)`	*ergibt*	`3`
`(+ 1 2 3 4.5)`	*ergibt*	`10.5`
`(+ 1 2 3 4.0)`	*ergibt*	`10.0`

13.2 - (Subtraktion)

Subtrahiert die zweite Zahl und folgende Zahlen von der ersten und gibt die Differenz zurück.

(- *[Zahl Zahl]* ...)

Falls Sie mehr als zwei Argumente `Zahl` angeben, wird die Summe der zweiten plus aller folgenden Zahlen von der ersten Zahl subtrahiert. Wenn Sie nur ein Argument `Zahl` angeben, wird diese Zahl von Null subtrahiert, d. h. die negative Zahl wird zurückgegeben. Werden keine Argumente angegeben, wird 0 zurückgegeben.

`(- 50 40)`	*ergibt*	`10`
`(- 50 40.0)`	*ergibt*	`10.0`
`(- 50 40.0 2.5)`	*ergibt*	`7.5`
`(- 8)`	*ergibt*	`-8`

13.3 * (Multiplikation)

Gibt das Produkt aller Zahlen zurück.

(* *[Zahl Zahl]* ...)

Wenn Sie nur ein Argument `Zahl` angeben, wird diese Zahl mit Eins multipliziert, d. h. die Zahl wird zurückgegeben. Werden keine Argumente angegeben, wird 0 zurückgegeben.

`(* 2 3)`	*ergibt*	`6`
`(* 2 3.0)`	*ergibt*	`6.0`
`(* 2 3 4.0)`	*ergibt*	`24.0`
`(* 3 -4.5)`	*ergibt*	`-13.5`
`(* 3)`	*ergibt*	`3`

13.4 / (Division)

Dividiert die erste Zahl durch das Produkt der verbleibenden Zahlen und gibt den Quotienten zurück.

(/ *[Zahl Zahl]* ...)

Falls Sie mehr als zwei Argumente *Zahl* angeben, dividiert diese Funktion die erste Zahl durch das Produkt der folgenden Zahlen und gibt den endgültigen Quotienten zurück. Wenn Sie nur ein Argument *Zahl* angeben, wird diese Zahl durch Eins dividiert, d. h. die Zahl wird zurückgegeben. Werden keine Argumente angegeben, wird 0 zurückgegeben.

```
(/ 100 2)                ergibt  50
(/ 100 2.0)              ergibt  50.0
(/ 100 20.0 2)           ergibt  2.5
(/ 100 20 2)             ergibt  2
(/ 4)                    ergibt  4
```

13.5 = (ist gleich)

Gibt T zurück, wenn alle Argumente numerisch gleich sind, und gibt andernfalls nil zurück.

(= *Zahl_Z_kette [Zahl_Z_kette]* ...)

Jedes Argument *Zahl_Z_kette* kann eine Zahl oder eine Zeichenkette enthalten.

```
(= 4 4.0)                ergibt  T
(= 20 388)               ergibt  nil
(= 2.4 2.4 2.4)          ergibt  T
(= 499 499 500)          ergibt  nil
(= "me" "me")            ergibt  T
(= "me" "you")           ergibt  nil
```

Siehe auch

Die Funktionen **eq** und **equal**

13.6 /= (ungleich)

Gibt T zurück, wenn die Argumente numerisch nicht gleich sind, und nil, falls Gleichheit herrscht.

(/= *Zahl_Z_kette [Zahl_Z_kette]* ...)

Jedes Argument *Zahl_Z_kette* kann eine Zahl oder eine Zeichenkette enthalten.

```
(/= 10 20)               ergibt  T
(/= "you" "you")         ergibt  nil
(/= 5.43 5.44)           ergibt  T
(/= 10 20)               ergibt  T
```

13.7 < (kleiner als)

Gibt T zurück, wenn das Argument numerisch kleiner ist als das Argument rechts von ihm, und gibt andernfalls nil zurück.

(< *Zahl_Z_kette [Zahl_Z_kette]* ...)

Jedes Argument *Zahl_Z_kette* kann eine Zahl oder eine Zeichenkette enthalten.

```
(< 10 20)                ergibt  T
(< "c" "b")              ergibt  T
(< 357 33.2)             ergibt  nil
(<= 2 3 88)              ergibt  T
(< 2 3 4 4)              ergibt  nil
```

13.8 <= (kleiner oder gleich)

Gibt `T` zurück, wenn das Argument numerisch kleiner als oder gleich dem Argument rechts von ihm ist, und gibt andernfalls nil zurück.

`(<=` *Zahl_Z_kette [Zahl_Z_kette]* `...)`

Jedes Argument *`Zahl_Z_kette`* kann eine Zahl oder eine Zeichenkette enthalten.

```
(< 10 20)                ergibt  T
(<= "b" "b")             ergibt  T
(< 357 33.2)             ergibt  nil
(<= 2 9 9)               ergibt  T
(<= 2 9 4 5)             ergibt  nil
```

13.9 > (größer als)

Gibt `T` zurück, wenn das Argument numerisch größer ist als das Argument rechts von ihm, und gibt andernfalls `nil` zurück.

`(>` *Zahl_Z_kette [Zahl_Z_kette]* `...)`

Jedes Argument *`Zahl_Z_kette`* kann eine Zahl oder eine Zeichenkette enthalten.

```
(> 120 17)               ergibt  T
(> "c" "b")              ergibt  T
(> 3.5 1792)             ergibt  nil
(>= 77 4 2)              ergibt  T
(> 77 4 4)               ergibt  nil
```

13.10 >= (größer oder gleich)

Gibt `T` zurück, wenn das Argument numerisch größer als oder gleich dem Argument rechts von ihm ist, und gibt andernfalls `nil` zurück.

`(>=` *Zahl_Z_kette [Zahl_Z_kette]* `...)`

Jedes Argument *`Zahl_Z_kette`* kann eine Zahl oder eine Zeichenkette enthalten.

```
(> 120 17)               ergibt  T
(>= "c" "c")             ergibt  T
(> 3.5 1792)             ergibt  nil
(>= 77 4 4)              ergibt  T
(>= 77 4 9)              ergibt  nil
```

13.11 ~(Bitweise NOT)

Gibt die bitweise Negation (1er-Komplement) des Arguments zurück.

`(~` *Ganzzahl*`)

```
(~ 3)                    ergibt  4
(~ 100)                  ergibt  101
(~ -4)                   ergibt   3
```

13.12 (Inkrement)

Gibt das Argument um 1 erhöht zurück.

`(1+` *Zahl*`)

```
(1+ 5)                   ergibt   6
(1+ -17.5)               ergibt  -16.5
```

13.13 (Dekrement)

Gibt das Argument um 1 verringert zurück.

(1- ***Zahl*****)**

`(1- 5)`	*ergibt*	`4`
`(1- -17.5)`	*ergibt*	`-18.5`

13.14 abs

Gibt den Absolutwert des Argumentes zurück.

(abs ***Zahl*****)**

`(abs 100)`	*ergibt*	`100`
`(abs -100)`	*ergibt*	`100`
`(abs -99.25)`	*ergibt*	`99.25`

13.15 acad_colordlg

Öffnet das AutoCAD-Standarddialogfeld für die Farbauswahl.

(acad_colordlg ***Farbnum [flag]*****)**

Das Argument *Farbnum* ist eine Ganzzahl zwischen 0 und 256 (einschließlich). Es gibt die AutoCAD-Farbnummer an, die als Vorgabewert angezeigt werden soll. Der Wert 0 für *Farbnum* entspricht der Vorgabe VONBLOCK, und der Wert 256 entspricht der Vorgabe VONLAYER. Durch Einstellung des fakultativen Argumentes *Flag* auf `nil` werden die Schaltflächen VONLAYER und VONBLOCK deaktiviert. Wird das Argument *Flag* weggelassen oder auf einen von nil verschiedenen Wert gesetzt, werden die Schaltflächen VONLAYER und VONBLOCK aktiviert.

Die Funktion **acad_colordlg** gibt die Farbnummer zurück, die vom Benutzer eingestellt wurde. Wenn der Benutzer das Dialogfeld verläßt, gibt **acad_colordlg** den Wert `nil` zurück.

Der folgende Code fordert den Benutzer auf, eine Farbe einzugeben, und stellt die vorgegebene Farbe auf Grün:

```
(acad_colordlg 3)
```

Extern definierte Funktion *acadapp* ARX-Anwendung

13.16 acad_helpdlg

Aktiviert die Hilfefunktion (veraltet).

(acad_helpdlg ***Hilfedatei Thema*****)**

Diese extern definierte Funktion wurde durch die integrierte Funktion **help** ersetzt. Sie dient dazu, die Kompatibilität mit früheren AutoCAD Releases zu erhalten. Eine ausführliche Beschreibung dieser Funktion finden Sie unter **"help"**.

Extern definierte Funktion *acadr14.lsp* AutoLISP-Funktion

13.17 acad_strlsort

Sortiert eine Liste von Zeichenketten alphabetisch.

(acad _strlsort ***Liste*****)**

Das Argument *Liste* legt die zu sortierende Liste von Zeichenketten fest. Die Funktion **acad_strlsort** gibt eine alphabetisch sortierte Liste derselben Zeichenketten aus. Wenn das Argument *Liste* nicht gültig ist oder wenn für das Sortieren nicht genügend Speicher zur Verfügung steht, Gibt **acad_strlsort** den Wert `nil` zurück.

Der folgende Code sortiert die Liste mit abgekürzten Monatsnamen:

```
(setq mon '("Jan" "Feb" "Mär" "Apr" "Mai" "Jun"
            "Jul" "Aug" "Sep" "Okt" "Nov" "Dez"))
(acad_strlsort mon)
```

Zurückgegeben wird folgende Liste:

```
("Apr" "Aug" "Dez" "Feb" "Jan" "Jul"
 "Jun" "Mär" "Mai" "Nov" "Okt" "Sep")
```

13.18 action_tile

Weist einer bestimmten Komponente in einem Dialogfeld eine auszuführende Aktion zu.

(action_tile ***Schlüssel Operationsausdruck*****)**

Die Argumente *Schlüssel* und *Operationsausdruck* sind Zeichenketten. Das Argument *Schlüssel* ist der Name der Dialogfeldkomponente, welche die Operation auslöst (als Attribut von Taste). Beim Argument *Schlüssel* wird zwischen Groß- und Kleinschreibung unterschieden. Das Argument *Operationsausdruck* wird ausgeführt, wenn die Komponente ausgewählt wird.

Die Operation, die mit **action_tile** zugewiesen wird, ersetzt die Vorgabewerte des Dialogfelds (mit Hilfe von **new_dialog** zugewiesen), bzw. das Attribut Operation der Dialogfeldkomponente, falls angegeben. Der Ausdruck kann sich auf den aktuellen Wert der Dialogfeldkomponente beziehen (auf ihr Attribut Wert) als $value, auf ihren Namen als $key, auf ihre anwendungsspezifischen Daten (eingestellt mit **client_data_tile**) als $data, ihre Rückmeldungs-Ursache als $reason und auf ihre Bildkoordinaten (wenn das Feld ein Bild-Schaltfeld ist) als $x und $y.

Anmerkung Sie können die AutoLISP- **Befehl**funktion nicht aus der **action_tile**funktion heraus aufrufen.

13.19 add_list

Fügt der aktuellen Dialogfeldliste eine Zeichenkette hinzu oder modifiziert diese.

(add_list ***Z_kette*****)**

Bevor Sie **add_list** anwenden, müssen Sie die Liste öffnen und mit einem Aufruf von **start_list** initialisieren. Abhängig von der Operation, die in **start_list** festgelegt ist, wird *Z_kette* entweder zur aktuellen Liste hinzugefügt oder ersetzt das aktuelle Listenobjekt.

Angenommen, die aktuelle DCL-Datei hat eine Popup-Liste oder ein Listenfeld mit einer Taste aus longlist, dann wird das folgende Codefragment die Liste initialisieren und die Text-Zeichenketten in llist hinzufügen.

```
(setq llist '("erste Zeile" "zweite Zeile" "dritte Zeile"))
(start_list "longlist")
(mapcar 'add_list llist)
(end_list)
```

Nach der Definition der Liste ändert das folgende Codefragment den Text in der zweiten Zeile in "2. Zeile".

```
(start_list "longlist" 1 0)
(add_list "2. Zeile")
(end_list)
```

13.20 ads

Gibt eine Liste der aktuell geladenen ADS-Anwendungen zurück.

(ads)

Jede Anwendung und ihr Pfad ist eine Zeichenkette in Anführungszeichen in der Liste.

(ads) *könnte ergeben:* ("files/progs/PROG1" "PROG2")

13.21 alert

Öffnet ein Warnfenster mit einer Fehler- oder Warnmeldung, die als Zeichenkette übergeben wird.

(alert ***Z_kette*****)**

Ein Warnfenster ist ein Dialogfeld mit einer Schaltfläche OK.

```
(alert "Diese Funktion steht nicht zur Verfügung!")
```

Mit Hilfe des Zeilenumbruchzeichens in *Z_kette* können Sie auch mehrzeilige Meldungen ausgeben lassen.

```
(alert "Diese Funktion steht nicht zur Verfügung!")
```

Anmerkung Zeilenlänge und -anzahl in einem Warnfenster sind abhängig von Rechnerplattform, Ausgabegerät und Fenster. AutoCAD verkürzt zu lange Zeichenketten auf die passende Länge.

13.22 alloc

Setzt die Segmentgröße auf eine bestimmte Anzahl von Knoten.

(alloc *Ganzzahl*)

Diese Funktion gibt die vorherige Segmentgröße zurück.

13.23 and

Gibt das logische AND für eine Liste mit Ausdrücken zurück.

(and *Ausdruck* ...)

Wenn einer der Ausdrücke `nil` zurückgibt, beendet diese Funktion die weitere Auswertung und gibt `nil` zurück, andernfalls gibt sie `T` zurück.

Bei den folgende Zuweisungen:

```
(setq a 103 b nil c "Zeichenkette")
```

gilt:

`(and 1.4 a c)`	*ergibt*	`T`
`(and 1.4 a b c)`	*ergibt*	`nil`

13.24 angle

Gibt den Winkel einer Linie, die durch ihre beiden Endpunkte definiert ist, im Bogenmaß zurück.

(angle *P1 P2*)

Der Winkel wird von der *X*-Achse der aktuellen Konstruktionsebene im Bogenmaß gemessen, mit im Gegenuhrzeigersinn zunehmenden Winkelwerten. Werden 3D-Punkte angegeben, so werden diese auf die aktuelle Konstruktionsebene projiziert.

`(angle '(1.0 1.0) '(1.0 4.0))`	*ergibt*	`1.5708`
`(angle '(5.0 1.33) '(2.4 1.33)`	*ergibt*	`3.14159`

Siehe auch

"Winkelkonvertierung"

13.25 angtof

Konvertiert eine Zeichenkette, die einen Winkel darstellt, in eine reale (Gleitkomma-) Zahl im Bogenmaß.

(angtof *Z_kette [Modus]*)

Das Argument *Z_kette* beschreibt einen Winkel anhand des Formats, das vom Argument `Modus` vorgegeben ist. Das Argument `Modus` definiert die Einheiten, in der die Zeichenkette formatiert ist, wie in der folgenden Tabelle gezeigt. Der Wert sollte sich in dem Bereich bewegen, der für die AutoCAD-Systemvariable AUNITS festgelegt ist. Wenn `Modus` weggelassen wird, verwendet **`angtof`** den aktuellen Wert von AUNITS.

Werte für Winkeleinheiten

Moduswert	Zeichenkettenformat
0	Grad
1	Grad/Minuten/Sekunden
2	Neugrad
3	Bogenmaß
4	Feldmaß

Die Zeichenkette `Z_kette` muß eine Zeichenkette sein, die **angtof** dem angegebenen `Modus` entsprechend verarbeiten kann. Die Zeichenkette kann dabei in der Form angegeben werden, in der **angtos** das Ergebnis zurückgibt, oder aber in einem anderen, von AutoCAD für Tastatureingaben unterstützten Format.

Die Funktionen **angtof** und **angtos** sind komplementär: Falls Sie **angtof** eine Zeichenkette übergeben, die mit **angtos** ermittelt wurde, gibt **angtof** garantiert einen gültigen Wert zurück, und umgekehrt (unter der Voraussetzung, daß die Werte für `Modus` übereinstimmen).

Wenn **angtof** erfolgreich ausgeführt wird, gibt es eine reale Zahl im Bogenmaß zurück, andernfalls gibt es `nil` zurück.

13.26 angtos

Konvertiert einen Winkelwert im Bogenmaß in eine Zeichenkette.

(angtos ***Winkel [Modus [Genauigkeit]]*****)**

Die Funktion **angtos** nimmt den Winkel `Winkel` (eine reale Zahl im Bogenmaß) und gibt ihn in einer Zeichenkette zurück, die den Einstellungen in `Modus`, `Genauigkeit`, der AutoCAD-Systemvariablen DIMZIN und der Bemaßungsvariablen BEMTNZ entspricht. Die Argumente `Modus` und `Genauigkeit` sind Ganzzahlen, welche Modus und Genauigkeit der Winkeleinheiten angeben.

Werte für Winkeleinheiten

Moduswert	Zeichenkettenformat
0	Grad
1	Grad/Minuten/Sekunden
2	Neugrad
3	Bogenmaß
4	Feldmaß

Das Argument `Genauigkeit` ist eine Ganzzahl, mit deren Hilfe die Zahl der Kommastellen festgelegt wird. `Modus` und `Genauigkeit` entsprechen den AutoCAD-Systemvariablen AUNITS und AUPREC. Wenn Sie diese Argumente weglassen, verwendet **angtos** die aktuellen Einstellungen von AUNITS bzw. AUPREC.

Die Funktion **angtos** akzeptiert auch negative Werte für `Winkel`, wandelt diese jedoch immer in positive Werte zwischen Null und 2 pi im Bogenmaß um, bevor die entsprechende Konvertierung durchgeführt wird.

```
(angtos 0.785398 0 4)        ergibt  "45.0000"
(angtos 0.785398 0 4) ergibt  "315.0000"
```

Die Variable UNITMODE beeinflußt die zurückgegebene Zeichenkette, wenn Feldmaß-Einheiten verwendet werden (ein `Modus`-Wert von 4). Ist UNITMODE = 0, werden Leerzeichen in die Zeichenkette eingefügt (Beispiel: "N 45d E"); ist UNITMODE = 1, enthält die Zeichenkette keine Leerzeichen (Beispiel: "N45dE").

Anmerkung Routinen, welche die Funktion **angtos** zur Anzeige beliebiger Winkel verwenden, (die nicht relativ zum Wert von ANGBASE sind), müssen den Wert von ANGBASE überprüfen und berücksichtigen.

Siehe auch

"Zeichenkettenkonvertierung"

13.27 append

Fügt eine beliebige Zahl von Listen zu einer Liste zusammen.

(append ***Liste*** **...)**

`(append '(a b) '(c d))` *ergibt* `(A B C D)`
`(append '((a)(b)) '((c)(d)))` *ergibt* `((A)(B)(C)(D))`

13.28 apply

Gibt eine Argumentliste an eine bestimmte Funktion weiter.

(apply '*Funktion Liste*)

Die Funktion **apply** arbeitet sowohl mit eingebauten Funktionen (subrs) als auch mit vom Benutzer definierten Funktionen (die entweder mit **defun** oder **lambda** erzeugt wurden).

`(apply '+ '(1 2 3))` *ergibt* `6`
`(apply 'strcat '("a" "b" "c"))` *ergibt* `"abc"`

13.29 arx

Gibt eine Liste der aktuell geladenen ARX-Anwendungen zurück.

(arx)

Jede Anwendung und ihr Pfad ist eine Zeichenkette in Anführungszeichen in der Liste.

`(arx)` *könnte* `("files/progs/PROG1" "PROG2")` *ergeben*

13.30 arxload

Lädt eine ARX-Anwendung.

(arxload ***Anwendung [beiFehler]*****)**

Das Argument *Anwendung* wird als Zeichenkette in Anführungszeichen oder als Variable eingegeben, die den Namen einer ausführbaren Datei enthält. Während des Ladens der Datei wird überprüft, ob es sich um eine gültige ARX-Anwendung handelt.

Wenn die **arxload**-Operation scheitert, wird normalerweise ein AutoLISP-Fehler erzeugt. Ist jedoch das Argument *beiFehler* vorhanden, gibt **arxload** bei einem Fehler den Wert dieses Arguments anstatt einer Fehlermeldung zurück.

Wurde die Anwendung erfolgreich geladen, wird der Anwendungsname zurückgegeben.

`(arxload "/myapps/appx")` *gibt bei Erfolg* `"/myapps/appx"` *zurück*

Wenn Sie versuchen, eine bereits geladene Anwendung zu laden, zeigt **arxload** folgende Meldung an und gibt den Anwendungsnamen zurück.

Anwendung "*Anwendung*" bereits geladen.

Sie sollten die zur Zeit geladenen ARX-Anwendungen mit der Funktion **arx** überprüfen, bevor Sie **arxload** verwenden.

13.31 arxunload

Beendet eine ARX-Anwendung.

(arxunload ***Anwendung [beiFehler]*****)**

Ist die Anwendung erfolgreich beendet, wird der Anwendungsname zurückgegeben, andernfalls wird eine Fehlermeldung zurückgegeben.

Geben Sie *Anwendung* als Zeichenkette in Anführungszeichen ein oder als Variable, die den Namen einer mit **arxload** geladenen Anwendung enthält. Der Anwendungsname muß auf genau die gleiche Weise wie bei der Funktion

arxload eingegeben werden. Wenn für die Anwendung bei **arxload** ein Pfad (Verzeichnisname) oder eine Erweiterung eingegeben wurde, so kann dies bei der Funktion **arxunload** weggelassen werden.

Der folgende Code beendet beispielsweise die vorher mit der Funktion **arxload** geladene Anwendung appx:

```
(arxunload "appx")    gibt bei Erfolg "appx" zurück
```

Wenn die **arxunload**-Operation scheitert, wird normalerweise ein AutoLISP-Fehler erzeugt. Ist jedoch das Argument *beiFehler* vorhanden, gibt **arxunload** bei einem Fehler den Wert dieses Arguments anstatt einer Fehlermeldung zurück. Diese Eigenschaft von **arxunload** entspricht der der Funktion **arxload** .

13.32 ASCII

Gibt den ASCII-Wert des ersten Zeichens einer Zeichenkette als Ganzzahl zurück.

(ascii ***Z_kette*****)**

```
(ascii "A")          ergibt  65
(ascii "A")          ergibt  97
(ascii "BIG")        ergibt 66
```

Dies entspricht der Funktion **ASC** in der Programmiersprache BASIC® .

13.33 assoc

Durchsucht eine Assoziationsliste nach einem Element und gibt den entsprechenden Eintrag der Liste zurück.

(assoc ***Element A_liste*****)**

Durchsucht die Assoziationsliste *A_liste* nach *Element* als Schlüsselelement und gibt den *A_liste*-Eintrag zurück. Wenn **assoc** den Eintrag *Element* nicht als Schlüsselelement in *A_liste* findet, wird nil zurückgegeben.

Bei dem folgenden Codefragment:

```
(setq al '((namensfeld) (breite 3) (groesse 4.7263) (tiefe 5)))
```

gilt:

```
(assoc 'groesse al)          ergibt  (SIZE 4.7263)
(assoc 'gewicht al)          ergibt  nil
```

Assoziationslisten werden oft zum Speichern von Daten eingesetzt, die mit einer *Taste* erreicht werden können. Die Funktion **subst** stellt ein praktisches Mittel dar, den mit einer Taste verbundenen Wert in der Assoziationsliste zu ersetzen.

13.34 atan

Gibt den Arkustangens einer Zahl im Bogenmaß zurück.

(atan ***Zahl1 [Zahl2]*****)**

Geben Sie **atan** nur ein Argument an, gibt die Funktion den Arkustangens von *Zahl1* im Bogenmaß zurück. Wenn Sie beide Argumente *Zahl1* und *Zahl2* übergeben, gibt *atan* den Arkustangens von *Zahl1*/*Zahl2* im Bogenmaß zurück. Ist *Zahl2* Null, gibt die Funktion einen Winkel von plus oder minus 1.570796 im Bogenmaß (+90 Grad oder -90 Grad) zurück, abhängig vom Vorzeichen von *Zahl1*. Der Bereich der zurückgegebenen Winkel liegt zwischen -pi/2 und +pi/2 im Bogenmaß.

```
(atan -0.5)      ergibt  -0.463648
(atan -1.0)      ergibt  -0.785398
(atan -1.0)      ergibt  -0.785398
(atan 2.0 -3.0)        ergibt  0.588003
(atan 2.0 -3.0)        ergibt  2.55359
(atan 1.0 -0.0)        ergibt  1.5708
```

Anmerkung Die Funktion **angtos** konvertiert den Wert im Bogenmaß, den **atan** zurückgibt, in einen Zeichenkettenwert.

```
(angtos (atan -1.0) 0 4)          ergibt  "315.0000"
(angtos (atan 2.0 -3.0) 0 4)      ergibt  "33.6901"
```

```
(angtos (atan 2.0 -3.0) 0 4)      ergibt  "146.3099"
(angtos (atan 1.0 0.0) 0 4)       ergibt  "90.0000"
```

13.35 atof

Konvertiert eine Zeichenkette in eine reale Zahl.

(atof *string*)

```
(atof "97.1")                ergibt  97.1
(atof "3")                   ergibt  3.0
(atof "3.9")                 ergibt  3.9
```

13.36 atoi

Konvertiert eine Zeichenkette in eine Ganzzahl.

(atoi *Z_kette*)

```
(atoi "97")                  ergibt  97
(atoi "3")                   ergibt  3
(atoi "3.9")                 ergibt  3
```

13.37 atom

Überprüft, ob ein Element ein Atom ist.

(atom *Element*)

Gibt `nil` zurück, wenn *`Element`* eine Liste ist, und gibt andernfalls `T` zurück. Ist *`Element`* keine Liste, wird es als Atom betrachtet.

Bei den folgenden Zuweisungen

```
(setq a '(x y z))
(setq b 'a)
```

gilt

```
(atom 'a)        ergibt  T
(atom a)         ergibt  nil
(atom 'b)        ergibt  T
(atom b)         ergibt  T
(atom '(a b c)   ergibt  nil
```

Einige Versionen von LISP interpretieren die Funktion **`atom`** unterschiedlich; seien Sie also vorsichtig, wenn Sie konvertierten Code verwenden.

13.38 atoms-family

Gibt eine Liste der im Augenblick definierten Symbole zurück.

(atoms-family *Format [Sym_liste]*)

Das Argument *`Format`* kann nur die Werte 0 oder 1 annehmen. Ist der Wert von *`Format`* 0, dann gibt **`atoms-family`** die Symbolnamen als Liste zurück. Ist *`Format`* 1, werden die Symbolnamen als Liste von Zeichenketten zurückgegeben. Die Funktion **`atoms-family`** sucht nach einer bestimmten Liste von Symbolnamen, wenn Sie das Argument *`Sym_liste`* angeben. Das Argument *`Sym_liste`* ist eine Liste von Zeichenketten, welche die Symbolnamen enthalten. Die Funktion **`atoms-family`** gibt eine Liste zurück, deren Typ über das Format (Symbole oder Zeichenketten) festgelegt wird und welche die Namen der definierten Symbole enthält. Für die nicht definierten Symbole wird `nil` zurückgegeben.

`(atoms-family 0)` *gibt eine Liste der im Augenblick definierten Symbole zurück*

Der folgende Code überprüft, ob die Symbole `CAR`, `CDR` und `XYZ` definiert wurden, und gibt die Liste als Zeichenketten zurück:

```
(atoms-family 1)
```

`'("CAR" "CDR" "XYZ"))` *ergibt* `("CAR" "CDR" nil )`

Das Ergebnis der Funktion zeigt, daß das Symbol `XYZ` nicht definiert worden ist.

13.39 autoarxload

Definiert Befehlsnamen zum Laden einer zugehörigen ARX-Datei.

(autoarxload ***Dateiname Bef_liste*****)**

Das Argument *Dateiname* ist eine Zeichenkette, in der die *arx*-Datei angegeben wird, die geladen wird, wenn einer der durch das Argument Bef_liste definierten Befehle in der Eingabeaufforderung Befehl eingegeben wird. Das Argument Bef_liste muß eine Liste von Zeichenketten sein. Die Funktion **autoarxload** gibt `nil` zurück.

Mit dem folgenden Code werden die Funktionen `C:APP1`, `C:APP2` und `C:APP3` definiert, die die Datei *bounsapp.arx* laden. Wenn einer der Befehle APP1, APP2 oder APP3 das erste Mal in der Eingabeaufforderung Befehl eingegeben wird, lädt die ARX-Anwendung die Datei, und der Befehl wird fortgesetzt.

```
(autoarxload "BONUSAPP" '("APP1" "APP2" "APP3"))
```

Warnung Die im Argument *Bef_liste* aufgeführten Befehle müssen von der Datei, die durch *Dateiname* spezifiziert ist, als Befehle definiert werden.

Extern definierte Funktion *acadr14.lsp* AutoLISP-Funktion

13.40 autoload

Definiert Befehlsnamen zum Laden einer zugehörigen AutoLISP-Datei.

(autoload ***Dateiname Bef_liste*****)**

Das Argument *Dateiname* ist eine Zeichenkette, in der die *arx*-Datei angegeben wird, die geladen wird, wenn einer der durch das Argument *Bef_liste* definierten Befehle in der Eingabeaufforderung Befehl eingegeben wird. Das Argument *Bef_liste* muß eine Liste von Zeichenketten sein. Die Funktion **autoload** gibt `nil` zurück.

Mit dem folgenden Code wird für die Funktionen `C:APP1`, `C:APP2` und `C:APP3` die Datei *bounsapp.arx* geladen. Wenn einer der Befehle APP1, APP2 oder APP3 das erste Mal in der Eingabeaufforderung Befehl eingegeben wird, wird die AutoLISP-Datei geladen, und der Befehl wird fortgeführt.

```
(autoload "BONUSAPP" '("APP1" "APP2" "APP3"))
```

Warnung Die im Argument *Bef_liste* aufgeführten Befehle müssen von der Datei, die durch *Dateiname* spezifiziert ist, als Befehle definiert werden.

Extern definierte Funktion *acadr14.lsp* AutoLISP-Funktion

13.41 autoxload

Definiert Befehlsnamen zum Laden einer zugehörigen ADS-Anwendung

(autoxload ***Dateiname Bef_liste*****)**

Das Argument *Dateiname* ist eine Zeichenkette, welche die ADS-Anwendung angibt, die bei der Eingabe eines durch das Argument *Bef_liste* definierten Befehls an der Eingabeaufforderung Befehl geladen wird. Das Argument *Bef_liste* muß eine Liste von Zeichenketten sein. Die Funktion **autoxload** gibt `nil` zurück.

Mit dem folgenden Code wird festgelegt, daß die Funktionen `C:APP1`, `C:APP2` und `C:APP3` die ADS-Anwendung *bounsapp* laden sollen. Das erste Mal, wenn einer der Befehle APP1, APP2 oder APP3 an der Eingabeaufforderung Befehl eingegeben wird, wird die ADS-Anwendung geladen und der Befehl wird fortgesetzt.

```
(autoxload "BONUSAPP" '("APP1" "APP2" "APP3"))
```

Warnung Die im Argument *Bef_liste* aufgeführten Befehle müssen von der Datei, die durch *Dateiname* spezifiziert ist, als Befehle definiert werden.

Extern definierte Funktion *acadr14.lsp* AutoLISP-Funktion

13.42 Boole

Stellt die allgemeine bitweise Boolesche Funktion dar.

(Boole ***Funktion Ganzzahl1 Ganzzahl2*** **...)**

Das Argument *Funktion* ist ein ganzzahliger Wert zwischen 0 und 15, der eine der 16 möglichen Booleschen Funktionen für zwei Variablen darstellt. Nachfolgende ganzzahlige Argumente werden mit der angegebenen Funktion und unter Berücksichtigung der folgenden Wahrheitstabelle bitweise logisch verarbeitet.

Boolesche Wahrheitstabelle

Ganzzahl1	Ganzzahl2	Funktions-Bit
0	0	8
0	1	4
1	0	2
1	1	1

Jedes Bit von *Ganzzahl1* wird mit dem entsprechenden Bit von *Ganzzahl2* verknüpft, wodurch sich eine horizontale Zeile in der Wahrheitstabelle ergibt. Das sich ergebende Bit hat entweder den Wert 0 oder 1, abhängig vom Wert des Bits von *Funktion*, das mit der entsprechenden Zeile der Tabelle übereinstimmt.

Wenn das entsprechende Bit in *Funktion* gesetzt ist, ist das resultierende Bit 1; andernfalls ist das resultierende Bit 0. Einige der Werte für *Funktion* sind äquivalent zu den booleschen Standard-Operationen AND, OR, XOR und NOT.

Bitwerte der Booleschen Funktionen

Funktion	Operation	Resultierendes Bit ist 1, wenn...
1	AND	Beide Eingabe-Bits sind 1
6	XOR	Nur eines der beiden Eingabe-Bits ist 1
7	OR	Eines oder beide Eingabe-Bits sind 1
8	NOR	Beide Eingabe-Bits sind 0 (Komplement von 1)

Folgendes Beispiel zeigt eine logische AND-Operation für die Werte 12 und 5:

```
(Boole 1 12 5)          ergibt  4
```

In ähnlicher Weise führt das nächste Beispiel eine logische XOR-Operation für die Werte 6 und 5 durch:

```
(Boole 6 6 5)           ergibt  3
```

Sie können auch andere Werte für *Funktion* einsetzen, die andere Boolesche Operationen durchführen, für die keine Standardnamen existieren. Hat *Funktion* den Wert 4, werden die sich ergebenden Bits gesetzt, wenn die entsprechenden Bits in *Ganzzahl2*, nicht aber in *Ganzzahl1* gesetzt sind. Somit gilt:

```
(Boole 4 3 14)          ergibt  12
```

13.43 boundp

Überprüft, ob ein Wert an ein Symbol gebunden ist.

(boundp ***Symbol*****)**

Ergibt T, wenn *Symbol* über einen mit ihm verbundenen Wert verfügt. Wenn kein Wert mit *Symbol* verbunden ist (oder wenn es mit nil verbunden ist), gibt **boundp** den Wert nil zurück. Wenn *Symbol* ein nicht definiertes Symbol ist, wird es automatisch erzeugt und mit nil verbunden.

Bei den folgenden Zuweisungen:

```
(setq a 2 b nil)
```

gilt:

```
(boundp 'a)             ergibt  T
(boundp 'b)             ergibt  nil
```

Die Funktion **atoms-family** stellt eine alternative Methode zur Verfügung, mit deren Hilfe die Existenz eines Symbols überprüft werden kann, ohne daß dabei dieses Symbol automatisch erzeugt wird.

13.44 car und cdr

Gibt das erste Element einer Liste zurück oder eine Liste, die alle Elemente bis auf das erste enthält.

(car *Liste*) und (cdr *Liste*)

Wenn *Liste* leer ist, gibt **car** den Wert nil zurück.

```
(car '(a b c))  ergibt  A
(car '((a b) c))        ergibt  (A B)
(car '())               ergibt  nil
```

Wenn *Liste* leer ist, gibt **cdr** den Wert nil zurück.

```
(cdr '(a b c))  ergibt  (B C)
(cdr '((a b) c))        ergibt  (C)
(cdr '())       ergibt  nil
```

Wenn das Argument *Liste* ein punktiertes Paar ist (siehe **"cons"**), gibt **cdr** das zweite Element zurück, ohne es in eine Liste aufzunehmen.

```
(cdr '(a . b))  ergibt  B
(cdr '(1 . "Text"))     ergibt  "Text"
```

AutoLISP unterstützt Verkettungen von **car** und **cdr** bis zu einer Tiefe von vier Ebenen. Die folgenden Kombinationen stellen gültige Funktionen dar:

caaaar	**cadaar**	**cdaaar**	**cddaar**
caaadr	**cadadr**	**cdaadr**	**cddadr**
caaar	**cadar**	**cdaar**	**cddar**
caadar	**caddar**	**cdadar**	**cdddar**
caaddr	**cadddr**	**cdaddr**	**cddddr**
caadr	**caddr**	**cdadr**	**cdddr**
caar	**cadr**	**cdar**	**cddr**

Diese Verkettungen entsprechen verschachtelten Aufrufen von **car** und **cdr**. Jedes *a* entspricht einem Aufruf von **car**, und jedes *d* entspricht einem Aufruf von **cdr**. Beispiel:

```
(caar x)    ist äquivalent zu    (car (car x))
(cdar x)    ist äquivaletnt zu   (cdr (car x))
(cadar x)   ist äquivalent zu    (car (cdr (car x)))
(cadr x)    ist äquivalent zu    (car (cdr x))
(cddr x)    ist äquivalent zu    (cdr (cdr x))
(caddr x)   ist äquivalent zu    (car (cdr (cdr x)))
```

In AutoLISP wird **cadr** häufig eingesetzt, um die *Y*-Koordinate eines 2D- oder 3D-Punktes zu ermitteln (das zweite Element einer Liste aus zwei bzw. drei realen Zahlen). Auf ähnliche Weise kann **caddr** dazu benutzt werden, die *Z*-Koordinate eines 3D-Punktes zu bestimmen. Bei den folgenden Zuweisungen:

```
(setq pt2 '(5.25 1.0))        ein 2D-Punkt
(setq pt3 '(5.25 1.0 3.0))    ein 3D-Punkt
```

gilt:

```
(car pt2)       ergibt  5.25
(cadr pt2)      ergibt  1.0
(caddr pt2)     ergibt  nil
(car pt3) ergibt  5.25
(cadr pt3)      ergibt  1.0
(caddr pt3)     ergibt  3.0
```

13.45 chr

Gibt das Zeichen als Zeichenkette der Länge 1 zurück, das dem angegebenen ganzzahligen ASCII-Zeichencode entspricht.

(chr ***Ganzzahl*****)**

```
(chr 65)  ergibt  "A"
(chr 66)  ergibt  "B"
(chr 97)  ergibt  "a"
```

Diese Funktion ähnelt der Funktion **chr$** in der Programmiersprache BASIC.

13.46 client_data_tile

Verbindet Anwendungsdaten mit einer Dialogfeldkomponente.

(client_data_tile ***Schlüssel Client_daten*****)**

Das Argument Schlüssel ist eine Zeichenkette, die eine Komponente angibt. Beim Argument *Schlüssel* wird zwischen Groß- und Kleinschreibung unterschieden. Die Daten bestehen aus einer Zeichenkette, die im Argument *Client_daten* angegeben wird. Ein Operationsausdruck oder eine Rückmeldungsfunktion kann sich auf die Zeichenkette als `$data` beziehen.

13.47 close

Schließt eine geöffnete Datei.

(close ***Dateideskr*****)**

Das Argument *Dateideskr* ist ein Dateideskriptor, der mit Hilfe der Funktion **open** ermittelt wurde. Die Funktion **close** verändert den Dateideskriptor nicht, er ist jedoch nicht mehr gültig. Daten, die zu einer geöffneten Datei hinzugefügt werden, werden nicht eher geschrieben, bis die Datei geschlossen ist. Die Funktion **close** gibt `nil` zurück, wenn *Dateideskr* gültig ist, andernfalls wird eine Fehlermeldung zurückgegeben.

Der folgende Code zählt die Linien in der Datei *somefile.txt* und setzt die Variable `ct` auf diese Anzahl.

```
(setq fil "SOMEFILE.TXT")
(setq x (open fil "r") ct 0)
(while (read-line x)
  (setq ct (1+ ct))
)
(close x)
```

13.48 command

Führt einen AutoCAD-Befehl aus.

(command ***[Argumente]*** **...)**

Die *Argumente* stellen AutoCAD-Befehle und deren Optionen dar. Argumente für die Funktion **command** können aus Zeichenketten, realen Zahlen, Ganzzahlen oder Punkten bestehen, je nachdem, wie dies von der Eingabeaufforderung des

entsprechenden Befehls erwartet wird. Eine leere Zeichenkette (`" "`) entspricht dem Drücken von RETURN auf der Tastatur. Der Aufruf von **command** ohne Angabe eines Arguments entspricht der Eingabe von ESC, und bricht die meisten AutoCAD-Befehle ab. Die Funktion **command** gibt `nil` zurück.

Die Funktion **command** wertet jedes Argument aus und schickt es als Antwort auf einander folgende Eingabeaufforderungen an AutoCAD. Befehlsnamen und Optionen werden als Zeichenketten, 2D- und 3D-Punkte als Liste von zwei bzw. drei realen Zahlen übergeben. AutoCAD erkennt Befehlsnamen nur, wenn es die Eingabeaufforderung Befehl anbietet.

Anmerkung Wenn Sie die Funktion **command** in einer *acad.lsp*- oder MNL-Datei benutzen, dann sollte diese Funktion nur von einer **defun**-Anweisung aus aufgerufen werden. Verwenden Sie die Funktion **S::STARTUP**, um Befehle zu definieren, die direkt zu Anfang der Zeichnungssitzung aktiviert werden müssen.

Das folgende Beispiel setzt zwei Variablen `P1` und `P2` mit zwei Punktwerten 1,1 und 1,5 gleich. Es verwendet dann die Funktion **command**, um den Befehl LINIE auszuführen und die beiden Punktwerte zu übergeben.

```
(setq P1 '(1 1) P2 '(1 5))
(command "line" P1 P2 "")
```

Wenn Ihre Anwendungen mit fremdsprachlichen Versionen von AutoCAD verwendet werden sollen, muß Befehlsnamen ein Unterstrich vorangestellt werden (_), damit sie übersetzt werden können. Wenn Sie den Punkt-Präfix verwenden (um die Verwendung neudefinbierter Befehle zu vermeiden), können Sie den Punkt und den Unterstrich in beliebiger Reihenfolge anordnen, d.h. sowohl `"._line"` als auch `"_.line"` sind gültig.

Befehle, die mit Hilfe der Funktion **command** eingegeben werden, werden nicht in der Befehlszeile wiedergegeben, wenn die Systemvariable CMDECHO (durch **setvar** und **getvar** zu verändern) auf 0 gesetzt wurde.

Die Anwender-Eingabefunktionen ***getxxx*** (**getangle, getstring, getint, getpoint**, usw.) können in der Funktion **command** nicht benutzt werden. Wird dies versucht, so wird die aktuelle Funktion abgebrochen und die folgende Meldung erscheint:

Fehler: AutoCAD hat diese Funktion zurückgewiesen

Wenn eine Benutzereingabe erforderlich ist, geben Sie vorher die Funktionen ***getxxx*** aus, oder ordnen Sie sie zwischen aufeinanderfolgenden **command** -Funktionsaufrufen an.

Für AutoCAD-Befehle, für die die Auswahl eines Objekts erforderlich ist (wie z. B. die Befehle BRUCH und STUTZEN), können Sie eine Liste zur Verfügung stellen, die mit Hilfe von **entsel** erzeugt wurde, anstatt einen Punkt für die Auswahl des Objekts anzubieten. Beispiele finden Sie in "Übergeben von Auswahlpunkten an AutoCAD-Befehle."

Die AutoCAD-Befehle DTEXT und SKIZZE lesen die Tastatur und das Digitalisiergerät direkt und können daher nicht mit der AutoLISP-Funktion **command** benutzt werden. Wird der Befehl 'SCRIPT mit der Funktion **command** verwendet, sollte dies der letzte Funktionsaufruf in der AutoLISP-Routine sein.

Um jeden Befehl, der mit der Funktion **command** verwendet wird, wird eine UNDO-Gruppe explizit erzeugt. Wenn ein Benutzer U (oder UNDO) eingibt, nachdem eine AutoLISP-Routine ausgeführt wurde, wird nur der letzte Befehl rückgängig gemacht. Durch weitere Eingaben von UNDO werden die in dieser Routine verwendeten Befehle rückwärts durchlaufen. Wenn Sie möchten, daß eine Befehlsgruppe (oder die gesamte Routine) als Gruppe behandelt wird, verwenden Sie die Optionen UNDO Begin und UNDO End.

Der Gebrauch des Symbols PAUSE

Wenn ein AutoCAD-Befehl aktiv ist, und das vordefinierte Symbol `PAUSE` wird als Argument an die Funktion **command** übergeben, wird die Funktion **command** unterbrochen, um die direkte Benutzereingabe zuzulassen.

Das Symbol `PAUSE` ist eine Zeichenkette, die einen einzigen umgekehrten Schrägstrich enthält. Sie können den umgekehrten Schrägstrich auch anstelle des Symbols `PAUSE` verwenden. Wird die Funktion **command** von einer Menüoption aufgerufen, dann wird durch den umgekehrten Schrägstrich das Lesen der Menüoption unterbrochen. Der AutoLISP-Ausdruck wird dann nur teilweise bearbeitet. Des weiteren kann es sein, daß der Pausenmechanismus in zukünftigen Versionen von AutoLISP durch einen anderen Wert ausgelöst wird. Es wird daher empfohlen, immer das Symbol `PAUSE` anstelle des umgekehrten Schrägstrichs zu verwenden.

Wird der umgekehrte Schrägstrich (\) in einer Zeichenkette verwendet, muß ihm ein weiterer umgekehrter Schrägstrich vorangestellt werden (\\).

Tritt `PAUSE` auf, wenn ein Befehl die Eingabe einer Textzeichenkette oder eines Attributwertes erwartet, dann wartet AutoCAD nur auf die Eingabe, wenn die Systemvariable TEXTEVAL nicht auf Null gesetzt ist. Andernfalls wartet

AutoCAD nicht auf die Benutzereingabe, sondern setzt das Symbol `PAUSE` (ein einzelner umgekehrter Schrägstrich) als Text ein.

Während die Funktion **command** auf Benutzereingaben wartet, wird diese Funktion als aktiv angesehen und der Benutzer kann keinen weiteren AutoLISP-Ausdruck zur Bearbeitung eingeben.

Das folgende Beispiel verdeutlicht den Gebrauch des Symbols `PAUSE` (der Layer NEU_LAY und der Block MEIN_BLOCK müssen vor dem Testen dieses Codes in der Zeichnung vorhanden sein):

```
(setq blk "MEIN_BLOCK")
(setq old_lay (getvar "clayer"))
(command "layer" "set" "NEU_LAY" "")
(command "insert" blk pause "" "" pause)
(command "layer" "set" old_lay "")
```

Das vorangehende Codefragment setzt den aktuellen Layer auf NEU_LAY, wartet auf die Benutzerauswahl eines Einfügepunkts für den Block MEIN_BLOCK (der mit *X*- und *Y*-Skalierungsfaktoren mit dem Wert 1 eingefügt wird) und wartet nochmals, um dem Benutzer die Auswahl eines Drehwinkels zu ermöglichen. Der aktuelle Layer wird dann auf den ursprünglichen zurückgesetzt.

Wenn die Funktion **command** eine `PAUSE` für den Befehl SELECT angibt, und es ist ein Satz PICKFIRST aktiv, wird mit dem Befehl SELECT der Satz PICKFIRST geladen, ohne auf die Benutzereingabe zu warten.

Warnung Die Unterbefehle Radius und Durchmesser der Eingabeaufforderung BEM verlangen unter gewissen Umständen nach zusätzlichen Eingaben. Dies kann zum Abbruch von AutoLISP-Programmen führen, die vor Release 11 geschrieben wurden und diese Befehle verwenden.

Siehe auch

Weitere Informationen über die Funktion **command** finden Sie im Abschnitt "Befehlsübergabe."

13.49 cond

Ist die wichtigste Bedingungsfunktion von AutoLISP.

(cond (*Test1 Ergebnis1* ...) ...)

Die Funktion **cond** akzeptiert eine beliebige Zahl von Listen als Argumente. Das erste Element jeder Liste wird (in der gegebenen Reihenfolge) bearbeitet, bis eines dieser Elemente einen andern Wert als `nil` zurückgibt. Anschließend werden die Ausdrücke des folgenden Tests bearbeitet, und der Wert des letzten Ausdrucks der Subliste wird zurückgegeben. Enthält die Subliste nur einen Ausdruck (wenn *`Ergebnis`* also fehlt) wird der Wert des Ausdrucks *`Test`* zurückgegeben.

Im folgenden Beispiel wird **cond** dazu eingesetzt, die Berechnung eines absoluten Werts durchzuführen:

```
(cond
  ((minusp a) (- a))
  (t a)
)
```

Wird die Variable `a` auf den Wert -10 gesetzt, ist das Ergebnis 10.

Wie gezeigt, kann **cond** als *case* Typ-Funktion verwendet werden. Im allgemeinen wird `T` als der letzte (Vorgabe-) *`test`*-Ausdruck verwendet. Betrachten wir ein weiteres einfaches Beispiel. Für eine gegebene, vom Benutzer als Antwort eingegebene Zeichenkette in der Variablen `s` testet diese Funktion die Antwort und gibt eine 1 zurück, wenn sie `Y` oder `y` ist, eine 0, wenn sie `N` oder `n` ist, und `nil` andernfalls.

```
(cond
  ((= s "Y") 1)
  ((= s "y") 1)
  ((= s "N") 0)
  ((= s "n") 0)
  (t nil)
)
```

13.50 cons

Die Grundfunktion zur Erstellung von Listen.

(cons neues_erstes_Element *Liste*)

Die Funktion **cons** nimmt ein Element (*neues_erstes_Element*) und eine *Liste*, und gibt das Ergebnis der Addition dieses Elementes zum Anfang der Liste zurück. Das erste Element kann ein Atom oder eine Liste sein.

```
(cons '(a) '(b c d)) ergibt ((A) B C D)
(cons '(a) '(b c d)) ergibt ((A) B C D)
```

Die Funktion **cons** akzeptiert auch ein Atom anstelle des Arguments *Liste* und erzeugt eine Struktur, die *punktiertes Paar* genannt wird. Bei der Darstellung eines punktierten Paars fügt AutoLISP einen Punkt zwischen dem ersten und zweiten Element ein. Mit Hilfe der Funktion **cdr** können Sie das zweite Atom eines punktierten Paars ermitteln.

```
(cons 'a 2)        ergibt (A . 2)
(car (cons 'a 2))  ergibt A
(cdr (cons 'a 2))  ergibt 2
```

Ein punktiertes Paar ist eine besondere Listenart, die nicht von allen Funktionen, die normale Listen verarbeiten, als Argument akzeptiert wird.

13.51 cos

Gibt den Kosinus eines im Bogenmaß ausgedrückten Winkels zurück.

(cos *Win*)

```
(cos 0.0)   ergibt 1.0
(cos pi)    eigibt -1.0
```

13.52 cvunit

Konvertiert einen Wert von einer Maßeinheit in eine andere.

(cvunit *Wert von in*)

Das Argument *Wert* ist der numerische Wert, den Sie konvertieren möchten. Es kann auch eine Liste mit zwei oder drei zu konvertierenden Zahlen sein (ein 2D- oder 3D-Punkt). Das Argument *von* ist die aktuelle Einheit des Werts, und *in* ist die Einheit, in die der Wert konvertiert werden soll. Die Argumente *von* und in können jeden Einheiten-Typ bezeichnen, der in der Datei *acad.unt* verzeichnet ist.

Nach erfolgreicher Operation gibt **cvunit** den konvertierten Wert zurück. Ist eine der Einheitenbezeichnungen unbekannt (der Name kann nicht in der Datei *acad.unt* gefunden werden), oder sind die Einheiten nicht kompatibel (wenn z. B. Gramm in Jahre konvertiert werden sollen), dann ist das Ergebnis von **cvunit** nil.

```
(cvunit 1 "minute" "second" ergibt 60.0
(cvunit 1 "gallon" "furlong"    ergibt nil
(cvunit 1.0 "inch" "cm")        ergibt 2.54
(cvunit 1.0 "acre" "sq yard")   ergibt 4840.0
(cvunit '(1.0 2.5) "ft" "in")   ergibt (12.0 30.0)
(cvunit '(1 2 3) "ft" "in") ergibt (12.0 24.0 36.0)
```

Anmerkung Wenn Sie mehrere Werte in der gleichen Weise umrechnen möchten, sollten Sie zuerst den Wert 1.0 umrechnen und den erhaltenen Wert als Skalierfaktor in Ihrer Funktion oder Berechnung einsetzen. Dies funktioniert bei allen vordefinierten Einheiten außer bei Temperaturen, weil dort feste Werte addiert bzw. subtrahiert werden müssen.

Siehe auch

"Konvertierung von Einheiten"

13.53 defun

Definiert eine Funktion.

(defun ***Symbol Argument-Liste Ausdruck*** **...)**

Die Funktion **defun** definiert eine Funktion mit dem Namen *Symbol* (der Funktionsname wird automatisch in Apostrophe gesetzt). Dem Funktionsnamen folgt eine Liste von Argumenten (die leer sein kann), wahlweise gefolgt von einem Schrägstrich und den Namen eines oder mehrerer lokaler Symbole für die Funktion. Der Schrägstrich muß von dem ersten lokalen Symbol und dem letzten Argument (falls vorhanden) jeweils durch mindestens ein Leerzeichen abgetrennt sein. Falls Sie keine Argumente oder lokale Symbole definieren, müssen Sie zwei Klammern () hinter den Funktionsnamen setzen.

Die folgenden Beispiele für *Argument-Liste* zeigen einige gültige und ungültige Werte:

`(defun meinfunkt (x y) ...)`	*Funktion nimmt zwei Argumente*
`(defun meinfunkt (/ a b) ...)`	*Funktion hat zwei lokale Symbole*
`(defun meinfunkt (x / temp) ...)`	*Ein Argument, ein lokales Symbol*
`(defun meinfunkt () ...)`	*Weder Argumente noch lokale Symbole*

Sie dürfen in einer Funktionsdefinition nicht mehreren Argumenten den gleichen Namen geben. Sie können jedoch eine lokale Variable als Argument definieren, die den gleichen Namen wie eine andere lokale Variable oder ein Argument hat:

`(defun fubar (a a / b) ...)`	*Ungültig*
`(defun fubar (a b / a a b) ...)`	*Gültig, aber sinnlos*

Enthält die Argument-/Symbolliste Doppeleinträge, dann wird der Name bei seinem ersten Auftreten verwendet und alle folgenden werden ignoriert.

Beim Aufruf der Funktion werden ein oder mehrere Ausdrücke berechnet, die der Argument-/Symbolliste folgen.

Die Funktion **defun** gibt den Namen der definierten Funktion zurück. Wird die definierte Funktion aufgerufen, dann werden die Argumente ausgewertet und mit den Argumentsymbolen verknüpft. Dabei können Sie die lokalen Symbole innerhalb der Funktion verwenden, ohne deren Verbindungen in Ebenen außerhalb der Funktion zu verändern. Die Funktion gibt das Ergebnis des zuletzt berechneten Ausdrucks zurück; alle vorher bearbeiteten Ausdrücke haben nur einen Nebeneffekt.

Die folgenden Beispiele zeigen die Definition neuer Funktion mit Hilfe von **defun** und die von den neuen Funktionen zurückgegebenen Werte:

```
(defun hinzufüg10 (x)
  (+ 10 x)
)     gibt HINZUFÜG10 zurück
(hinzufüg10 5)              gibt 15 zurück
(hinzufüg10 -7.4)                gibt 2.6 zurück
```

and

```
(defun Punkte (x y / temp)
  (setq temp (strcat x "..."))
  (strcat temp y)
)     gibt DOTS zurück
(Punkte "a" "b")                       gibt "a...b" zurück
(Punkte "von" "nach")            gibt "von...nach" zurück
```

Warnung Verwenden Sie niemals den Namen einer eingebauten Funktion oder eines eingebauten Symbols als *Symbol*. Sie können sonst nicht mehr auf diese Funktion zugreifen. Um eine Liste der eingebauten und vordefinierten Funktionen zu erhalten, schlagen Sie unter **"atoms-family".** nach.

Siehe auch

"Symbol- und Funktionsbearbeitung"

13.54 dictadd

Fügt dem angegebenen Wörterbuch ein nicht-grafisches Objekt hinzu.

(dictadd ***Elem_name Symbol NeuesObjekt*****)**

Fügt das Objekt *NeuesObjekt* dem Wörterbuch hinzu. Das Argument ist der Schlüsselbegriff des Objektes, das zum Wörterbuch hinzugefügt werdensoll; muß ein eindeutiger Name sein, der noch nicht im Wörterbuch vorhanden ist. Das durch spezifizierte Objekt enspricht nur einem nicht-graphischen Objekt.

Als allgemeine Regel gilt, daß jedes zu einem Wörterbuch hinzugefügte Objekt in diesem Wörterbuch nur einmal vorhanden sein darf. Dies ist insbesondere ein Problem, wenn Gruppen-Objekte zum Gruppen-Wörterbuch hinzugefügt werden. Wird dasselbe Gruppen-Objekt mit verschiedenen Schlüsselbegriffen hinzugefügt, ergeben sich doppelte Gruppen-Namen, wodurch die Funktion in eine Endlosschleife geraten kann.

13.55 dictnext

Findet das nächste Element in einem Wörterbuch.

(dictnext ***Elem_name [Erstes]*****)**

Das Argument *Elem_name* ist ein Elementname, der das zu suchende Wörterbuchobjekt festlegt.

Wenn **dictnext** wiederholt verwendet wird, gibt es jedes Mal den jeweils nächsten Eintrag im festgelegten Wörterbuch zurück. Die Funktion **dictsearch** legt den aufzurufenden Eintrag fest. Wenn das Argument *Erstes* vorhanden ist und einen Wert ergibt, der nicht `nil` ist, wird im Wörterbuch zurückgeblättert und der erste Eintrag aufgerufen. Befinden sich keine weiteren Einträge im Wörterbuch, ist das Ergebnis `nil`. Gelöschte Wörterbucheinträge werden niemals zurückgegeben.

Informationen zum Elementnamen des leitenden Wörterbuchs finden Sie unter **"namedobjdict"**.

Anmerkung Wenn Sie einmal begonnen haben, den Inhalt eines Wörterbuchs zu durchlaufen, geht die Position im ursprünglichen Wörterbuch verloren, wenn Sie der Funktion **dictnext** einen anderen Wörterbuchnamen übergeben. Es ist mit anderen Worten nur ein globaler Durchlauf zur Verwendung mit dieser Funktion möglich.

Wird ein Eintrag gefunden, wird er als eine Liste punktierter Paare zurückgegeben, die aus DXF-Codes und -Werten bestehen.

13.56 dictremove

Entfernt einen Eintrag aus dem angegebenen Wörterbuch.

(dictremove ***Elem_name Symbol*****)**

Entfernt den Eintrag, der mit *Symbol* angegeben ist, aus dem Wörterbuch, das mit *Elem_name* angegeben ist.

Wenn *Elem_name* ungültig ist oder *Symbol* nicht gefunden wird, gibt **dictremove** den Wert `nil` zurück. Bei erfolgreicher Durchführung gibt **dictremove** den Elementnamen des entfernten Eintrags zurück.

Vorgabemäßig wird durch Entfernen eines Eintrages aus einem Wörterbuch der Eintrag nicht aus der Datenbank gelöscht. Dies muß durch Aufruf der Funktion **entdel** erfolgen. Ausnahmen zu dieser Regel sind zur Zeit Gruppen und Mlinienstile. Der Code, mit dem diese Eigenschaften realisiert werden, erfordert, daß die Datenbank und diese Wörterbücher auf dem aktuellen Stand sein müssen, und daher wird das Element automatisch gelöscht, wenn es aus dem Wörterbuch entfernt wird (mit **dictremove**).

Die Funktion **dictremove** erlaubt es nicht, einen Mlinienstil aus dem Mlinienstil-Wörterbuch zu entfernen, wenn hierauf eine aktive Referenz durch eine Mlinie in der Datenbank besteht.

13.57 dictrename

Benennt einen Eintrag im Wörterbuch um.

(dictrename ***Elem_name AltesSymbol NeuesSymbol*****)**

Führt eine Umbenennung eines Schlüsselbegriffs eines Eintrages im Wörterbuch von *AltesSymbol* auf *NeuesSymbol* durch. Das Wörterbuch wird mit *Elem_name* angegeben.

Wenn *AlterName* im Wörterbuch nicht vorhanden ist, *Elem_name* ungültig ist, *NeuerName* ungültig ist oder *NeuerName* bereits im Wörterbuch vorhanden ist, gibt **dictrename** den Wert `nil` zurück.

13.58 dictsearch

Durchsucht ein Wörterbuch nach einem Element.

(dictsearch ***Elem_name Symbol [Setzfolg]*****)**

Das Argument *Elem_name* ist ein Elementname, der das zu suchende Wörterbuchobjekt festlegt. Das Argument *Symbol* ist eine Zeichenkette, die das Element innerhalb des Wörterbuchs festlegt.

Findet **dictsearch** einen Eintrag für das gegebene Element, gibt es das Element in dem für **dictnext** beschriebenen Format zurück. Wird kein derartiger Eintrag gefunden, ist das Ergebnis nil.

Normalerweise hat **dictsearch** keinen Einfluß auf die Reihenfolge der Einträge, die von **dictnext** aufgerufen werden. Wenn jedoch **dictsearch** erfolgreich durchgeführt wird, und das Argument *Setzfolg* vorhanden und nicht nil ist, wird der Eintragszähler von **dictnext** so eingestellt, daß der folgende Aufruf von **dictnext den Eintrag zurückgibt, der nach dem durch diesen Aufruf von dictsearch gegebenen Eintrag liegt.**

Im folgenden Beispiel wird **dictsearch** zum Auffinden der Definitionsliste der G2-Gruppe verwendet (dieser Code geht davon aus, daß eine Gruppe mit dem Namen G2 in der aktuellen Zeichnung vorhanden ist).

```
(setq grp (dictsearch (namedobjdict) "ACAD_GROUP"))
(setq g2 (dictsearch (cdr (assoc -1 grp)) "G2"))
```

Informationen zum Elementnamen des leitenden Wörterbuchs finden Sie unter **"namedobjdict"**.

13.59 dimx_tile und dimy_tile

Ermittelt die Abmessungen einer bestimmten Dialogfeldkomponente.

(dimx_tile Schlüssel) und (dimy_tile Schlüssel)

Die Funktion **dimx_tile** gibt die Breite einer Dialogfeldkomponente zurück, während **dimy_tile** ihre Höhe ermittelt. Bei beiden Funktionen ist das Argument Schlüssel eine Zeichenkette, welche die Bezeichnung der Komponente enthält. Das Argument Schlüssel unterscheidet zwischen Groß- und Kleinschreibung.

Die zurückgegebenen Koordinaten sind die innerhalb der Komponente maximal erlaubten; da die Koordinaten mit Null beginnen, liefern diese Funktionen einen Wert kleiner als die gesamte *X*- oder *Y*-Abmessung (*X*-1 und *Y*-1). Die Funktionen **dimx_tile** und **dimy_tile** werden zur Verwendung mit **vector_image**, **fill_image** und **slide_image** bereitgestellt, bei denen es erforderlich ist, daß Sie absolute Dialogfeldkomponenten angeben.

13.60 distance

Gibt den 3D-Abstand zwischen zwei Punkten zurück.

(distance ***P1 P2*****)**

```
(distance '(1.0 2.5 3.0) '(7.7 2.5 3.0))   gibt  6.7  zurück
(distance '(1.0 2.0 0.5) '(3.0 4.0 0.5))   gibt  2.82843  zurück
```

Sind einer oder beide der angegebenen Punkte 2D-Punkte, ignoriert **distance** die *Z*-Koordinate vom angegebenen 3D-Punkt und gibt die 2D-Entfernung zwischen den Punkten zurück, wie sie auf die aktuelle Konstruktionsebene projiziert werden.

Siehe auch

"Geometrische Funktionen"

13.61 distof

Konvertiert eine Zeichenkette, die eine reale (Gleitkomma-) Zahl darstellt, in einen realen Wert.

(distof ***Z_kette [Modus]*****)**

Das Argument *Modus* definiert die Einheiten, in der die Zeichenkette formatiert ist. Der Wert sollte in dem Bereich liegen, der für die AutoCAD-Systemvariable LUNITS erlaubt ist, wie in der folgenden Tabelle gezeigt ist. Wenn *Modus* weggelassen wird, verwendet **distof** den aktuellen Wert von LUNITS.

Werte für lineare Einheiten

Moduswert	Zeichenkettenformat
1	Exponential

2	Dezimalschreibweise
3	Engineering (Fuß und Zoll in Dezimalschreibweise)
4	Architectural (Fuß und Zoll in Bruchschreibweise)
5	Bruchschreibweise

Das Argument `Z_kette` muß eine Zeichenkette sein, die **distof** korrekt auf den angegebenen `Modus` analysieren kann. Die Zeichenkette kann dabei in der Form angegeben werden, in der **rtos** das Ergebnis zurückgibt, oder aber in einem anderen, von AutoCAD für Tastatureingaben unterstützten Format. Die Funktionen **distof** und **rtos** sind komplementär. Falls Sie **distof** eine Zeichenkette übergeben, die mit **rtos** ermittelt wurde, gibt **distof** garantiert einen gültigen Wert zurück, und umgekehrt (unter der Voraussetzung, daß die Werte für Modus übereinstimmen).

Anmerkung Die Funktion **distof** behandelt die Modi 3 und 4 gleich. Wenn `Modus` also als Einheit 3 (engineering) oder 4 (architectural) angibt, und `Z_kette` in einem dieser Formate angegeben wird, gibt **distof** den korrekten rellen Wert zurück.

Wenn **distof** erfolgreich ausgeführt wird, gibt es eine reale Zahl zurück, andernfalls gibt es `nil` zurück.

13.62 done_dialog

Schließt ein Dialogfeld.

(done_dialog ***[Status]*****)**

Die Funktion **done_dialog** darf nur aus einem Operationsausdruck oder einer Rückmeldungsfunktion heraus aufgerufen werden (siehe auch **"action_tile"**).

Wenn Sie das fakultative Argument `Status` angeben, muß es als positive Ganzzahl angegeben werden, die **start_dialog** zurückgibt, anstelle von `1` für OK oder `0` für Abbrechen. Die Bedeutung von Werten für `Status`, die größer als 1 sind, ist von Ihrer Anwendung abhängig.

Die Funktion **done_dialog** gibt die Position des Dialogfelds beim Verlassen als Liste eines zweidimensionalen Punkts (*X*,*Y*) zurück. Sie können diesen Punkt bei einem nachfolgenden Aufruf von **new_dialog** übergeben, um das Dialogfeld an der vom Benutzer gewählten Stelle wieder zu öffnen.

Anmerkung Wenn Sie eine Rückmeldefunktion für das Schaltfeld bereitstellen, dessen Schlüssel `"accept"` oder `"cancel"` ist (normalerweise die Schaltfelder OK und Abbrechen), muß die Rückmeldefunktion **done_dialog** explizit aufrufen. Ist dies nicht der Fall, kann der Benutzer das Dialogfeld unter Umständen nicht mehr verlassen. Falls Sie für diese Schaltflächen keine explizite Rückmeldung vereinbaren und die Standardschaltflächen zum Verlassen verwenden, verarbeitet AutoCAD diese automatisch. Außerdem muß eine explizite AutoLISP-Aktion für die Schaltfläche `"accept"` einen `Status` von 1 spezifizieren (oder einen durch die Anwendung definierten Wert); andernfalls gibt **start_dialog** den Vorgabewert 0 zurück, durch den es so erscheint, als ob das Dialogfeld abgebrochen wurde.

13.63 end_image

Beendet die Erstellung des aktuellen Dialogfeldbilds.

(end_image)

Diese Funktion stellt das Komplement zu **start_image** dar.

13.64 end_list

Beendet die Verarbeitung der aktuellen Dialogfeldliste.

(end_list)

Diese Funktion stellt das Komplement zu **start_list** dar.

13.65 entdel

Löscht Objekte (Elemente) oder macht das Löschen eines zuvor gelöschten Objekts wieder rückgängig.

(entdel ***Elem_name*****)**

Das vom Attribut *Elem_name* bezeichnete Element wird gelöscht, falls es sich gegenwärtig in der Zeichnung befindet. Die Funktion **entdel** macht das Löschen des Elements wieder rückgängig (fügt es also wieder in die Zeichnung ein), wenn es zuvor in der aktuellen Bearbeitungssitzung gelöscht wurde. Gelöschte Elemente werden endgültig aus der Zeichnung gelöscht, wenn die Bearbeitung dieser Zeichnung beendet wird. Die Funktion **entdel** kann sowohl grafische als auch nichtgrafische Elemente löschen.

```
(setq e1 (entnext))    Weist e1 den Namen des ersten
                       Elements der Zeichnung zu
(entdel e1)            ;Löscht Element e1
(entdel e1)            ;Macht das Löschen von Element e1 rückgängig.
```

Die Funktion **entdel** bearbeitet nur Hauptelemente. Attribute und Polylinien-Kontrollpunkte können nicht unabhängig von ihren übergeordneten Elementen gelöscht werden. Sie können die Funktion **command** verwenden, um die Befehle ATTEDIT oder PEDIT auszuführen, um untergeordnete Elemente zu ändern.

Innerhalb einer Blockdefinition ist es nicht möglich, Elemente zu löschen. Sie können jedoch den gesamten Block mit Hilfe von **entmake** neu definieren und dabei das zu löschende Element ignorieren.

13.66 entget

Ermittelt die Definitionsdaten eines Objekts (Elements).

(entget ***Elem_name [Anw_liste]*****)**

Die Funktion **entget** gibt eine Liste mit den Definitionsdaten des Elements *Elem_name* zurück. Dies gilt sowohl für grafische als auch für nichtgrafische Elemente. Falls Sie auch *Anw_liste* (eine optionale Liste von registrierten Anwendungsnamen) angeben, gibt **entget** auch die erweiterten Elementdaten zurück, die mit den angegebenen Anwendungen verbunden sind.

Die Daten, die von **entget** zurückgegeben werden, sind in Form einer Assoziationsliste kodiert, aus der Sie mit Hilfe von **assoc** Einträge herausziehen können. Den Objekten in der Liste sind AutoCAD DXF-Gruppencodes für jeden Teil der Elementdaten zugeordnet.

Nehmen wir an, das zuletzt gezeichnete Objekt in der Zeichnung ist eine Linie von Punkt (1,2) zu Punkt (6,5). Sie können den Namen des letzten Objektes mit der Funktion **entlast** feststellen und diesen Namen an **entget** weitergeben.

```
(entget (entlast))
```

Die zurückgegebene Liste könnte so aussehen:

```
((-1 . <Elementname: 60000014>)   Elementname
  (0 . "LINE")                    Object type
  (8 . "0")                       Layer
  (10 1.0 2.0 0.0)                Start point
  (11 6.0 6.0 0.0)                Endpoint
)
```

Die DXF-Gruppencodes, die von AutoLISP verwendet werden, weichen leicht von den Gruppencodes einer DXF-Datei ab. Die AutoLISP DXF-Gruppencodes sind dokumentiert in appendix C, "DXF-Gruppencodes".

Wie bei DXF werden die Element-Header-Einträge (Farbe, Linientyp, Objekthöhe, Attribut-Folgen-Flag und Elementreferenz) nur dann exportiert, wenn sie von den Vorgabewerten abweichen. Anders als bei DXF werden optionale Elementdefinitionsfelder exportiert, unabhängig davon, ob sie den Vorgabewerten entsprechen oder nicht. Dies

vereinfacht die Verarbeitung. Programme können so für allgemeine Algorithmen immer davon ausgehen, daß diese Felder vorhanden sind. Ein weiterer Unterschied zu DXF liegt in der Darstellung assoziierter Koordinaten, da *X*, *Y* und *Z* in einer Punktliste zusammengefaßt werden (10 1.0 2.0 3.0) und nicht in separaten Gruppen der Form 10, 20 und 30 erscheinen.

Das Element -1 am Anfang der Liste enthält den Namen dieses Elements. Die einzelnen punktierten Paare, welche die Werte darstellen, können mit **assoc** extrahiert werden, wobei **cdr** verwendet wird, um deren Werte zu ermitteln.

Die Sublisten für Punkte sind keine punktierten Paare wie die anderen. Es gilt die Konvention, daß der **cdr**-Wert der Subliste den Gruppenwert darstellt. Weil ein Punkt durch eine Liste von zwei (oder drei) realen Zahlen dargestellt wird, besteht die gesamte Liste aus drei (bzw. vier) Elementen. Der **cdr**-Wert der Gruppe ist die Liste, die den Punkt darstellt, so daß die Konvention erfüllt ist, daß **cdr** immer den Wert der Liste zurückgibt.

Wenn Sie Funktionen zur Verarbeitung der Elementlisten schreiben, stellen Sie sicher, daß sie unabhängig von der Reihenfolge der Sublisten sind. Verwenden Sie **assoc**, um dies zu garantieren. Die -1 - Gruppe, die den Elementnamen enthält, erlaubt es, daß Änderungsoperationen die Elementliste akzeptieren und vermeidet es, daß der Elementname in einer parallelen Struktur gehalten werden muß. Ein Sequenzendelement am Ende einer Polylinie oder ein Satz von Attributen enthält eine -2 - Gruppe, deren **cdr**-Wert der Elementname des Headers dieses Elementes ist. Hierdurch kann der Header in einem Subelement gefunden werden, indem vorwärts zum Sequenzendelement gegangen, und dann **cdr** der -2 - Gruppe als Elementname verwendet wird, um das zugehörige Haupt-Element wiederzufinden.

Warnung Bevor **entget** auf Kontrollpunkte angewendet wird, sollten Sie den Header des Polylinienelements schreiben oder lesen. Gehört der Kontrollpunkt nicht zum zuletzt bearbeiteten Polylinienelement, können Informationen zur Weite (Gruppen 40 und 41) verlorengehen.

Alle mit einem Objekt verbundenen Punkte werden in dem Objekt-Koordinatensystem (OCS) des Objektes ausgedrückt. Für Punkt, Linie, 3DLine, 3DFläche, 3DPolyLinie, 3DNetz und Bemaßungsobjekte entspricht das OCS dem WKS (Die Objekt-Punkte sind Punkte im Weltkoordinatensystem). Für alle anderen Objekte kann das OCS aus dem WKS und der Extrusionsrichtung des Objektes (seiner 210-Gruppe) abgeleitet werden. Bei der Arbeit mit Objekten, die unter Verwendung von Koordinatensystemen gezeichnet wurden, die nicht dem WKS entsprechen, kann es sein, daß Sie die Punkte mit der Funktion trans in das WKS oder in das aktuelle BKS konvertieren müssen.

13.67 entlast

Gibt den Namen des zuletzt erzeugten, nicht gelöschten Hauptobjekts (Elements) in der Zeichnung zurück.

(entlast)

Die Funktion **entlast** wird häufig dazu verwendet, den Namen eines neuen Elements zu ermitteln, das gerade mit Hilfe von **command** zur Zeichnung hinzugefügt wurde. Zu diesem Zweck muß sich das Element nicht auf dem Bildschirm oder auf einem getauten Layer befinden.

`(setq e1 (entlast))`	*Setzt* `e1` *auf den Namen des letzten Haupt-Elements der Zeichnung*
`(setq e2 (entnext e1))`	*Setzt* `e2` *auf nil (oder auf ein Attribut oder einen Kontrollpunkt-Subelement-Namen)*

Wenn Ihre Anwendung den Namen des letzten nicht gelöschten Elementes benötigt, (Hauptelement *oder* Subelement), definieren Sie anstelle von **entlast** eine Funktion, wie den folgenden Aufruf.

```
(defun lastent (/ a b)
  (if (setq a (entlast))          Ruft das letzte Hauptelement ab
    (while (setq b (entnext a))   Wenn Subelemente folgen, Schleife, bis keine weiteren
      (setq a b)                  Subelemente vorhanden
    )
  )
  a                               Rückgabe des letzten Hauptelementes
)                                 oder Subelements
```

13.68 entmake

Erzeugt ein neues Element (grafisches Objekt) in der Zeichnung.

(entmake ***[Elem_liste]*****)**

Das Argument *Elem_liste* muß eine Liste von Elementdefinitionsdaten sein, deren Format dem von der Funktion **entget** zurückgegebenen entspricht. Das Argument *Elem_liste* muß dabei die gesamte Information enthalten, die zur Definition des Elements benötigt wird. Die Funktion **entmake** kann sowohl grafische als auch nichtgrafische Elemente definieren. Wenn erforderliche Definitionsdaten weggelassen werden, gibt **entmake** den Wert nil zurück und das Objekt wird zurückgewiesen. Werden optionale Informationen (wie zum Beispiel der Layer) nicht gefunden, dann verwendet **entmake** den Vorgabewert.

Erzeugt **entmake** erfolgreich ein neues Element, so gibt die Funktion eine Liste der Definitionsdaten des Elements zurück. Wenn **entmake** nicht in der Lage ist, das Element zu erzeugen, gibt es nil zurück.

Eine Methode zur Erzeugung eines neuen Elements besteht darin, die Definitionsdaten eines Elements mit der Funktion **entget** zu lesen, diese zu verändern und die geänderten Daten anschließend **entmake** zu übergeben. Vor der Erzeugung eines neuen Elements stellt **entmake** sicher, daß gültige Namen für Layer, Linientyp und Farbe vergeben wurden. Wird ein neuer Layername angegeben, so wird dieser Layer von **entmake** automatisch erzeugt. Die Funktion **entmake** überprüft auch Block-, Bemaßungs-, Textstil- und Symbolnamen, wenn dies durch den Elementtyp erforderlich wird.

Der Elementtyp (zum Beispiel KREIS oder LINIE) muß das erste oder zweite Feld von *Elem_liste* sein. Steht er an zweiter Stelle, so darf nur der Elementname vorausgehen. Dies entspricht dem von **entget** zurückgegebenen Format. In diesen Fällen wird der Elementname ignoriert, wenn das neue Element erzeugt wird. Enthält *Elem_liste* eine Objektreferenz, so wird diese ebenfalls ignoriert.

Der folgende Code erzeugt einen roten Kreis mit Radius 1 um den Mittelpunkt (4,4). Die optionalen Angaben über Layer und Linientyp wurden nicht angegeben und nehmen daher die Vorgabewerte an.

```
(entmake
  '((0 . "CIRCLE")          Element-Typ
    (62 . 1)                Farb-
    (10 4.0 4.0 0.0)        Mittelpunkt
    (40 . 1.0)              Radius
  )
)
```

Anmerkung Objekte, die auf gefrorenen Layern erzeugt werden, werden erst wieder regeneriert, wenn der entsprechende Layer getaut wird.

Komplexe Elemente

Ein komplexes Element (eine Blockdefinition, eine Polylinie oder eine Blockreferenz, die Attribute enthält) kann durch mehrere Aufrufe von **entmake** erzeugt werden, bei denen die entsprechenden Subelemente (Attribute oder Kontrollpunkte) definiert werden. Stellt **entmake** fest, daß ein komplexes Element erstellt wird, dann erzeugt die Funktion eine temporäre Datei, um die Definitionsdaten aufzunehmen. Für jedes **entmake** wird eine Überprüfung durchgeführt, ob die temporäre Datei vorhanden ist, und falls dies so ist, werden die neuen Daten zur Datei hinzugefügt. Ist die Definition des komplexen Elements abgeschlossen (durch Anfügen des entsprechenden Seqend- oder Endblk-Elements), werden die komplexen Daten nochmals überprüft, und das komplexe Element wird der Zeichnung hinzugefügt. Die Beendigung einer Blockdefinition ((**entmake** eines Endblk) gibt den Blocknamen und nicht die Elementdatenliste zurück, die normalerweise zurückgegeben wird.

Anmerkung Sie können mit **entmake** keine Ansichtsfensterobjekte erzeugen.

Werden Daten empfangen, die dem Elementtyp nicht entsprechen, wird weder das Element noch das gesamte komplexe Element akzeptiert. Eine Blockdefinition kann nicht verschachtelt sein und auch nicht auf sich selbst verweisen. Allerdings kann sie Verweise auf andere Blockdefinitionen enthalten. Sämtliche Elemente eines komplexen Elements können entweder nur im Modell- oder nur im Papierbereich existieren, nicht aber in beiden Bereichen.

Ein Code der Gruppe 66 wird nur für Einfügeobjekte berücksichtigt (was soviel wie *Attribute folgen* bedeutet). Für Polylinienelemente wird für den Gruppencode 66 ein Wert von 1 erzwungen (*Kontrollpunkte folgen*), für alle anderen Elemente wird der Vorgabewert 0 eingesetzt. Das einzige Element, was auf ein Polylinienelement folgen kann, ist ein Kontrollelement.

Kein Teil eines komplexen Elements wird in Ihre Zeichnung eingefügt, solange die Definition nicht vollständig ist. Um die Erzeugung eines komplexen Elements abzubrechen, rufen Sie **entmake** ohne Angabe von Argumenten auf. Hierdurch wird die temporäre Datei gelöscht und `nil` zurückgegeben.

Die Elemente block und endblk können zum Erzeugen einer neuen Block-Definition herangezogen werden. Neue Blöcke werden automatisch in die Symboltabelle eingetragen, wo auf sie verwiesen werden kann.

Anwendungen können Polygone mit einer beliebig großen Anzahl an Flächen als Vielflächennetze darstellen. Die Elementstruktur von AutoCAD begrenzt die Zahl der Kontrollpunkte, die ein beliebiges Flächenelement enthalten kann. Sie können komplexere Polygone darstellen, indem Sie diese in dreieckige Keile aufteilen. AutoCAD stellt dreieckige Keile als Flächen mit vier Kontrollpunkten dar, wobei zwei nebeneinanderliegende Kontrollpunkte denselben Wert besitzen. Die Kanten sollten als unsichtbar definiert werden, damit keine sichtbaren Artefakte dieser Unterteilung gezeichnet werden. Mit dem Befehl PNETZ wird diese Unterteilung automatisch vorgenommen, doch wenn Anwendungen Polyflächen-Netze direkt erstellen, muß diese von den Anwendungen selbst durchgeführt werden.

Die Anzahl der Scheitelpunkte pro Fläche ist in dieser Unterteilung der Schlüsselparameter. Die Systemvariable PFACEVMAX stellt einer Anwendung die Zahl der Kontrollpunkte pro Flächenelement zur Verfügung. Diese Systemvariable hat den Wert 4 und ist schreibgeschützt.

Warnung Wenn **entmake** einen Block erzeugt, kann es einen vorhandenen Block überschreiben. Die Funktion **entmake** führt keine Überprüfung auf Namens-Konflikte in der Block-Definitionstabelle durch, verwenden Sie daher vor ihrem Einsatz die Funktion **tblsearch** , um sicherzustellen, daß der Name des neuen Blockes nur einmal vorhanden ist. Allerdings kann der Einsatz von **entmake** bei der erneuten Definition von unbenannten Blöcken hilfreich sein, die im nächsten Abschnitt beschrieben wird.

Unbenannte Blöcke

Die Blockdefinitionstabelle in einer Zeichnung kann unbenannte Blöcke enthalten. Unbenannte Blöcke werden erzeugt, um Schraffurmuster und Assoziativbemaßung zu unterstützen. Sie können auch durch **entmake** für eigene Zwecke der Anwendung erzeugt werden und enthalten im allgemeinen Elemente, die der Benutzer nicht direkt manipulieren kann.

Der Gruppencode 2 (Blockname) einer Bemaßungseinheit ist für die Funktion **entmake** optional. Wird der Blockname aus der Elementdefinitionsliste ausgelassen, erstellt AutoCAD einen neuen. Ansonsten erstellt AutoCAD die Bemaßung unter Verwendung des gegebenen Namens.

Der Name (Gruppe 2) eines unbekannten Blocks ist `*U`*nnn*, wobei *nnn* eine von AutoCAD erzeugte Nummer ist. Außerdem wird das niederwertige Bit des *Blocktyp Flags* (Gruppe 70) eines unbekannten Blocks auf 1 gesetzt. Wenn **entmake** einen Block erzeugt, dessen Name mit einem Sternchen (`*`) beginnt und dessen Unbenannt-Bit gesetzt ist, behandelt AutoCAD diesen Block als unbenannten Block und weist ihm einen Namen zu. Zeichen hinter dem * in der an **entmake** übergebenen Namenszeichenkette werden ignoriert. Nach Erzeugen des Blocks gibt **entmake** dessen Namen zurück. Erzeugen Sie einen Block durch mehrfachen Aufruf von **entmake**, so wird der Name nach folgendem erfolgreichen Aufruf zurückgegeben:

```
(entmake "endblk")
```

Bei jedem Öffnen einer Zeichnung werden alle unbenannten Blöcke, auf die nicht verwiesen wird, aus der Blockdefinitionstabelle gelöscht. Unbenannte (eingefügte) Blöcke, auf die verwiesen wird, werden nicht gelöscht. Sie können **entmake** dazu benutzen, eine Blockreferenz (Einfügen) auf einen unbenannten Block zu erzeugen (Sie können jedoch keinen unbenannten Block an den Befehl EINFÜGEN übergeben). Sie können den Block mit **entmake** auch neu definieren. Die Elemente des Blocks (nicht aber das Blockelement selbst) können mit Hilfe von **entmod** modifiziert werden.

Anmerkung Obwohl ein unbenannter Block mit Referenz permanent wird, kann sich der aus Zahlen bestehende Teil seines Namens von Sitzung zu Sitzung ändern. Anwendungen können nicht davon ausgehen, daß die Namen unbenannter Blöcke konstant bleiben.

13.69 entmakex

Erstellt ein neues Objekt oder Element, versieht es mit einer Referenz und einem Elementnamen (ohne einen Eigentümer zuzuweisen) und gibt den neuen Namen des Elements zurück.

(entmakex ***[Elem_liste]***)

Das Argument `Elem_liste` muß eine Liste von Elementdefinitionsdaten sein, deren Format dem von der Funktion **entget** zurückgegebenen entspricht. Das Argument `Elem_liste` muß dabei die gesamte Information enthalten, die zur Definition des Elements oder Objektes benötigt wird. Die Funktion **entmakex** kann sowohl grafische als auch nichtgrafische Objekte definieren. Fehlen benötigte Definitionsdaten, gibt **entmakex** als Ergebnis `nil` zurück, und das

Objekt wird nicht akzeptiert. Werden optionale Informationen (wie zum Beispiel der Layer) weggelassen, dann verwendet **entmakex** den Vorgabewert.

Erstellt **entmakex** erfolgreich ein neues Element, gibt es den Elementnamen zurück. Wenn **entmakex** nicht in der Lage ist, das Element zu erzeugen, gibt es `nil` zurück.

Warnung Objekte und Elemente ohne Eigentümer werden *nicht* in *.dwg*- oder *.dxf*-Dateien geschrieben. Gehen Sie sicher, daß Sie nach der Verwendung von **entmakex** einen Eigentümer festlegen. Sie können beispielsweise mit Hilfe von **dictadd** ein Wörterbuch für den Besitz eines Objekts einrichten.

13.70 entmod

Modifiziert die Definitionsdaten eines Objekts (Elements).

(entmod ***Elem_liste*****)**

Die Funktion **entmod** verarbeitet eine Liste (*`Elem_liste`*) in dem Format, das **entget** zurückgibt, und sie aktualisiert die Datenbankinformationen für das Element, dessen Name in der Gruppe -1 von *`Elem_liste`* enthalten ist. Der Hauptmechanismus, mit dem AutoLISP die Datenbank aktualisiert, ist das Ermitteln der Elemente mit **entget**, dann wird die Definitionsliste des Elements verändert, und anschließend wird das Element in der Datenbank mit Hilfe von **entmod** aktualisiert. Die Funktion **entmod** kann sowohl grafische als auch nichtgrafische Objekte ändern.

```
(setq en (entnext))     Setzt en auf den Namen des ersten Elementes in der Zeichnung
(setq ed (entget en))   Setzt ed auf die Elementdaten des Elementes en
(setq ed
  (subst (cons 8 "0")
    (assoc 8 ed)        Ändert die Layer-Gruppe in ed auf Layer 0
    ed
  )
)
(entmod ed)         Ändert den Layer des Elementes en in der Zeichnung
```

Die Modifikationsmöglichkeiten von **entmod** unterliegen einigen Einschränkungen. So können weder Typ noch Referenz eines Elements verändert werden. Wenn Sie dies durchführen wollen, führen Sie einfach **entdel** aus, und erstellen Sie mit den Funktionen **command** oder **entmake** ein neues Element. Vor Ausführung von **entmod** muß sichergestellt sein, daß AutoCAD alle Objekte bekannt sind, auf die in der Elementliste verwiesen wird. Also müssen auch Textstil, Linientyp, Symbol und Blockname in einer Zeichnung definiert sein, bevor diese in einer Elementliste von **entmod** verarbeitet werden können. Eine Ausnahme bilden die Layernamen: **entmod** erzeugt einen neuen Layer mit den Vorgabewerten, die von der Option Neu des Befehls LAYER verwendet werden, wenn ein zuvor undefinierter Layer in einer Elementliste benannt wird.

Für Elementfelder, die Gleitkommawerte enthalten (z. B. Objekthöhe), erlaubt **entmod** auch ganzzahlige Werte und konvertiert diese in Fließkommzahlen. In gleicher Weise konvertiert **entmod** eine Gleitkommazahl, die für ein Ganzzahl-Elementfeld (z. B. Farbnummer) eingegeben wurde, in eine Ganzzahl.

Die Funktion **entmod** überprüft die übergebene Liste auf Konsistenz. Wenn ein schwerer Fehler erkannt wird, wird die Datenbank nicht aktualisiert, und es wird `nil` zurückgegeben. Andernfalls gibt **entmod** die an die Funktion als Argument übergebene Liste zurück. Die Funktion **entmod** ist nicht in der Lage, interne Felder, wie zum Beispiel den Elementnamen in der Gruppe -2 eines Seqend-Elements zu verändern. Versuche, diese Felder zu verändern, werden ignoriert.

Wenn **entmod** ein Hauptelement aktualisiert, verändert die Funktion das Element und aktualisiert die Bildschirmdarstellung (einschließlich der Subelemente). Aktualisiert **entmod** ein Subelement (einen Polylinienkontrollpunkt oder ein Blockattribut), wird das Subelement in der Datenbank aktualisiert, die Zeichnung auf dem Bildschirm jedoch nicht. Nachdem alle gewünschten Änderungen an den Subelementen eines beliebigen Elements erfolgt sind, kann die Zeichnung auf dem Bildschirm mit Hilfe der Funktion **entupd** neu gezeichnet werden.

Anmerkung Sie können mit der Funktion **entmod** keine Ansichtsfensterelemente modifizieren; die Sichtbarkeit eines Bereichs von einem Element kann jedoch auf 0 oder 1 gesetzt werden (ausgenommen für Ansichtsfensterobjekte). Verwenden Sie **entmod**, um ein Element innerhalb einer Blockdefinition zu verändern, so betrifft diese Änderung alle Stellen, an denen der Block in der Zeichnung erscheint.

Bevor **entmod** auf Kontrollpunkte angewendet wird, sollten Sie den Header des Polylinienelements schreiben oder lesen. Gehört der Kontrollpunkt nicht zum zuletzt bearbeiteten Polylinienelement, können Informationen zur Weite (Gruppen 40 und 41) verlorengehen.

Warnung Sie können **entmod** dazu verwenden, Elemente innerhalb einer Blockdefinition zu modifizieren, dies kann jedoch dazu führen, daß Sie einen Block erzeugen, der auf sich selbst verweist. In diesem Fall hält AutoCAD das System an.

13.71 entnext

Gibt den Namen des nächsten Objekts (Elements) in der Zeichnung zurück.

(entnext ***[Elem_name]*****)**

Wird die Funktion **entnext** ohne Angabe von Argumenten aufgerufen, gibt sie den Namen des ersten nicht gelöschten Elements in der Datenbank zurück. Wird **entnext** mit einem Elementnamen als Argument *Elem_name* aufgerufen, gibt sie den Namen des ersten nicht gelöschten Elements zurück, das auf *Elem_name* in der Datenbank folgt. Falls kein weiteres Element in der Datenbank gefunden wird, ist das Ergebnis `nil`. Die Funktion **entnext** gibt sowohl die Namen von Haupt- als auch von Subelementen zurück.

Die mit Hilfe der Funktion **ssget** ausgewählten Elemente sind Hauptelemente, keine Blockattribute oder Polylinienkontrollpunkte. Sie erhalten Zugang zur internen Struktur dieser komplexen Elemente, wenn Sie die Subelemente mit **entnext** durchlaufen. Haben Sie den Namen des gewünschten Subelements gefunden, können Sie es wie jedes andere Element manipulieren. Das übergeordnete Element zu einem mit **entnext** gefundenen Subelement finden Sie, indem Sie mit **entnext** bis zu einem Seqenzendelement weitergehen und dann die Gruppe -2 aus diesem Element herausziehen, die den Namen des Hauptelements enthält.

```
(setq en (entnext))          Setzt el auf den Namen des ersten Elementes in der Zeichnung
(setq e2 (entnext e1))  Setzt e2 auf den Namen des Elements;  wie bereits e1
```

13.72 entsel

Fordert den Benutzer zur Auswahl eines einzelnen Objekts (Elements) durch Bestimmung eines Punkts auf.

(entsel ***[Anfrage]*****)**

Die Funktion **entsel** gibt eine Liste zurück, deren erstes Element der Elementname des ausgewählten Objekts ist und deren zweites Element die Koordinaten des Auswahlpunkts (in Bezug auf das aktuelle BKS) darstellt. Eine für *Anfrage* angegebene Zeichenkette dient dazu, den Benutzer nach dem Objekt zu fragen. Andernfalls verwendet das System den vorgegebenen Text Objekte auswählen.

Anmerkung Der von **entsel** zurückgegebene Auswahlpunkt repräsentiert *keinen* Punkt, der auf dem ausgewählten Objekt liegt. Der zurückgegebene Punkt ist die Position des Fadenkreuzes zum Zeitpunkt der Auswahl. Der Zusammenhang zwischen dem Auswahlpunkt und dem Objekt ist abhängig von der Größe der Pickbox und dem aktuellen Zoom-Maßstab.

Eine Liste der Form, wie sie von **entsel** zurückgegeben wird, kann AutoCAD als Antwort auf jede der Eingabeaufforderungen zur Objektwahl eingegeben werden. Die Liste wird von AutoCAD genauso behandelt, als wäre das Objekt am angegebenen Punkt mit einem Zeigegerät ausgewählt worden.

Die folgende AutoCAD-Befehlssequenz verdeutlicht den Einsatz der Funktion **entsel** und den der zurückgegebenen Liste:

Befehl: **Linie**
Von Punkt: **1,1**
Zu Punkt: **6,6**
Zu Punkt: ENTER
Befehl: **(setq e (entsel "Bitte wählen Sie ein Objekt: "))**
Bitte wählen Sie ein Objekt: **3,3**
(<Elementname: 60000014> (3.0 3.0 0.0))

In einigen Fällen kann es bei der Bearbeitung von Objekten erforderlich werden, daß Sie gleichzeitig ein Objekt auswählen und die Koordinaten des Auswahlpunkts bestimmen möchten. Beispiele hierfür in AutoCAD können beim Objektfang gefunden werden, sowie in den Befehlen BRUCH, STUTZEN und DEHNEN. Die Funktion **entsel** erlaubt es AutoLISP-Programmen, diese Operation durchzuführen. Dabei wird ein einzelnes Objekt durch einen auszuwählenden Punkt bestimmt. Die aktuelle Einstellung von Ofang wird von der Funktion ignoriert (kein Objektfang), wenn Sie sie während der Ausführung der Funktion nicht ausdrücklich anfordern. Die Funktion **entsel** erkennt Schlüsselworte an, die von einem vorherigen Aufruf von **initget** stammen.

13.73 entupd

Aktualisiert die Bildschirmdarstellung eines Objekts (Elements).

(entupd ***Elem_name*****)**

Wird der Kontrollpunkt einer Polylinie oder ein Blockattribut mit Hilfe von **entmod** verändert, erfolgt keine Aktualisierung des komplexen Elements auf dem Bildschirm. Die Funktion **entupd** kann dazu verwendet werden, eine Aktualisierung einer veränderten Polylinie oder eines veränderten Blocks auf dem Bildschirm auszulösen. Diese Funktion kann mit dem Elementnamen eines beliebigen Teils der Polylinie oder des Blocks aufgerufen werden, es muß nicht das Hauptelement sein. Obwohl **entupd** in erster Linie für die Bearbeitung von Polylinien und Blöcken mit Attributen gedacht ist, kann die Funktion für jedes beliebige Objekt aufgerufen werden. Dabei wird immer das gesamte Element einschließlich aller Subelemente auf dem Bildschirm aktualisiert.

Anmerkung Wird **entupd** auf ein verschachteltes Element (ein Element innerhalb eines Blocks) oder auf einen Block, der verschachtelte Elemente enthält, angewendet, kann es vorkommen, daß einige Elemente nicht regeneriert werden. Um eine vollständige Regeneration zu gewährleisten, setzen Sie den Befehl REGEN ein.

Nehmen wir an, das erste Element einer Zeichnung sei eine Polylinie mit mehreren Kontrollpunkten, dann gilt

```
(setq e1 (entnext))      Setzt e1 auf den Namen des Elementes der Polylinie
(setq e2 (entnext e1))        Setzt e2 auf den ersten Kontrollpunkt
(setq ed (entget e2)) Setzt ed auf die Kontrollpunkt-Daten
(setq ed
  (subst '(10 1.0 2.0)
    (assoc 10 ed)        Ändert die Lage des Kontrollpunktes in ed
    ed                        auf den Punkt (1,2)
  )
)
(entmod ed)                   Bewegt den Kontrollpunkt in der Zeichnung
(entupd e1)       Regeneriert das Polylinien-Element e1
```

13.74 eq

Stellt fest, ob zwei Ausdrücke identisch sind.

(eq ***Ausdr1 Ausdr2*****)**

Die Funktion *eq* bestimmt, ob *Ausdr1* und *Ausdr2* mit demselben Objekt verbunden sind (zum Beispiel durch **setq**). Gibt T zurück, wenn die beiden Ausdrücke identisch sind, und gibt andernfalls nil zurück.

Sie können diese Funktion auch verwenden, um festzustellen, ob zwei Listen identisch sind. Bei den folgende Zuweisungen:

```
(setq f1 '(a b c))
(setq f2 '(a b c))
(setq f3 f2)
```

gilt:

```
(eq f1 f3)     ergibt   nil, f1 und f3 sind nicht dieselben Listen
(eq f3 f2)     ergibt   T, f3 und f2 sind exakt dieselbe Liste
```

Siehe auch

Die Funktionen **=** (equal to) und **equal**

13.75 equal

Stellt fest, ob zwei Ausdrücke gleich sind.

(equal ***Ausdr1 Ausdr2 [ungenau]*****)**

Die Funktion **equal** bestimmt, ob *Ausdr1* und *Ausdr2* dasselbe Ergebnis haben. Werden zwei reale Zahlen miteinander verglichen (oder zwei Listen mit realen Zahlen, wie bei Punkten), können zwei *identische* Zahlen leicht voneinander abweichen, wenn unterschiedliche Berechnungsmethoden angewandt wurden. Aus diesem Grund kann im dritten Argument, *ungenau*, die maximale Abweichung angegeben werden, bei der die beiden Ausdrücke *Ausdr1* und *Ausdr2* noch als gleich angesehen werden.

Bei den folgende Zuweisungen:

```
(setq f1 '(a b c))
(setq f2 '(a b c))
(setq f3 f2)
(setq a 1.123456)
(setq b 1.123457)
```

gilt:

`(equal f1 f3)`	*ergibt*	`T`
`(equal f3 f2)`	*ergibt*	`T`
`(equal a b)`	*ergibt*	`nil`
`(equal a b 0.000001)`	*ergibt*	`T`

Obwohl eine von **`equal`** festgestellte Gleichheit zweier Listen mit Hilfe von **`eq`** nicht immer bestätigt werden kann, werden Atome, die **`equal`** als gleich einstuft, auch von **`eq`** so bewertet. Umgekehrt gilt, daß eine mit **`eq`** festgestellte Identität zweier Listen oder Atome von der Funktion **`equal`** auch als gleich bewertet wird.

Siehe auch

Die Funktionen **`= (ist gleich)`** und **`eq`**

13.76 *error*

Durch den Benutzer zu definierende Funktion zur Fehlerbehandlung.

`(*error*` ***Z_kette*****`)`**

Wenn **`*error*`** nicht `nil` ist, wird es immer dann als Funktion ausgeführt, wenn eine AutoLISP-Fehlerbedingung vorliegt. AutoCAD übergibt eine Zeichenkette, die die Fehlerbeschreibung enthält, als Argument an die Funktion **`*error*`**.

Die Funktion des folgenden Programmbeispiels erledigt dieselbe Aufgabe wie die normale AutoLISP-Fehlerbehandlungsroutine: Druckerfehler und Beschreibung.

```
(defun *error* (msg)
  (princ "error: ")
  (princ msg)
  (princ)
)
```

Ihre Funktion **`*error*`** kann Aufrufe der Funktion **`command`** ohne Argumente beinhalten (zum Beispiel `(command)`). Dies bedeutet, daß ein zuvor mit **`command`** ausgeführter AutoCAD-Befehl abgebrochen wird.

Siehe auch

Siehe auch "Fehlerbehebung" für ein Beispiel zu einem Fehlerbehandlungsprogramm, welches die Zeichenkette testet, die von den Funktionen **`exit`** und **`quit`** zurückgegeben wird.

13.77 e

val

Gibt die Berechnung eines AutoLISP-Ausdrucks als Ergebnis zurück.

(eval ***Ausdr*****)**

Bei den folgende Zuweisungen:

```
(setq a 123)
(setq b 'a)
```

gilt:

```
(eval 4.0)                ergibt  4.0
(eval (abs -10))          ergibt  10
(eval a)                  ergibt  123
(eval b)                  ergibt  123
```

13.78 exit

Erzwingt die Beendigung der aktuellen Anwendung.

(exit)

Wird die Funktion **exit** aufgerufen, so wird die Meldung Verlassen/Beenden Abbrechen ausgegeben und zur AutoCAD-Eingabeaufforderung Befehl zurückgekehrt.

Siehe auch

Die Funktion **quit**

13.79 exp

Gibt die Konstante *e* (eine reale Zahl) potenziert auf einen bestimmten Wert zurück (den natürlichen Antilogarithmus).

(exp ***num*****)**

```
(atan -1.0)          ergibt  -2.71828
(atan -2.2)          ergibt  -9.02501
(exp -0.4)                   ergibt  0.67032
```

13.80 expand

Weist Knotenraum durch Anforderung einer bestimmten Anzahl von Segmenten zu.

(expand ***Ganzzahl*****)**

Siehe auch

Siehe "Manuelle Zuordnung" für weitere Informationen über die Funktion **expand**.

13.81 expt

Gibt eine Zahl potenziert auf einen bestimmten Wert zurück.

(expt ***Zahl Exponent*****)**

Sind beide Argumente Ganzzahlen, ist auch der Rückgabewert eine Ganzzahl. Andernfalls ist das Ergebnis eine reale Zahl. Beispiele:

```
(expt 2 4)                ergibt  16
(expt 3.0 2.0)            ergibt  9.0
```

13.82 fill_image

Zeichnet ein ausgefülltes Rechteck in der aktuellen Dialogfeldbildkomponente.

(fill_image ***x1 y1 Breite Höhe Farbe*****)**

Die Funktion **fill_image** muß zwischen den Aufrufen von **start_image** und **end_image** verwendet werden. Der von dem Argument *Farbe* definierte Parameter ist eine AutoCAD-Farbnummer oder eine der in der folgenden Tabelle dargestellten logischen Farbnummern.

Symbolische Namen für Farbattribute

Farbnummer	Mnemonischer ADI-Befehlscode	Beschreibung
-2	BGLCOLOR	Aktueller Hintergrund des AutoCAD-Grafikbildschirms
-15	DBGLCOLOR	Aktuelle Hintergrundfarbe des Dialogfelds
-16	DFGLCOLOR	Aktuelle Vordergrundfarbe des Dialogfelds (Textfarbe)
-18	LINELCOLOR	Farbe der Umrandung des aktuellen Dialogfelds

Die erste Ecke (oben links) des Rechtecks liegt bei (*x1*,*y1*) und die zweite Ecke (rechts unten) liegt in der relativen Entfernung (*Breite*,*Höhe*) von der ersten Ecke (*Breite* und *Höhe* müssen positive Werte sein). Der Ursprung (0,0) ist die obere linke Bildecke. Die Koordinaten der unteren rechten Ecke erhalten Sie, indem Sie die Bemaßungsfunktionen aufrufen (**dimx_tile** und **dimy_tile**).

13.83 findfile

Durchsucht den AutoCAD-Bibliotheksuchpfad nach der angegebenen Datei.

(findfile ***Dateiname*****)**

Die Funktion **findfile** macht keine Annahmen über den Dateityp oder die Dateierweiterung von *Dateiname*. Wird in *Dateiname* kein Laufwerk-/Verzeichnispräfix angegeben, so durchsucht **findfile** den AutoCAD-Bibliothekspfad. Ist ein Laufwerk-/Verzeichnispräfix vorhanden, durchsucht **findfile** nur das angegebene Verzeichnis. Die Funktion **findfile** gibt immer einen vollständigen Laufwerks-/Verzeichnis-/Dateinamen oder `nil`, wenn sie angegebene Datei nicht gefunden wird.

Betrachten wir die folgenden Beispiele. Wenn das aktuelle Verzeichnis */acad* ist und die Datei *abc.lsp* enthält,

gibt `(findfile "abc.lsp") das Ergebnis "/acad/abc.lsp"` *zurück*

Wenn Sie eine Zeichnung im Verzeichnis */acad/drawings* bearbeiten, ist die ACAD-Environment-Variable auf */acad/support* eingestellt, und die Datei *xyz.txt* existiert nur im Verzeichnis */acad/support*.

`(findfile "xyz.txt")` *ergibt* `"/acad/support/xyz.txt"`

Ist die Datei *nosuch* in keiner der im Bibliotheksuchpfad angegebenen Verzeichnisse vorhanden, dann gilt:

`(findfile "nosuch")` *ergibt* `nil`

Der vollständige Dateiname, den **findfile** zurückgibt, kann in der Funktion **open** eingesetzt werden.

13.84 fix

Konvertiert eine reale Zahl in den nächstkleineren ganzzahligen Wert.

(fix ***Zahl*****)**

Die Funktion **fix** begrenzt *Zahl* auf die nächstkleinere Ganzzahl, indem sie die Nachkommastellen wegläßt.

```
(fix 3)                    gibt  3  zurück
(fix 3.7)                  gibt  3  zurück
```

Anmerkung Ist *Zahl* größer als die größte darstellbare Ganzzahl (+2.147.483.647 oder -2.147.483.648 auf einem 32-Bit-Rechner), so gibt **fix** eine verkürzte reale Zahl zurück. (Ganzzahlen, die zwischen AutoLISP und AutoCAD ausgetauscht werden, sind jedoch auf durch 16 Bit darstellbare Werte beschränkt.)

13.85 float

Konvertiert eine Zahl in eine reale Zahl.

(float ***Zahl*****)**

```
(float 3)                  gibt  3.0  zurück
(float 3.75)               gibt  3.75  zurück
```

13.86 foreach

Berechnet einen Ausdruck für alle Elemente einer Liste.

(foreach ***Name Liste Ausdruck*** **...)**

Arbeitet *Liste* ab, weist jedem Element *Name* zu und berechnet jeden Ausdruck für jedes Element der Liste. Die Zahl der zu berechnenden Ausdrücke ist beliebig. Die Funktion **foreach** gibt das Ergebnis des zuletzt berechneten *Ausdr*. zurück.

```
(foreach n '(a b c) (print n))
```

ist äquivalent zu

```
(print a)
(print b)
(print c)    und ergibt  c
```

mit dem Unterschied, daß **foreach** nur das Ergebnis des zuletzt bearbeiteten Ausdrucks zurückgibt.

13.87 gc

Erzwingt eine Abfallsammlung und macht so Knotenraum frei.

(gc)

Siehe auch

"Knotenraum" für eine genauere Erklärung der Abfallsammlung.

13.88 gcd

Gibt den größten gemeinsamen Nenner zweier Ganzzahlen zurück.

(gcd ***Ganzzahl1 Ganzzahl2*****)**

Die Argumente *Ganzzahl1* und *Ganzzahl2* müssen Ganzzahlen größer als 0 sein.

```
(gcd 81 57)                gibt  3  zurück
(gcd 12 20)                gibt  4  zurück
```

13.89 get_attr

Ermittelt den DCL-Wert eines Dialogfeldattributs.

(get_attr Schlüssel *Attribute*)

Das Argument Schlüssel ist eine Zeichenkette, die eine Dialogfeldkomponente definiert und zwischen Groß- und Kleinschreibung unterscheidet. Das Argument *Attribute* gibt den Namen des Attributs an, wie es in der DCL-Beschreibung der Komponente erscheint. Die Argumente Schlüssel und *Attribute* sind Zeichenketten. Der zurückgegebene Wert ist der Anfangswert des Attributs, wie er in seiner DCL-Beschreibung angegeben ist, der Wert gibt *nicht* die Änderungen des Zustandes des Feldes wider, die durch Benutzereingaben oder durch Aufrufe von **set_tile** hervorgerufen wurden. Gibt den Wert des Attributs als Zeichenkette zurück.

13.90 get_tile

Ermittelt den aktuellen Laufzeitwert einer Dialogfeldkomponente.

(get_tile Schlüssel)

Das Argument Schlüssel ist eine Zeichenkette, die eine Dialogfeldkomponente definiert und zwischen Groß- und Kleinschreibung unterscheidet. Es gibt den Wert der Komponente als Zeichenkette zurück.

13.91 getangle

Erwartet die Benutzereingabe eines Winkels und gibt diesen Winkel im Bogenmaß zurück.

(getangle *[P] [Anfrage]*)

Das Argument *pt* ist ein 2D-Basispunkt im aktuellen BKS, und das Argument *Anfrage* ist eine als Eingabeaufforderung angezeigte Zeichenkette. Das Argument *P* wird, sofern angegeben, als erster von zwei Punkten interpretiert, so daß der Benutzer AutoLISP den Winkel durch Identifizieren eines weiteren Punktes angeben kann. Sie können zwar einen 3D-Basispunkt angeben, der Winkel wird jedoch immer in der aktuellen Konstruktionsebene gemessen.

Die Funktion **getangle** mißt Winkel in Null-Bogenmaß-Richtung (die durch die Systemvariable ANGBASE bestimmt wird), wobei die Winkelwerte im Gegenuhrzeigersinn zunehmen. Der zurückgegebene Winkel wird im Bogenmaß ausgedrückt, bezogen auf die aktuelle Konstruktionsebene (die *XY*-Ebene des aktuellen BKS, an der aktuellen Erhebung).

Der Benutzer kann auch einen Winkel im aktuellen Winkelformat von AutoCAD eingeben. Obwohl dieses auf Grad, Neugrad oder ein anderes Format eingestellt sein kann, gibt diese Funktion Winkel immer im Bogenmaß zurück. Der Benutzer kann AutoLISP einen Winkel auch dadurch zeigen, daß er zwei 2D-Punkte auf dem Grafikbildschirm identifiziert. AutoCAD zeichnet eine Gummibandlinie vom ersten Punkt zur aktuellen Position des Fadenkreuzes, um dem Benutzer eine visuelle Hilfestellung zu geben.

Es ist wichtig, den Unterschied zwischen dem Eingabewinkel und dem Winkel zu verstehen, den **getangle** zurückgibt. Die Winkel, die an **getangle** übergeben werden, basieren auf den aktuellen Einstellungen von ANGDIR und ANGBASE. Wenn ein Winkel jedoch vorliegt, wird er im Gegenuhrzeigergegensinn gemessen (ANGDIR wird ignoriert), mit Null Bogenmaß als aktuelle Einstellung von ANGBASE. Der folgende Code stellt dar, auf welche Weise verschiedene Argumente verwendet werden können.

```
(setq ang (getangle))
(setq ang (getangle '(1.0 3.5)))
(setq ang (getangle "Wohin? "))
(setq ang (getangle '(1.0 3.5) "Wohin? "))
```

Der Benutzer kann keinen weiteren AutoLISP-Ausdruck als Antwort auf eine **getangle**-Anfrage eingeben.

Siehe auch

Abbildung und Vergleich zu der Funktion **getorient**

13.92 getcfg

Ruft Anwendungsdaten aus dem Teil AppData der Datei *acad.cfg* ab.

(getcfg *cfg_Name*)

Das Argument *cfg_Name* ist eine Zeichenkette (maximale Länge 347 Zeichen), die den zu suchenden Abschnitt und den Parameter enthält, der zu ermitteln ist. Wenn *cfg_Name* nicht gültig ist, gibt **getcfg** den Wert nil zurück. Das Argument cfg_Name muß eine Zeichenkette der folgenden Form sein:

"AppData/*Anw_name*/*Abschn_name*/.../*Param_name*"

Nimmt man zum Beispiel an, daß der Parameter Wandstärke im Abschnitt AppData/ArchZeug den Wert 8 hat,

gibt (getcfg "AppData/ArchZeug/WandSt") *den Wert* "8" zurück

Siehe auch

Die Funktion **setcfg**

13.93 getcname

Findet den lokalisierten oder englischen Namen eines AutoCAD-Befehls.

(getcname *Bef_name*)

Das Argument *Bef_name* gibt den lokalisierten oder unterstrichenen englischen Befehlsnamen an, der bis zu 64 Zeichen enthalten kann. Ist vor *Bef_name* kein Unterstrich vorhanden (vorausgesetzt, es handelt sich um den lokalisierten Befehlsnamen), gibt **getcname** den unterstrichenen englischen Befehlsnamen zurück. Ist vor *Bef_name* ein Unterstrich vorhanden, gibt **getcname** den lokalisierten Befehlsnamen zurück. Diese Funktion gibt nil, wenn *Bef_name* kein gültiger Befehlsname ist.

In einer französischen Version von AutoCAD gilt folgendes:

(getcname "ETIRER") *gibt* "_STRETCH" *zurück*
(getcname "_STRETCH") *gibt* "ETIRER" *zurück*

13.94 getcorner

Erwartet die Benutzereingabe einer zweiten Ecke für ein Rechteck.

(getcorner *P [Anfrage]*)

Die Funktion **getcorner** verlangt die Angabe eines Basispunkts, *pt*, auf der Basis des aktuellen BKS als Argument, und zieht ein Rechteck von diesem Punkt bis zur aktuellen Position des Fadenkreuzes, das der Benutzer über den Bildschirm bewegt. Das Argument *Anfrage* ist eine Zeichenkette, die als Eingabeaufforderung angezeigt wird. Die Funktion **getcorner** gibt einen Punkt im aktuellen BKS zurück, ähnlich wie **getpoint**. Gibt der Benutzer einen 3D-Punkt an, wird seine *Z*-Koordinate ignoriert. Die aktuelle Elevation wird als *Z*-Kooordinate benutzt.

Der Benutzer kann keinen weiteren AutoLISP-Ausdruck als Antwort auf eine **getcorner**-Anfrage eingeben.

13.95 getdist

Erwartet die Benutzereingabe einer Entfernung.

(getdist *[P] [Anfrage]*)

Das Argument *P* stellt einen 2D- oder 3D-Basispunkt im aktuellen BKS dar. Falls angegeben, wird *pt* als erster von zwei Punkten interpretiert, und der Benutzer wird nur nach dem zweiten Punkt gefragt. Das Argument *Anfrage* ist eine Zeichenkette, die als Eingabeaufforderung angezeigt wird.

Eine Entfernung kann dadurch ermittelt werden, daß zwei Punkte ausgewählt werden, oder nur der zweite, wenn ein Basispunkt vorgegeben ist. Der Benutzer kann auch eine Entfernung im aktuellen Entfernungsformat von AutoCAD eingeben. Obwohl das aktuelle Entfernungs-Format auf Fuß und Zoll (architectural) eingestellt sein kann, gibt die Funktion **getdist** eine Entfernung immer als reale Zahl zurück.

Die Funktion **getdist** zeichnet eine Gummibandlinie vom ersten Punkt zur aktuellen Position des Fadenkreuzes, um dem Benutzer eine visuelle Hilfestellung zu geben.

Wird ein 3D-Punkt angegeben, ist auch die gemessene Entfernung dreidimensional. Wird jedoch Bit 64 der Funktion **initget** gesetzt, ignoriert **getdist** die Z-Komponente von 3D-Punkten und gibt eine 2D-Entfernung zurück.

```
(setq dist (getdist))
(setq dist (getdist '(1.0 3.5)))
(setq dist (getdist "Wie weit "))
(setq dist (getdist '(1.0 3.5) "Wie weit? "))
```

Der Benutzer kann keinen weiteren AutoLISP-Ausdruck als Antwort auf eine **getdist**-Anfrage eingeben.

13.96 getenv

Gibt den Wert einer Umgebungsvariable als Zeichenkette zurück.

(getenv *Var_name***)**

Das Argument *Var_name* ist eine Zeichenkette, die den Namen der zu lesenden Variable angibt. Wenn diese Variable nicht existiert, gibt **getenv** den Wert `nil` zurück.

Ist zum Beispiel die Umgebungsvariable ACAD auf das Verzeichnis */acad/support* gesetzt, und die Variable NOSUCH existiert nicht, dann gilt

```
(getenv "ACAD")      ergibt  "/acad/support"
(getenv "NOSUCH")    ergibt  nil
```

Anmerkung Bei UNIX -Systemen beziehen sich ACAD und `acad` auf zwei getrennte Umgebungsvariablen, da diese Betriebssysteme zwischen Groß- und Kleinschreibung unterscheiden.

13.97 getfiled

Fordert den Benutzer mit dem AutoCAD-Standard-Dateidialogfeld zur Eingabe eines Dateinamens auf und gibt diesen Dateinamen zurück.

(getfiled *Titel Vorgabe Erw Flags***)**

Das Argument *Titel* gibt den Dialogfeldtitel an, *Vorgabe* gibt den zu verwendenden Vorgabewert für den Dateinamen an (der eine leere Zeichenkette [""] sein kann), und *Erw* ist der Vorgabewert für die Dateierweiterung. Wenn *Erw* als leere Zeichenkette [""] übergeben wird, ist der Vorgabewert `*` (alle Dateitypen). Wenn der Dateityp `dwg` in das Argument *Erw* eingeschlossen wird, zeigt die Funktion **getfiled** eine Bildvoransicht im Dialog an. Das Argument *Flags* ist ein ganzzahliger Wert (ein Bit-kodiertes Feld), das die Eigenschaften des Dialogfelds steuert. Um mehr als eine Option auf einmal zu aktivieren, addieren Sie die gewünschten Werte, damit *Flags* Werte zwischen 0 und 15 annehmen kann.

Flag-Wert = 1 (Bit 0)

Setzen Sie dieses Bit, wenn Sie zur Eingabe eines Namens einer neu zu *erzeugenden* Datei auffordern möchten. Setzen Sie dieses Bit nicht, wenn Sie eine bestehende Datei *öffnen* möchten. Im zweiten Fall zeigt das Dialogfeld eine Fehlermeldung in seinem unteren Bereich an, wenn der Name einer Datei eingegeben wurde, die nicht existiert.

Ist dieses Bit gesetzt und der Benutzer wählt einen bereits existierenden Dateinamen aus, zeigt AutoCAD ein Warnfenster an und bietet die Auswahl zwischen Fortführen oder Abbrechen der Operation.

Flag-Wert = 2 (Bit 1)

Sperrt das Schaltfeld Eingeben. Dieses Bit wird gesetzt, wenn **getfiled** aufgerufen wird, während ein anderes Dialogfeld noch aktiv ist. (Andernfalls wird das andere Dialogfeld geschlossen).

Ist dieses Bit nicht gesetzt, ist die Schaltfläche Eingeben aktiviert. Wenn der Benutzer die Schaltfläche auswählt, wird das Dialogfeld geschlossen und **getfiled** gibt den Wert 1 zurück.

Flag-Wert = 4 (Bit 2)

Erlaubt die Eingabe einer beliebigen Dateinamenerweiterung oder keiner Erweiterung.

Wird dieses Bit nicht gesetzt, akzeptiert **getfiled** *nur* die Erweiterung, die im Argument *Erw* angegeben wurde, und hängt diese auch an den Dateinamen an, wenn der Benutzer im Textfeld Datei keine Erweiterung angibt.

Flag-Wert = 8 (Bit 3)

Ist dieses Bit gesetzt und Bit 0 ist *nicht* gesetzt, durchsucht **getfiled** die Bibliothek nach dem eingegebenen Dateinamen. Findet **getfiled** die Datei und ihr Verzeichnis im Bibliotheksuchpfad, wird der Pfad entfernt und nur der Dateiname zurückgegeben. (Der Pfadname wird nicht entfernt, wenn eine Datei gleichen Namens in einem anderen Verzeichnis existiert).

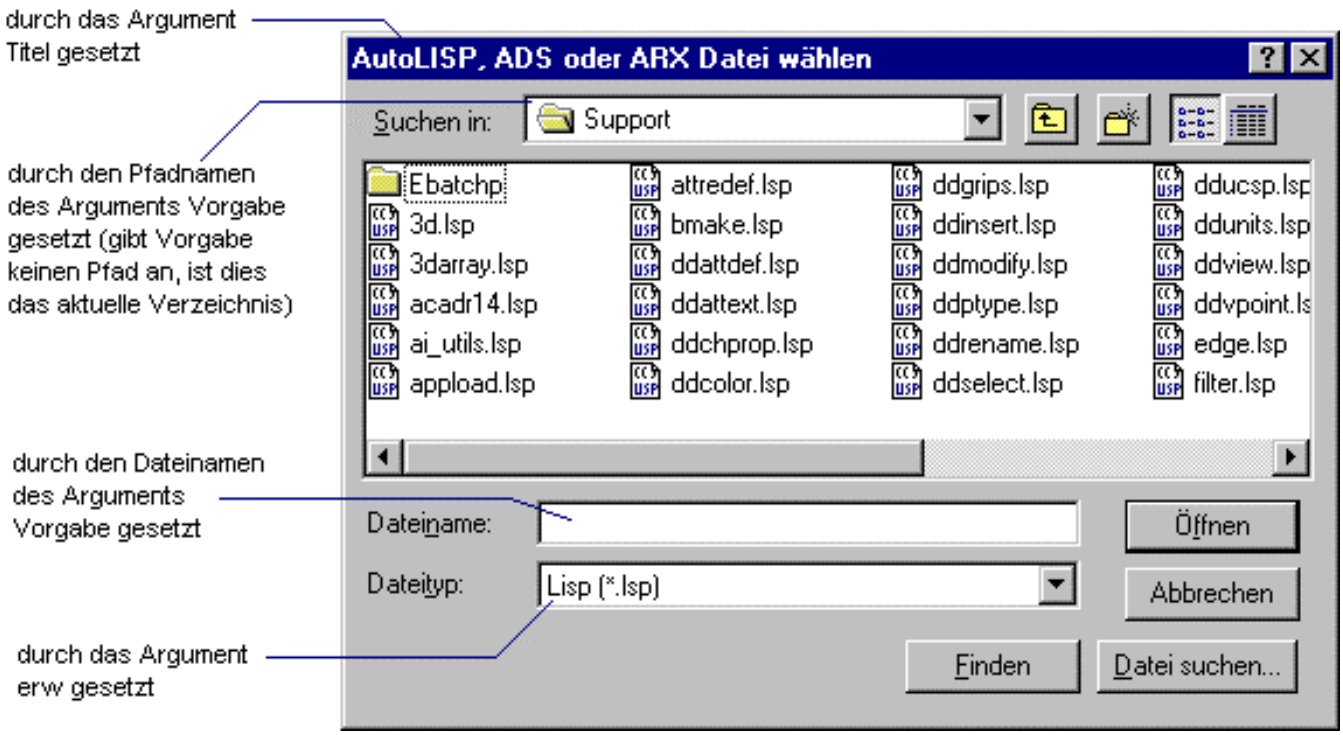
Ist dieses Bit nicht gesetzt, gibt **getfiled** den gesamten Dateinamen inklusive Pfadnamen zurück.

Setzen Sie dieses Bit, wenn Sie mit Hilfe des Dialogfelds eine existierende Datei öffnen, und den Dateinamen in einer Zeichnung (oder anderen Datenbank) speichern möchten.

Wenn in das Dialogfeld vom Benutzer ein Dateiname eingegeben wird, gibt **getfiled** eine Zeichenkette zurück, in der der Dateiname angegeben wird, andernfalls wird `nil` zurückgegeben.

Der folgende Aufruf von **getfiled** erzeugt das Dialogfeld Lisp-Datei wählen:

```
(getfiled "Lisp-Datei wählen" "/acadr14/win/support/" "lsp" 8)
```



Beispiel für ein getfile-Dialogfeld

Die Funktion **getfiled** erzeugt ein Dialogfeld, das eine Liste von Dateien mit einem bestimmten Erweiterungstyp enthält. Mit Hilfe dieses Dialogfelds können Sie verschiedene Laufwerke und Verzeichnisse durchsuchen, eine existierende Datei auswählen oder einen neuen Dateinamen angeben.

13.98 getint

Erwartet die Benutzereingabe einer Ganzzahl und gibt diese Ganzzahl zurück.

(getint ***[Anfrage]*****)**

Das Argument *Anfrage* ist eine fakultative Zeichenkette, die als Eingabeaufforderung angezeigt wird. Die Funktion **getint** gibt die Ganzzahl oder `nil` zurück.

```
(setq num (getint))
(setq num (getint "Geben Sie eine Zahl ein: "))
```

Die an **getint** übergebenen Werte können im Bereich von -32,768 bis +32,767 liegen. Der Anwender kann als Antwort auf eine Anforderung von **getint** keinen anderen AutoLISP-Ausdruck eingeben.

Siehe auch

"Die getxxx-Funktionen" und die Funktion if

13.99 getkword

Erwartet die Benutzereingabe eines Schlüsselworts und gibt dieses Schlüsselwort zurück.

(getkword ***[Anfrage]*****)**

Legen Sie gültige Schlüsselwörter mit der Funktion **initget** fest, bevor Sie die Funktion **getkword** aufrufen. Das Argument *Anfrage* ist eine Zeichenkette, die als Eingabeaufforderung angezeigt wird. Die Funktion **getkword** gibt daraufhin das Schlüsselwort zurück, das der vom Benutzer eingegebenen Zeichenkette entspricht. AutoCAD macht einen weiteren Versuch, wenn die Eingabe keinem Schlüsselwort entspricht. Wenn die Eingabe Null ist (RETURN), gibt **getkword** den Wert `nil` zurück (wenn die Eingabe von Null zulässig ist). Diese Funktion gibt auch dann `nil` zurück, wenn **initget** zuvor nicht aufgerufen wurde, um ein oder mehrere Schlüsselwörter zu definieren.

Das folgende Beispiel stellt einen möglichen ersten Aufruf von **initget** dar, der für den nachfolgenden Aufruf von **getkword** eine Schlüsselwortliste erstellt (Ja und Nein) und die Null-Eingabe ausschließt (der Wert von Bits ist gleich 1):

```
(initget 1 "Ja Nein")
(setq x (getkword "Sind Sie sicher? (Ja oder Nein) "))
```

Dieser Code fordert den Benutzer zu einer Eingabe auf und setzt das Symbol `x` entweder auf `Ja` oder `Nein`, abhängig von der Antwort des Benutzers. Wenn die Antwort keinem der Schlüsselwörter entspricht oder wenn der Benutzer eine Null-Antwort gibt, gibt AutoCAD in einem erneuten Aufruf die im Argument *Anfrage* enthaltene Zeichenkette an. Ist kein Argument *Anfrage* vorhanden, antwortet AutoCAD mit der folgenden Aufforderung:

Nochmaliger Versuch:

Der Benutzer kann keinen weiteren AutoLISP-Ausdruck als Antwort auf eine **getkword**-Anfrage eingeben.

Siehe auch

"Die getxxx-Funktionen" und die Funktion if

13.100 getorient

Erwartet die Benutzereingabe eines Winkels und gibt diesen Winkel im Bogenmaß zurück.

(getorient ***[P] [Anfrage]*****)**

Diese Funktion entspricht der Funktion **getangle**, mit dem Unterschied, daß der von **getorient** zurückgegebene Winkelwert nicht durch die Systemvariablen ANGBASE und ANGDIR beeinflußt wird. Die Eingabe eines Winkels durch den Benutzer hängt jedoch immer noch von den aktuellen Einstellungen von ANGDIR und ANGBASE ab.

Das Argument *P* ist ein 2D-Basispunkt im aktuellen BKS, und das Argument *Anfrage* ist eine als Eingabeaufforderung angezeigte Zeichenkette. Das Argument *P* wird, sofern angegeben, als erster von zwei Punkten interpretiert, so daß der Benutzer AutoLISP den Winkel durch Identifizieren eines weiteren Punktes angeben kann. Sie können zwar einen 3D-Basispunkt angeben, der Winkel wird jedoch immer in der aktuellen Konstruktionsebene gemessen.

Die Funktion **getorient** mißt Winkel mit dem Null-Bogenmaß in Richtung rechts (Osten) und Winkel, die im Gegenuhrzeigersinn größer werden. Wie bei **getangle** wird auch bei **getorient** der zurückgegebne Winkel im Bogenmaß bezogen auf die aktuelle Konstruktionsebene angegeben. Die Winkel, die an **getorient** übergeben werden, basieren auf den aktuellen Einstellungen von ANGDIR und ANGBASE. Sobald jedoch ein Winkel angegeben ist, wird er im Gegenuhrzeigersinn gemessen, wobei die Null-Bogenmaße nach rechts orientiert sind (während ANGDIR und ANGBASE ignoriert werden). Daher müssen Sie eine Konvertierung vornehmen, wenn Sie einen abweichenden Null-Grad-Grundwert oder eine abweichende Richtung für größer werdende Winkel mit Hilfe des Befehls EINHEITEN oder der Systemvariablen ANGBASE und ANGDIR wählen.

Benutzen Sie die Funktion **getangle**, wenn Sie einen Drehwert benötigen (relativer Winkel). Benutzen Sie die Funktion **getorient**, um eine Ausrichtung zu erhalten (absoluter Winkel).

Der Benutzer kann keinen weiteren AutoLISP-Ausdruck als Antwort auf eine **getorient**-Anfrage eingeben.

Siehe auch

Die Funktionen"Die getxxx-Funktionen" und **getangle** und **if**

13.101 getpoint

Erwartet die Benutzereingabe eines Punkts und gibt diesen Punkt zurück.

(getpoint ***[P] [Anfrage]*****)**

Das Argument *P* ist ein 2D- oder ein 3D-Basispunkt im aktuellen BKS, und das Argument Anfragist eine als Eingabeaufforderung angezeigte Zeichenkette. Der Benutzer kann einen Punkt definieren, indem er entweder auf eine Koordinate im aktuellen Einheitsformat zeigt oder eine andere Koordinate eingibt. Wenn das Argument P vorliegt, zeichnet AutoCAD von diesem Punkt eine Gummibandlinie zu der aktuellen Position des Fadenkreuzes. Der zurückgegebene Wert ist ein im aktuellen BKS ausgedrückter 3D-Punkt.

```
(setq p (getpoint))
(setq p (getpoint "Wo? ")) "))
(setq p (getpoint '(1.5 2.0) "Zweiter Punkt: "))
```

Der Benutzer kann keinen weiteren AutoLISP-Ausdruck als Antwort auf eine **getpoint**-Anfrage eingeben.

Siehe auch

Die Funktionen"Die getxxx-Funktionen" und **getcorner** und **if**

13.102 getreal

Erwartet die Benutzereingabe einer realen Zahl und gibt diese reale Zahl zurück.

(getreal ***[Anfrage]*****)**

Das Argument *Anfrage* ist eine Zeichenkette, die als Eingabeaufforderung angezeigt wird.

```
(setq val (getreal))
(setq val (getreal "Skalierfaktor: "))
```

Der Benutzer kann keinen weiteren AutoLISP-Ausdruck als Antwort auf eine **getreal**-Anfrage eingeben.

13.103 getstring

Erwartet die Benutzereingabe einer Zeichenkette und gibt diese Zeichenkette zurück.

(getstring ***[cr] [Anfrage]*****)**

Wenn das Argument *cr* verwendet wird und nicht `nil` ist, darf die eingegebene Zeichenkette Leerzeichen enthalten (und muß mit einem RETURN enden). Andernfalls wird die Eingabe-Zeichenkette durch ein Leerzeichen oder ein RETURN beendet. Das Argument *Anfrage* ist eine Zeichenkette, die als Eingabeaufforderung angezeigt wird.

Enthält die Zeichenkette mehr als 132 Zeichen, werden von der Funktion nur die ersten 132 Zeichen der Kette zurückgegeben. Enthält die eingegebene Zeichenkette das Zeichen für den umgekehrten Schrägstrich (\), wird dieses Zeichen in einen doppelten umgekehrten Schrägstrich (\\) konvertiert. Diese Maßnahme ist erforderlich, damit der zurückgegebene Wert Verzeichnisnamen enthalten kann, die von anderen Funktionen benutzt werden können.

```
(setq s (getstring "Wie lautet Ihr Vorname? ")) "))
```

Durch Eingabe von **Hans** wird `s` auf `"Hans"` gesetzt

```
(setq s (getstring T "Wie lautet Ihr voller Name? ")) "))
```

Durch Eingabe von **Hans Mustermann** wird `s` auf `"Hans Mustermann"` gesetzt

```
(setq s (getstring "Geben sie einen Dateinamen ein: "))
```

Durch Eingabe von **\acad\meinzeich** wird `s` auf `"\\acad\\meinzeich"` gesetzt

Der Benutzer kann keinen weiteren AutoLISP-Ausdruck als Antwort auf eine **getstring**-Anfrage eingeben.

Siehe auch

Die Funktion **getkword** als Beispiel für eine Routine, bei welcher der Benutzer eine oder mehrere bekannte Optionen (Schlüsselwörter) eingeben muß.

13.104 getvar

Ermittelt den Wert einer AutoCAD-Systemvariablen.

(getvar ***Var_name*****)**

Das Argument *Var_name* ist eine Zeichenkette, die eine Systemvariable benennt. Wenn *Var_name* keine gültige Systemvariable ist, gibt **getvar** den Wert `nil` zurück.

Angenommen der Rundungsradius ist bei 0.25 Einheiten festgelegt, dann gilt:

`(getvar "FILLETRAD")` *gibt* `0.25` *zurück*

Eine Liste der aktuellen AutoCAD-Systemvariablen finden Sie in der *AutoCAD-Befehlsreferenz.*

Siehe auch

Die Funktion **setvar**

13.105 graphscr

Zeigt den AutoCAD-Grafikbildschirm an.

(graphscr)

Die Funktion **graphscr** gibt immer `nil` zurück.

Diese Funktion entspricht dem Befehl GRAPHSCR oder dem Drücken der Funktionstaste Bildumschaltung. Die Funktion **textscr** ist das Komplement von **graphscr**.

13.106 grclear

Löscht den Inhalt des aktuellen Ansichtsfensters (veraltet).

(grclear)

Die Funktion **grclear** gibt immer `nil` zurück.

Diese Funktion läßt die Befehl/-Eingabeaufforderungs-, Status- und Menübereiche unverändert. Sie können die Funktion **redraw** anwenden, um den vormaligen Inhalt des Grafikbildschirms zu reaktivieren. Bei Ein-Bildschirm-Systemen, schaltet die Funktion **grclear** zum Grafikbildschirm um, bevor der Inhalt des aktuellen Ansichtsfensters gelöscht wird.

13.107 grdraw

Zeichnet einen Vektor zwischen zwei Punkten im aktuellen Ansichtsfenster.

(grdraw ***von nach Farbe [Markierung]*****)**

Die Argumente *von* und *nach* sind 2D- oder 3D-Punkte (Listen aus zwei oder drei realen Zahlen), welche die Endpunkte der Vektoren entsprechend des aktuellen BKS spezifizieren. AutoCAD beschneidet den Vektor der Bildschirmgröße entsprechend. Die Funktion **grdraw** zeichnet den Vektor mit der im ganzzahligen Argument *Farbe* angegebenen Farbe, wobei -1 *XOR ink* angibt, wodurch alles, was überzeichnet wird, in Komplementärdarste-llung dargestellt wird, und sich selbst beim Überzeichnen löscht. Wenn das angegebene ganzzahlige Argument *Markierung* ungleich Null ist, wird mit Hilfe der für den Bildschirm vorgegebenen Markierungsmethode der Vektor gezeichnet (normalerweise gestrichelt). Wird das Argument *Markierung* ausgelassen oder ist es gleich Null, verwendet die Funktion **grdraw** den gängigen Anzeigemodus. Diese Funktion gibt immer `nil` zurück.

Siehe auch

die Funktion **grvecs** für eine Routine, die mehrere Vektoren zeichnet

13.108 grread

Liest Werte von jedem AutoCAD-Eingabegerät ein.

(grread ***[Spur] [Alle_Tasten [Cur_Typ]]*****)**

Diese Funktion wird nur von speziellen AutoLISP-Routinen benötigt. Die meisten Eingaben für AutoLISP sollten mit den verschiedenen **get***xxx*–Funktionen erhalten werden.

Wenn das Argument *Spur* angegeben und nicht `nil` ist, können Koordinaten von einem Zeigegerät abgelesen werden, während es bewegt wird. Liegt das Argument *Alle_Tasten* vor, führt die Funktion **grread** je nach dem angegebenen Code Funktionen aus. Das Argument *Cur-Typ* kann zur Steuerung des angezeigten Cursortyps verwendet werden. Die Argumente *Alle_Tasten* und *Cur-Typ* sind beide Ganzzahlen und werden im folgenden erklärt:

Alle_Tasten

Die Bit-Code-Werte des Arguments *Alle_Tasten* können addiert werden, um die Funktionalität zu kombinieren.

1 (Bit 0) Liefert *Zeichnen-Modus*-Koordinaten. Wenn dieses Bit gesetzt ist, und der Benutzer das Zeigegerät bewegt, anstatt eine Schaltfläche auszuwählen oder eine Taste zu drücken, gibt die Funktion **grread** eine Liste zurück, in der das erste Element vom Typ 5 ist und das zweite Element die Koordinaten *(X,Y)* der aktuellen Position des Zeigegeräts (Maus oder Digitalisierer) angibt. Auf diese Weise funktioniert bei AutoCAD das Ziehen von Objekten.

2 (Bit 1) Gibt *alle* Schlüsselwerte zurück, u. a. auch die Tastencodes von Funktionen und Cursor, bewegt den Cursor jedoch *nicht*, wenn der Benutzer eine Cursortaste drückt.

4 (Bit 2) Benutzt den im Argument *Cur_Typ* angegebenen Wert, um die Cursoranzeige zu steuern.

8 (Bit 3) Keine Anzeige der Unterbrechungsmeldung error: console, wenn der Benutzer die Tasten CTRL+C drückt.

Cur_Typ

Der Wert des Arguments *Alle_Tasten* für Bit 2 muß definiert werden, damit die Werte des Arguments *Cur_Typ* wirksam werden können.

0 Anzeige des normalen Fadenkreuzes.

1 Keine Anzeige eines Cursors (kein Fadenkreuz).

2 Anzeige des "Ziel"-Cursors zur Objekt-Auswahl.

Anmerkung Das Argument *Cur_Typ* kann nur den Cursor-Typ im aktuellen Aufruf der Funktion **grread** beeinflussen.

Anmerkung Es besteht die Möglichkeit, daß in zukünftigen AutoCAD-Versionen zusätzliche Steuerbits definiert werden.

Die Funktion **grread** gibt eine Liste zurück, deren erstes Element ein Code ist, der den Eingabetyp festlegt. Das zweite Element der Liste ist je nach Eingabetyp entweder eine ganze Zahl oder ein Punkt. Die zurückgegebenen Werte sind in der folgenden Liste zusammengestellt.

Zurückgegebene Werte der Funktion grread

Erstes Element		**Zweites Element**	
Wert	**Eingabetyp**	**Wert**	**Beschreibung**
2	Eingabe über Tastatur	unterschiedlich	Zeichencode
3	Ausgewählter Punkt	3D-Punkt	Koordinaten des Punktes
4	Bildschirm-/Pull-Down-Menüelement (vom Zeigegerät)	0 bis 999 1001 bis 1999 2001 bis 2999 3001 bis 3999 ... usw. bis 16001 bis 16999	Bildschirm-Menüfeld-Nr. POP1-Menüfeld-Nr. POP2-Menüfeld-Nr. POP3-Menüfeld-Nr. ... -POP16-Menüfeld-Nr.
5	Zeigegerät (wird nur zurückgegeben, wenn die Spuraufzeichnung aktiviert ist)	3D-Punkt	Koordinaten des Zugmodus
6	BUTTONS-Menüelement	0 bis 999 1000 bis 1999 2000 bis 2999 3000 bis 3999	SCHALTFLÄCHEN1-Menüschaltflächen-Nr. SCHALTFLÄCHEN2-Menüschaltflächen-Nr.

			SCHALTFLÄCHEN3-Menüschaltflächen-Nr. SCHALTFLÄCHEN4-Menüschaltflächen-Nr.
7	TABLET1-Menüelement	0 bis 32767	Eingegebene Feld-Nr.
8	TABLET2-Menüelement	0 bis 32767	Eingegebene Feld-Nr.
9	TABLET3-Menüelement	0 bis 32767	Eingegebene Feld-Nr.
10	TABLET4-Menüelement	0 bis 32767	Eingegebene Feld-Nr.
11	AUX-Menüelement	0 bis 999 1000 bis 1999 2000 bis 2999 3000 bis 3999	AUX1-Menüschaltflächen-Nr. AUX2-Menüschaltflächen-Nr. AUX3-Menüschaltflächen-Nr. AUX4-Menüschaltflächen-Nr.
12	Taste auf dem Zeigegerät (folgt einer Rückgabe eines Typ 6- oder Typ 11-Codes)	3D-Punkt	Koordinaten des Punktes

Durch Eingabe von ESC während die Funktion **grread** aktiv ist, wird das Programm AutoLISP über die Tastatur abgebrochen (es sein denn, dies wurde über das Argument *Alle_Tasten* nicht zugelassen). Jede andere Eingabe wird direkt an die Funktion **grread** weitergegeben, wodurch der Anwendung die vollständige Steuerung des Eingabegeräts übertragen wird.

Wenn der Benutzer die Taste auf dem Zeigegerät in einem Bildschirm- oder Pull-Down-Menüfeld drückt, gibt die Funktion **grread** einen Typ 6- oder Typ 11-Code zurück, in einem direkt folgenden Aufruf wird jedoch kein Typ 12-Code zurückgegeben: Der Typ 12-Code folgt nur dann auf einen Typ 6 oder Typ 11, wenn die Taste des Zeigegeräts gedrückt wird, während sich das Gerät im Grafikbereich des Bildschirms befindet.

Es ist wichtig, daß die Code 12-Daten aus dem Pufferspeicher gelöscht werden, bevor eine weitere Operation mit Hilfe der Taste auf dem Zeigegerät oder mit einer Hilfsschaltfläche durchgeführt werden soll. Dazu müssen Sie die Funktion **grread** wie folgt verschachteln:

```
(setq code_12 (grread (setq code (grread))))
```

Diese Zeichenfolge erfaßt den Wert der Code 12-Liste als Eingabedatenfolge von dem Gerät.

Anmerkung Aufgrund der Tatsache, daß Eingaben auf den verschiedenen von AutoCAD unterstützten Plattformen unterschiedlich gehandhabt werden, ist es möglich, daß die Funktion **grread** unerwartete Resultate zurückgibt.

- Das vorgegebene Zeigegerät bei Plattformen, die mit einer System-Maus arbeiten, gibt einen Code 11 anstatt des Codes 6 zurück.
- Beim Macintosh wird durch das Pop-Menü ein Code 11 zurückgegeben, nicht der Code 4. Weiterhin wird auf dem Macintosh ein Doppelklick als Code 11 zurückgegeben (nicht als Code 6), auf den ein Code 5-Koordinaten-Paar folgt, wenn dies im aktuellen Ansichtsfenster ausgewählt wird. Und ein Doppelklick in einem anderen als dem aktuellen Ansichtsfenster wird durch ein Code 3-Koordinaten-Paar zurückgegeben, auf das ein Code 11 folgt.

13.109 grtext

Schreibt Text in die Statuszeile oder in den Bildschirmmenübereich.

(grtext ***[Feld Text [Markierung]]*****)**

Das Argument *Feld* ist eine ganze Zahl, welche die Position festlegt, an der Text geschrieben werden soll. Das Argument *Text* ist eine Zeichenkette, die den Text festlegt, der in das Bildschirmmenü oder an der angegebenen Position in die Statuszeile geschrieben werden soll. Ist das Argument *Text* zu lang für den zur Verfügung stehenden Raum, wird es verkürzt. Das Argument Markierung ist eine ganze Zahl, die eine Bildschirmmenüposition auswählt oder eine Auswahl aufhebt. Die Werte dieser Argumente sind je nach Bildschirmposition, an der Sie schreiben möchten, unterschiedlich. Diese Funktion zeigt den übergebenen Text im Menübereich, sie ändert nicht den darunterliegenden Menüpunkt. Die Funktion **grtext** kann auch ohne jedes Argument aufgerufen werden, um für alle Textbereiche ihren

Standardwert wiederherzustellen. Bei erfolgreicher Durchführung gibt die Funktion **grtext** die Zeichenkette zurück, die als Argument *Text* übergeben wurde, und bei nicht erfolgreicher Durchführung wird nil zurückgegeben.

Bildschirmmenübereich

Sie legen eine Bildschirmmenüposition fest, wenn Sie für das Argument *Feld* einen positiven Wert oder den Wert Null definieren. Gültige Werte für das Argument *Feld* liegen zwischen 0 und der größten Bildschirmmenüfeld-nummer minus 1. Die Systemvariable SCREENBOXES enthält die maximale Anzahl der Bildschirmmenüfelder. Wenn für das Argument *Markierung* eine positive ganze Zahl angegeben ist, markiert die Funktion, **grtext** den Text in dem entsprechenden Feld. Sobald ein Feld markiert wird, wird die Markierung jedes beliebigen anderen Feldes automatisch aufgehoben. Hat das Argument *Markierung* den Wert Null, wird die Markierung des Menüelements aufgehoben. Ist dem Argument *Markierung* ein negativer Wert zugeordnet, wird es ignoriert. Auf einigen Plattformen muß der Text zuerst ohne das Argument *Markierung* geschrieben und anschließend markiert werden. Die Markierung einer Bildschirmmenüposition ist nur möglich, wenn sich der Cursor nicht in diesem Bereich befindet.

Bereich der Statuszeile

Wenn bei Aufruf der Funktion **grtext** der Wert des Arguments *Quader* -1 beträgt, schreibt die Funktion den Text in den Modusbereich der Statuszeile. Die Länge des Modusbereichs der Statuszeile ist je nach Bildschirm unterschiedlich (die meisten Bildschirme lassen mindestens 40 Zeichen zu). Der folgende Code benutzt den **$(linelen)** DIESEL-Ausdruck, um die Länge des Modusbereichs der Statuszeile darzustellen.

```
(setq modelen (menucmd "M=$(linelen)"))
```

Wenn der Wert des Arguments *Feld* -2 beträgt, schreibt die Funktion **grtext** den Text in den Koordinatenbereich der Statuszeile. Wenn die Spuraufzeichnung der Koordinaten eingeschaltet wird, werden alle in dieses Feld hineingeschriebenen Werte überschrieben, sobald über das Zeigegerät ein anderes Koordinatenpaar übermittelt wird. Bei beiden Werten (-1 oder -2) von *Quader* wird das Argument *Markierung* ignoriert.

13.110 grvecs

Zeichnet mehrere Vektoren auf dem Grafikbildschirm.

(grvecs ***V-liste [Trans]*****)**

Das Argument *V-liste* ist eine Vektorliste, die eine Reihe ganzer Zahlen für Farboptionen und zwei Punktelisten enthält. Das Argument *Trans* ist eine Transformationsmatrix, die Sie benutzen können, um die Position oder Proportion der in Ihrer Vektorliste definierten Vektoren zu verändern. Diese Matrix ist eine Liste, die ihrerseits aus vier Listen mit je vier realen Zahlen besteht.

Das Format des Arguments *V-liste* sieht folgendermaßen aus:

```
([Farbe1] von1 nach1 [Farbe2] von2 nach2 ...)
```

Der die Farbe definierende Wert gilt solange für alle folgenden Vektoren, bis durch das Argument *V-liste* eine andere Farbe festgelegt wird. AutoCAD-Farben liegen im Bereich von 0-255. Wenn der Farbwert größer als 255 ist, werden die darauf folgenden Vektoren in *XOR ink* gezeichnet, wodurch alles, was überzeichnet wird, in Komplementärdarstellung dargestellt wird, und sich selbst beim Überzeichnen löscht. Ist der Wert für die Farbe kleiner als Null, wird der Vektor markiert. Die Art der Markierung hängt vom Bildschirm ab. Die meisten Bildschirmtreiber zeigen die Markierung durch eine gestrichelte Linie an, aber einige zeichnen einen markierten Vektor in einer bestimmten Farbe.

Ein Punktlisten-Paar, *von* und *nach*, definiert die Endpunkte des im aktuellen BKS dargestellten Vektors. Diese Punkte können zwei- oder dreidimensional sein. Sie müssen diese Punkte paarweise - als zwei aufeinanderfolgende Punktlisten - übergeben, sonst kann **grvecs** nicht erfolgreich aufgerufen werden.

AutoCAD verkürzt die Vektoren, damit sie auf den Bildschirm passen. Wenn **grvecs** erfolgreich aufgerufen wird, gibt die Funktion nil zurück.

Der folgende Code zeichnet fünf vertikale Linien in verschiedenen Farben auf den Grafikbildschirm:

```
(grvecs '(1 (1 2)(1 5)      Zeichnet eine rote Linie von (1,2) nach (1,5)
         2 (2 2)(2 5)       Zeichnet eine gelbe Linie von (2,2) nach (2,5)
         3 (3 2)(3 5)       Zeichnet eine grüne Linie von (3,2) nach (3,5)
         4 (4 2)(4 5)       Zeichnet eine cyanfarbene Linie von (4,2) nach (4,5)
         5 (5 2)(5 5)       Zeichnet eine blaue Linie von (5,2) nach (5,5)
) )
```

Die folgende Matrix stellt eine einheitliche Skalierung von 1.0 und eine Parallelverschiebung von 5.0,5.0,0.0 dar. Wird diese Matrix auf die oben aufgeführte Vektorliste angewandt, werden sie um 5.0,5.0,0.0 verschoben.

```
'(   (1.0 0.0 0.0 5.0)
    (0.0 1.0 0.0 5.0)
    (0.0 0.0 1.0 0.0)
    (0.0 0.0 0.0 1.0)
)
```

Siehe auch

die Funktion **nentselp** für weitere Informationen über Transformationsmatrizen

13.111 handent

Gibt einen Objektnamen (Elementnamen) auf der Basis seiner Referenz zurück.

(handent ***Referenz*****)**

Wenn eine Elementreferenz-Zeichenkette das Argument *Referenz* bildet, gibt die Funktion **handent** den Elementnamen zurück, der mit der Referenz in der aktuellen Bearbeitungssitzung verbunden ist. Die Funktion **handent** gibt den Elementnamen sowohl von grafischen als auch von nichtgrafischen Elementen zurück.

Wenn der Funktion **handent** eine ungültige oder eine in der aktuellen Zeichnung von keinem Element benutzte Referenz zugewiesen wird, wird `nil` zurückgegeben. Die Funktion **handent** gibt Elemente zurück, die in der aktuellen Bearbeitungssitzung gelöscht worden sind. Mit Hilfe der Funktion **entdel** können Sie die Entfernung wieder rückgängig machen.

Der Name eines Elements kann sich von einer Bearbeitungssitzung zur nächsten ändern, die Referenz eines Elements bleibt jedoch unverändert. In einer bestimmten Bearbeitungssitzung kann der Code

`(handent "5A2")` *zurückgeben:* `<Elementname: 60004722>`

Ein Aufruf, der zwar in der gleichen Zeichnung, aber in unterschiedlichen Bearbeitungssitzungen gebraucht wird, kann verschiedene Elementnamen zurückgeben. Liegt der Elementname vor, kann er dazu benutzt werden, das Element mit jeder der zu Elementen gehörenden Funktionen zu manipulieren.

13.112 help

Aktiviert die Hilfefunktion.

(help ***[Hilfedatei [Thema [Befehl]]]*****)**

Das Argument *Hilfedatei* ist eine Zeichenkette, die eine Hilfedatei festlegt. Wenn Sie eine AutoCAD-Hilfedatei (*.ahp*) festlegen, verwendet die Funktion **help** die AutoCAD-Hilfeanzeige, um den Inhalt dieser Datei anzuzeigen. Wenn Sie eine Windows-Hilfedatei (*.hlp*) festlegen, verwendet die Funktion **help** das Programm WinHelp, um den Inhalt dieser Datei anzuzeigen. Wenn das Argument *Hilfedatei* eine leere Zeichenkette (`""`) ist oder weggelassen wird, verwendet AutoCAD die AutoCAD-Hilfedatei der Vorgabe. Das Argument *Thema* ist ein Schlüsselwort, welches das Thema festlegt, das von der Hilfefunktion zuerst angezeigt wird. Wenn das Argument *Thema* eine leere Zeichenkette (`""`) ist, zeigt die Hilfe die Einleitung der Hilfedatei an. Das Argument *Befehl* ist eine Zeichenkette, die den ursprünglichen Zustand des Hilfefensters festlegt, wie in der folgenden Tabelle beschrieben.

Werte für das Argument Befehl

Zeichenkette	Beschreibung
HELP_CONTENTS	Zeigt das erste Thema in der Hilfedatei an.
HELP_HELPONHELP	Zeigt Anweisungen zur Verwendung der Hilfe an.
HELP_PARTIALKEY	Zeigt den Suchdialog unter Verwendung der Zeichenkette an, die als Thema aufgerufen wurde, als ersten Suchtext an.

Wenn Sie eine Windows-Hilfedatei festlegen, kann das Argument `Befehl` auch eine Zeichenkette sein, die von dem Argument *Befehl* der Winhelp()-Funktion verwendet wird, die durch die WinHelp API in der Microsoft Windows SDK definiert ist.

Die Dateinamenerweiterung ist beim Argument `Hilfedatei` nicht erforderlich. Wird eine Dateinamenerweiterung angegeben, sucht AutoCAD nur nach dieser Datei. Erfolgt dies nicht, gelten folgende Regeln für die Suche: append *.hlp* an, ansonsten verwenden Sie *.ahp*.

Die einzige Fehlerbedingung, welche die Funktion **help** der Anwendung zurückgibt, ist die Existenz der Datei, die durch das Argument `Hilfedatei` festgelegt wird. Alle anderen Fehler werden dem Benutzer über ein Dialogfeld mitgeteilt. Die Funktion **help** gibt die Zeichenkette `Hilfedatei` zurück, wenn sie erfolgreich ausgeführt wird und `nil`, wenn ein Fehler auftritt. Wenn Sie **help** ohne Argumente verwenden, gibt die Funktion bei erfolgreicher Ausführung eine leere Zeichenkette (`""`) zurück und `nil`, wenn ein Fehler auftritt.

Der folgende Code ruft **help** auf, um die Information über `MEINBEFEHL` in der Hilfedatei *achelp.ahp* anzuzeigen.

```
(help "achelp.ahp" "meinbefehl")
```

Siehe auch

"Anpassen der Online-Dokumentation" für Informationen über das Erzeugen von AutoCAD-Hilfedateien. Die Funktion **setfunhelp** verknüpft kontextbezogene Hilfe mit einem benutzerdefinierten Befehl (bei Betätigung von F1).

13.113 if

Wertet Ausdrücke je nach Bedingung aus.

(if ***Testausdr Dannausdr [Sonstausdr]*****)**

Wenn `Testausdr` nicht `nil` ist, wird `Dannausdr` ausgewertet, andernfalls wird `Sonstausdr` ausgewertet. Die Funktion **if** gibt den Wert des ausgewählten Ausdrucks zurück. Wenn `Sonstausdr` fehlt, und `Testausdr` ist `nil`, dann gibt die Funktion **if** den Wert `nil` zurück.

```
(if (= 1 3) "JA!!" "nein.") ergibt "nein."
(if (= 2 (+ 1 1)) "JA!!")   ergibt "JA!!"
(if (= 2 (+ 3 4)) "JA!!")   ergibt nil
```

Siehe auch

Die Funktion **progn**

13.114 initget

Führt Schlüsselwörter ein, die beim nächsten Aufruf einer Benutzereingabefunktion gebraucht werden.

(initget ***[Bits] [Z_kette]*****)**

Die Funktionen, die Schlüsselworte berücksichtigen, sind **getint**, **getreal**, **getdist**, **getangle**, **getorient**, **getpoint**, **getcorner**, **getkword**, **entsel**, **nentsel** und **nentselp**.Die Funktion **getstring** ist die einzige Benutzereingabefunktion, die Schlüsselwörter nicht berücksichtigt.

Das Argument `Bits` besteht aus einer binärcodierten ganzen Zahl, die bestimmte Typen von Benutzereingaben zuläßt oder nicht zuläßt. Das Argument `Z_kette` definiert eine Liste der Schlüsselwörter. Die Schlüsselwörter werden vom nächsten Aufruf einer Benutzereingabefunktion überprüft, wenn der Benutzer nicht den erwarteten Eingabetyp eingibt (zum Beispiel einen Punkt bei der Funktion **getpoint**). Entspricht die Benutzerein-gabe einem Schlüsselwort aus der Liste, gibt die Funktion dieses Schlüsselwort als Zeichenkettenergebnis zurück. Die Anwendung kann die Schlüsselwörter überprüfen und die Operation, die mit jedem einzelnen verbunden ist, durchführen. Wenn die Benutzereingabe nicht dem erwarteten Typ entspricht und kein passendes Schlüsselwort vorliegt, fordert AutoCAD den

Benutzer auf, einen erneuten Versuch zu unternehmen. Die Bitwerte und Schlüsselwörter der Funktion **initget** gelten nur für den nächsten Aufruf einer Benutzereingabefunktion.

Die Funktion **initget** gibt immer `nil` zurück.

Definiert die Funktion **initget** ein Steuerbit, und die Anwendung ruft eine Benutzereingabefunktion auf, für die das Bit keine Bedeutung hat, dann wird dieses Bit ignoriert. Die Bits können in beliebiger Kombination addiert werden, um einen Wert zwischen 0 und 255 zu bilden. Wenn kein Argument B*its* übergeben wird, wird Null (keine Bedingungen) angenommen. Erfüllt die Benutzereingabe eine oder mehrere festgelegte Bedingungen nicht (wenn zum Beispiel der Wert Null enthalten ist, obwohl dieser Wert nicht zulässig ist), dann fordert AutoCAD den Benutzer mit einer Meldung auf, einen erneuten Versuch zu unternehmen.

Durch die Funktion initget definierte Eingabeoptionen

Bitwert	Beschreibung
1 (Bit 0)	Verhindert, daß der Benutzer lediglich durch Drücken von RETURN auf eine Anfrage antworten kann.
2 (Bit 1)	Verhindert, daß der Benutzer durch die Eingabe von Null auf eine Anfrage antworten kann.
4 (Bit 2)	Verhindert, daß der Benutzer durch Eingabe eines negativen Wertes antwortet.
8 (Bit 3)	Erlaubt es dem Benutzer, einen Punkt außerhalb der aktuellen Grenzen der Zeichnung einzugeben. Diese Bedingung gilt für die nächste Benutzer-Eingabefunktion, auch wenn die AutoCAD-Systemvariable LIMCHECK zur Zeit gesetzt ist.
16 (Bit 4)	(Zur Zeit nicht belegt)
32 (Bit 5)	Verwendet gestrichelte Linien, um Gummibandlinien oder Felder zu zeichnen. Bei Funktionen, mit denen der Benutzer durch die Auswahl einer Position auf dem Grafikbildschirm einen Punkt definieren kann, sorgt dieser Bitwert dafür, daß die Gummibandlinie oder das Feld mit einer gestrichelten und nicht mit einer durchgezogenen Linie gezeichnet werden. (Einige Bildschirmtreiber setzen eine bestimmte Farbe anstelle der gestrichelten Linie ein.) Ist der Wert der Systemvariablen POPUPS 0, ignoriert AutoCAD dieses Bit.
64 (Bit 6)	Verbietet die Eingabe einer *Z*-Koordinate in die Funktion `getdist` und läßt eine Anwendung sicherstellen, daß diese Funktion einen 2D-Abstand zurückgibt.
128 (Bit 7)	Erlaubt die beliebige Eingabe, als ob es sich um ein Schlüsselwort handelt. Dabei werden zunächst alle anderen Steuerbits und in der Liste enthaltene Schlüsselwörter berücksichtigt. Dies hat Vorrang vor Bit 0; wenn Bit 7 und 0 gesetzt sind und der Benutzer RETURN drückt, wird eine leere Zeichenkette zurückgegeben.

Anmerkung Zukünftige AutoLISP-Versionen können in der Funktion **initget** zusätzliche Steuerbits verwenden, vermeiden Sie daher das Setzen von nicht in der Tabelle gezeigten Bits.

Die speziellen Steuerungswerte werden nur von den Funktionen unter dem Oberbegriff **get*xxx*** berücksichtigt, für die sie sinnvoll sind.

Benutzereingabefunktionen und anwendbare Steuerbits

Funktion	Berück-sichtigt Schlüssel-wörter	Steuer-bitwerte					
		Keine null (1)	Keine Null (2)	Keine negativ (4)	Keine Grenzen (8)	Verwendet gestrichelte Linien (32)	2D Abs (64)
`getint`	▬	▬	▬	▬			
`getreal`	▬	▬	▬	▬			
`getdist`	▬	▬	▬	▬		▬	▬
`getangle`	▬	▬	▬			▬	
`getorient`	▬	▬	▬			▬	
`getpoint`	▬	▬			▬	▬	
`getcorner`	▬	▬			▬	▬	
`getkword`	▬	▬					
`entsel`	▬						
`nentsel`	▬						
`nentselp`	▬						

Spezifikationen für Schlüsselwörter

Das Argument *`Z_kette`* wird nach den folgenden Regeln interpretiert:

Jedes Schlüsselwort ist vom folgenden durch ein oder mehrere Leerzeichen getrennt. `"Breite Höhe Tiefe"` definiert zum Beispiel drei Schlüsselwörter.

Jedes Schlüsselwort kann nur Buchstaben, Zahlen und Bindestriche (-) enthalten.

Es gibt zwei Möglichkeiten, Schlüsselwörter abzukürzen:

- Der gewünschte Teil des Schlüsselworts wird durch Großbuchstaben hervorgehoben, während der Rest des Schlüsselwortes aus Kleinbuchstaben besteht. Der Teil in Großbuchstaben kann sich an einer beliebigen Stelle des Schlüsselwortes befinden (zum Beispiel `"LType"`, `"beendeN"` oder `"obEn"`).
- Das *gesamte* Schlüsselwort besteht nur aus Großbuchstaben, und es schließt sich direkt ein Komma an, auf das die gewünschten Zeichen folgen (zum Beispiel `"LTYPE,LT"`). In diesem Fall müssen die Zeichen der Abkürzung die ersten Buchstaben des Schlüsselworts enthalten, das heißt, daß `"OBEN,E"` ungültig ist.

Die beiden Beispiele `"LType"` und `"LTYPE,LT"` sind äquivalent: Wenn der Benutzer **LT** eingibt (Groß- oder Kleinbuchstaben), reicht diese Eingabe aus, um das Schlüsselwort zu identifizieren. Der Benutzer darf Zeichen eingeben, die dem fraglichen Teil des Schlüsselwortes *folgen*, vorausgesetzt sie treten mit den Kennbuchstaben nicht in Konflikt. In dem Beispiel könnte der Benutzer auch **LTY** oder **LTYP** eingeben, aber **L** reicht nicht aus.

Wenn das Argument *`Z_kette`* das Schlüsselwort ganz in Groß- *oder* in Kleinbuchstaben zeigt, ohne daß ein Komma und eine notwendige Zeichenkombination folgen, erkennt AutoCAD das Schlüsselwort nur dann, wenn es als ganzes Wort eingegeben wird.

Die Funktion **`initget`** bietet Unterstützung für lokalisierte Schlüsselwörter. Die folgende Syntax für die Schlüsselwort-Zeichenkette ermöglicht die Eingabe des übersetzten Schlüsselworts, während sie das sprachenunabhängige Schlüsselwort zurückgibt:

"*lokal1 lokal2 localn _unabh1 unabh2 unabhn*"

Hierbei sind *`lokal1`* bis *`localn`* die übersetzten Schlüsselwörter, und *`unabh1`* bis *`unabhn`* sind die sprachunabhängigen Schlüsselwörter.

Die Anzahl der übersetzten Schlüsselwörter muß mit der Anzahl der sprachunabhängigen Schlüsselwörter übereinstimmen. Das erste sprachunabhängige Schlüsselwort beginnt mit einem Unterstrich, wie im folgenden Beispiel dargestellt:

```
(initget "Abc Def _Ghi Jkl")
(getkword "\nGeben Sie eine Option ein (Abc/Def): ")
```

Die Eingabe von **A** ergibt Ghi und die Eingabe von **_J** ergibt Jkl.

Siehe auch

"Steuern der Bedingungen von Benutzereingabefunktionen"

13.115 inters

Ermittelt den Schnittpunkt zweier Linien.

(inters ***P1 P2 P3 P4 [Aufseg]*****)**

Die Argumente *P1* und *P2* sind die Endpunkte der ersten Linie, und *P3* und *P4* sind die Endpunkte der zweiten Linie. Ist das Argument *Aufseg* vorhanden und beträgt `nil`, wird die Länge der beiden von den vier Argumenten *P* definierten Linien als unendlich betrachtet. Die Funktion **inters** gibt ihren Schnittpunkt an, auch wenn sich dieser Punkt jenseits einer oder beider Linien befindet. Wenn das Argument *Aufseg* weggelassen wird oder nicht `nil` ist, muß der Schnittpunkt auf beiden Linien liegen, oder **inters** gibt `nil` zurück. Die Funktion **inters** gibt `nil` zurück, wenn die beiden Linien sich nicht schneiden.

Alle Punkte werden gemäß des aktuellen BKS ausgedrückt. Sind alle vier Punktargumente dreidimensional, sucht die Funktion **inters** nach dreidimensionalen Schnittpunkten. Ist einer der Punkte zweidimensional, projiziert die Funktion **inters** die Linien auf die aktuelle Konstruktionsebene und sucht nur nach 2D-Schnittpunkten.

```
(setq a '(1.0 1.0) b '(9.0 9.0))
(setq c '(4.0 1.0) d '(4.0 2.0))
(inters a b c d)            ergibt  nil
(inters a b c d T)          ergibt  nil
(inters a b c d nil)        ergibt  (4.0 4.0)
```

13.116 itoa

Gibt die Konvertierung einer Ganzzahl in eine Zeichenkette zurück.

(itoa ***Ganzzahl*****)**

Das Argument *Ganzzahl* gibt eine Ganzzahl an.

```
(itoa 33)                                   ergibt  "33"
(itoa -17)                                  ergibt  "-17"
```

13.117 lambda

Definiert eine unbenannte Funktion.

(lambda ***Argumente Ausdruck*** **. . .)**

Benutzen Sie die Funktion **lambda**, wenn das Definieren von neuen Funktionen zu aufwendig ist. Diese Funktion verdeutlicht zudem die Absicht des Programmierers, indem sie die Funktion an der Stelle darlegt, an der sie benötigt wird. Sie gibt den Wert ihres letzten Arguments *Ausdruck* zurück und wird häufig im direkten Zusammenhang mit den Funktionen **apply** und/oder **mapcar** benutzt, um eine Funktion auf einer Liste auszuführen.

```
(apply '(lambda (x y z)
          (* x (- y z))
        )
        '(5 20 14)
)                           ergibt  30
```

and

```
(setq counter 0)
(mapcar '(lambda (x)
```

```
            (setq counter (1+ counter))
            (* x 5)
          )
          '(2 4 -6 10.2)
)                                ergibt  (10 20 -30 51.0)
```

13.118 last

Gibt das letzte Element einer Liste zurück.

(last ***lst*****)**

```
(last '(a b c d e))        ergibt  E
(last '(a b c (d e)))      ergibt  (D E)
```

Wie dargestellt, kann die Funktion **last** ein Atom oder eine Liste zurückgeben.

Anmerkung Auf den ersten Blick scheint die Benutzung der Funktion **last** die ideale Methode zur Ermittlung der *Y*-Koordinate eines Punkts zu sein. Für 2D-Punkte (Liste mit zwei realen Zahlen) trifft dies auch zu, aber für 3D-Punkte gibt die Funktion **last** die *Z*-Koordinate zurück. Damit Funktionen richtig mit 2D- oder 3D-Punkten arbeiten, verwenden Sie **cadr** zur Bestimmung der *Y*-Koordinaten und **caddr** zur Bestimmung der *Z*-Koordinaten.

13.119 length

Gibt eine Ganzzahl zurück, welche die Anzahl der Elemente in einer Liste angibt.

(length ***lst*****)**

```
(length '(a b (c d)))      ergibt   4
(length '(a b (c d)))      ergibt   3
(length '())               ergibt   0
```

13.120 list

Nimmt eine beliebige Anzahl von Ausdrücken und faßt sie in einer Liste zusammen.

(list Ausdruck ...)

```
(list 'a 'b 'c)            ergibt  (A B C)
(list 'a '(b c) 'd)        ergibt  (A (B C) D)
(list 3.9 6.7)             ergibt  (3.9 6.7)
```

In AutoLISP wird diese Funktion häufig eingesetzt, um Variablen für 2D- oder 3D-Punkte zu definieren (Listen mit zwei oder drei realen Zahlen).

Alternativ zum Gebrauch der Funktion **list** können Sie sich mit der Funktion **quote** explizit auf eine Liste beziehen, wenn diese keine Variablen oder nicht definierten Elemente enthält. Das Zeichen für das Hochkommist als die Funktion quote definierta (**'**) ist als die Funktion quote definiert.

'(3.9 6.7) *ist gleichbedeutetend mit* (list 3.9 6.7)

Diese Tatsache kann bei der Erstellung von Assoziationslisten und beim Definieren von Punkten hilfreich sein.

Siehe auch

Die Funktion **quote**

13.121 listp

Überprüft, ob ein Element eine Liste ist.

(listp ***Element*****)**

Gibt `T` zurück, wenn *`Element`* eine Liste ist, und gibt andernfalls `nil` zurück.

```
(listp '(a b c))           ergibt  T
(listp 'a)                 ergibt  nil
(listp 4.343)              ergibt  nil
```

Anmerkung Da `nil` sowohl ein Atom als auch eine Liste ist, gibt die Funktion **`listp`** den Wert `T` zurück, wenn `nil` übergeben wird.

```
(listp nil)                ergibt   T
```

13.122 load_dialog

Lädt eine DCL-Datei.

`(load_dialog` ***DCL_datei*****`)`**

Das Argument *`DCL_datei`* ist eine Zeichenkette, die eine zu ladende DCL-Datei festlegt. Wenn das Argument *`DCL_datei`* keine Dateierweiterung angibt, wird *.dcl* angenommen. Gibt eine positive Ganzzahl zurück (`DCL_bez`), wenn erfolgreich ausgeführt, und gibt eine negative Ganzzahl zurück, wenn die Datei nicht geöffnet werden kann. Die `DCL_bez` wird als Referenz bei darauffolgenden Aufrufen von **`new_dialog`** und **`unload_dialog`** verwendet.

Die Funktion **`load_dialog`** sucht entsprechend dem AutoCAD-Bibliotheksuchpfad nach Dateien.

Diese Funktion stellt das Komplement zu **`unload_dialog`** dar. Eine Anwendung kann mehrere DCL-Dateien mit mehreren Aufrufen von **`load_dialog`** laden.

13.123 load

Wertet die AutoLISP-Ausdrücke in einer Datei aus.

`(load` ***Dateiname [beiFehler]*****`)`**

Das Argument *`Dateiname`* ist eine Zeichenkette, die den Dateinamen darstellt. Wenn das Argument *`Dateiname`* keine Dateierweiterung angibt, wird *.lsp* angenommen. Scheitert die Ausführung der Funktion **`load`**, wird der Wert des Arguments *`beiFehler`* zurückgegeben. Liegt das Argument *`beiFehler`* nicht vor, verursacht ein Scheitern der Funktion **`load`** einen AutoLISP-Fehler. Ist die Operation erfolgreich, gibt die Funktion **`load`** den Wert des letzten Ausdrucks in der Datei zurück.

Der *`Dateiname`* kann ein Verzeichnis-Präfix enthalten, wie in "*/funktion/test1*". Für DOS-basierte Systeme ist auch ein Laufwerksbuchstabe zulässig. Ein Schrägstrich (`/`) oder zwei umgekehrte Schrägstriche (`\\`) sind gültige Verzeichnis-Trennzeichen. Wenn Sie in die Zeichenkette *`Dateiname`* kein Verzeichnis-Präfix aufnehmen, sucht **`load`** im AutoCAD-Bibliothekspfad nach der angegebenen Datei. Wird sie an irgendeiner Stelle dieses Pfads gefunden, lädt die Funktion **`load`** diese Datei.

Ist das Argument **`beiFehler`** eine gültige AutoLISP-Funktion, erfolgt die Auswertung. Das Argument *`beiFehler`* sollte in den meisten Fällen eine Zeichenkette oder ein Atom sein. Unter dieser Voraussetzung kann eine AutoLISP-Anwendung die Funktion **`load`** aufrufen und hat so andere Möglichkeiten, auf ein eventuelles Scheitern der Anwendung zu reagieren.

Angenommen die Datei */fred/test1.lsp* enthält

```
(defun MEIN_FUNKT1 (x)
...Funktionskörper...
)
(defun MEIN_FUNKT2 (x)
...Funktionskörper...
```

und die Datei *test2.lsp* existiert nicht, dann gilt,

```
(load "/fred/test1")        ergibt   MEIN_FUNKT2
(load "\\fred\\test1")      ergibt   MEIN_FUNKT2
(load "/fred/test1" "bad")  ergibt   MEIN_FUNKT2
(load "test2" "bad")        ergibt   "bad"
(load "test2")              verursacht einen AutoLISP-Fehler
```

Die Funktion **`load`** kann aus einer anderen AutoLISP-Funktion heraus oder auch rekursiv (in der geladenen Datei) angewendet werden.

Siehe auch

Die **`defun`** Funktion und "Symbol- und Funktionsbearbeitung"

13.124 log

Gibt den natürlichen Logarithmus einer Zahl als reale Zahl zurück.

(log ***Zahl*****)**

```
(log 4.5)                    ergibt  1.50408
(log 1.22)                   ergibt  0.198851
```

13.125 logand

Gibt das Ergebnis der logischen bitweisen Operation AND einer Liste von Ganzzahlen zurück.

(logand ***Ganzzahl Ganzzahl*****...)**

```
(apply '+ '(7 15 3))                     ergibt  3
(apply '+ '(2 3 15))                     ergibt  2
(logand 8 3 4)               ergibt  0
```

13.126 logior

Gibt das Ergebnis der logischen bitweisen Operation Inklusiv-OR einer Liste von Ganzzahlen zurück.

(logior ***Ganzzahl Ganzzahl*****...)**

```
(logior 1 2 4)               ergibt  7
(logior 9 3)                 ergibt  11
```

13.127 lsh

Gibt die logische bitweise Verschiebung einer Ganzzahl um eine bestimmte Bitzahl zurück.

(lsh ***Ganzzahl Anzahlbits*****)**

Wenn *Anzahlbits* positiv ist, wird *Ganz* nach links verschoben, ist es negativ, wird nach rechts geschoben. In jedem Fall wird ein 0-Bit in die freiwerdende Stelle eingesetzt, die herausgeschobenen Bits werden verworfen. Der zurückgegebene Wert ist positiv, wenn das höchstwertige Bit (Bit Nummer 31) nach dem Schiebeoperator eine 0 enthält, andernfalls ist er negativ.

```
(lsh 2 1)                    ergibt  4
(lsh 2 1)                    ergibt  1
(lsh 40 2)                   ergibt  160
```

13.128 mapcar

Gibt eine Liste des Ergebnisses einer durchgeführten Funktion mit den einzelnen Elementen einer oder mehrerer Listen zurück, die der Funktion als Argumente dienen.

(mapcar ***Funktion Liste1 . . . Liste_n*****)**

Die Anzahl der Listen muß der Anzahl der Argumente entsprechen, die vom Argument *Funktion* gefordert werden.

```
(setq a 10 b 20 c 30)
(mapcar '1+ (list a b c)) ergibt  (11 21 31)
```

ist äquivalent zu

```
(1+ a)
(1+ b)
(1+ c)
```

Der einzige Unterschied ist, daß die Funktion **mapcar** eine Liste der Ergebnisse zurückgibt.

Die Funktion **lambda** kann eine unbekannte Funktion angeben, die durch **mapcar** ausgeführt werden soll. Dies ist dann hilfreich, wenn einige der Funktionsargumente konstant sind oder auf andere Art und Weise zur Verfügung stehen.

```
(mapcar '(lambda (x)
                    (+ x 3)
                     )
                   '(10 20 30)
```

ergibt `(13 23 33)`

13.129 max

Gibt die größten der angegebenen Zahlen zurück.

(max ***Zahl Zahl*** **...)**

```
(max 4.07 -144)          ergibt  4.07
(max -88 19 5 2)         ergibt  19
(max 2.1 4 8)            ergibt  8.0
```

13.130 mem

Zeigt den aktuellen Zustand des AutoLISP-Speichers an.

(mem)

Zeigt den aktuellen Status des AutoLISP-Speichers an und gibt `nil` zurück. Es zeigt folgende Information an:

Nodes ist die Gesamtzahl der bislang zugewiesenen Knoten und sollte gleich der Knotensegmentgröße multipliziert mit der Anzahl der Segmente sein.

Free nodes ist die Anzahl der Knoten, die zur Zeit als Ergebnis einer Abfallsammlung in der Liste der *freien* Knoten aufgeführt werden.

Segments ist die Anzahl der zugewiesenen Knotensegmente.

Allocate ist die aktuelle Segmentgröße.

Collections ist die Zahl der Abfallsammlungen, sei es automatisch oder erzwungen.

Siehe auch

Kapitel 15, "Speicherverwaltung"

13.131 member

Durchsucht eine Liste nach einem Ausdruck und gibt den Rest der Liste beginnend mit dem ersten Auftreten des gesuchten Ausdrucks zurück.

(member ***Ausdruck Liste*****)**

Wenn *`Ausdruck`* in *`Liste`* nicht auftritt, gibt **`member`** den Wert `nil` zurück.

```
(member 'c '(a b c d e))    ergibt  (C D E)
(member 'q '(a b c d e))    ergibt  nil
```

13.132 menucmd

Aktiviert Menübefehle oder definiert und ermittelt den Status von Menüoptionen.

(menucmd ***Z_kette*****)**

Das Argument *`Z_kette`* ist eine Zeichenkette, die den Menüabschnitt und den ihm zuzuweisenden Wert definiert. Das Argument *`Z_kette`* hat folgende Parameter.

"*`Menü_bereich=Wert`*"

Die erlaubten Werte des *`Menü_bereich`*, die in der folgenden Tabelle gezeigt werden, entsprechen den in den Untermenüreferenzen enthaltenen Werten. Weitere Informationen finden Sie unter "Verweise auf Pull-Down- oder Cursormenüs."

Zeichenkettenwerte für Menü_bereich

Zeichenkette für Menü _ bereich	Menüabschnitt
B1-B4	Für BUTTONS-Menüs 1 bis 4
A1-A4	Für AUX-Menüs 1 bis 4
P0-P16	Für Pull-Down-(POP-)Menüs 0 bis 16
I	Für Bilddatei-Menüs
S	Für das SCREEN-Menü
T1-T4	Für TABLET-Menüs 1 bis 4
M	Für DIESEL-Zeichenkettenausdrücke
`Gmenugroup.nametag`	Legt eine Menügruppe und einen Menübezeichner fest.

Der Parameter *`Wert`* spezifiziert den Wert, der *`Menü_bereich`* zugeordnet wird.

Die Funktion **`menucmd`** kann zwischen Teilbereichen in einem AutoCAD-Menü wechseln. Diese Funktion kann die Anzeige von Menüs auch erzwingen. Auf diese Weise können die AutoLISP-Programme Bildkomponentenmenüs verwenden, um andere Menüs anzuzeigen, aus denen der Benutzer wählen kann. AutoLISP-Programme können Menüoptionen auch aktivieren, deaktivieren und mit Markierungen versehen.

Der folgende Code zeigt das Bilddateimenü `MOREICONS` an.

```
(menucmd "I=moreicons")      Lädt das Bilddateimenü MOREICONS
(menucmd "I=*")              Zeigt das Menü an
```

Der folgende Code überprüft den Status der dritten Menüoption im Pull-Down-Menü `POP11`. Wenn die Menüoption zur Zeit freigegeben ist, wird sie durch die Funktion **`menucmd`** gesperrt.

```
(setq s (menucmd "P11.3=?"))      Ermittelt den Status der Menüoption
(if (= s "")                      Wenn der Status eine leere Zeichenkette ist, sperrt,
  (menucmd "P11.3=~")             die Menüoption
)
```

Der genannte Code ist nicht narrensicher. Menüoptionen können nicht nur aktiviert oder deaktiviert, sondern auch mit Markierungen versehen werden. Der Code `(menucmd "P11.3=?")` kann `"!."` liefern, und so anzeigen, daß die Menüoption gerade überprüft wird. Der genannte Code würde annehmen, daß die Menüoption deaktiviert ist und würde deshalb fortfahren, ohne sie zu deaktivieren. Falls der Code einen Aufruf der Funktion **`wcmatch`** enthält, könnte er den Status auf das Auftreten des Tildenzeichens (~) überprüfen und dann entsprechende Maßnahmen einleiten.

Die Funktion **`menucmd`** ermöglicht es AutoLISP-Programmen auch, Nutzen aus der DIESEL-Sprache zu ziehen. Einige Dinge sind mit DIESEL einfacher durchzuführen als mit dem äquivalenten AutoLISP-Code. Der folgende Code gibt eine Zeichenkette zurück, die den aktuellen Tag und das Datum beinhaltet:E

```
(menucmd "M=$(edtime,$(getvar,date),DDDD\",\" D MONTH YYYY)")
ergibt    "Sunday, 16 July 1995"
```

Siehe auch

Kapitel 6, "Programmierschnittstellen", für weitere Informationen über die Ermittlung des Menü-Bezeichnungs-Status in AutoLISP, und Kapitel 5, "Statuszeilen- konfiguration und Makroprogrammiersprache DIESEL", für Informationen über die Benutzung von DIESEL

13.133 menugroup

Überprüft, ob eine Menügruppe geladen ist.

`(menugroup` ***Gruppenname*****`)`**

Das Argument *`Gruppenname`* ist eine Zeichenkette, die den Menügruppennamen festlegt. Wenn *`Gruppenname`* einer geladenen Menügruppe entspricht, gibt die Funktion die *`Gruppenname`* Zeichenkette zurück; im anderen Fall gibt sie nil zurück.

13.134 min

Gibt die kleinste der angegebenen Zahlen zurück.

(min ***Zahl Zahl...*****)**

```
(min 683 -10.0)          ergibt  -10.0
(min 73 2 48 5)          ergibt    2
(min 2 4 6.7)            ergibt    2.0
```

13.135 minusp

Überprüft, ob eine Zahl negativ ist.

(minusp ***Zahl*****)**

Gibt `T` zurück, wenn *`Zahl`* negativ ist, und gibt andernfalls `nil` zurück.

```
(minusp -1)                ergibt  T
(minusp -4.293)            ergibt  T
(minusp 830.2)             ergibt  nil
```

13.136 mode_tile

Definiert den Modus einer Dialogfeldkomponente.

(mode_tile ***Schlüssel Modus*****)**

Das Argument *`Schlüssel`* ist eine Zeichenkette, die eine Dialogfeldkomponente definiert und zwischen Groß- und Kleinschreibung unterscheidet. Das Argument *`Modus`* ist ein ganzzahliger Wert. Die Werte des Arguments *`Modus`* werden in der folgenden Tabelle beschrieben.

Werte des Arguments Modus

Wert	Beschreibung
0	Komponenten aktivieren
1	Komponenten deaktivieren
2	Komponenten fokussieren
3	Inhalt eines Eingabefelds auswählen
4	Bildmarkierung an- oder ausschalten

13.137 namedobjdict

Gibt den Elementnamen des in der aktuellen Zeichnung benannten Objektwörterbuchs an, das die Grundlage aller nichtgrafischen Elemente in der Zeichnung bildet.

(namedobjdict)

Unter Verwendung des Namens, der von dieser Funktion zurückgegeben wird, und der Wörterbuchzugriffsfunktionen kann eine Anwendung auf die nichtgrafischen Objekte in einer Zeichnung zugreifen.

13.138 nentsel

Fordert den Benutzer auf, durch Definieren eines Punkts ein Objekt (Element) auszuwählen, und gewährt Zugriff auf die Definitionsdaten, die in einem komplexen Objekt enthalten sind.

(nentsel ***[Anfrage]*****)**

Das Argument *`Anfrage`* ist eine Zeichenkette, die als Eingabeaufforderung angezeigt wird. Fehlt dieses Argument, wird die Eingabeaufforderung Objekte wählen aktiviert.

Die Funktion **nentsel** fordert den Benutzer auf, ein Objekt zu wählen. Der aktuelle Ofang-Modus wird ignoriert, wenn der Benutzer ihn nicht ausdrücklich anfordert. Um zusätzliche Unterstützung in der Eingabeaufforderung Befehl zu geben, berücksichtigt **nentsel** Schlüsselworte, die bei einem vorherigen Aufruf von **initget** definiert wurden.

Wenn das gewählte Objekt nicht komplex (d. h. keine Polylinie und kein Block) ist, gibt **nentsel** dieselbe Information zurück, wie **entsel**. Ist das gewählte Objekt jedoch eine Polylinie, gibt die Funktion **nentsel** eine Liste zurück, die den Namen des Subelements (Kontrollpunkt) und den Auswahlpunkt enthält. Diese entspricht der Liste, die von **entsel** zurückgegeben wird, anstelle des Polylinien-Headers wird jedoch der Name des gewählten Auswahlpunkts zurückgegeben. Die Funktion **nentsel** gibt immer den Anfangskontrollpunkt des gewählten Polyliniensegments zurück. Wenn Sie zum Beispiel das dritte Segment einer Polylinie auswählen, wird Ihnen der dritte Kontrollpunkt zurückgegeben. Das Subelement Seqend wird von **nentsel** für eine Polylinie niemals zurückgegeben.

Anmerkung Eine optimierte Polylinie (Element lwpolyline) ist in der Zeichnung als einzelnes Element definiert, es enthält keine Subelemente.

Wenn Sie ein Attribut innerhalb einer Blockreferenz auswählen, werden der Attributsname und der Auswahlpunkt zurückgegeben. Ist ein gewähltes Objekt Bestandteil einer Blockreferenz, die kein Attribut ist, gibt die Funktion, **nentsel** eine Liste mit vier Elementen zurück.

Das erste Element der Liste, die nach der Auswahl eines Objekts innerhalb eines Blocks zurückgegeben wird, ist der Name des gewählten Elements. Das zweite Element ist eine Liste, welche die Koordinaten des Punktes enthält, der zum Auswählen des Objekts benutzt wurde.

Das dritte Element wird als die Modell-Welt-Transformationsmatrix bezeichnet. Dies ist eine Liste, die aus vier Sublisten besteht, von denen jede einen Satz von Koordinaten enthält. Diese Matrix kann eingesetzt werden, um die Punkte für die Elementdefinition von einem internen Koordinatensystem, dem sogenannten Modellkoordinatensystem (MKS), in das Weltkoordinatensystem (WKS) zu übertragen. Der Einfügepunkt des Blocks, der das gewählte Element enthält, definiert den Ursprung des MKS. Die Ausrichtung des BKS bei der Erstellung des Blocks bestimmt die Ausrichtung der Achsen des MKS.

Das vierte Element ist eine Liste, die den Elementnamen des Blocks mit dem gewählten Objekt enthält. Befindet sich das gewählte Objekt in einem verschachtelten Block (ein Block innerhalb eines Blocks), enthält die Liste zusätzlich noch die Elementnamen aller Blöcke, in denen das gewählte Objekt verschachtelt ist, die Liste beginnt mit dem innersten Block und endet mit dem Namen des äußersten Blocks, der in die Zeichnung eingefügt wurde.

```
(<Elementname: Elem_name1>  Name des Elements

  (Px Py Pz)                Auswahlpunkt
  ( (X0 Y0 Z0)              Modell-Welt-Transformationsmatrix
    (X1 Y1 Z1)
    (X2 Y2 Z2)
    (X3 Y3 Z3)
  )
  (<Elementname:Elem_name2> Name des am tiefsten verschachtelten Blocks, der das gewählte
     .                      Objekt enthält
     .
  <Elementname:Elem_namen>) Name des äußersten Blocks,
)                           der das gewählte Objekt enthält
```

Wenn der Elementname und die Modell-Welt-Transformationsmatrix ermittelt wurde, können Sie die Elementdefinitions-Datenpunkte vom MKS ins WKS transformieren. Verwenden Sie **entget** und **assoc** für den Elementnamen, um die Definitionspunkte ausgedrückt in MKS-Koordinaten zu erhalten. Die Modell-Welt-Transformationsmatrix, die von **nentsel** zurückgegeben wird, hat denselben Zweck, wie die von **nentselp** zurückgegebene, sie ist jedoch eine 4x3-Matrix, die als Reihe aus vier Punkten weitergegeben wird und auf der Konvention beruht, daß ein Punkt eher durch eine Zeile als durch eine Spalte dargestellt werden kann. Die Transformation wird durch die folgende Matrix-Multiplikation dargestellt:

$$\begin{bmatrix} X' & Y' & Z' & 1.0 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1.0 \end{bmatrix} \cdot \begin{bmatrix} M_{00} & M_{01} & M_{02} \\ M_{10} & M_{11} & M_{12} \\ M_{20} & M_{21} & M_{22} \\ M_{30} & M_{31} & M_{32} \end{bmatrix}$$

Somit sehen die Gleichungen, mit Hilfe derer die neuen Koordinaten abgeleitet werden, folgendermaßen aus:

$$X' = XM_{00} + YM_{10} + ZM_{20} + M_{30}$$
$$Y'''' = XM_{01} + YM_{11} + ZM_{21} + M_{31}$$
$$Z' = XM_{02} + YM_{12} + ZM_{22} + M_{32}$$

Die *Mij*, wobei 0 *i*, *j* 2 ist, sind die Koordinaten der Modell-Welt-Transformationsmatrix; *X*, *Y*, *Z* ist der Elementdefinitions-Datenpunkt ausgedrückt in MKS-Koordinaten, und *X'*, Y', Z' ist der sich ergebende Elementdefinitions-Datenpunkt, ausgedrückt in WKS-Koordinaten.

Anmerkung Dies ist die einzige AutoLISP-Funktion, bei der eine Matrix dieses Typs verwendet wird, die Funktion **`nentselp`** gibt eine Matrix zurück, die ähnlich der von anderen AutoLISP- und ADSRX-Funktionen verwendeten ist.

Siehe auch

Die Funktionen"Elementnamen-Funktionen" sowie die Funktionen **`entsel`** und **`if`**

13.139 nentselp

Funktioniert ähnlich wie die Funktion `nentsel`, benötigt jedoch keine Benutzereingabe.

`(nentselp` *[Anfrage] [P]*`)`

Zusätzlich zum optionalen Argument `Anfrage` akzeptiert die Funktion **`nentselp`** einen Auswahlpunkt als fakultatives Argument. Dies ermöglicht die Auswahl von Objekten ohne Benutzereingabe. Die Funktion **`nentselp`** gibt eine 4x4 - Transformationsmatrix zurück, die wie folgt definiert ist:

$$\begin{vmatrix} M_{00} & M_{01} & M_{02} & M_{03} \\ M_{10} & M_{11} & M_{12} & M_{13} \\ M_{20} & M_{21} & M_{22} & M_{32} \\ M_{30} & M_{31} & M_{32} & M_{33} \end{vmatrix}$$

Die ersten drei Spalten der Matrix legen die Skalierung und die Drehung fest. Die vierte Spalte stellt einen Vektor für die Parallelverschiebung dar.

Die Funktionen, die eine Matrix dieses Typs verwenden, behandeln einen Punkt als Spaltenvektor der Dimension 4. Der Punkt wird in *homogenen Koordinaten* ausgedrückt, bei denen das vierte Element des Punktvektors ein *Skalierfaktor* ist, der normalerweise auf 1.0 gesetzt ist. Die letzte Zeile der Matrix, der Vektor [*M*30 *M*31 *M*32 *M*33], hat den Nennwert [0 0 0 1] und wird zur Zeit von den Funktionen, die dieses Matrixformat verwenden, ignoriert. Bei dieser Konvention ist die Anwendung einer Transformation auf einen Punkt eine Matrix-Multiplikation, die wie folgt durchgeführt wird:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} M_{00} & M_{01} & M_{02} & M_{03} \\ M_{10} & M_{11} & M_{12} & M_{13} \\ M_{20} & M_{21} & M_{22} & M_{23} \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1.0 \end{bmatrix}$$

Aus dieser Multiplikation gehen die folgenden Einzelkoordinaten des Punktes hervor:

$$X' = XM_{00} + YM_{01} + ZM_{02} + M_{03}(1.0)$$
$$Y''' = XM_{10} + YM_{11} + ZM_{12} + M_{13}(1.0)$$
$$Z' = XM_{20} + YM_{21} + ZM_{22} + M_{23}(1.0)$$

Wie aus diesen Gleichungen hervorgeht, sind der Skalierfaktor und die unterste Zeile der Matrix wirkungslos und werden daher ignoriert.

13.140 new_dialog

Eröffnet ein neues Dialogfeld und zeigt es an; kann auch eine Vorgabeoperation angeben.

(new_dialog ***Dialogname DCL_bez [Operation [Bildschirm_P]]*****)**

Das Argument *Dialogname* ist eine Zeichenkette, die das Dialogfeld definiert, und das Argument and *DCL_bez* bezeichnet die DCL-Datei. (Ihren Wert müssen Sie aus dem Aufruf der Funktion **load_dialog** erhalten haben.)

Das Argument *Operation*, das definiert sein muß, wenn Sie das Argument *Bildschirm-pt* festlegen, ist eine Zeichenkette, die einen AutoLISP-Ausdruck enthält, der als Vorgabeoperation zu benutzen ist. Möchten sie keine Vorgabeoperation definieren, geben Sie das Argument *Operation* als leere Zeichenkette (`""`) ein. Das Argument *Bildschirm_P* ist eine 2D-Punktliste, die mit Hilfe der Koordinaten *X,Y* die Position des Dialogfelds auf dem Bildschirm definiert. Der Punkt gibt normalerweise die linke obere Ecke des Dialogfeldes an, dies hängt jedoch wie das Einheitensystem, in dem die Position definiert wird, von der jeweiligen Plattform ab. Geben Sie den Punkt als `'(-1 -1),` ein, wird das Dialogfeld in der vorgegebenen Position geöffnet (in der Mitte des AutoCAD-Grafikbildschirms).

Bei erfolgreicher Durchführung gibt **new_dialog** den Wert `T` zurück, andernfalls wird `nil` zurückgegeben.

Ihre Anwendung muß **new_dialog** aufrufen, bevor **start_dialog** aufgerufen wird. Die gesamte Initialisierung eines Dialogfeldes - wie zum Beispiel das Setzen von Komponentenwerten, das Erzeugen von Bildern oder Listen für Listenfelder sowie die Durchführung verwandter Operationen mit einzelnen Komponenten (unter Zuhilfenahme der Funktion **action_tile**) - muß nach dem Aufruf der Funktion **new_dialog** und vor dem Aufruf der Funktion **start_dialog** erfolgen.

Die Vorgabeoperation wird ausgewertet, wenn der Benutzer eine aktive Komponente wählt, der durch die Funktion **action_tile** oder in DCL weder eine Operation noch eine Rückmeldung ausdrücklich zugeordnet worden sind.

Anmerkung Überprüfen Sie immer den Status, den **new_dialog** zurückgibt. Der Aufruf von **start_dialog** kann zu unvorhergesehenen Ergebnissen führen, wenn der Aufruf von **new_dialog** nicht funktioniert hat.

13.141 not

Überprüft, ob ein Element nil ergibt.

(not ***Element*****)**

Gibt `T` zurück, wenn *Element* `nil` ist, und gibt andernfalls `nil` zurück.

```
(setq a 123 b "Zeichenkette" c nil)
(not a)                          ergibt  nil
(not b)                          ergibt  nil
(not c)                          ergibt  T
(not '())                        ergibt  T
```

Normalerweise wird die Funktion **null** für Listen benutzt, und die Funktion **not** wird bei anderen Datentypen in Verbindung mit einer Steuerfunktion angewendet.

13.142 nth

Gibt das nte Element einer Liste zurück.

(nth ***n Liste*****)**

Das Argument *n* ist die Nummer des zurückzugebenden Elements. (Null ist das erste Element.) Wenn *n* größer als die höchste Elementnummer von *Liste* ist, gibt **nth** den Wert `nil` zurück.

```
(nth 3 '(a b c d e))        ergibt  D
```

```
(nth 0 '(a b c d e))        ergibt  A
(nth 5 '(a b c d e))        ergibt  nil
```

13.143 null

Überprüft, ob ein Element an nil gebunden ist.

(null ***Element*****)**

Gibt `T` zurück, wenn *`Element`* an `nil` gebunden ist, und gibt andernfalls `nil` zurück.

```
(setq a 123 b "Zeichenkette" c nil)
(null a)                    ergibt  nil
(null b)                    ergibt  nil
(null c)                    ergibt  T
(null '())                  ergibt  T
```

13.144 numberp

Überprüft, ob ein Element eine reale Zahl oder eine Ganzzahl ist.

(numberp ***Element*****)**

Gibt `T` zurück, wenn *`Element`*eine reale Zahl oder eine Ganzzahl ist, und gibt andernfalls `nil` zurück.

```
(setq a 123 b 'a)
(numberp 4)                 ergibt  T
(numberp 3.8348)                    ergibt  T
(numberp "Howdy")                   ergibt  nil
(numberp 3.8348)                    ergibt  T
(null b)                    ergibt  nil
(numberp (eval b))                  ergibt  T
```

13.145 open

Öffnet eine Datei für den Zugriff durch die AutoLISP-E/A-Funktionen.

(open ***Dateiname Modus*****)**

Das Argument *`Dateiname`* ist eine Zeichenkette, die den Namen und die Dateierweiterung der zu öffnenden Datei angibt. Das Argument *`Modus`* ist die Steuermarke für Lesen/Schreiben. Sie muß eine Zeichenkette mit einem einzigen Kleinbuchstaben sein. Die folgende Tabelle enthält die gültigen Buchstaben für *`Modus`*.

Modus-Optionen der Funktion open

Open-Modus	Beschreibung
`"r"`	Öffnet die Datei zum Lesen. Wenn *`Dateiname`* nicht vorhanden ist, gibt **`open`** den Wert `nil` zurück.
`"w"`	Öffnet die Datei zum Schreiben. Gibt es das Argument *`Dateiname`* nicht, wird eine neue Datei erstellt und geöffnet. Existiert das Argument *`Dateiname`* bereits, werden seine Daten überschrieben.
`"a"`	Öffnet die Datei für Anhängungen. Gibt es das Argument *`Dateiname`* nicht, wird eine neue Datei erstellt und geöffnet. Existiert das Argument *`Dateiname`* bereits, wird es geöffnet und der Zeiger wird am Ende der bereits existierenden Daten positioniert, so daß neue Daten, die Sie zu der Datei hinzufügen, an die bestehenden Daten angehängt werden.

Daten, die in eine geöffnete Datei übertragen wurden, werden erst nach dem Schließen der Datei mit der Funktion **`close`** geschrieben.

Anmerkung Bei DOS-Systemen schreiben einige Programme und Texteditoren Textdateien mit einer Dateiendemarke (STR Z, Dezimal-ASCII-Code 26) am Ende des Textes. Beim Lesen einer Textdatei gibt DOS einen Dateiendestatus zurück, wenn eine STR Z-Marke gefunden wird, auch wenn auf diese Marke noch weitere Daten folgen. Wenn Sie beabsichtigen, den Modus `"a"` von OPEN zu verwenden, um Daten an Dateien anzuhängen, die von einem anderen

Programm erzeugt wurden, stellen Sie sicher, daß dieses andere Programm keine STRG-Z-Marke an das Ende seiner Textdateien anfügt.

Die Funktion **open** gibt einen Dateideskriptor zurück, der von anderen E/A-Funktionen verwendet wird. Dem Dateideskriptor muß ein Symbol der Funktion **setq** zugewiesen sein.

```
(setq a (open "dateiname.ext" "r"))
```

Wenn die im folgenden Beispiel genannten Dateien *nicht* existieren, gilt:

```
(setq f (open "neu.tst" "w"))     ergibt  <Datei #nnn>
(setq f (open "nichtda.dat" "r"))       ergibt  nil
(setq f (open "logdatei" "a"))    ergibt  <Datei #nnn>
```

Das Argument *Dateiname* kann ein Verzeichnispräfix enthalten, wie in */test/funkt3*. Bei DOS-Systemen ist außerdem die Angabe eines Laufwerksbuchstabens zulässig, und Sie müssen den umgekehrten Schrägstrich (\) anstelle des Schrägstrichs (/) verwenden. Beachten Sie jedoch, daß Sie zwei umgekehrte Schrägstriche (\\) schreiben, um einen umgekehrten Schrägstrich in einer Zeichenkette zu erhalten.

```
(setq f (open "/x/neu.tst" "w")) ergibt  <Datei #nnn>
(setq f (open "nichtda.dat" "r"))       ergibt  nil
```

13.146 or

Gibt das logische OR einer Liste von Ausdrücken zurück.

(or *Ausdruck* ...)

Die Funktion **or** wertet die Ausdrücke auf der Suche nach einem Nicht-`nil`-Ausdruck von links nach rechts aus. Wenn ein solcher Ausdruck gefunden wird, beendet **or** die weitere Auswertung und gibt `T` zurück. Wenn alle Ausdrücke `nil` sind, gibt **or** den Wert `nil` zurück.

```
(or nil 45 '())             ergibt  T
(or nil '())                ergibt  nil
```

13.147 osnap

Gibt als Ergebnis einer Anwendung des Ofang-Modus auf einen festgelegten Punkt einen 3D-Punkt zurück.

(osnap *P Modus*)

Das Argument *Modus* ist eine Zeichenkette, die aus einem oder mehreren gültigen Ofang-Bezeichnern besteht, wie z B. `mid`, `cen` usw., die durch Kommata getrennt sind.

```
(setq P1 (getpoint))
(setq P2 (osnap P1 "cen"))
(setq P3 (osnap P1 "end,sch"))
```

Der von der Funktion **osnap** zurückgegebene Punkt ist von der aktuellen 3D-Ansicht und der Festlegung der Systemvariablen APERTURE abhängig.

13.148 polar

Gibt den 3D-Punkt im BKS in einem festgelegten Winkel und Abstand von einem Punkt zurück.

(polar *P Winkel Abstand*)

Das Argument *Winkel* wird im Bogenmaß relativ zur *X*-Achse ausgedrückt, wobei die Winkel in Richtung gegen den Uhrzeigersinn größer werden. Obwohl das Argument *P* ein 3D-Punkt sein kann, hängt das Argument *Winkel* immer direkt von der aktuellen Konstruktionsebene ab.

```
(polar '(1 1 3.5) 0.785398 1.414214)  ergibt  (2.0 2.0 3.5)
```

13.149 prin1

Druckt einen Ausdruck in die Befehlszeile oder schreibt einen Ausdruck in eine geöffnete Datei.

(prin1 *[Ausdruck [Dateideskr]]*)

Das Argument *Ausdruck* muß keine Zeichenkette sein. Wenn das Argument *Dateideskr* vorhanden und der Dateideskriptor für eine zum Schreiben geöffnete Datei ist, wird das Argument *Ausdruck* genau so in die Datei geschrieben, wie es auf dem Bildschirm erscheint. Es wird nur der angegebene *Ausdruck* gedruckt, es ist kein Zeilenvorschub oder Leerzeichen enthalten.

```
(setq a 123 b '(a))
(prin1 'a)          druckt  A         und gibt  A  zurück
(prin1 a)           druckt  123       und gibt  123  zurück
(prin1 b)           druckt  (A)       und gibt  (A)  zurück
(prin1 "Hallo")     druckt  "Hallo"   und gibt  "Hallo"  zurück
```

Jedes der oben angefügten Beispiele wird auf dem Bildschirm angezeigt, da kein Argument *Dateideskr* definiert war. Angenommen, das Argument `f` ist ein gültiger Dateideskriptor für eine zum Schreiben geöffnete Datei, dann schreibt

```
(prin1 "Hallo" f)
```

`"Hallo"` in die angegebene Datei und gibt `"Hallo"` zurück.

Ist das Argument *Ausdruck* eine Zeichenkette, die Steuerzeichen enthält, erweitert die Funktion **prin1** diese Zeichen, wie in der folgenden Tabelle dargestellt, durch einen vorangestellten \.

Steuercodes

Code	Beschreibung
`\\`	\-Zeichen
`\"`	"-Zeichen
`\e`	Escape-Zeichen
`\n`	Zeilenumbruchzeichen
`\r`	Zeichen für die Eingabetaste
`\t`	Tab-Zeichen
`\`*nnn*	Zeichen, dessen Oktalcode *nnn* entspricht
`\U+`*XXXX*	Unicode-Folge
`\M+N`*XXXX*	Unicode-Folge

Die folgenden Beispiele zeigen den Gebrauch von Steuerzeichen:

```
(prin1 (chr 2))     druckt  "\002"  und gibt  "\002"     zurück
(prin1 (chr 10))    druckt  "\n"    und gibt  "\n"     zurück
```

Die Funktion **prin1** kann ohne Argumente aufgerufen werden. In diesem Fall gibt sie die leere Zeichenkette zurück (und druckt sie). Benutzen Sie die Funktion **prin1** (ohne Argument) als letzten Ausdruck einer benutzerdefinierten Funktion, wird nach Beendigung der Funktion nur eine Leerzeile gedruckt, so daß die Anwendung die Funktion automatisch beenden kann.

Siehe auch

"Anzeigen von Meldungen in der Befehlszeile"

13.150 princ

Druckt einen Ausdruck in die Befehlszeile oder schreibt einen Ausdruck in eine geöffnete Datei.

(princ *[Ausdruck [Dateideskr]]*)

Diese Funktion entspricht der Funktion **prin1**, die Steuerzeichen im Argument Ausdruck werden jedoch ohne Erweiterung gedruckt.

Generell ist die Funktion **prin1** dafür bestimmt, Ausdrücke in einer Weise zu schreiben, die mit der Funktion **load** kompatibel ist, während die Funktion **princ** sie so druckt, daß sie für Funktionen wie **read-line** lesbar ist.

13.151 print

Druckt einen Ausdruck in die Befehlszeile oder schreibt einen Ausdruck in eine geöffnete Datei.

(print ***[Ausdruck [Dateideskr]]*****)**

Diese Funktion entspricht der Funktion **prin1**, druckt jedoch vor dem Argument Ausdruck ein Zeilenumbruchzeichen und nach Ausdruck ein Leerzeichen.

13.152 progn

Wertet jeden Ausdruck der Reihe nach aus und gibt den Wert des letzten Ausdrucks zurück.

(progn ***[Ausdruck]*** **...)**

Sie können die Funktion **progn** verwenden, um mehrere Ausdrücke auszuwerten, wo nur ein Ausdruck erwartet wird.

```
(if (= a b)
  (progn
    (princ "\nA = B ")
    (setq a (+ a 10) b (- b 10))
  )
)
```

Die Funktion **if** wertet normalerweise einen Ausdruck *dann* aus, wenn der Test-Ausdruck einen Wert ergibt, der von `nil` verschieden ist. Im genannten Beispiel wird die Funktion **progn** benutzt, um statt dessen zwei Ausdrücke auszuwerten.

13.153 prompt

Zeigt im Eingabeaufforderungsbereich Ihres Bildschirms eine Zeichenkette an.

(prompt ***Anfrage*****)**

Bei AutoCAD-Konfigurationen mit zwei Bildschirmen zeigt **prompt** die *`Anfrage`* auf beiden Bildschirmen und ist daher der Funktion **princ** vorzuziehen.

`(prompt "Neuer Wert: ")` *zeigt* `Neuer Wert:` *auf dem/den Bildschirm(en) an*

Die Funktion **prompt** ergibt `nil`.

13.154 quit

Erzwingt die Beendigung der aktuellen Anwendung.

(quit)

Wird die Funktion **quit** aufgerufen, so wird die Fehlermeldung Abbruch ausgegeben und zur AutoCAD-Eingabeaufforderung zurückgekehrt.

Siehe auch

Funktion **exit**

13.155 quote

Gibt einen Ausdruck zurück, ohne ihn ausgewertet zu haben.

(quote ***Ausdruck*****)**

Dies kann auch folgendermaßen geschrieben werden

```
'expr
(quote a)                ergibt  A
(quote cat)              ergibt  CAT
(quote (a b))            ergibt  (A B)
'a                       ergibt  A
'cat                     ergibt  CAT
'(a b)                   ergibt  (A B)
```

Die letzten drei Beispiele können nicht über die Tastatur als direkte Antwort auf eine Eingabeaufforderung von AutoCAD eingegeben werden. Beachten Sie, daß derartige Eingaben mit dem Zeichen (oder ! beginnen müssen, um als AutoLISP-Ausdruck erkannt zu werden.

13.156 read

Gibt die erste Liste oder das erste Atom zurück, das von einer Zeichenkette übertragen wurde.

(read ***[Z_kette]*****)**

Das Argument `Z_kette` darf außer in einer Liste oder Zeichenkette keine Leerzeichen enthalten. Die Funktion **read** gibt das in den entsprechenden Datentyp konvertierte Argument zurück:

```
(read "hallo")                ergibt das Atom        HALLO
(read "hallo ih")             ergibt das Atom        HALLO
(read "\"Hi Ihr alle\"")      ergibt die Zeichenkette        "Hi Ihr alle"
(read "(a b c)")              ergibt die Liste               (A B C)
(read "(a b c) (d)")          ergibt die Liste               (A B C)
(read "1.2300")               ergibt die reale Zahl  1.23
(read "87")                   ergibt die ganze Zahl  87
(read "87 3.2")               ergibt die ganze Zahl  87
```

13.157 read-char

Gibt den dezimalen ASCII-Code zurück, der das Zeichen aus dem Tastatureingabepuffer oder aus einer geöffneten Datei darstellt.

(read-char ***[Dateideskr]*****)**

Wenn kein `Dateideskr` angegeben wird und sich keine Zeichen im Tastatureingabepuffer befinden, wartet **read-char** auf eine Tastatureingabe (gefolgt von RETURN). Nimmt man zum Beispiel an, daß der Tastatureingabepuffer leer ist, wartet die Funktion

```
(read-char)
```

auf eine Eingabe. Wenn Sie ABC gefolgt von RETURN eingeben, gibt die Funktion **read-char** 65 (den dezimalen ASCII-Code des Buchstabens A) zurück. Die folgenden drei Aufrufe der Funktion **read-char** geben jeweils 66, 67 und 10 (Zeilenvorschub) zurück. Erfolgt dann noch ein weiterer Aufruf der Funktion **read-char**, erwartet sie wieder eine Eingabe.

Die verschiedenen Betriebssysteme, unter denen AutoCAD läuft, verwenden unterschiedliche Konventionen, um in einer ASCII-Textdatei das Zeilenende kenntlich zu machen. So verwenden UNIX-basierte Systeme zum Beispiel ein einzelnes Zeilenumbruchzeichen (Zeilenvorschub [LF], ASCII-Code 10), während DOS-basierte Systeme zum gleichen Zweck eine Zeichenkombination benutzen (Wagenrücklauf [CR]/LF, ASCII-Codes 13 und 10). Um die Entwicklung von AutoLISP-Programmen zu erleichtern, akzeptiert die Funktion **read-char** alle diese Konventionen und gibt immer ein einziges Zeilenumbruchzeichen (ASCII-Code 10) zurück, wenn sie ein Zeilenendezeichen (oder eine entsprechende Zeichenfolge) erkennt.

13.158 read-line

Liest eine Zeichenkette von der Tastatur oder aus einer geöffneten Datei.

(read-line ***[Dateideskr]*****)**

Wenn die Funktion **read-line** das Ende der Datei findet, gibt sie `nil` zurück, andernfalls die gelesene Zeichenkette.

Angenommen, `f` ist eine gültige Dateihinweismarke, dann gibt die Funktion

```
(read-line f)
```

die nächste Eingabezeile der Datei zurück. Wenn das Dateiende erreicht ist, gibt sie `nil` zurück.

13.159 redraw

Zeichnet ein aktuelles Ansichtsfenster oder ein definiertes Objekt (Element) im aktuellen Ansichtsfenster neu.

(redraw ***[Elem_name [Modus]]*****)**

Die Auswirkung der Funktion **redraw** hängt von den angegebenen Argumenten ab. Wird sie ohne jedes Argument aufgerufen, zeichnet sie das aktuelle Ansichtsfenster neu. Wird sie mit dem Argument *Elem_name* (Elementname) aufgerufen, zeichnet sie das definierte Element neu. Das Argument *Modus* ist ein ganzzahliger Wert, der die Sichtbarkeit und das Markieren des Elements steuert.

Modi der Funktion redraw

redraw-Modus	Operation
1	Element anzeigen
2	Element verbergen (ausblenden)
3	Element markieren
4	Markierung eines Elementes aufheben

Ist das Argument *Elem_name* der Header eines komplexen Elements (eine Polylinien- oder Blockreferenz mit Attributen), verarbeitet es das Hauptelement und alle seine Subelemente, wenn das Argument *Modus* positiv ist. Ist das Argument *Modus* negativ, berücksichtigt die Funktion **redraw** nur das Header-Element.

Die Funktion **redraw** gibt immer `nil` zurück.

Anmerkung Die Markierung von Elementen (Modus 3) muß in Abstimmung zum Rückgängigmachen der Markierungen (Modus 4) verwendet werden.

Der Befehl NEUZEICH hat keine Wirkung auf markierte oder ausgeblendete Elemente, der Befehl REGEN erzwingt jedoch das Neuzeichnen der Elemente in ihrer normalen Darstellung.

13.160 regapp

Registriert einen Anwendungsnamen mit der aktuellen AutoCAD-Zeichnung, um die Verwendung erweiterter Objektdaten vorzubereiten.

(regapp ***Anwendung*****)**

Das Argument *Anwendung* ist eine bis zu 31 Zeichen lange Zeichenkette, die den Konventionen der Symbolnamengebung entspricht. Ein Anwendungsname kann Buchstaben, Ziffern und die Sonderzeichen Dollarzeichen ($), Bindestrich (-) und Unterstrich (_) enthalten. Er darf keine Leerzeichen enthalten. Buchstaben im Namen werden in Großbuchstaben umgewandelt.

Ist unter dem gleichen Namen bereits eine Anwendung registriert, gibt diese Funktion `nil` zurück, andernfalls gibt sie den Namen der Anwendung zurück.

Nach erfolgreicher Registrierung wird der Anwendungsname in die Symboltabelle APPID eingetragen. Diese Tabelle enthält eine Liste der Anwendungen, die in der Zeichnung erweiterte Daten benutzen.

```
(regapp "ADESK_4153322344")
(regapp "DESIGNER-v2.1-124753")
```

Anmerkung Sie sollten einen eindeutigen Anwendungsnamen wählen. Dies können Sie mit einem Benennungsschema sicherstellen, das den Firmen- oder Produktnamen und eine eindeutige Zahl verwendet (zum Beispiel Ihre Telefonnummer oder das aktuelle Datum/die aktuelle Uhrzeit). Die Nummer der Produktversion kann in den Anwendungsnamen integriert sein oder von der Anwendung in einem separaten, ganze oder reale Zahlen umfassenden Feld gespeichert werden, zum Beispiel `(1040 2.1)`.

13.161 rem

Dividiert die erste Zahl durch die zweite und gibt den Rest zurück.

(rem ***Zahl Zahl*** **...)**

Bei mehr als zwei Zahlen dividiert die Funktion **rem** das Ergebnis der Division von erster durch zweite Zahl durch die dritte usw.

```
(rem 42 12)          ergibt  6
(rem 12.0 16)         ergibt  12.0
(rem 26 7)           ergibt  5
(rem 5 2)            ergibt  1
(rem 26 7 2)         ergibt  1
```

13.162 repeat

Wertet jeden Ausdruck so oft wie angegeben aus und gibt den Wert des letzten Ausdrucks zurück.

(repeat ***Ganzzahl Ausdruck*** **...)**

Das Argument *Ganzzahl* muß positiv sein.

```
(setq a 10 b 100)
(repeat 4
            (setq a (+ a 10))
            (setq b (+ b 100))
)           Setzt a auf 50, setzt b auf 500, und gibt 500  zurück
```

13.163 reverse

Gibt eine Liste mit der Umkehrung ihrer Elemente zurück. (reverse ***Liste*****)**

```
(reverse '((a) b c))          ergibt    (C B (A))
```

13.164 rtos

Konvertiert eine Zahl in eine Zeichenkette.

(rtos ***Zahl [Modus [Genauigkeit]]*****)**

Die Funktion **rtos** gibt eine Zeichenkette als Darstellung von *Zahl* entsprechend der Einstellungen von *Modus*, *Genauigkeit*, sowie der Systemvariablen UNITMODE und DIMZIN zurück. Die Argumente *Modus* und *Genauigkeit* sind Ganzzahlen, mit denen der Modus und die Genauigkeit der linearen Einheiten gewählt werden, wie in der folgenden Tabelle gezeigt.

Werte für lineare Einheiten

Moduswert	Zeichenkettenformat
1	Wissenschaftliche Schreibweise
2	Dezimalschreibweise
3	Engineering (Fuß und Zoll in Dezimalschreibweise)
4	Architectural (Fuß und Zoll in Bruchschreibweise)
5	Bruchschreibweise

Die Argumente *Modus* und *Genauigkeit* entsprechen den Systemvariablen LUNITS und LUPREC. Wenn Sie diese Argumente weglassen, verwendet **rtos** die aktuellen Einstellungen von LUNITS und LUPREC. Die Variable UNITMODE hat dann Auswirkungen auf die zurückgegebene Zeichenkette, wenn die Engineering- oder Architectural-Schreibweise oder die Bruchschreibweise ausgewählt werden (ein Wert des Arguments *Modus* von 3, 4 oder 5).

Siehe auch

"Zeichenkettenkonvertierung," wo die Erläuterung von **rtos** fortgesetzt wird.

13.165 set

Ordnet den Wert eines in Anführungszeichen angegebenen Symbolnamens einem Ausdruck zu.

(set ***Symbol Ausdruck*****)**

Gibt den Wert des Ausdrucks zurück.

`(set 'a 5.0)`	*ergibt*	`5.0`	*und setzt das Symbol*	`A`
`(set (quote b) 'a)`	*ergibt*	`A`	*und setzt das Symbol*	`B`

Wird die Funktion **set** mit einem nicht in Anführungszeichen angegebenen Symbolnamen verwendet, kann sie einem anderen Symbol *indirekt* einen neuen Wert zuweisen. Zum Beispiel:

`(set b 640)` *ergibt* `640`

und ordnet den Wert 640 dem Symbol *a* zu, da dies der Inhalt von Symbol `b` ist.

Siehe auch

Funktion **setq**

13.166 set_tile

Setzt den Wert einer Dialogfeldkomponente fest.

(set_tile ***Schlüssel Wert*****)**

Das Argument *`Schlüssel`* ist eine Zeichenkette, welche die Komponente definiert, und das Argument *`Wert`* ist eine Zeichenkette, die den neuen, zuzuweisenden Wert (ursprünglich durch das Attribut `Wert` gesetzt) angibt.

13.167 setcfg

Schreibt Anwendungsdaten in den Abschnitt AppData der Datei *acad.cfg*.

(setcfg ***cfg_Name cfg_Wert*****)**

Das Argument *`cfg_Name`* ist eine Zeichenkette (maximal 132 Zeichen lang), die den Abschnitt und den Parameter definiert, welche den Wert des Arguments *`cfg_Wert`* (maximal 347 Zeichen lang) festlegen. Wenn *`cfg_Name`* nicht gültig ist, gibt **setcfg** den Wert `nil` zurück. Das Argument *`cfg_Name`* muß eine Zeichenkette der folgenden Form sein:

`"AppData/`*Anwendungsname*`/`*Abschnittsname*`/.../`*Param_name*`"`

Der folgende Code setzt den Parameter WallThk im Abschnitt AppData/ArchStuff auf 8 und gibt die Zeichenkette "8" zurück:

`(setcfg "AppData/ArchStuff/WallThk" "8")` *ergibt* `"8"`

Siehe auch

Funktion **getcfg**

13.168 setfunhelp

Registriert einen benutzerdefinierten Befehl mit der Hilfefunktion, so daß die entsprechenden Hilfedateien und Hilfethemen aufgerufen werden, wenn der Benutzer Hilfe zu diesem Befehl anfordert.

(setfunhelp ***c:dname [Hilfedatei [Thema [Befehl]]]*****)**

Das Argument *`c:dname`* ist eine Zeichenkette, die den vom Benutzer definierten Befehl angibt (die Funktion **C:*XXX***) und muß das Präfix **c:** enthalten. Anders als der an die Funktion **defun** übergebene Funktionsname muß das Argument *`Funktion`* für **setfunhelp** eine Zeichenkette in Anführungszeichen sein. Die verbleibenden drei optionalen Argumente beschreiben den Hilfeaufruf. Sie sind ebenfalls Zeichenketten und entsprechen den Argumenten, die der Funktion **help** übergeben werden. Bei erfolgreicher Durchführung gibt die Funktion **setfunhelp** die Zeichenkette zurück, die als *`c:dname`* übergeben wurde, andernfalls gibt sie `nil` zurück.

Die Dateinamenerweiterung ist beim Argument `Hilfedatei` nicht erforderlich. Wird eine Dateinamenerweiterung angegeben, sucht AutoCAD nur nach dieser Datei. Erfolgt dies nicht, gelten folgende Regeln für die Suche: Arbeiten Sie mit AutoCAD für Windows/NT, geben Sie . *.klp* an, ansonsten verwenden Sie *.ahp*. Wenn kein *<Dateiname>.ahp* gefunden wird, wird der *<Dateiname>* ohne Dateierweiterung gesucht. Beachten Sie, daß nach der Datei ohne Erweiterung zuletzt gesucht wird, in UNIX wird daher vor *acad* zunächst nach *acad.ahp* gesucht.

Beachten Sie, daß diese Funktion nur überprüft, ob das Argument `c:dname` das Präfix `c:` hat. Sie überprüft *nicht*, ob die Funktion `c:dname` vorhanden ist, und auch nicht die Richtigkeit der anderen Argumente.

Wenn Sie die Funktion **defun** verwenden, um eine Funktion **C:*XXX*** zu definieren, entfernt diese den Namen der Funktion aus den Funktionen, die von **setfunhelp** registriert wurden (falls vorhanden). Deshalb sollte **setfunhelp** erst nach dem Aufruf der Funktion **defun**, die den benutzerdefinierten Befehl festlegt, aufgerufen werden.

Im folgenden Beispiel wird **defun** zur Definition des benutzerdefinierten Befehls MEINEFUN verwendet. Die Funktion **setfunhelp** dient zur Registrierung des Funktionsnamens **C:MEINEFUN** und verbindet den Namen mit dem Thema `meinefun` in der Datei *mhilfe.ahp*.

```
(defun c:meinefun ()
  ...
  (getint "gimme: ")
  ...
)
(setfunhelp "c:meinefun" "mhilfe.ahp" "meinefun")
```

Befehl: **meinefun**

gimme: **'help**

Angenommen, die AutoCAD-Hilfedatei *mhilfe.ahp* existiert im Verzeichnis Support. AutoCAD zeigt das Hilfe-Dialogfeld mit dem Thema meinefun aus der Datei *meineanw.ahp* an.

Siehe auch

Funktionen **defun** und **help**

13.169 setq

Ordnet den Wert eines oder mehrerer Symbole den zugehörigen Ausdrücken zu.

(setq ***Symbol1 Ausdruck1 [Symbol2 Ausdruck2]*** **...)**

Dies ist die Grundzuordnungsfunktion von AutoLISP. Die Funktion **setq** kann im Laufe eines einzigen Funktionsaufrufs mehrere Symbole zuordnen, gibt jedoch nur das letzte Argument `Ausdruck` zurück.

```
(setq a 5.0)              ergibt  5.0
```

und setzt das Symbol `a` auf 5.0. Immer, wenn `a` berechnet wird, wird die reale Zahl 5.0 ausgewertet.

```
(setq b 123 c 4.7)        ergibt  4.7
(setq s "it")             ergibt  "it"
(setq x '(a b))           ergibt  (A B)
```

Siehe auch

"AutoLISP-Variablen"

13.170 setvar

Setzt eine AutoCAD-Systemvariable auf einen definierten Wert.

(setvar ***Var_name Wert*****)**

Wird die Funktion **setvar** erfolgreich ausgeführt, gibt sie den Wert der Systemvariablen zurück. Sie müssen den Variablennamen in Anführungszeichen setzen.

```
(setvar "FILLETRAD" 0.50) ergibt  0.5
```

und setzt den Rundungsradius in AutoCAD auf 0.5 Einheiten. Für Systemvariablen mit ganzzahligen Werten muß das angegebene Argument `Wert` zwischen -32,768 und +32,767 liegen.

Einige AutoCAD-Befehle erhalten Systemvariablenwerte, bevor Eingabeaufforderungen ausgegeben werden. Wenn Sie mit der Funktion **setvar** einen neuen Wert setzen, während ein Befehl ausgeführt wird, wird der neue Wert eventuell erst beim nächsten AutoCAD-Befehl wirksam.

Anmerkung Wenn Sie den Befehl **setvar** verwenden, um die AutoCAD-Systemvariable ANGBASE zu ändern, wird das Argument Wert im Bogenmaß ausgewertet. Dies ist anders als beim AutoCAD-Befehl SETVAR, der dieses Argument als Grad interpretiert. Wenn Sie mit der Funktion **setvar** die AutoCAD-Systemvariable SNAPANG ändern, wird das Argument Wert entsprechend der von AutoCAD vorgegebenen Richtung für den Winkel 0, der als *Osten* oder *3 Uhr* definiert ist, als Bogenmaß interpretiert. Dies ist auch anders als beim Befehl SETVAR,der dieses Argument entsprechend der Einstellung von ANGBASE als Grad interpretiert.

Anmerkung Der Befehl ZURÜCK macht keine Änderungen rückgängig, die für die Systemvariable CVPORT mit der Funktion **setvar** durchgeführt wurden.

Eine Liste der aktuellen AutoCAD-Systemvariablen finden Sie in "Systemvariablen" in der *AutoCAD Befehlsreferenz*.

Siehe auch

Funktion **getvar**

13.171 sin

Gibt den Sinus eines Winkels als reale Zahl in Bogenmaß zurück.

(sin *Winkel*)

Das Argument `Winkel` muß ein in Bogenmaß ausgedrückter Winkel sein.

`(sin 1.0)`	*ergibt*	`0.841471`
`(sin 0.0)`	*ergibt*	`0.0`

13.172 setview

Erstellt eine Ansicht für ein angegebenes Ansichtsfenster

(setview *Ansichtsbeschr [Ansichtsfens_bez]*)

Das Argument `Ansichtsbeschr` ist eine Elementdefinitionsliste, die der von der Funktion **tblsearch** zurückgegebenen entspricht, wenn sie auf die Symboltabelle VIEW angewendet wird. Das fakultative Argument `Ansichtsfens_bez` kennzeichnet, welches Ansichtsfenster die neue Ansicht erhalten soll. Die `Ansichtsfens_bez`-Nummer kann mit der Systemvariablen CVPORT abgerufen werden. Wenn `Ansichtsfens_bez` 0 ist, erhält das aktuelle Ansichtsfenster die neue Ansicht. Bei erfolgreicher Durchführung gibt die Funktion **setview** die `Ansichtsbeschr` zurück.

13.173 slide_image

Zeigt ein AutoCAD-Dia in der aktuellen Dialogfeldbildkomponente an.

(slide_image *X1 Y1 Breite Höhe Dianame*)

Die Dias können entweder Diadateien (*.sld*) oder Teile einer Diabibliotheksdatei (*.slb*) sein: Das Argument `Dianame` gibt es an, wie Sie es für den Befehl ZEIGDIA oder eine Menü-Datei angeben würden (siehe "Erstellen von Bildern"):

`Dianame` *oder* `Biblname(Dianame)`

Die erste Ecke (oben links) des Dias - sein Einfügepunkt - liegt bei (`X1`,`Y1`), und die zweite Ecke (rechts unten) liegt in der relativen Entfernung (`Breite`,`Höhe`) von der ersten Ecke (`Breite` und `Höhe` müssen positive Werte sein). Der Ursprung (0,0) ist die obere linke Bildecke. Die Koordinaten der unteren rechten Ecke erhalten Sie, indem Sie die Bemaßungsfunktionen aufrufen (**dimx_tile** und **dimy_tile**).

13.174 snvalid

Sucht im Symboltabellennamen nach gültigen Zeichen.

(snvalid *Symbolname [Flag]*)

Das Argument *Symbolname* ist eine Zeichenkette, die einen Symboltabellennamen festlegt. Das optionale Argument *Flag* gibt an, ob vertikale Balken-Zeichen in *Symbolname* zulässig sind, oder nicht. Das Argument *Flag* kann 1 oder 0 sein (Voreinstellung ist 0). Die Funktion **snvalid** gibt T zurück, wenn *Symbolname* ein gültiger Symboltabellenname ist, andernfalls gibt sie nil zurück.

Symboltabellennamen dürfen nur aus alphanumerischen Zeichen und den Sonderzeichen Dollar ($), Unterstrich (_), und Bindestrich (-) bestehen. Eine leere Zeichenkette ist kein gültiger Name.

13.175 sqrt

Gibt die Quadratwurzel einer Zahl als reale Zahl zurück.

(sqrt *num*)

(sqrt 4)	*ergibt*	2.0
(sqrt 2.0)	*ergibt*	1.41421

13.176 ssadd

Fügt einem Auswahlsatz ein Objekt (Element) hinzu oder erstellt einen neuen Auswahlsatz.

(ssadd *[Elem_name [Ausw_satz]]*)

Wird die Funktion **ssadd** ohne Argument aufgerufen, erstellt sie einen neuen leeren Auswahlsatz. Wird sie mit dem Argument *Elem_name* aufgerufen, das nur einen einzelnen Elementnamen darstellt, erstellt die Funktion **ssadd** einen neuen Auswahlsatz, der nur dieses eine Element enthält. Wird sie mit einem Elementnamen und dem Argument *Ausw_satz* aufgerufen, fügt die Funktion **ssadd** das genannte Element dem Auswahlsatz hinzu. Die Funktion **ssadd** gibt immer den neuen oder geänderten Auswahlsatz zurück.

Wenn Sie einem Satz ein neues Element hinzufügen, wird das neue Element einem bestehenden Satz hinzugefügt, und der als Argument *Ausw_satz* angegebene Satz wird als Ergebnis zurückgegeben. Wird der Satz also anderen Variablen zugeordnet, spiegeln diese auch die Erweiterung des Satzes wider. Ist das genannte Element in dem Satz bereits enthalten, wird die Operation der Funktion **ssadd** ignoriert, und es erfolgt keine Fehlermeldung.

(setq e1 (entnext))	*Setzt* e1 *auf den Namen des ersten Elementes in der Zeichnung*
(setq ss (ssadd))	*Setzt* ss *auf einen leeren Auswahlsatz*
(ssadd e1 ss)	*ergibt* ss *mit dem Elementnnamen* e1 *hinzugefügt*
(setq e2 (entnext e1))	*Ermittelt das auf* e1 *folgende Element*
(ssadd e2 ss)	*ergibt* ss *mit dem Elementnnamen* e2 *hinzugefügt*

13.177 ssdel

Löscht ein Objekt (Element) aus einem Auswahlsatz.

(ssdel *Elem_name Ausw_satz*)

Die Funktion **ssdel** löscht das Element *Elem_name* aus dem Auswahlsatz *Ausw_satz* und gibt den Namen des Auswahlsatzes *Ausw_satz* zurück. Beachten Sie, daß das Element tatsächlich aus dem Auswahlsatz gelöscht wird. Es wird nicht ein neuer Satz zurückgegeben, in dem das Element gelöscht ist. Wenn das Element nicht im Satz vorhanden ist, wird nil zurückgegeben.

Nimmt man zum Beispiel an, daß der Elementname e1 im Auswahlsatz Elem_name vorhanden ist und Elementname e2 nicht, dann ergibt

(ssdel e1 Elem_name)	*den Auswahlsatz* Elem_name *ohne Element* e1
(ssdel e2 Elem_name)	nil *(keine Änderung von* Elem_name)

13.178 ssget

Fordert den Benutzer auf, Objekte (Elemente) zu wählen, und gibt einen Auswahlsatz zurück.

(ssget ***[Modus] [P1 [P2]] [P_liste] [Filterliste]*****)**

Das Argument *Modus* ist eine Zeichenkette, welche die Objektauswahlmethode definiert. Gültige Modi sind "F", "PP", "K", "KP", "L", "V", "I" und "Z", die den Auswahlmethoden Fenster, FPolygon, Kreuzen, (KPolygon), Letztes, Vorher, Impliziert und Zaun entsprechen. Ein weiterer optionaler Wert für *Modus* ist "X", wodurch die gesamte Datenbank ausgewählt wird. Die Argumente *P1* und *P2* definieren die für die Auswahl wichtigen Punkte. Wird ein Punkt ohne das Argument *Modus* angegeben, entspricht dies der Objektwahl mit einem einzelnen Punkt. Die aktuelle Festlegung des Ofang-Modus wird von dieser Funktion ignoriert, es sei denn, Sie fordern dies ausdrücklich an, wenn Sie mit der Funktion arbeiten. Das Argument *Filterliste* ist eine Assoziationsliste, die Objekteigenschaften definiert. Objekte, die dem Argument *Filterliste* entsprechen, werden dem Auswahlsatz hinzugefügt. Werden alle Argumente ausgelassen, fordert die Funktion **ssget** den Benutzer mit der Eingabeaufforderung Objekte wählen zur Eingabe auf, was einen interaktiven Aufbau des Auswahlsatzes ermöglicht.

Auswahlsätze können sowohl Objekte aus dem Papierbereich als auch aus dem Modellbereich enthalten, wenn der Auswahlsatz jedoch in einer Operation eingesetzt wird, werden Objekte des momentan nicht aktiven Bereichs herausgefiltert. Von der Funktion **ssget** zurückgegebene Auswahlsätze enthalten nur Hauptelemente (keine Attribute oder Polylinien-Kontrollpunkte).

`(ssget)`	*Fordert den Benutzer zu einer allgemeinen Objektwahl auf und nimmt diese Objekte in einen Auswahlsatz auf*
`(ssget "P")`	*Erzeugt einen Auswahlsatz mit den zuletzt gewählten Objekten*
`(ssget "L")`	*Erzeugt einen Auswahlsatz mit dem zuletzt sichtbaren Objekt, das zur Datenbank hinzugefügt wurde*
`(ssget "I")`	*Erzeugt einen Auswahlsatz der Objekte im implizierten Auswahlsatz (die mit eingeschalteter Funktion PICKFIRST ausgewählt wurden)*
`(ssget '(2 2))`	*Erzeugt einen Auswahlsatz des Objektes, das durch den Punkt (2,2) verläuft*
`(ssget "W" '(0 0) '(5 5))`	*Erzeugt einen Auswahlsatz der Objekte innerhalb des Fensters von (0,0) bis (5,5)*
`(ssget "C" '(0 0) '(1 1))`	*Erzeugt einen Auswahlsatz der Objekte, die das Feld von (0,0) bis (1,1) kreuzen*
`(ssget "X")`	*Erzeugt einen Auswahlsatz aller Objekte in der Datenbank*
`(ssget "X" Filterliste)`	*Scannt die Datenbank und erzeugt einen Auswahlsatz von Objekten, welche dem Argument Filterliste entsprechen*
`(ssget Filterliste)`	*Fordert den Benutzer zu einer allgemeinen Objektauswahl auf und nimmt nur die Objekte in einen Auswahlsatz auf, die dem Argument Filterliste entsprechen*
`(ssget "P" Filterlist)`	*Erzeugt einen Auswahlsatz mit den zuletzt gewählten Objekten, welche dem Argument Filterliste entsprechen*

Die folgenden Beispiele für **ssget** erfordern es, daß eine Liste von Punkten an die Funktion übergeben wird. Die Variable `P_liste` darf keine Punkte enthalten, durch die Segmente der Länge Null definiert werden.

```
(setq pt_list '((1 1)(3 1)(5 2)(2 4)) )
```

`(ssget "WP" pt_list)`	*Erzeugt einen Auswahlsatz aller Objekte in dem durch* `P_liste` *definierten Polygon*
`(ssget "CP" pt_list)`	*Erzeugt einen Auswahlsatz aller Objekte, die das durch* `P_liste` *definierten Polygon kreuzen und in ihm enthalten sind*
`(ssget "F" pt_list)`	*Erzeugt einen Auswahlsatz aller Objekte*

	die den durch `P_list` *definierten Zaun schneiden*
`(ssget "WP" P_liste Filterliste)`	*Erzeugt einen Auswahlsatz aller Objekte in dem durch* `P_liste` *definierten Polygon, die dem Argument Filterliste entsprechen*

Die gewählten Objekte werden nur dann markiert, wenn die Funktion **ssget** ohne Argumente benutzt wird. Auswahlsätze verbrauchen Temporärdateiplätze von AutoCAD, so daß AutoLISP nicht mehr als 128 Auswahlsätze gleichzeitig geöffnet haben darf. Wenn diese Grenze erreicht wird, weigert sich AutoCAD, weitere Auswahlsätze zu erzeugen und gibt `nil` an alle Aufrufe von **ssget** zurück. Möchten Sie einen nicht mehr benötigten Auswahlsatz schließen, setzen Sie ihn auf `nil`.

Die Variable eines Auswahlsatzes kann AutoCAD als Antwort auf eine beliebige Eingabeaufforderung Objekte wählen übergeben werden, bei der die Auswahl durch "Letztes" gültig ist. Dann werden alle Objekte in der Auswahlsatzvariablen gewählt.

13.178.1 Auswahlsatzfilter

Auswahlsatzfilterlisten können mit jedem der Auswahlmodi verwendet werden. Sie können einen Auswahlsatz erhalten, der alle Objekte eines bestimmten Typs auf einem bestimmten Layer oder in einer bestimmten Farbe einschließt.

Das folgende Beispiel gibt einen Auswahlsatz zurück, der nur aus blauen Linien besteht, die zum Auswahlsatz "Implizit" gehören (Objekte, die gewählt werden, während die Systemvariable PICKFIRST aktiv ist):

```
(ssget "I" '((0 . "LINE") (62 . 5)))
```

Mit der Filterliste der Funktion **ssget** können Sie alle Objekte wählen, die erweiterte Daten für eine bestimmte Anwendung enthalten. Dazu benutzen Sie den Gruppencode -3, wie in dem folgenden Beispiel gezeigt.

```
(ssget "V" '((0 . "CIRCLE") (-3 ("ANWNAME"))))
```

Hier werden alle Kreise gewählt, die Daten für die Anwendung `"ANWNAME"` enthalten.

Weitere Informationen finden Sie unter "Auswahlsatz-Filterlisten" und "Suchen nach erweiterten Daten (Extended Data)."

13.178.1.1 Relationale Tests

Soweit nicht anders definiert, wird für jedes Element der *`Filterliste`* ein relationaler Test angenommen. Für numerische Gruppen (Ganzzahlen, reale Zahlen, Punkte und Vektoren) können Sie andere Relationen definieren, indem Sie einen -4-Gruppencode einbeziehen, der einen relationalen Operator definiert. Der Wert einer -4-Gruppe ist eine Zeichenkette, die den Testoperator anzeigt, der auf die nächste Gruppe in der Filterliste angewendet werden soll.

```
(ssget "X" '((0 . "CIRCLE") (-4 . ">=") (40 . 2.0)))
```

wählt alle Kreise, deren Radius (Gruppencode 40) größer oder gleich 2.0 ist.

Die folgende Tabelle zeigt die möglichen Operatoren.

Relationale Operatoren für die Auswahlsatz-Filterlisten

Operator	Beschreibung
`"*"`	Alles möglich (immer richtig)
`"="`	Gleich
`"!="`	Ungleich
`"/="`	Ungleich
`"<>"`	Ungleich
`"<"`	Kleiner
`"<="`	Kleiner als oder gleich
`">"`	Größer als
`">="`	Größer als oder gleich
`"&"`	Bitweise AND (nur für Gruppen mit Ganzzahlen)
`"&="`	Bitweise mit der Maske übereinstimmend (nur für Gruppen

mit Ganzzahlen)

Der Gebrauch von relationalen Operatoren hängt von der Art der Gruppe ab, die Sie testen:

- Alle relationalen Operatoren, mit Ausnahme der bitweise arbeitenden (`"&"` und `"&="`), sind sowohl für die Gruppen mit realen als auch für die mit ganzen Zahlen gültig.
- Die bitweise arbeitenden Operatoren `"&"` und `"&="` sind nur für Gruppen mit ganzen Zahlen gültig. Das bitweise AND, `"&"`, ist richtig, wenn (*(Ganzzahlgruppe & Filter) /= 0*) gilt - das heißt, wenn beliebige in der Maske gesetzte Bits auch in der Ganzzahlgruppe gesetzt sind. Der Operator "Bitweise mit der Maske übereinstimmend", `"&="`, ist richtig, wenn (*(Ganzzahlgruppe & Filter) = Filter*) - das heißt, wenn alle in der Maske gesetzten Bits auch im Argument `Ganzzahlgruppe` gesetzt sind. (Im Argument `Ganzzahlgruppe` können noch weitere Bits gesetzt sein, diese werden aber nicht überprüft.)
- Für Gruppen, die Punkte enthalten, können die *X*-, *Y*- und *Z*- Tests in einer einzigen Zeichenkette zusammengefaßt werden, wobei die einzelnen Operatoren durch Kommata getrennt sind (zum Beispiel `">,>,*"`). Wird ein Operator aus der Zeichenkette ausgelassen, (`"=,<>"` umgeht zum Beispiel den *Z*-Test), wird der Operator "alles möglich", `"*"`, angenommen.
- Richtungsvektoren (Gruppen-Typ 210) können nur mit den Operatoren `"*"`, `"="` und `"!="` verglichen werden (oder mit einer der äquivalenten Zeichenketten für "ungleich").
- Sie können die relationalen Operatoren nicht mit Zeichenkettengruppen verwenden; statt dessen müssen Sie Platzhalter-Tests benutzen.

13.178.1.2 Logische Gruppierung von Filtertests

Die oben beschriebenen relationalen Operatoren sind binäre Operatoren. Sie können Gruppen auch testen, indem Sie verschachtelte Boolesche Ausdrücke erstellen, welche die in der folgenden Tabelle beschriebenen Gruppierungsoperatoren einsetzen. Die Gruppierungsoperatoren werden, wie die relationalen Operatoren, durch -4-Gruppen spezifiziert. Sie sind zu Paaren zusammengefaßt und müssen in der Filterliste im richtigen Verhältnis stehen, oder der Aufruf der Funktion **`ssget`** schlägt fehl. Die Zahl der Operanden, die diese Operatoren umfassen können, hängt von der jeweiligen Operation ab.

Gruppierungsoperatoren für die Auswahlsatz-Filterlisten

Anfangs-Operator	Umfaßt	Schluß-Operator
"<AND"	Einen oder mehr Operanden	"AND>"
"<OR"	Einen oder mehr Operanden	"OR>"
"<XOR"	Zwei Operanden	"XOR>"
"<NOT"	Einen Operanden	"NOT>"

Im Zusammenhang mit den Gruppierungsoperatoren verstehen wir unter einem *Operanden* eine Elementfeldgruppe, einen relationalen Operator gefolgt von einer Elementfeldgruppe oder einen von diesen Operatoren geschaffenen verschachtelten Ausdruck. Es folgt ein Beispiel für den Einsatz von Gruppierungsoperatoren in einer Filterliste:

```
(ssget "X" '((-4 . "<OR")
             (-4 . "<AND")
             (0 . "CIRCLE")
             (40 . 1.0)
             (-4 . "AND>")
             (-4 . "<AND")
             (0 . "LINE")
             (8 . "ABC")
             (-4 . "AND>")
             (-4 . "OR>"))
)
```

Auf diese Weise werden alle Kreise mit dem Radius 1.0 und alle Linien auf dem Layer `"ABC"` gewählt.

Da die Gruppierungsoperatoren nicht zwischen Groß- und Kleinschreibung unterscheiden, können Sie auch kleine Buchstaben benutzen: `"<and"`, `"and>"`, `"<or"`, `"or>"`, `"<xor"`, `"xor>"`, `"<not"`, und `"not>"`.

13.179 ssgetfirst

Legt fest, welche Objekte ausgewählt und mit Griffen versehen werden.

(ssgetfirst)

Liefert eine Liste von zwei Auswahlsätzen, die denen ähneln, die **sssetfirst** übergeben wurden. Das erste Element in der Liste stellt einen Auswahlsatz von Elementen dar, die mit Griffen versehen, jedoch nicht ausgewählt wurden. Das zweite Listenelement entspricht einem Auswahlsatz von Elementen, die sowohl mit Griffen versehen als auch ausgewählt wurden. Ein (oder beide) Elemente der Liste können `nil` sein.

Anmerkung Es können nur Elemente aus dem Modell- und Papierbereich der aktuellen Zeichnung mit dieser Funktion analysiert werden, keine nicht-grafischen Objekte oder Elemente in anderen Blockdefinitionen.

13.180 sslength

Gibt eine Ganzzahl zurück, welche die Anzahl der Objekte (Elemente) in einem Auswahlsatz enthält.

(sslength *Ausw_satz*)

Ist die Anzahl größer als 32,767, wird sie als reale Zahl zurückgegeben. Auswahlsätze enthalten keine Elemente doppelt.

`(setq sset (ssget "L"))`	*Plaziert das letzte Objekt in Auswahlsatz* `sset`
`(sslength sset)`	*Ergibt* `1`

13.181 ssmemb

Überprüft, ob ein Objekt (Element) in einem Auswahlsatz enthalten ist.

(ssmemb *Elem_name Ausw_satz*)

Ist dies der Fall, gibt **ssmemb** den Elementnamen (*Elem_name*) zurück. Ist dies nicht der Fall, wird `nil` zurückgegeben. Nimmt man zum Beispiel an, daß der Elementname `e1` im Auswahlsatz `Ausw_satz1` vorhanden ist und Elementname `e2` nicht, dann ergibt

`(ssmemb e1 Ausw_satz1)`	*den Elementnamen* `e1`
`(ssmemb e2 Ausw_satz1)`	`nil`

13.182 ssname

Gibt den Objektnamen (Elementnamen) des indizierten Elements eines Auswahlsatzes zurück.

(ssname *Ausw_satz Index*)

Das Argument *Index* muß eine Ganzzahl sein. Ist das Argument *Index* negativ oder größer als die höchste Elementnummer in dem Auswahlsatz, wird `nil` zurückgegeben. Das erste Element des Satzes hat den Index Null. Elementnamen in Auswahlsätzen, die Sie nach Anwendung der Funktion **ssget** erhalten haben, gehören immer zu Hauptelementen. Subelemente (wie Attribute und Kontrollpunkte von Polylinien) werden nicht zurückgegeben. (Die Funktion **entnext** ermöglicht den Zugriff darauf. Siehe **"entnext".**)

`(setq sset (ssget))`	*Erzeugt einen Auswahlsatz mit dem Namen* `sset`
`(setq ent1 (ssname sset 0))`	*Ermittelt den Namen des ersten Elementes in* `sset`
`(setq ent4 (ssname sset 3))`	*Ermittelt den Namen des vierten Elementes in* `sset`

Zum Zugriff auf Elemente hinter dem 32767. Element in einem Auswahlsatz müssen Sie das Argument *Index* als reale Zahl übergeben. Beispiel:

`(setq entx (ssname sset 50843.0))` *Ermittelt den Namen des 50844. Elementes in* `sset`

13.183 ssnamex

Findet Informationen zur Art und Weise der Erstellung eines Auswahlsatzes wieder.

(ssnamex ***Ausw_satz [Index]*****)**

Diese Funktion gibt den Namen des Elements zurück, das durch `Index` vom Auswahlsatz `Ausw_satz` angegeben wurde, zusammen mit Daten, die die Art und Weise der Auswahl des Elements beschreiben. Ist das Argument `Index` nicht vorhanden, gibt diese Funktion eine Liste der Namen aller Elemente im Auswahlsatz zurück, zusammen mit Daten zurück, die die Art und Weise der Auswahl aller Elemente beschreiben.

Die Daten, die von **ssnamex** zurückgegeben werden, sind eine oder mehrere Listen, die Informationen zur Beschreibung eines Elements und dessen Auswahlmethode enthalten, oder ein Polygon, mit dem ein oder mehrere Elemente ausgewählt wurden. Jede Unterliste, die die Beschreibung der Auswahl eines bestimmten Elements enthält, besteht aus drei Teilen: Der Auswahlmethoden-ID (eine Ganzzahl >= 0), dem Namen des ausgewählten Elements und den spezifischen Daten zur Auswahlmethode, die die Art und Weise der Elementauswahl beschreiben.

((*sel_id1 Elem_name1* (*Daten*)) (*sel_id2 Elem_name2* (*Daten*)) **. . .**)

In der folgenden Tabelle sind die Auswahlmethoden-IDs aufgelistet.

Auswahlmethoden-IDs

ID	Beschreibung
0	nicht spezifisch (z. B. Letzte, Alle etc.)
1	Auswahl
2	Fenster oder WPolygon
3	Kreuzen oder CPolygon
4	Begrenzungsmarke

Jede Unterliste mit der Beschreibung eines Polygons, das bei der Elementauswahl verwendet wurde, erscheint als Polygon-ID (eine Ganzzahl < 0) gefolgt von Punktbeschreibungen.

(*Polygon_ID P_beschr_1 P_beschr_n* **. . .**)

Die Aufzählung der Polygon-IDs beginnt bei -1, und jede zusätzliche Polygon-ID steigt um -1 an. Je nach Ansichtspunkt wird ein Punkt folgendermaßen dargestellt: als unendliche Linie, als Strahl oder als Liniensegment. Eine Punktbeschreibung umfaßt drei Teile: Eine Punkt-Deskriptor-ID (die beschriebene Elementart), den Startpunkt des Elements und einen optionalen Einheitsvektor, der entweder die Richtung beschreibt, in die die unendliche Linie verläuft, oder ein Vektor, der den Abstand zur anderen Seite des Liniensegments beschreibt.

(*P_beschr_ID Basis_P [Einh_oder_Abst_Vek]*)

In der folgenden Tabelle sind die gültigen Punkt-Deskriptor-IDs aufgelistet.

Punkt-Deskriptor-IDs

ID	Beschreibung
0	Unendliche Linie
1	Strahl
2	Liniensegment

Der `Einh_oder_Abst_Vek` wird zurückgegeben, wenn der Ansichtspunkt einen anderen Wert als 0,0,1 aufweist.

Die mit der Elementauswahlart Auswahl (Typ 1) verbundenen `Daten` stellen eine einzelne Punktbeschreibung dar. Folgender Eintrag wird für die Auswahl eines Elements gegeben, das bei 1,1 in der Draufsicht des WKS ausgewählt wurde:

(1 <Elementname: 60000064> (0 (1.0 1.0 0.0)))

Die `Daten`, die mit einem Element verbunden sind, das mit Fenster, WPolygon, Kreuzen oder CPolygon ausgewählt wurde, stellen eine Ganzzahl-ID des Polygons dar, mit dem das Element ausgewählt wurde. Es ist Aufgabe der Anwendung, die Polygon-IDs zuzuordnen und die Verbindung zwischen Polygon und den damit ausgewählten Elementen herzustellen. Folgender Eintrag wird für eine Einheit gegeben, die durch Kreuzen ausgewählt wurde (wobei die Polygon-ID -1 lautet).

((3 <Elementname: 60000024> -1) (-1 (0 (5.14828 7.05067 0.0))
(0 (7.13676 7.05067 0.0)) (0 (7.13676 4.62785 0.0))
(0 (5.14828 4.62785 0.0))))

Die Daten, die mit der Auswahl von Begrenzungsmarken verbunden sind, stellen eine Liste von Punktbeschreibungen für die Punkte dar, an denen sich die Begrenzungsmarke und das Element sichtbar schneiden. Folgender Eintrag wird für eine fast vertikale Linie gegeben, die dreimal von einer Z-förmigen Begrenzungsmarke geschnitten wird.

((4 <Elementname: 60000024> (0 (5.28135 6.25219 0.0))
(0 (5.61868 2.81961 0.0)) (0 (5.52688 3.75381 0.0))))

Anmerkung Es können nur Auswahlsätze mit Elementen aus dem Modell- und Papierbereich der aktuellen Zeichnung mit dieser Funktion wiedergefunden werden, *keine* nicht-grafischen Objekte oder Elemente in anderen Blockdefinitionen.

13.184 sssetfirst

Legt fest, welche Objekte ausgewählt und mit Griffen versehen werden.

(sssetfirst ***Griffsatz*** **[*Picksatz*])**

Der Auswahlsatz von Objekten, die mit dem Argument `Griffsatz` angegeben werden, wird mit Griffen versehen, und der Auswahlsatz von Objekten, die mit dem Argument `Picksatz` angegeben werden, wird sowohl mit Griffen versehen als auch ausgewählt. Wenn Objekte in beiden Auswahlsätzen gemeinsam vorhanden sind, versieht **sssetfirst** nur den Auswahlsatz mit Griffen und wählt ihn aus, der durch `Picksatz` angegeben wird (der durch `Griffsatz` angegebene Satz wird nicht mit Griffen versehen). Wenn `Griffsatz` `nil` ist, versieht **sssetfirst** den `Picksatz`-Satz mit Griffen und wählt ihn aus. Die Funktion **sssetfirst** gibt eine Liste der zwei als Auswahlsätze übergebenen Variablen zurück.

Anmerkung Rufen Sie **ads_ssetfirst()** *nicht* auf, wenn AutoCAD gerade einen Befehl ausführt.

13.185 startapp

Startet eine Windows-Anwendung.

(startapp ***Anw_befehl)*** **[*Datei*])**

Das Argument `Anw_befehl` ist eine Zeichenkette, die eine auszuführende Anwendung definiert. Wenn `Anw_befehl` keinen vollständigen Pfadnamen enthält, sucht die Funktion in der Umgebungsvariablen PATH nach der Anwendung. Das Argument `Datei` ist eine Zeichenkette, die den Namen der zu öffnenden Datei definiert. Gibt bei erfolgreicher Ausführung eine Ganzzahl größer als 0 zurück, andernfalls wird 0 zurückgegeben.

Der folgende Code startet den Notizblock von Windows und öffnet die Datei *acad.lsp*.

```
(startapp "notepad" "acad.lsp")
```

Wenn in einem Argument Leerzeichen vorhanden sind, muß es in doppelten Anführungszeichen stehen. Zum Beispiel wird zum Editieren der Datei *meinzeug.txt* mit dem Notizblock die folgende Syntax verwendet:

```
(startapp "notepad.exe" "\"meinzeug.txt\"")
```

Extern definierte Funktion ARX-Anwendung *acadapp*

13.186 start_dialog

Zeigt ein Dialogfeld an und akzeptiert die Benutzereingabe.

(start_dialog)

Zuerst müssen Sie durch einen Aufruf der Funktion **new_dialog** das Dialogfeld initialisieren. Das Dialogfeld bleibt so lange aktiv, bis ein Operationsausdruck oder eine Rückmeldungsfunktion **done_dialog** aufruft. Normalerweise gehört **done_dialog** zu der Komponente, deren `Schlüssel "annehmen"` ist (typischerweise die Schaltfläche OK) und zur Komponente, deren `Schlüssel "abbrechen"` ist (typischerweise die Schaltfläche Abbrechen).

Die Funktion **start_dialog** hat keine Argumente. Sie gibt den optionalen `Status` zurück, der an **done_dialog** übergeben wurde. Der Standardwert ist `1`, wenn der Benutzer OK gedrückt hat, `0`, wenn der Benutzer Abbrechen gedrückt hat, oder `-1`, wenn alle Dialogfelder mit **term_dialog** beendet wurden. Wird der Funktion **done_dialog**

jedoch ein ganzzahliges Argument *status* weitergegeben, das größer als 1 ist, gibt die Funktion **start_dialog** diesen Wert zurück, dessen Bedeutung von der jeweiligen Anwendung abhängt.

13.187 start_image

Beginnt die Erstellung eines Bilds in der Dialogfeldkomponente.

(start_image ***Schlüssel*****)**

Aufeinanderfolgende Aufrufe von **fill_image**, **slide_image** und **vector_image** beeinflussen dieses Bild, bis die Anwendung die Funktion **end_image** aufruft. Das Argument *Schlüssel* ist eine Zeichenkette, welche die Dialogfeldkomponente angibt. Beim Argument *Schlüssel* wird zwischen Groß- und Kleinschreibung unterschieden.

Anmerkung Verwenden Sie die Funktion **set_tile** *nicht* zwischen den Aufrufen der Funktionen **start_image** und **end_image**.

13.188 start_list

Beginnt die Verarbeitung einer Liste im Listenfeld oder in der Dialogfeldkomponente "Pop-Up-Liste".

(start_list ***Schlüssel [Operation [Index]]*****)**

Das Argument *Schlüssel* ist eine Zeichenkette, welche die Dialogfeldkomponente angibt. Beim Argument *Schlüssel* wird zwischen Groß- und Kleinschreibung unterschieden. Das Argument *Operation* ist ein ganzzahliger Wert, dessen Bedeutung in der folgenden Tabelle zusammengefaßt ist.

Listenfeld-Codes für die Funktion start_list

Wert	Beschreibung
1	Gewählten Listeninhalt ändern
2	Neuen Listeneintrag anhängen
3	Alte Liste löschen und neue erstellen (die Vorgabe)

Das Argument *Index* wird ignoriert, es sei denn der Aufruf der Funktion **start_list** beginnt eine Änderungs-Operation (1). In diesem Fall zeigt *Index* den Listeneintrag an, der durch den anschließenden Aufruf von **add_list** geändert werden soll. Der *Index* basiert auf Null. Wenn Sie *Operation* nicht definieren, wird der Vorgabewert 3 genommen (Erzeugen einer neuen Liste), wenn Sie zwar das Argument *Operation*, aber nicht das Argument *Index* definieren, lautet der vorgegebene Indexwert 0.

Aufeinanderfolgende Aufrufe von **add_list** beeinflussen die durch **start_list** begonnene Liste, bis die Anwendung die Funktion **end_list** aufruft.

Anmerkung Verwenden Sie die Funktion **set_tile** *nicht* zwischen den Aufrufen der Funktionen **start_list** und **end_list**.

13.189 strcase

Gibt eine Zeichenkette zurück, in der alle alphabetischen Zeichen entweder in Groß- oder Kleinbuchstaben umgewandelt wurden.

(strcase ***Z_kette [wie]*****)**

Wenn *wie* weggelassen wird oder `nil` ergibt, werden alle alphabetischen Zeichen in *Z_kette* in Großbuchstaben umgewandelt. Wenn *wie* übergeben wird und nicht `nil` ist, werden alle alphabetischen Zeichen in *Z_kette* in Kleinbuchstaben umgewandelt.

```
(strcase "Muster")          ergibt  "MUSTER"
(strcase "Muster" T)        ergibt  "Muster"
```

Die Funktion **strcase** dient zur korrekten Behandlung der Groß- und Kleinbuchstaben-Umsetzung des aktuell konfigurierten Zeichensatzes (siehe "Unterstützung von Fremdsprachen").

13.190 strcat

Gibt eine Zeichenkette zurück, welche die Verkettung mehrerer Zeichenketten ist.

(strcat ***Z_kette1 [Z_kette2]...*****)**

```
(strcat "a" "bout")        ergibt  "about"
(strcat "a" "b" "c")       ergibt  "abc"
(strcat "a" "" "c")        ergibt "ac"
```

13.191 strlen

Gibt eine Ganzzahl zurück, welche der Anzahl der Zeichen in einer Zeichenkette entspricht.

(strlen ***[Z_kette]...*****)**

Liegen mehrere Argumente *Z_kette* vor, gibt diese Funktion die Summe der Länge aller Argumente zurück. Wird das Argument ausgelassen oder eine leere Zeichenkette eingegeben, ist das Ergebnis 0 (Null).

```
(strlen "abcd")            ergibt  4
(strlen "ab")              ergibt  2
(strlen "eins" "zwei" "vier"ergibt  10
(strlen)                   ergibt  0
(strlen)                   ergibt  0
```

13.192 subst

Durchsucht eine Liste nach einem alten Element und gibt eine Kopie der Liste zurück, in der das alte Element jedesmal durch ein neues Element ersetzt wird.

(subst ***neu_Element alt_Element Liste*****)**

Wenn *alt_Element* in *Liste* nicht gefunden wird, gibt **subst** *Liste* unverändert zurück.

```
(setq Muster '(a b (c d) b))
(subst 'qq 'b Muster) ergibt  (A QQ (C D) QQ)
(subst 'qq 'z Muster) ergibt  (A B (C D) B)
(subst 'qq '(c d) Muster)  ergibt  (A B QQ B)
(subst '(qq rr) '(c d) Muster)   ergibt  (A B (QQ RR) B)
(subst '(qq rr) 'z Muster) ergibt  (A B (C D) B)
```

Sie können die Funktion **subst** mit der Funktion **assoc** kombinieren, um den mit dem Argument Schlüssel in einer Assoziationsliste verbundenen Wert ganz einfach zu ersetzen.

```
(setq wer '((zuerst Peter) (dann q) (zuletzt öffentlich)))
(setq alt (assoc 'zuerst wer)          setzt alt auf  (ZUERST PETER)
      neu '(zuerst p)                  setzt neu auf  (ZUERST P)
)
(subst neu alt wer)  ergibt   ((ZUERST P)(DANN Q)(ZULETZT ÖFFENTLICH))
```

13.193 substr

Gibt eine untergeordnete Zeichenkette einer Zeichenkette zurück

(substr ***Z_kette Anfang [Länge]*****)**

Die Funktion **substr** beginnt an der Zeichenposition des durch *Anfang* definierten Zeichens des Arguments *Z_kette* und wird für die im Argument *Länge* definierten Zeichen fortgesetzt. Ist das Argument *Länge* nicht definiert, endet die untergeordnete Zeichenkette erst mit dem Ende des Arguments *Zeichenkette*. Die Argumente *Anfang* und *Länge* müssen positive Ganzzahlen sein.

Das erste Zeichen von *Z_kette* ist das Zeichen Nummer 1. Dies unterscheidet sich von anderen Funktionen, bei denen Elemente einer Liste verarbeitet werden (wie z. B. **nth** und **ssname**) und bei denen das erste Element als 0 gezählt wird.

```
(substr "abcde" 2)         ergibt  "bcde"
(substr "abcde" 2 1)       ergibt  "b"
(substr "abcde" 3 2)       ergibt  "cd"
```

13.194 tablet

Ermittelt und setzt Digitalisier-(Tablett-)kalibrierungen.

(tablet ***Code [Reihe1 Reihe2 Reihe3 Richtung]*****)**

Abhängig von der durch *Code* angegebenen Ganzzahl ermittelt **tablet** entweder die aktuelle Kalibrierung des Digitalisiergerätes oder stellt die Kalibrierwerte ein. Wenn *Code* 0 ist, gibt **tablet** die aktuelle Kalibrierung zurück. Ist das Argument *Code* 1, muß sich die Einstellung der neuen Kalibrierung anschließen: *Reihe1, Reihe2, Reihe3,* und *Richtung*.

Code

Eine Ganzzahl. Wenn der von Ihnen übergebene *code* 0 ist, gibt **tablet** die aktuelle Kalibrierung zurück. In diesem Fall müssen die anderen Argumente weggelassen werden. Wenn der von Ihnen übergebene Code 1 ist, stellt **tablet** die Kalibrierung entsprechend der folgenden Argumente ein. In diesem Fall müssen Sie die anderen Argumente angeben.

Reihe1, **Reihe2**, **Reihe3**

Drei 3D-Punkte. Diese drei Argumente definieren die drei Zeilen der Transformationsmatrix des Tabletts.

Richtung

Ein 3D-Punkt. Hierbei handelt es sich um den Vektor (ausgedrückt im Weltkoordinatensystem oder WKS), der senkrecht auf der Ebene steht, welche die Tablettoberfläche darstellt.

Anmerkung Wenn die definierte *Richtung* nicht normalisiert ist, korrigiert **tablet** sie, so daß das Argument *Richtung*, das zurückgegeben wird, wenn Sie die Kalibrierung festsetzen, daher von dem Wert abweichen kann, den Sie übergeben haben. Aus ähnlichen Gründen muß das dritte Element in *Reihe3* (*Z*) immer gleich 1 sein: **Die Funktion tablet gibt es auch als 1 zurück, wenn für das Argument** *Reihe3* in der Liste ein anderer Wert definiert ist.

Wenn **tablet** scheitert, gibt die Funktion `nil` zurück und setzt die Systemvariable ERRNO auf einen Wert, der die Fehlerursache anzeigt (siehe Kapitel 16, "AutoLISP-Fehlercodes und Fehlermeldungen"). Dies kann passieren, wenn der Digitalisierer kein Tablett ist.

Eine sehr einfache Transformation, die mit der Funktion **tablet** durchgeführt werden kann, ist die Identitätstransformation:

```
(tablet 1 '(1 0 0) '(0 1 0) '(0 0 1) '(0 0 1))
```

Ist diese Transformation aktiv, erhält AutoCAD vom Tablett quasi Rohkoordinaten für den Digitalisierer. Wenn Sie zum Beispiel den Punkt mit den Digitalisiererkoordinaten (5000,15000) wählen, versteht AutoCAD diesen Punkt als den Punkt mit den entsprechenden Koordinaten in Ihrer Zeichnung.

Mit der Systemvariablen TABMODE können Routinen von AutoLISP das Tablett ein- und ausschalten.

Siehe auch

"Kalibrieren von Tabletts" für zusätzliche Informationen über die Tablett-Transformationsmatrix

13.195 tblnext

Findet das nächste Element in einer Symboltabelle.

(tblnext ***Tabell_name [Erster]*****)**

Das Argument *Tablell_name* ist eine Zeichenkette, welche die Symboltabelle angibt. Gültige Werte für *Tablell_name* sind "LAYER", "LTYPE", "ANSICHT", "STIL", "BLOCK", "BKS", "APP_ID", "BEMSTIL" und "AFENSTER". Die Zeichenkette muß nicht in Großbuchstaben eingegeben werden.

Anmerkung Da die Funktion **vports** die aktuelle AFENSTER-Tabelleninformation zurückgibt, kann es einfacher sein, **vports** zu benutzen und nicht **tblnext**, um diese Information abzufragen.

Wird die Funktion **tblnext** wiederholt angewendet, gibt sie gewöhnlich jedesmal den nächsten Eintrag der definierten Tabelle zurück. Mit der Funktion **tblsearch** kann eingestellt werden, daß der *nächste* Eintrag abgerufen wird. Wenn das Argument *Erster* vorhanden ist und einen Wert ergibt, der nicht nil ist, wird in der Symboltabelle zurückgeblättert und der erste Eintrag aufgerufen. Befinden sich keine weiteren Einträge in der Tabelle, wird nil zurückgegeben. Gelöschte Tabelleneinträge werden nie zurückgegeben.

Wird ein Eintrag gefunden, wird er als eine Liste punktierter Paare zurückgegeben, die aus DXF-Codes und -Werten bestehen.

```
(tblnext "layer" T)    Ermittelt den ersten Layer
```

kann zurückgeben

```
((0 . "LAYER")         Symbol-Typ
  (2 . "0")            Symbolname
  (70 . 0)             Flags
  (62 . 7)             Farbnummer, negativ wenn aus
  (6 . "CONTINUOUS")   Linientypname
)
```

Bitte beachten Sie, daß es keine -1-Gruppe gibt. AutoCAD erinnert sich an den letzten zurückgegebenen Eintrag jeder Tabelle und gibt mit jedem folgenden Aufruf der Funktion **tblnext** für diese Tabelle den nächsten Eintrag zurück. Wenn Sie eine Tabelle scannen, sollten Sie das zweite Argument unbedingt als ungleich nil definieren, damit in der Tabelle zurückgegangen und der erste Eintrag zurückgegeben wird.

In der Blocktabelle ermittelte Einträge enthalten auch eine -2-Gruppe mit dem Elementnamen des ersten Elements in der Blockdefinition (wenn vorhanden). Nimmt man also an, es gibt einen Block mit der Bezeichnung BOX,

```
(tblnext "block") Ermittelt Blockdefinition
```

kann zurückgeben

```
((0 . "BLOCK")  Symboltyp
  (2 . "BOX")   Symbolname
  (70 . 0)      Flags
  (10 9.0 2.0 0.0)    Ursprung X,Y,Z
  (-2 . <Elementname: 40000126>)  Erstes Element
)
```

Der Elementname in der -2-Gruppe wird von den Funktionen **entget** und **entnext** akzeptiert, von anderen Funktionen zum Zugriff auf Elemente jedoch nicht. Zum Beispiel können Sie dieses Element nicht mit der Funktion **ssadd** in einen Auswahlsatz einfügen. Wenn Sie den Namen des -2-Gruppe-Elements für die Funktion **entnext** angeben, können Sie die Elemente scannen, die eine Blockdefinition enthalten; **entnext** gibt nach dem letzten Element in der Blockdfinition nil zurück.

Anmerkung Enthält ein Block keine Elemente, ist die von der Funktion **tblnext** zurückgegebene -2-Gruppe der Elementname des Elements "endblk".

13.196 tblobjname

Gibt den Elementnamen eines festgelegten Symboltabelleneintrags zurück.

(tblobjname ***Tabell_name Symbol*****)**

Die Funktion **tblobjname** durchsucht die Symboltabelle *Tabell_name* nach dem Symbolnamen *Symbol* und gibt den Elementnamen dieses Symboltabelleneintrags zurück.

Der Elementname, der von **tblobjname** zurückgegeben wird, kann mit den Operationen **entget** und **entmod** verwendet werden.

13.197 tblsearch

Durchsucht eine Symboltabelle nach einem Symbolnamen.

(tblsearch ***Tabell_name Symbol [Setzfolg]*****)**

Die Funktion **tblsearch** durchsucht die Symboltabelle *Tabell_name* nach dem Symbolnamen *Symbol*. Beide Namen werden automatisch durchgängig groß geschrieben. Findet **tblsearch** einen Eintrag für den gegebenen Symbolnamen, gibt es den Eintrag in dem für **tblnext** beschriebenen Format zurück. Wird kein derartiger Eintrag gefunden, ist das Ergebnis `nil`.

```
(tblsearch "style" "standard")   Textstile
```

kann zurückgeben

```
((0 . "STYLE")   Symbolname
  (70 . 0)       Flags
  (40 . 0.0)     Festgelegte Höhe
  (41 . 1.0)     Breitenfaktor
  (50 . 0.0)     Neigungswinkel
  (71 . 0)       Generierungs-Flags
  (3 . "txt")    Primär-Schriftdatei
  (4 . "")       Big-Font-Datei
)
```

Normalerweise hat **tblsearch** keinen Einfluß auf die Reihenfolge der Einträge, die von **tblnext** aufgerufen werden. Wenn jedoch **tblsearch** erfolgreich durchgeführt wird, und das Argument *Setzfolg* vorhanden und nicht `nil` ist, wird der Eintragszähler von **tblnext** so eingestellt, daß der nächste Aufruf von **tblnext** den Eintrag zurückgibt, der nach dem durch diesen Aufruf von **tblsearch** gegebenen Eintrag liegt.

13.198 term_dialog

Schließt alle aktuellen Dialogfelder, als ob der Benutzer jedes einzelne geschlossen hätte.

(term_dialog)

Wird eine Anwendung beendet, während noch eine DCL-Datei geöffnet ist, ruft AutoCAD automatisch die Funktion **term_dialog** auf. Diese Funktion wird in erster Linie dafür eingesetzt, verschachtelte Dialogfelder abzubrechen. Die Funktion **term_dialog** gibt immer `nil` zurück.

13.199 terpri

Druckt einen Zeilenvorschub in die Befehlszeile.

(terpri)

Die Funktion **terpri** wird nicht für E/A-Funktionen für Dateien eingesetzt. Wenn Sie einen Zeilenvorschub in eine Datei schreiben möchten, benutzen Sie die Funktionen **prin1**, **princ** oder **print**.

13.200 textbox

Mißt ein definiertes Textobjekt und gibt die diagonalen Koordinaten eines Feldes zurück, das den Text enthält.

(textbox ***Elem_liste*****)**

Das Argument *Elem_liste* ist eine Elementdefinitionsliste in der Form, wie sie von **entget** zurückgegeben wird. Es muß ein Textobjekt definieren. Fehlen im Argument *Elem_liste* Felder, welche Textparameter (nicht den Text selbst) definieren, werden die aktuellen (oder vorgegebenen) Einstellungen benutzt. Wird die Funktion **textbox** erfolgreich durchgeführt, gibt sie eine Liste mit zwei Punkten zurück, andernfalls gibt sie `nil` zurück.

Die kürzeste Liste, die von der Funktion **textbox** erkannt wird, enthält nur den Text selber.

`(textbox '((1 . "Hallo Leute."))))` *kann zurückgeben* `((0.0 0.0 0.0) (0.8 0.2 0.0))`

In diesem Fall würde die Funktion **textbox** die aktuellen Vorgaben für Texte benutzen, um die fehlenden Parameter zu liefern.

Die von der Funktion **textbox** zurückgegebenen Punkte beschreiben den Begrenzungsrahmen des Textobjekts so, als ob sein Einfügepunkt die Koordinaten (0,0,0) hätte und sein Drehwinkel 0 betragen würde. Die zuerst zurückgegebene Liste enthält im allgemeinen den Punkt (0.0 0.0 0.0), es sei denn, das Textobjekt ist schräg oder vertikal angeordnet oder enthält Buchstaben mit Unterlängen (zum Beispiel *g* und *p*). Der Wert der ersten Punktliste definiert den Abstand des Texteinfügepunkts zur unteren linken Ecke des kleinsten den Text umschließenden Rechtecks. Die zweite Punktliste bezeichnet die obere rechte Ecke dieses Rechtecks. Ohne Berücksichtigung der festgestellten Textrichtung beschreibt die zurückgegebene Punktliste immer die untere linke und die obere rechte Ecke dieses Begrenzungsrahmens.

13.201 textpage

Wechselt vom Grafik- zum Textbildschirm.

(textpage)

Die Funktion **textpage** ist äquivalent zur Funktion **textscr**. Diese Funktion gibt immer `nil` zurück.

13.202 textscr

Wechselt vom Grafik- zum Textbildschirm (vergleichbar mit der Funktionstaste zur Bildschirmumschaltung bei AutoCAD).

(textscr)

Die Funktion **textscr** gibt immer `nil` zurück.

13.203 trace

Unterstützt die Fehlersuche in AutoLISP.

(trace ***Funktion*** **...)**

Die Funktion **trace** setzt das Trace-Flag für die definierten *Funktionen*. Jedesmal, wenn ein definiertes Argument *Funktion* ausgewertet wird, erscheint eine Protokollierungsanzeige, die den Funktionseintrag zeigt (eingerückt auf die Ebene der Aufruftiefe) und das Ergebnis der Funktion druckt.

`(trace mein_funk)` *ergibt* `MEIN_FUNK`

und setzt das Trace-Flag für die Funktion **MEIN_FUNK**. Die Funktion **trace** gibt den letzten an sie weitergegebenen Funktionsnamen zurück.

Siehe auch

Die Funktion **untrace**

13.204 trans

Überträgt einen Punkt (oder eine Verschiebung) von einem Koordinatensystem in ein anderes.

(trans ***P von in [Verschiebung]*****)**

Das Argument *P* ist eine aus drei realen Zahlen bestehende Liste, die entweder als 3D-Punkt oder als 3D-Verschiebung (Vektor) verstanden werden kann. Das Argument *von* gibt das Koordinatensystem an, in dem *P* ausgedrückt wird, und *in* gibt das Koordinatensystem des zurückgegebenen Punktes an. Ist das optionale Argument *Verschiebung* vorhanden und nicht `nil`, legt es fest, daß das Argument *P* nicht wie ein Punkt behandelt werden soll, sondern wie eine 3D-Verschiebung. Die Argumente *von* und *in* können ganzzahlige Codes (wie in der folgenden Tabelle), Elementnamen oder 3D-Hochzugsvektoren sein.

Code	Koordinatensystem
0	Welt (WCS)

1 Benutzer (aktuelles BKS)

2 Bildschirm:
mit Code 0 oder 1 AKS des aktuellen Ansichtfensters
mit Code 3 AKS des aktuellen Modellbereich-Ansichtsfensters

3 Papierbereich-AKS (PBAKS) (*nur* mit Code 2)

Wenn Sie einen Elementnamen für die Argumente `von` oder `in` verwenden, muß er so übergeben werden, wie er von den Funktionen **`entnext`**, **`entlast`**, **`entsel`**, **`nentsel`** und **`ssname`** zurückgegeben wurde. Hiermit können Sie einen Punkt zu und vom Objektkoordinatensystem (OKS) eines bestimmten Objektes übertragen. (Für einige Objekte ist das OKS äquivalent zum WKS; für diese Objekte ist die Umsetzung zwischen OKS und WKS eine leere Operation.) Ein 3D-Hochzugsvektor (eine Liste von drei realen Zahlen) ist eine andere Methode, die Umsetzung zum und vom OKS eines Objektes vorzunehmen. Dies kann jedoch nicht bei Objekten angewendet werden, bei denen OKS und WKS gleich sind.

Die Funktion **`trans`** gibt einen 3D-Punkt (oder eine Verschiebung:) in dem mit *in* angeforderten Koordinatensystem zurück. Angenommen, es handelt sich um ein um 90° gegen den Uhrzeigersinn um die Welt-*Z*-Achse gedrehtes BKS, dann gilt:

`(trans '(1.0 2.0 3.0) 0 1)` *ergibt (2.0 -1.0 3.0)*
`(trans '(1.1 2.0 3.0) 0 0)` *ergibt (2.0 -1.0 3.0)*

Koordinatensysteme werden ausführlicher erläutert in "Konvertierung von Koordinatensystemen."

Um zum Beispiel eine Linie zu zeichnen, die vom Einfügepunkt eines Textteils ausgeht (ohne Ofang zu verwenden), müssen Sie den Einfügepunkt des Textobjekts von seinem OKS in sein WKS konvertieren.

`(trans` *Texteinfügepunkt Text_Ename* `1)`

Sie können das Resultat dann der Eingabeaufforderung Von Punkt zuführen.

Umgekehrt müssen Sie Punkt- (oder Verschiebungs-) Werte in ihr Ziel-OKS umwandeln, bevor Sie sie an die Funktion **`entmod`** übergeben. Wenn Sie zum Beispiel einen Kreis (ohne den Befehl SCHIEBEN zu verwenden) um die BKS-relative Verschiebung (1,2,3) verschieben wollen, müssen Sie die Verschiebung vom BKS ins OKS des Kreises umwandeln:

```
(trans '(1 2 3) 1 Kreis_Ename)
```

Dann addieren Sie die daraus resultierende Verschiebung zum Kreismittelpunkt.

Wenn Sie zum Beispiel einen vom Benutzer eingegebenen Punkt haben und ermitteln möchten, welchem Ende einer Linie er näher liegt, dann konvertieren Sie den Benutzerpunkt aus dem BKS in das AKS.

```
(trans Benutzerpunkt 1 2)
```

Dann konvertieren Sie die beiden Endpunkte der Linie aus dem OKS in das AKS.

```
(trans Endpunkt Linie_Ename 2)
```

Anschließend können Sie den Abstand zwischen dem Benutzerpunkt und den beiden Endpunkten der Linie berechnen (dabei ignorieren Sie die *Z*-Koordinaten), um festzustellen, welchem Endpunkt der Benutzerpunkt tatsächlich näher liegt.

Die Funktion **`trans`** kann auch 2D-Punkte transformieren. Dabei *fügt* die Funktion für die *Z*-Koordinate einen passenden Wert ein. Die eingesetzte *Z*-Komponente hängt davon ab, welches Koordinatensystem mit dem Argument `von` angegeben wurde und ob der Wert als Punkt oder als Verschiebung konvertiert werden soll. Soll der Wert als Verschiebung konvertiert werden, ist der *Z*-Wert immer 0.0; soll der Wert dagegen als Punkt konvertiert werden, wird der einzufügende *Z*-Wert wie in der folgenden Tabelle beschrieben ermittelt.

Aus	Eingefügter Z-Wert
WKS	0.0
BKS	Aktuelle Erhebung
OKS	0.0
AKS	Auf die aktuelle Konstruktionsebene projiziert (BKS-*XY*-Ebene + aktuelle Erhebung)
PSDCS	Auf die aktuelle Konstruktionsebene projiziert (BKS-*XY*-Ebene + aktuelle Erhebung)

Konvertierte Z-Werte für 2D-Punkte
rte Z-

13.205 type

Gibt den Typ eines angegebenen Elements zurück.

(type ***Element*****)**

Die Typen werden in Form von einem der in der folgenden Tabelle dargestellten Atome wiedergegeben:

Symboltypen

Typ	Beschreibung	Typ	Beschreibung
REAL	Gleitkomma-Zahlen	SUBR	Interne Funktionen
FILE	Dateideskriptoren	EXSUBR	Externe Funktionen (ADS)
STR	Zeichenketten	PICKSET	Auswahlsätze
INT	Ganzzahlen	ENAME	Elementnamen
SYM	Symbole	PAGETB	Funktionspaging-Tabellen
LIST	Listen (und Benutzerfunktionen)		

Für Elemente, die `nil` ergeben (zum Beispiel ein nicht zugewiesenes Symbol), wird `nil` zurückgegeben.

Bei den folgende Zuweisungen:

```
(setq a 123 r 3.45 s "Hallo!" x '(a b c))
(setq f (open "Name" "r"))
```

gilt:

```
(type 'a)                ergibt SYM
(type a)                 ergibt INT
(type f)                 ergibt FILE
(type r)                 ergibt REAL
(type s)                 ergibt STR
(type x)                 ergibt LIST
(type +)                 ergibt SUBR
(type nil)               ergibt nil
```

Im folgenden Beispiel wird die Funktion **type** verwendet:

```
(defun isint (a)
   (if (= (type a) 'INT)   ist TYPE eine ganze Zahl?
     T                     ja, ergibt T
     nil                   nein, ergibt nil
   )
)
```

13.206 unload_dialog

Beendet eine DCL-Datei.

(unload_dialog ***DCL_bez*****)**

Beendet die mit dem Argument *DCL_bez* verbundene DCL-Datei (die aus einem vorangegangenen Aufruf der Funktion **new_dialog** hervorgegangen ist).

Gibt immer `nil` zurück.

13.207 untrace

Löscht den Inhalt des Protokoll-Flags für die definierte Funktion.

(untrace ***Funktion*****...)**

Gibt den letzten Funktionsnamen zurück.

Der folgende Code löscht den Inhalt des Trace-Flags für die Funktion **MEIN-FUNK**:

```
(untrace mein_funk)                    ergibt   MEIN-FUNK
```

Siehe auch

Die Funktion **trace**

13.208 vector_image

Zeichnet einen Vektor in das aktuelle Dialogfeldbild .

(vector_image ***x1 y1 x2 y2 Farbe*****)**

Diese Funktion zeichnet einen Vektor in das aktuelle Dialogfeldbild (geöffnet durch **start_image**) von dem Punkt (*x1*,*y1*) zum Punkt (*x2*,*y2*). Der von dem Argument *Farbe* definierte Parameter ist eine AutoCAD-Farbnummer oder eine der in der folgenden Tabelle dargestellten logischen Farbnummern.

Der Ursprung (0,0) ist die obere linke Bildecke. Die Koordinaten der unteren rechten Ecke erhalten Sie, indem Sie die Bemaßungsfunktionen aufrufen (**dimx_tile** und **dimy_tile**).

Symbolische Namen für das Farbattribut

Farb-nummer	Mnemonische ADI-Abkürzung	Beschreibung
-2	BGLCOLOR	Aktuelle Hintergrundfarbe des AutoCAD-Grafikbildschirms
-15	DBGLCOLOR	Aktuelle Hintergrundfarbe der Dialogfelder
-16	DFGLCOLOR	Aktuelle Vordergrundfarbe der Dialogfelder (für Text)
-18	LINELCOLOR	Aktuelle Linienfarbe der Dialogfelder

13.209 ver

Gibt eine Zeichenkette zurück, welche die aktuelle AutoLISP-Versionsnummer enthält.

(ver)

Die Funktion **ver** sollte (mit der Funktion **equal**) verwendet werden, um die Kompatibilität von Programmen zu überprüfen. Die Zeichenkette sieht folgendermaßen aus:

```
"AutoLISP Release X.X (nn)"
```

wobei *X.X* die aktuelle Versionsnummer und *nn* eine Sprachbeschreibung aus zwei Buchstaben darstellt.

```
(ver)    kann ergeben    "AutoLISP Release 14.0 (en)"
```

Es folgen einige Beispiele für Sprachenbeschreibungen aus zwei Buchstaben:

(en) US/UK (es) Spanish (fr) French

(de) German (it) Italian

13.210 vports

Gibt eine Liste von Ansichtsfensterdeskriptoren für die aktuelle Ansichtsfensterkonfiguration zurück.

(vports)

Jeder Ansichtsfensterdeskriptor ist eine Liste, die aus der Ansichtsfensternummer und den Koordinaten der unteren linken und der oberen rechten Ecke des Ansichtsfensters besteht.

Ist die AutoCAD-Systemvariable TILEMODE auf 1 (ein) gesetzt, beschreibt die zurückgegebene Liste die Ansichtsfensterkonfiguration, die mit dem AutoCAD-Befehl AFENSTER erstellt wurde. Die Ecken der Ansichtsfenster werden mit Werten zwischen 0.0 und 1.0 ausgedrückt, wobei (0.0, 0.0) die untere linke und (1.0, 1.0) die obere rechte Ecke des Grafikbereichs des Bildschirms darstellt. Ist die Systemvariable TILEMODE auf 0 (aus) gesetzt, beschreibt die zurückgegebene Liste die mit dem Befehl MANSFEN erstellten Ansichtsfensterobjekte. Die Ecken des Ansichtsfensterobjekts werden in Papierbereichkoordinaten ausgedrückt. Wenn die Systemvariable TILEMODE auf "aus" gesetzt ist, wird das Ansichtsfenster 1 immer in Papierbereichkoordinaten ausgedrückt.

Angenommen, Sie haben eine Konfiguration mit einem Ansichtsfenster, und die Systemvariable TILEMODE ist auf "ein" gesetzt, dann kann die Funktion **vports** folgendes zurückgeben:

```
((1 (0.0 0.0) (1.0 1.0)))
```

Angenommen, vier gleich große Ansichtsfenster sind in den vier Ecken Ihres Bildschirms angeordnet, und die Systemvariable TILEMODE ist auf ‚ein" gesetzt; dann kann die Funktion **vports** folgendes zurückgeben:

```
( (5 (0.5 0.0) (1.0 0.5))
  (2 (0.5 0.5) (1.0 1.0))
  (3 (0.0 0.5) (0.5 1.0))
  (4 (0.0 0.0) (0.5 0.5)) )
```

Der Deskriptor des aktuellen Ansichtsfensters steht immer an erster Stelle der Liste. Im genannten Beispiel stellt die Ansichtsfensternummer 5 das aktuelle Ansichtsfenster dar.

13.211 wcmatch

Berücksichtigt Platzhalter in einer Zeichenkette.

(wcmatch *Z_kette Muster*)

Die Funktion **wcmatch** vergleicht das Argument *Z_kette* mit dem Argument *Muster*, um herauszufinden, ob sie einander entsprechen. Ist dies der Fall, wird T zurückgegeben, andernfalls wird nil zurückgegeben. Die Argumente *Z_kette* und *Muster* können beide entweder Zeichenketten in Anführungszeichen oder Variablen sein. Das Argument *Muster* kann die dem Platzhalter entsprechenden gesuchten Zeichen enthalten, die in der folgenden Tabelle aufgeführt sind. Nur die ersten (ungefähr) 500 Zeichen von *Z_kette* und *Muster* werden miteinander verglichen, darüberhinausgehende Zeichen werden ignoriert.

Platzhalterzeichen

Zeichen	Definition
# (Pfund)	Entspricht jeder einzelnen numerischen Ziffer
@ (at)	Entspricht jedem einzelnen alphabetischen Zeichen
. (Punkt)	Entspricht jedem einzelnen nicht-alphanumerischen Zeichen
* (Sternchen)	Entspricht jeder Zeichenreihe, leere Zeichenreihen eingeschlossen, und kann im Suchmuster an jeder beliebigen Stelle stehen: am Anfang, in der Mitte, am Ende
? (Fragezeichen)	Entspricht jedem einzelnen Zeichen
~ (Tilde)	Ist es das erste Zeichen des Musters, entspricht es allen Kombinationen, *außer* dem Muster
[...]	Entspricht *jedem* der eingeschlossenen Zeichen
[~...]	Entspricht jedem einzelnen *nicht* eingeschlossenen Zeichen
– (Bindestrich)	In Klammern geschrieben, um für ein einzelnes Zeichen einen Bereich zu definieren
, (Komma)	Trennt zwei Muster
` (umgekehrtes Anführungszeichen)	Läßt Sonderzeichen unberücksichtigt (liest das nächste Zeichen in seiner Grundbedeutung)

`(wcmatch "Name" "???,~*m*,N*")` *ergibt* `T`

Auf diese Weise wird die Zeichenkette `Name` überprüft, um festzustellen, ob sie mit dem Zeichen `N` beginnt. Sie können in einem Muster Kommata verwenden, um mehr als eine Bedingung für das Muster einzugeben. Das folgende Beispiel führt drei Vergleiche durch:

`(wcmatch "Name" "???,~*m*,N*")` *ergibt* `T`

Wird eine der drei Musterbedingungen erfüllt, gibt die Funktion **`wcmatch`** `T` zurück. In diesem Fall sind die Tests: `Name` hat drei Zeichen (false); `Name` enthält kein `m` (false); und `Name` beginnt mit `N` (true). Mindestens eine Bedingung ist erfüllt worden, also gibt dieser Ausdruck `T` zurück.

Der Vergleich unterscheidet zwischen Groß- und Kleinschreibung, folglich müssen Groß- bzw. Kleinbuchstaben übereinstimmen. Es ist angebracht, Variablen und Werte zu verwenden, die von AutoLISP-Funktionen für *`Z_kette`* und *`Muster`* zurückgegeben wurden.

Möchten Sie in einer Zeichenkette ein Platzhalterzeichen suchen, können Sie das umkehrte Apostroph (`) verwenden, um die *Sonderzeichenbedeutung* dieses Zeichens unberücksichtigt zu lassen. Escape bedeutet, daß das auf das einzelne umgekehrte Anführungszeichen folgende Zeichen nicht als Platzhalterzeichen gelesen wird, es wird mit seinem ursprünglichen Wert verglichen. Um zum Beispiel ein Komma irgendwo in der Zeichenkette `Name` zu suchen, müssen Sie folgendes eingeben:

`(wcmatch "Name" "*`,*")` *ergibt* `nil`

Anmerkung Da in zukünftigen AutoLISP-Versionen weitere Platzhalterzeichen hinzukommen können, ist es ratsam, alle nicht-alphanumerischen Zeichen im Muster unberücksichtigt zu lassen, um die zukünftige Kompatibilität zu gewährleisten.

Die beiden Programmiersprachen C und AutoLISP setzen als Escape-Zeichen den umgekehrten Schrägstrich (\) ein, also benötigen Sie zwei umgekehrte Schrägstriche (\\), um einen umgekehrten Schrägstrich in einer Zeichenkette darzustellen. Um in der Zeichenkette `Name` einen umgekehrten Schrägstrich zu suchen, müssen Sie folgendes eingeben:

`(wcmatch "Name" "*`\ *")` *ergibt* `nil`

Alle in eckigen Klammern . . stehenden Zeichen ([. . .]) werden in ihrer Grundbedeutung gelesen, so daß es keinen Grund gibt, sie unberücksichtigt zu lassen, mit folgenden Ausnahmen: Das Tilden-Zeichen (~) wird nur in seiner Grundbedeutung gelesen, wenn es *nicht* das erste Zeichen in eckigen Klammern ist (wie in `"[A~BC]"`); andernfalls wird es als Negationszeichen gelesen, was bedeutet, daß **`wcmatch`** die Übereinstimmung aller Zeichen untersucht, *außer* denen, die auf das Tildenzeichen folgen (wie in `"[~ABC]"`). Der Bindestrich (-) wird nur dann in seiner Grundbedeutung gelesen, wenn er an erster oder an letzter Stelle in der Klammer steht (zum Beispiel `"[-ABC]"` oder `"[ABC-]"`), oder wenn er auf eine an erster Stelle stehende Tilde folgt (zum Beispiel `"[~-ABC]"`). Andernfalls wird

der Bindestrich (-) innerhalb von eckigen Klammern benutzt, um für ein bestimmtes Zeichen einen Wertbereich zu definieren. Der Bereich ist nur für einzelne Zeichen möglich, so entspricht `"STR[1-38]"` `STR1`, `STR2`, `STR3` und `STR8`, und `"[A-Z]"` entspricht jedem einzelnen Großbuchstaben.

Das Zeichen (`"]"`)) wird auch in seiner Grundbedeutung gelesen, wenn es sich um das erste Zeichen in der eckigen Klammer handelt oder wenn es auf eine an erster Stelle stehende Tilde folgt (zum Beispiel `"[ ]ABC]"` oder `"[~]ABC]"`).

13.212 while

Wertet einen Testausdruck aus; ist dieser nicht nil, werden andere Ausdrücke ausgewertet; wiederholt diesen Vorgang, bis der Testausdruck nil ergibt.

(while ***Testausdr Ausdruck*** **...)**

Die Funktion **while** wird fortgeführt, bis *Testausdr* `nil` ist. Dann wird der letzte Wert des letzten Ausdruck zurückgegeben.

Der folgende Code ruft die Benutzerfunktion **SOME-FUNC** zehnmal auf, wobei die Variable `test` auf 1 bis 10 gesetzt ist. Anschließend wird 11 zurückgegeben, was dem Wert des zuletzt ausgewerteten Ausdrucks entspricht.

```
(setq test 1)
(while (<= test 10)
  (some-func test)
  (setq test (1+ test))
)
```

13.213 write-char

Schreibt ein Zeichen auf den Bildschirm oder in eine geöffnete Datei.

(write-char ***Zahl [Dateideskr]*****)**

Das Argument *Zahl* ist einerseits der dezimale ASCII-Code für das zu schreibende Zeichen und andererseits der von der Funktion **write-char** zurückgegebene Wert.

`(write-char 67)` *ergibt* `67`

und schreibt den Buchstaben C auf den Bildschirm. Angenommen, `f` ist der Deskriptor für eine geöffnete Datei, dann gilt:

`(write-char 67 f)` *ergibt* `67`

und schreibt den Buchstaben `C` in diese Datei.

Die verschiedenen Betriebssysteme, unter denen AutoCAD läuft, arbeiten mit unterschiedlichen Konventionen, um das Zeilenende in einer ASCII-Textdatei anzuzeigen. UNIX-Systeme benutzen zum Beispiel ein einzelnes Zeilenumbruchzeichen (LF, ASCII-Code 10), während DOS-Systeme für den gleichen Zweck eine Tastenkombination einsetzen (CR/LF, ASCII-Codes 13 und 10). Um die Entwicklung von AutoLISP-Programmen zu erleichtern, konvertiert die Funktion **write-char** ein Zeilenumbruchzeichen (ASCII-Code 10) in ein Zeilenendezeichen (oder eine Folge von Zeichen), das von dem Betriebssystem verwendet wird, mit dem Sie gerade arbeiten. Also gilt für ein DOS-System

`(write-char 10 f)` *ergibt* `10`

Dabei wird jedoch die Zeichenfolge CR/LF (ASCII-Codes 13 und 10) in die Datei geschrieben. Die Funktion **write-char** kann kein NUL-Zeichen (ASCII-Code 0) in eine Datei schreiben.

Siehe auch

appendix A, "ASCII-Codes", Eine Liste von ASCII-Codes finden Sie in appendix A, "In diesem Anhang werden die Standard-ASCII-Codes aufgeführt. Die Oktalzahl eignet sich für Zeichen- oder Zeichenfolgenkonstanten im Format \nnn. In Ihrem S <XRefEnd >

13.214 write-line

Schreibt eine Zeichenkette auf den Bildschirm oder in eine geöffnete Datei.

(write-line ***Z_kette [Dateideskr]*****)**

Diese Funktion gibt das Argument `Z_kette` regulär in Anführungszeichen zurück, läßt die Anführungszeichen jedoch weg, wenn das Argument in eine Datei geschrieben wird. Angenommen, `f` ist ein gültiger Dateideskriptor für eine geöffnete Datei, dann gibt die Funktion

`(write-line "Test" f)` *den Text* `Test` *aus und gibt* `"Test"` *zurück*

13.215 xdroom

Gibt die Größe des Bereichs für erweiterte Daten (X-Daten) zurück, der für ein Objekt (Element) zur Verfügung steht.

(xdroom ***Elem_name*****)**

Bei nicht erfolgreicher Durchführung gibt **xdroom** `nil` zurück. Da es für den Umfang erweiterter Daten, die der Definition eines Elements zugewiesen werden können, eine Beschränkung gibt (momentan 16 KB) und weil verschiedene Anwendungen dem gleichen Element erweiterte Daten zuweisen können, ist diese Funktion als Kontrollmechanismus notwendig, so daß eine Anwendung überprüfen kann, ob Platz für die erweiterten Daten zur Verfügung steht. Diese Funktion kann in Verbindung mit der Funktion **xdsize** aufgerufen werden, welche die Größe einer Liste erweiterter Daten zurückgibt.

Es folgt ein Beispiel, das den für erweiterte Daten eines Ansichtsfensterobjekts zur Verfügung stehenden Platz ermittelt. Angenommen, die Variable `vpname` enthält den Namen eines Ansichtsfensterobjekts, dann gilt

`(xdroom vpname)` *ergibt* `16162`

In diesem Beispiel stehen 16,162 Bytes der ursprünglichen 16,383 Bytes des Platzes für die erweiterten Daten zur Verfügung, was bedeutet, daß 221 Bytes belegt sind. Mit der Funktion **xdsize** können Sie den für erweiterte Daten zur Verfügung stehenden Platz ermitteln.

13.216 xdsize

Gibt die Größe (in Byte), die eine Liste in Anspruch nimmt, wenn Sie in Form von erweiterten Daten mit einem Objekt (Element) verbunden ist.

(xdsize ***Liste*****)**

Bei nicht erfolgreicher Durchführung gibt **xdsize** `nil` zurück. Das Argument `Liste` muß eine gültige Liste erweiterter Daten sein, die einen zuvor mit Hilfe der Funktion **regapp** registrierten Anwendungsnamen enthält. Felder in geschweiften Klammern (Gruppencode 1002) müssen im richtigen Verhältnis vorhanden sein. Ein ungültiges Argument `Liste` erzeugt einen Fehler und plaziert den entsprechenden Fehlercode in der Variablen ERRNO . Enthalten die erweiterten Daten einen nicht registrierten Anwendungsnamen, erscheint die folgende Fehlermeldung (vorausgesetzt, die Variable CMDECHO ist auf ‚ein" gesetzt):

Ungültiger Anwendungsname in der Gruppe 1001

Das Argument `Liste` kann mit einem -3-Gruppen-Code beginnen (der Satzmarke für erweiterte Daten), dies ist jedoch keine Bedingung. Da erweiterte Daten Informationen aus verschiedenen Anwendungen enthalten können, muß die Liste von Klammern umschlossen werden.

```
(-3 ("MEINANW" (1000 . "SUITOFARMOR")
               (1002 . "{")
               (1040 . 0.0)
               (1040 . 1.0)
               (1002 . "}")
    )
)
```

Es folgt das gleiche Beispiel ohne den -3-Gruppen-Code. Diese Liste besteht nur aus der Funktion **cdr** des ersten Beispiels, wichtig ist jedoch, daß die umschließenden Klammern vorhanden sind:

```
( ("MEINANW" (1000 . "SUITOFARMOR")
             (1002 . "{")
             (1040 . 0.0)
             (1040 . 1.0)
             (1002 . "}")
   )
)
```

13.217 xload

Lädt eine ADS- Anwendung.

(xload ***Anwendung [beiFehler]*****)**

Das Argument *Anwendung* wird als Zeichenkette in Anführungszeichen oder als Variable eingegeben, die den Namen einer ausführbaren Datei enthält. Wenn die Datei geladen ist, wird sichergestellt, daß es sich um eine gültige ADS-Anwendung handelt. Die ADS-Programmversion, ADS selber und die laufende AutoLISP-Version werden auf Kompatibilität überprüft.

Wenn die **xload**-Operation scheitert, wird normalerweise ein AutoLISP-Fehler erzeugt. Ist jedoch das Argument *beiFehler* vorhanden, gibt **xload** bei einem Fehler den Wert dieses Arguments anstatt einer Fehlermeldung zurück.

Wurde die Anwendung erfolgreich geladen, wird der Anwendungsname zurückgegeben.

`(xload "/meinanw/xapp")` *ergibt bei Erfolg* `"/meinanw/xapp"`

Wenn Sie versuchen, eine bereits geladene Anwendung zu laden, zeigt **xload** folgende Meldung an

Anwendung "*Anwendung*" bereits geladen.

Weiterhin wird der Anwendungsname zurückgegeben. Sie können die zur Zeit geladenen ADS-Anwendungen mit der Funktion **ads** überprüfen, bevor Sie **xload** verwenden.

13.218 xunload

Beendet eine ADS-Anwendung.

(xunload ***Anwendung [beiFehler]*****)**

Ist die Anwendung erfolgreich beendet, wird der Anwendungsname zurückgegeben, andernfalls wird eine Fehlermeldung zurückgegeben.

Geben Sie *Anwendung* als Zeichenkette in Anführungszeichen ein oder als Variable, die den Namen einer mit **xload** geladenen Anwendung enthält. Der Anwendungsname muß auf genau die gleiche Weise wie bei der Funktion **xload** eingegeben werden. Wenn für die Anwendung bei **xload** ein Pfad (Verzeichnisname) eingegeben wurde, so kann dies bei der Funktion **xunload** weggelassen werden.

Die folgende Funktion beendet beispielsweise die vorher mit der Funktion **xload** geladene Anwendung:

`(xunload "ame")` *ergibt bei Erfolg* `"ame"`

Wenn die **xunload**-Operation scheitert, wird normalerweise ein AutoLISP-Fehler erzeugt. Ist jedoch das Argument *beiFehler* vorhanden, gibt **xunload** bei einem Fehler den Wert dieses Arguments anstatt einer Fehlermeldung zurück. Diese Eigenschaft von **xunload** entspricht der der Funktion **xload**.

13.219 zerop

Überprüft, ob eine Zahl Null ergibt.

(zerop *Zahl*)

Gibt `T` zurück, wenn *`Zahl`* Null ist, andernfalls wird `nil` zurückgegeben.

`(zerop 0)`	*ergibt*	`T`
`(zerop 0.0)`	*ergibt*	`T`
`(zerop 0.0001)`	*ergibt*	`nil`

14 Zugriff auf extern definierte Befehle und Systemvariablen

AutoCAD-Befehle und Systemvariablen, die von ARX- oder AutoLISP-Anwendungen definiert wurden, werden als *extern definiert* bezeichnet. Auf extern definierte Befehle greifen AutoLISP-Anwendungen anders zu als auf interne AutoCAD-Befehle. Viele extern definierte Befehle haben ihre eigene Programmierschnittstelle, die es AutoLISP-Anwendungen erlaubt, ihre Funktionalität zu nutzen.

Weitere Informationen über die in diesem Kapitel beschriebenen Befehle siehe *AutoCAD Befehlsreferenz*.

14.1 3DSIN

Liest eine 3D Studio (*.3ds*)-Datei ein

(c:3dsin *Modus [Multimat Erstell] Datei*)

Das Argument `Modus` ist eine Ganzzahl, die festlegt, ob der Befehl interaktiv (`Modus` = 1) oder nicht-interaktiv (`Modus` = 0) benutzt werden soll. Wenn `Modus` auf 0 gesetzt wird, sind die Argumente `Multimat` und `Erstell` zusätzlich zu dem Argument `Datei` erforderlich. Wenn `Modus` auf 1 gesetzt wird, ist nur das Argument `Datei` erforderlich. Das Argument `Datei` ist eine Zeichenkette, in der die zu importierende .3ds-Datei angegeben wird. Die .3ds-*Erweiterung ist erforderlich.*

Der folgende Aufruf öffnet die 3D Studio-Datei *globe.3ds* zum Einlesen und fordert dann den Benutzer zur Eingabe von Parametern auf:

```
(c:3dsin 1 "globe.3ds")
```

Die Argumente `Multimat` und `Erstell` sind Ganzzahlen, die festlegen, wie Objekte aus mehreren Materialien behandelt und neue Objekte erstellt werden sollen.

Modus 0 (nicht-interaktiv) hat Argumente, die festlegen, wie mit Objekten aus mehreren Materialien umgegangen werden soll und wie neue Objekte erstellt werden sollen. Mit diesem Modus werden immer alle Objekte aus der *.3ds*-Datei eingelesen.

Argumentwerte von Multimat und Erstell

Argument	Beschreibung
Multimat	Legt fest, wie Objekte aus mehreren Materialien behandelt werden sollen: 0 - Erstellt für jedes Material ein neues Objekt 1 - Weist dem neuen Objekt das erste Material zu
Erstell	Legt fest, wie neue Objekte erstellt werden sollen: 0 - Erstellt einen Layer für jedes 3DS-Objekt 1 - Erstellt einen Layer für jede 3DS-Farbe 2 - Erstellt einen Layer für jedes 3DS-Material 3 - Plaziert alle neuen Objekte auf einen einzigen Layer

Dieser Aufruf liest die gesamte Datei *globe.3ds* ohne Benutzereingaben ein, trennt Objekte aus mehreren Materialien und plaziert alle neuen Objekte auf denselben Layer:

```
(c:3dsin 0 0 3 "globe.3ds")
```

Extern definierte Funktion *render* ARX-Anwendung

14.2 3DSOUT

Erstellt eine 3D Studio-Datei

(c:3dsout *Asatz Amodus Div Glätt Verschw Datei*)

Die Funktion **`c:3dsout`** hat sechs erforderliche Argumente. Das Argument `Asatz` ist ein Auswahlsatz, der alle zu erstellenden AutoCAD-Objekte enthält. Das Argument `Amodus` ist eine Ganzzahl, die den Ausgabemodus für die Darstellung der AutoCAD-Daten angibt. Momentan ist die 3DSOUT-Ausgabe identisch, ganz gleich, ob `Amodus` auf 0 oder 1 gesetzt ist. Das Argument `div` ist eine Ganzzahl, die angibt, wie AutoCAD-Objekte in 3D Studio-Objekte gesplittet werden. Wenn `div` auf 0 gesetzt wird, wird für jeden AutoCAD-Layer ein Objekt erzeugt. Wenn `div` auf 1

gesetzt wird, wird ein Objekt für jede AutoCAD-Farbe erzeugt, und wenn es auf 2 gesetzt wird, wird ein Objekt für jeden AutoCAD-Objekt-Typ erzeugt. Das Argument `Glätt` ist eine Ganzzahl, die den Grenzwinkel für die automatische Glättung festlegt.Wenn `Glätt` auf -1 gesetzt wird, erfolgt keine automatische Glättung, wird der Wert 0-360 eingestellt, erzeugt AutoCAD eine Glättung, wenn der Winkel zwischen den Flächennormalen unter diesem Wert liegt. Das Argument `Verschw` ist eine reale Zahl, die den Abstandsgrenzwert für die Verschweißung benachbarter Scheitelpunkte angibt. Wenn `Verschw` auf einen Wert gesetzt wird, der kleiner als 0 ist, ist die Verschweißung gesperrt, wird es auf einen Wert größer oder gleich 0 gesetzt, verschweißt AutoCAD Kontrollpunkte, die näher als dieser Wert zusammenliegen. Das letzte Argument, `Datei`, ist eine Zeichenkette, welche den Namen der zu erzeugenden 3D Studio-Datei angibt. Die *.3ds*-Erweiterung ist erforderlich.

Der folgende Aufruf exportiert die gesamte Zeichnung und erstellt 3D Studio-Objekte ausgehend vom Layer der Zeichnung mit einem Glättungsgrenzwert von 30 Grad und einem Verschweißungsabstand von 0.1:

```
(c:3dsout (ssget "X") 0 0 30 0.1 "testav.3ds")
```

Extern definierte Funktion *render* ARX-Anwendung

14.3 AUSRICHTEN

Verschiebt Objekte parallel, dreht sie und ermöglicht dadurch ihre Ausrichtung an anderen Objekten

(ausrichten ***arg1 arg2 ...*****)**

Reihenfolge, Anzahl und Typ der Argumente für die Funktion **ausrichten** entsprechen genau der Eingabe in der Eingabeaufforderung. Um eine leere Antwort zu kennzeichnen (RETURN), verwenden Sie `nil` oder eine leere Zeichenkette (`""`). Bei erfolgreicher Durchführung gibt **ausrichten** den Wert `T` zurück, andernfalls wird `nil` zurückgegeben.

Im folgenden Beispiel werden zwei Paare von Quell- und Zielpunkten festgelegt, mit denen eine 2D-Bewegung durchgeführt wird.

```
(setq As (ssget))
(ausrichten As s1 d1 s2 d2 "" "2d")
```

Anmerkung AutoLISP-Unterstützung für die Funktion **ausrichten** wird durch die Benutzung der Bibliothek SAGET implementiert.

Extern definierte Funktion *geom3d* ARX-Anwendung

14.4 ASEADMIN

Führt Verwaltungsfunktionen für externe Datenbank-Befehle aus.

(command "_aseadmin" ***arg1 arg2 ...*****)**

Reihenfolge, Anzahl und Typ der Argumente entsprechen genau denen, die Sie bei der Eingabeaufforderung eingeben würden.

Extern definierte Funktion *ase* ARX-Anwendung

14.5 ASEEXPORT

Erstellt Verknüpfungsinformationen für ausgewählte Objekte

(command "_aseexport" ***arg1 arg2 ...*****)**

Reihenfolge, Anzahl und Typ der Argumente entsprechen genau denen, die Sie bei der Eingabeaufforderung eingeben würden.

Extern definierte Funktion *ase* ARX-Anwendung

14.6 ASELINKS

Manipuliert Verknüpfungen zwischen Objekten und externen Datenbankdaten

`(command "_aselinks"` *arg1 arg2 ...*`)`

Reihenfolge, Anzahl und Typ der Argumente entsprechen genau denen, die Sie bei der Eingabeaufforderung eingeben würden.

Extern definierte Funktion *ase* ARX-Anwendung

14.7 ASEROWS

Manipuliert Tabellendaten von externen Datenbankdaten

`(command "_aserows"` *arg1 arg2 ...*`)`

Reihenfolge, Anzahl und Typ der Argumente entsprechen genau denen, die Sie bei der Eingabeaufforderung eingeben würden.

Extern definierte Funktion *ase* ARX-Anwendung

14.8 ASESELECT

Erstellt einen Auswahlsatz aus Text- und Grafik-Auswahlsätzen

`(command "_aseselect"` *arg1 arg2 ...*`)`

Reihenfolge, Anzahl und Typ der Argumente entsprechen genau denen, die Sie bei der Eingabeaufforderung eingeben würden.

Extern definierte Funktion *ase* ARX-Anwendung

14.9 ASESQLED

Führt SQL-Anweisungen aus

`(command "_asesqled"` *arg1 arg2 ...*`)`

Reihenfolge, Anzahl und Typ der Argumente entsprechen genau denen, die Sie bei der Eingabeaufforderung eingeben würden.

Extern definierte Funktion *ase* ARX-Anwendung

14.10 HINTERGRUND

Stellt den Hintergrund eines gerenderten Bilds ein

`(c:hintergrund` *Modus [Optionen]*`)`

Mit dem Befehl HINTERGRUND können Sie den Hintergrund eines gerenderten Bildes einstellen. Dieser Befehl enthält vier Modi, die in der folgenden Tabelle beschrieben werden. Jeder Modus wird durch ein Zeichenketten-Argument angegeben, wobei die `Optionen`-Argumente vom `Modus` abhängig sind.

HINTERGRUND-Modi

Modus	Beschreibung
SOLID	Verwendung einer einzelnen Farbe als Hintergrund
GRADIENT	Erzeugt einen dreifarbigen Übergang-Hintergrund
IMAGE	Verwendet eine Bilddatei als Hintergrund
MERGE	Verschmelzen mit dem aktuellen Bildschirm

Extern definierte Funktion *render* ARX-Anwendung

14.10.1 SOLID

Der Modus Solid steuert, ob eine spezielle Volumenkörper-Farbe als Hintergrund verwendet wird, oder ob die von AutoCAD spezifizierte Hintergrundfarbe verwendet wird.

`(c:hintergrund "solid" "acad" |` *Farbe* `)`

Die Argumente des Modus "solid" werden in der folgenden Tabelle beschrieben:

SOLID - Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
ACAD	STR	Legt fest, daß die AutoCAD-Hintergrundfarbe verwendet werden soll	"ACAD"
Farbe	LIST (von realen Zahlen)	Einzustellende Farbe	Keine

14.10.2 GRADIENT

Dieser Modus erzeugt einen Übergang mit zwei oder drei Farben, der in einem beliebigen Winkel gedreht ist.

`(c:hintergrund "gradient"` *Farbe1 Farbe2 Farbe3 [Winkel [Horizont [Höhe]]]*`)`

Die Argumente im Modus "gradient" werden in der folgenden Tabelle beschrieben.

GRADIENT - Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Farbe1	LIST (von realen Zahlen)	Farbe des oberen Bandes des Übergangs	Keine
Farbe2	LIST (von realen Zahlen)	Farbe des mittleren Bandes des Übergangs	Keine
Farbe3	LIST (von realen Zahlen)	Farbe des unteren Bandes des Übergangs	Keine
Winkel	REAL	Drehung des Horizonts gegen den Uhrzeigersinn	0.0
Horizont	REAL	Mitte eines Übergangs mit zwei Farben oder Mitte des mittleren Bandes eines Übergangs mit drei Farben	0.5
Höhe	REAL	Prozentsatz (0.0->1.0) der Höhe des mittleren Bandes des Übergangs	0.3

```
(c:hintergrund "gradient"
  '(1.0 0.0 0.0) '(0.0 1.0 0.0) '(0.0 0.0 1.0) (30.0 0.6 0.1))
```

Erzeugt einen Übergang mit drei Farben, obere Farbe rot, mittlere Farbe grün, untere Farbe blau, gedreht um 30 Grad mit einem schmalen (10 %) grünen Band, dessen Mitte sich 60 % über dem Boden befindet.

14.10.3 IMAGE

Dieser Modus verwendet eine Bilddatei als Hintergrund. Es kann entweder beliebig neu skaliert werden, oder so, daß es auf den Bildschirm paßt. Es kann auch gekachelt oder zugeschnitten werden.

(c:hintergrund "image" ***Datei [fit [Winkel | [SkalierungX [SkalierungY [AbstandX [AbstandY [Feld]]]]]]]]*****)**

Die Argumente des Modus "image" werden in der folgenden Tabelle beschrieben:

IMAGE - Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Datei	STR	Name der Datei	Keiner
Eingepaßt	STR	Modus Eingepaßt: FIT - Neuskalieren sowohl von x als auch y, um die Ausgabe anzugleichen FIT ASPECT - Neuskalieren mit gleicher x- und y-Skalierung und so anpassen, daß die kleinere Abmessung in die Ausgabe paßt	FIT
Winkel	REAL	Ignoriert	0.0
SkalierungX	REAL	Die x-Skalierung	1.0
SkalierungY	REAL	Die y-Skalierung	1.0
AbstandX	REAL	Der x-Abstand (Mitte des Bildes gegenüber der Mitte der Ausgabe)	0.0
AbstandY	REAL	Der y-Abstand	0.0
Feld	INT	Kacheln: 0 - Zuschneiden 1 - Kacheln	1

```
(c:hintergrund "image" "valley_l.tga" "fit")
```

Verwendet valley_l.tga als Hintergrund, und das Bild wird neu skaliert, so daß es auf den Bildschirm paßt.

14.10.4 MERGE

Dieser Modus dient zum Verschmelzen der Hintergrundes der aktuellen Sicht als Hintergrund des neuen Bildes. Es sind keine Parameter vorhanden.

(c:hintergrund "merge")

14.10.5 ENVIRONMENT

Dieser Modus setzt die globale Reflection Map entweder auf eine angegebene Datei oder auf den aktuellen Hintergrund, das aktuelle Bild, den aktuellen Übergang oder den aktuellen Volumenkörper.

(c:hintergrund "environment" ***lock | Datei*****)**

Die Argumente im Modus "environment" werden in der folgenden Tabelle beschrieben.

ENVIRONMENT - Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Sperren	STR	Legt fest, daß die globale Environment-Map mit dem aktuellen Hintergrund verriegelt werden soll. Der Parameter ist "LOCK".	LOCK

Datei	STR	Name der Umgebungs-Datei.	Keiner

```
(c:hintergrund "environment" "lock")
```

14.11 KAL

Ruft den Online-Geometrierechner auf und gibt die Werte des berechneten Ausdrucks zurück

(KAL *Ausdruck*)

Das Argument `Ausdruck` ist eine Zeichenkette in Anführungszeichen. Eine Beschreibung der gültigen Werte finden Sie unter "'CAL" in der *AutoCAD Befehlsreferenz.*

Das folgende Beispiel benutzt KAL in einem AutoLISP-Ausdruck mit der Funktion **trans** :

```
(trans (kal "[1,2,3]+MID") 1 2)
```

Extern definierte Funktion *geomcal* ARX-Anwendung

14.12 NEBEL

Ermöglicht es Ihnen, Abstand von der Kamera hinzuzufügen

(c:nebel *aktiviert [Farbe [Kurze_Entf [Weite_Entf [Naher_Prozentsatz [Ferner_Prozentsatz [Hintergrund]]]]]]*)

Mit dem Befehl NEBEL können Sie Bildschirminformation über den Abstand von Objekten vom Kameraobjektiv angeben. Um den Nebel zu maximieren, addieren Sie Weiß zum Bild, um die Bildtiefenerzeugung zu maximieren, addieren Sie Schwarz. Dieser Befehl enthält sieben Modi, die in der folgenden Tabelle beschrieben werden.

NEBEL - Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
aktiviert	STR	Schaltet Nebel ein und aus, ohne die anderen Einstellungen zu beeinflussen.	EIN (Nebel einschalten)
Farbe	3D-Punkt	Die Farbeinstellungen sind die vorgegebenen AutoCAD-Farbwerte.	(111)
Kurze_Entf	REAL	Definiert, wo Nebel beginnt.	0.0
Weite_Entf	REAL	Definiert, wo Nebel endet.	1.0
Naher_Prozentsatz	REAL	Definiert den Prozentsatz für den Nebel zu Beginn der Bank.	0.0
Ferner_Prozentsatz	REAL	Definiert den Prozentsatz für den Nebel am Ende der Bank.	1.0
Hintergrund	STR	Wendet Nebel auf den Hintergrund und die Geometrie an.	AUS (Funktion Nebel nicht auf den Hintergrund anwenden)

Nil oder fehlende nachgestellte Argumente werden nicht geändert.

Extern definierte Funktion *render* ARX-Anwendung

14.13 LICHT

Erstellt, modifiziert und löscht Lichter und Beleuchtungseffekte

(c:licht ***Modus [Optionen]*****)**

Mit dem Befehl LICHT können Sie ein neues Licht hinzufügen und ein vorhandenes Licht modifizieren oder löschen. Diese Funktion hat acht Modi, die in der folgenden Tabelle beschrieben werden. Jeder Modus wird durch ein Zeichenketten-Argument angegeben, wobei die `Optionen`-Argumente vom `Modus` abhängig sind.

LICHT - Funktionsmodi

Modus	Beschreibung
A	Stellt die Intensität des Umgebungslichts ein oder ruft sie neu auf
D	Löscht vorhandene Lichtquellen
L	Listet alle Lichtquellen in der Zeichnung auf oder gibt eine Definition der angegebenen Lichtquelle zurück
M	Modifiziert vorhandene Lichtquellen
ND	Erstellt eine neue entfernte Lichtquelle
NP	Erstellt ein neues Punktlicht
NS	Erstellt ein neues Spotlicht
R	Benennt eine vorhandene Lichtquelle um

Anmerkung Dieser Befehl kann im Papierbereich *nicht* verwendet werden.

Extern definierte Funktion *render* ARX-Anwendung

14.13.1 A - Umgebungslicht

Stellt die Intensität des Umgebungslichts ein oder ruft sie neu auf.

(c:licht "A" ***[Intensität [Farbe]]*****)**

Das Argument `Intensität` ist eine reale Zahl von 0.0 bis 1.0; wenn `Intensität` weggelassen wird, ist die Voreinstellung 1.0. Das Argument `Farbe` ist eine Liste, mit der ein beliebiger RGB-Tripel angegeben wird, wenn er weggelassen wird, ist die Voreinstellung (1.0 1.0 1.0).

Um beispielsweise die Intensität des Umgebungslichts auf 0.6 einzustellen, geben Sie ein:

```
(c:licht "A" 0.6)
```

Um die aktuelle Intensität des Umgebungslichts auszugeben, geben Sie das Argument `Intensität` nicht an.

`(c:licht "A")` *gibt* `(0.3 (1.0 1.0 1.0))` *zurück, was eine Intensität von 0.3 und eine Farbeinstellung von (1.0 1.0 1.0) bedeutet*

14.13.2 D - Löscht Lichtquellen

Löscht vorhandene Lichtquellen.

(c:licht "D" ***Name*****)**

Das Argument `Name` ist eine Zeichenkette, die den Namen der zu löschenden Lichtquelle angibt.

```
(c:licht "D" "OLDLGT")
```

14.13.3 L - Listet Lichtquellen auf

Listet alle Lichtquellen in der Zeichnung auf oder gibt eine Definition der angegebenen Lichtquelle zurück.

(c:licht "L" ***[Name]*****)**

Das Argument `Name` ist eine Zeichenkette, die den Namen der aufzulistenden Lichtquelle angibt. Wenn das Argument `Name` weggelassen wird, gibt `c:licht` eine Liste aller in der Zeichnung definierten Lichtquellen. Wenn Sie `Name` angeben, gibt **`c:licht`** die Definition der genannten Lichtquelle.

`(c:licht "L")` *gibt* `("SPOT1")`

```
(c:licht "L" "SPOT1")       gibt:
  ("S" <Elementname: 600039c6> 157.22 (-97.8002 1.21954e-14 70.0222)
(13.8456 20.103 0.0) (0.944 0.89824 0.736) 0 12.0 16.0 3.0 "EIN")
```

14.13.4 M - Modifiziert Lichtquellen

Modifiziert vorhandene Lichtquellen.

`c:licht "M"` ***Name [Intensität [von [nach [Farbe [Größe der Shadow-Map [max. Licht [min. Licht [Schattenweichheit [Schatten [Schattenobjekte [Monat [Tag [Stunde [Minute*** **`[Sommerzeit [geogr. Breite [geogr. Länge`** ***[Lichtintensitätsverlust]]]]]]]]]]]]]]]]]]*)**

Die Argumente des Modus "Modifizieren" werden in der folgenden Tabelle beschrieben:

LICHT - "M" Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name einer einzelnen Lichtquelle	Keiner
Intensität	REAL	Eine reale Zahl zwischen 0.0 und dem vorgegebenen Maximum	Basiert auf der Lichtabnahme
von	LIST	Position der Lichtquelle	Aktueller Ausgangsblickpunkt
nach	LIST	Ziel der Lichtquelle	Aktueller Zielblickpunkt
Farbe	LIST	Ein beliebiges RGB-Tripel	1.0, 1.0, 1.0
Größe der Shadow-Map	INT	Ganzzahl zwischen 0 und 4096 (Größe in Pixel auf einer Seite der Shadow-Map)	0
max. Licht	REAL	Winkel des Helligkeitsstrahls in Grad (muß zwischen 1 und 160 liegen)	44.0
min. Licht	REAL	Winkel, der den Bereich der raschen Lichtabnahme einschließt in Grad (muß zwischen 0 und 160 liegen und größer sein als der Wert für das maximale Licht)	45.0
Schattenweichheit	REAL	Reale Zahl zwischen 0.0 und 10.0	0.0
Schatten	STR	Ein- und Ausschalten von Schattenwurf. Gültige Werte sind: "aus" (kein Schattenwurf) und "ein" (Schattenwurf)	0.0
Schattenobjekte	ENAME	Eine Auswahl von Objekten, die die Shadow-Maps begrenzen.	0.0
Monat	INT	Ganzzahl zwischen 1 und 12	9
Tag	INT	Ganzzahl zwischen 1 und 31	21
Stunde	INT	Ganzzahl zwischen 0 und 24	15
Minute	INT	Ganzzahl zwischen 0 und 59	0
Sommerzeit	STR	Sommerzeitumschaltung. Gültige Werte sind: "aus" (keine Sommerzeit) und "ein" (Sommerzeit)	"aus"
Geogr. Breite	REAL	Reale Zahl zwischen 0 und 90	37.62

Geogr. Länge	REAL	Reale Zahl zwischen 0 und 180	122.37
Zeitzone	INT	Ganzzahl zwischen -12 und 12, die die Stunden der Greenwicher Zeit (GMT) repräsentieren	8 (PST)
Lichtintensitäts-verlust	INT	0 = keine Abnahme 1 = invers-lineare Abnahme 2 = invers-quadratische Abnahme	1

Die Argumente max. Licht und min. Licht wirken sich nur auf Spotlichter aus. Sie müssen sie auf `nil` setzen, wenn Sie eine neue entfernte Lichtquelle erstellen.

Wenn ein Argument `nil` ist oder am Ende der Argumentliste ausgelassen wurde, behält es seinen aktuellen Wert. Wenn ein Argument für die Art der Lichtquelle, die Sie modifizieren wollen, nicht gilt, setzen Sie es auf `nil` - oder lassen Sie es aus, wenn es am Ende der Argumentliste steht.

Der folgende Code ändert zum Beispiel die Farbe der entfernten Lichtquelle D1 in Blau.

```
(c:licht "M" "D1" nil nil nil '(0.0 0.0 1.0))
```

14.13.5 ND - Neue entfernte Lichtquelle

Erstellt eine neue entfernte Lichtquelle.

`(c:licht "ND"` ***Name [Intensität [von [nach [Fabe]]]]***)

Die Argumente für den Modus "Neue entfernte Lichtquelle" werden in der folgenden Tabelle beschrieben:

LICHT - "ND"-Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name einer einzelnen Lichtquelle	Keiner
Intensität	REAL	Eine reale Zahl zwischen 0.0 und dem vorgegebenen Maximum	Basiert auf der Lichtabnahme
von	LIST	Position der Lichtquelle	Aktueller Ausgangsblickpunkt
nach	LIST	Ziel der Lichtquelle	Aktueller Zielblickpunkt
Farbe	LIST	Ein beliebiges RGB-Tripel	1.0, 1.0, 1.0
Größe der Shadow-Map	INT	Ganzzahl zwischen 0 und 4096 (Größe in Pixel auf einer Seite der Shadow-Map)	0
max. Licht	REAL	Winkel des Helligkeitsstrahls in Grad (muß zwischen 1 und 160 liegen)	44.0
min. Licht	REAL	Winkel, der den Bereich der raschen Lichtabnahme einschließt in Grad (muß zwischen 0 und 160 liegen und größer sein als der Wert für das maximale Licht)	45.0
Schattenweich-heit	REAL	Reale Zahl zwischen 0.0 und 10.0	0.0
Schatten	STR	Ein- und Ausschalten von Schattenwurf. Gültige Werte sind: "aus" (kein Schatten) und "ein" (Schattenwurf)	0.0
Schattenobjekte	ENAME	Eine Auswahl von Objekten, die die Shadow-Maps begrenzen.	0.0

Monat	INT	Ganzzahl zwischen 1 und 12	9
Tag	INT	Ganzzahl zwischen 1 und 31	21
Stunde	INT	Ganzzahl zwischen 0 und 24	15
Minute	INT	Ganzzahl zwischen 0 und 59	0
Sommerzeit	STR	Sommerzeitumschaltung. Gültige Werte sind: "aus" (keine Sommerzeit) und "ein" (Sommerzeit)	"aus"
Geogr. Breite	REAL	Reale Zahl zwischen 0 und 90	37.62
Geogr. Länge	REAL	Reale Zahl zwischen 0 und 180	122.37
Zeitzone	INT	Ganzzahl zwischen -12 und 12, die die Stunden der Greenwicher Zeit (GMT) repräsentieren	8 (PST)

14.13.6 NP - Neues Punktlicht

Erstellt ein neues Punktlicht.

`(c:licht "NP"` ***Name [Intensität [von [***`nil` ***[Farbe [Größe der Shadow-Map [***`nil` ***[***`nil` ***[Schattenweichheit [Schatten [Lichtintensitätsverlust [Schattenobjekte]]]]]]]]]]***`)`

Die Argumente für den Modus "Neues Punktlicht" werden in der folgenden Tabelle beschrieben:

LICHT - "NP"- Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name einer einzelnen Lichtquelle	Keine
Intensität	REAL	Eine reale Zahl zwischen 0.0 und dem vorgegebenen Maximum	Basiert auf der Lichtabnahme
von	LIST	Position der Lichtquelle	Aktueller Ausgangsblickpunkt
Farbe	LIST	Ein beliebiges RGB-Tripel	1.0, 1.0, 1.0
Größe der Shadow-Map	INT	Ganzzahl zwischen 0 und 4096 (Größe in Pixel auf einer Seite der Shadow-Map)	0
Schattenweich-heit	REAL	Reale Zahl zwischen 0.0 und 10.0	0.0
Schatten	STR	Ein- und Ausschalten von Schattenwurf. Gültige Werte sind: "aus" (kein Schatten) und "ein" (Schattenwurf)	0.0
Lichtintensitäts-verlust	INT	0 = keine Abnahme 1 = invers-lineare Abnahme 2 = invers-quadratische Abnahme	1
Schattenobjekte	ENAME	Eine Auswahl von Objekten, die die Shadow-Maps begrenzen.	0.0

Drei Argumente - `nach` (nach `von`), `max. Licht` und `min. Licht` (nach `Größe der Shadow-Map`) - wirken sich nicht auf Punktlichter aus. Sie müssen sie auf `nil` setzen, wenn Sie ein neues Punktlicht erstellen.

Zum Beispiel erzeugt der folgende Code ein neues Punktlicht mit der Bezeichnung NEUPT1.

```
(c:licht "NP" "NEUPT1")
```

NEUPT1 hat die vorgegebene Intensität, die derzeitige Einstellung der Lichtabnahme, das vorgegebene Positions-Aussehen an der aktuellen Sicht und die vorgegebene Farbe weiß.

Anmerkung Bei Punktlichtquellen ist die vorgegebene maximale Intensität abhängig von der momentan eingestellten Punkt-/Spotlicht-Lichtabnahme. Ohne Abnahme ist sie 1.00; bei invers linearer Abnahme ist sie der maximale Abstand der Grenzen der Zeichnung; und bei invers quadratischer Abnahme ist sie das Quadrat des maximalen Abstands der Grenzen.

14.13.7 NS - Neues Spotlicht

Erstellt ein neues Spotlicht.

`(c:licht "NS"` ***Name [Intensität [von [nach [Farbe [***`Größe der Shadow-Map`*** [max. Licht [min. Licht [Schattenweichheit [Schatten [Lichtintensitätsverlust [Schattenobjekte]]]]]]]]]]***`)`

Die Argumente des Modus "Neues Spotlicht" werden in der folgenden Tabelle beschrieben:

LICHT - "NS"- Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name einer einzelnen Lichtquelle	Keine
Intensität	REAL	Eine reale Zahl zwischen 0.0 und dem vorgegebenen Maximum	Basiert auf der Lichtabnahme
von	LIST	Position der Lichtquelle	Aktueller Ausgangsblick-punkt
nach	LIST	Ziel der Lichtquelle	Aktueller Zielblickpunkt
Farbe	LIST	Ein beliebiges RGB-Tripel	1.0, 1.0, 1.0
Größe der Shadow-Map	INT	Ganzzahl zwischen 0 und 4096 (Größe in Pixel auf einer Seite der Shadow-Map)	0
max. Licht	REAL	Winkel des Helligkeitsstrahls in Grad (muß zwischen 1 und 160 liegen)	44.0
min. Licht	REAL	Winkel, der den Bereich der raschen Lichtabnahme einschließt in Grad (muß zwischen 0 und 160 liegen und größer sein als der Wert für das maximale Licht)	45.0
Schattenweichheit	REAL	Reale Zahl zwischen 0.0 und 10.0	0.0
Schatten	STR	Ein- und Ausschalten von Schattenwurf. Gültige Werte sind: "aus" (kein Schatten) und "ein" (Schattenwurf)	0.0
Lichtintensitätsverlust	INT	0 = keine Abnahme 1 = invers-lineare Abnahme 2 = invers-quadratische Abnahme	1
Schattenobjekte	ENAME	Eine Auswahl von Objekten, die die Shadow-Maps begrenzen.	0.0

Zum Beispiel erzeugt der folgende Code ein neues Spotlicht mit der Bezeichnung NEUSPOT.

```
(c:licht "NS" "NEUSPOT" 137.82 '(12.0 6.0 24.0)
  '(78.0 78.0 24.0) nil nil 30.0 32.0)
```

NEUSPOT ist ein Spotlicht mit einer Intensität von 137.82. Seine Farbe ist die Vorgabe (weiß). Die Position des Spotlichts ist (12,6,24), sein Zielpunkt ist (78,78,24). Sein Kegel hat eine Breite von 32 Grad mit einem maximalen Helligkeitsbereich von 30 Grad.

Anmerkung Bei Spotlichtquellen ist die maximale Intensität von der momentanen Einstellung der Lichtabnahme des Punkt-/Spotlichts abhängig. Ohne Abnahme beträgt sie 1.00; bei invers-linearer Abnahme entspricht sie der maximalen Zeichnungsausdehnung; und bei invers-quadratischer Abnahme ist sie gleich dem Quadrat der maximalen Ausdehnung.

14.13.8 R - Umbenennen von Lichtquellen

Benennt eine Lichtquelle um.

(c:licht "R" ***alt_Name neu_Name*****)**

Das Argument *alt_Name* ist eine Zeichenkette, die den Namen der umzubenennenden Lichtquelle angibt. Das Argument *neu_Name* ist eine Zeichenkette, die den neuen Namen für die Lichtquelle festlegt.

```
(c:licht "R" "PUNKT5" "SEITPT")
```

14.14 LSBEARB

Erzeugt oder modifiziert Landschaftsobjekte

(c:lsbearb Modus ***[Optionen]*****)**

Der Befehl LSBEARB dient dazu, Landschaftsobjekte in der Zeichnung zu erzeugen oder zu modifizieren.

(c:lsbearb "LIST" Objekt)

Listet die Attribute des angegebenen Landschaftsobjektes auf.

(c:lsbearb Objekt Höhe [Position [Ausrichtung]])

Modifiziert ein Landschaftsobjekt, wobei *Objekt* der AutoCAD-Name (entsel) des zu modifizierenden Objektes ist. Höhe, Position und Ausrichtung können auf `nil` gesetzt sein, wobei in diesem Fall der Wert unverändert bleibt.

Die Argumente des Befehls LSBEARB werden in der folgenden Tabelle beschrieben:

LSBEARB-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Objekt	ENAME	Referenz des Landschaftsobjektes	Keine
Höhe	REAL	Höhe des Objektes in Zeicheneinheiten	Keine
Position	LIST (von realen Zahlen)	Die Position der Basis des Objektes	Keine
Ausrichtung	INT	Bestimmt die Geometrie und die Ausrichtung des Eintrags: 0 - Einzelne Fläche, auf die Kamera ausgerichtet 1 - Einzelne Fläche, nicht auf die Kamera ausgerichtet 2 - Sich kreuzende Flächen, nicht auf die Kamera ausgerichtet 3 - Sich kreuzende Flächen, auf die Kamera ausgerichtet	Keine

```
(c:lsbearb Objekt Höhe [Position [Ausrichtung]])
```

Modifiziert ein Landschaftsobjekt, wobei Objekt der AutoCAD-Name (entsel) des zu modifizierenden Objektes ist. Höhe, Position und Aurichtung können `nil` sein, wobei in diesem Fall der Wert unverändert bleibt.

```
(c:lsbearb <ename> 35.0 '(10.0 23.0) nil)
```

Modifiziert ein Landschaftsobjekt, wobei <ename> der AutoCAD-Name (entsel) des zu modifizierenden Objektes ist. Höhe, Position und Ausrichtung können `nil` sein, wobei in diesem Fall der Wert unverändert bleibt.

```
(c:lsbearb "LIST" <ename>)
```

Gibt eine Liste mit dem Namen, der Höhe, der Position und der Kamera-Ausrichtung des angegebenen Objektes zurück.

Extern definierte Funktion *render* ARX-Anwendung

14.15 LSBIBL

Verwaltet die Landschaftsbibliothek

(c:lsbibl Modus *[Optionen]*)

Die Argumente des Befehls LSBIBL werden in der folgenden Tabelle beschrieben:

LSBIBL-Argumente

Modus	Beschreibung
ADD	Hinzufügen eines Eintrages zu einer Landschaftsbibliothek
DELETE	Entfernen eines Eintrages aus der Landschaftsbibliothek
MODIFY	Ändern eines Eintrages in einer Landschaftsbibliothek
OPEN	Öffnet eine Landschaftsbibliothek
SAVE	Speichert die aktuelle Landschaftsbibliothek
LIST	Listet die Einträge in der aktuellen Landschaftsbibliothek auf

14.15.1 ADD

Hinzufügen eines Eintrages zur aktuellen Bibliothek

(c:lsbibl "ADD" *Name Texture-Map Opacity-Map Ausrichtung*)

Die Argumente des Modus ADD werden in der folgenden Tabelle beschrieben:

LSBIBL - "ADD"-Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	Zeichen-kette	Name des Eintrags in der Landschafts-bibliothek	Keiner
Texture-Map	STR	Name der Bilddatei für den Eintrag.	Keiner
Opacity-Map	STR	Name des Opacity-Bildes für den Eintrag.	Keiner
Ausrichtung	INT	Bestimmt die Geometrie und die Ausrichtung des Eintrags: 0 - Einzelne Fläche, auf die Kamera ausgerichtet 1 - Einzelne Fläche, nicht auf die Kamera ausgerichtet 2 - Sich kreuzende Flächen, nicht auf die Kamera ausgerichtet 3 - Sich kreuzende Flächen, auf die Kamera ausgerichtet	Keine

```
(c:lsbibl "ADD" "Maple tree" "maple.tga" "mapleo.tga" 0)
```

Hinzufügen eines Eintrages mit der Bezeichnung "Maple tree" zur aktuellen Landschaftsbibliothek

14.15.2 DELETE

Entfernen eines Eintrages aus der aktuellen Bibliothek

(c:lsbibl "DELETE" *Name*)

Der Modus "DELETE" hat ein Argument, das in der folgenden Tabelle beschrieben wird.

LSBIBL - "DELETE"-Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name des Eintrags in der Landschaftsbibliothek	Keiner

```
(c:lsbibl "delete" "Maple tree")
```

Entfernt den Eintrag mit der Bezeichnung "Maple tree" aus der aktuellen Landschaftsbibliothek

14.15.3 MODIFY

(c:lsbibl "MODIFY" Name Textur-Map [Opacity-Map [Ausrichtung]])

Ändert einen Eintrag in der aktuellen Bibliothek, Texture-Map, Opacity-Map und Ausrichtung können `nil` sein, wobei in diesem Fall der Wert unverändert bleibt.

Die Argumente des Modus "MODIFY" werden in der folgenden Tabelle beschrieben:

LSBIBL - "MODIFY"-Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name des Eintrags in der Landschaftsbibliothek	Keiner
Textur-Map	STR	Name der Bilddatei für den Eintrag	Keiner
Opacity-Map	STR	Name des Opacity-Bilds für den Eintrag.	Keiner
Ausrichtung	INT	Bestimmt die Geometrie und die Ausrichtung des Eintrags: 0 - Einzelne Fläche, auf die Kamera ausgerichtet 1 - Einzelne Fläche, nicht auf die Kamera ausgerichtet 2 - Sich kreuzende Flächen, nicht auf die Kamera ausgerichtet 3 - Sich kreuzende Flächen, auf die Kamera ausgerichtet	Keine

```
(c:lsbibl "MODIFY" "Maple tree" nil nil 2)
```

Ändert "Maple tree" auf sich kreuzenden Flächen, nicht auf die Kamera ausgerichtet.

14.15.4 OPEN

Öffnet eine neue Bibliothek und macht sie zur aktuellen Bibliothek.

(c:lsbibl "OPEN" *Name*)

Der Modus "OPEN" hat ein Argument, das in der folgenden Tabelle beschrieben wird.

LSBIBL - "OPEN"-Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Dateiname der zu öffnenden	Keiner

```
(c:lslib "OPEN" "TREES.LLI")
```

Liest die Datei TREES.LLI und macht sie zur aktuellen Landschaftsbibliothek.

14.15.5 SAVE

Speichert die aktuelle Landschaftsbibliothek unter dem angegebenen Namen.

(c:lsbibl "SAVE" ***Name*****)**

Der Modus "SAVE" hat ein Argument, das in der folgenden Tabelle beschrieben wird.

LSBIBL - "SAVE"-Modus-Argument

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Dateiname der zu speichernden Landschaftsbibliothek	Keiner

```
(c:lslib "SAVE" "TREES.LLI")
```

Schreibt die Datei TREES.LLI.

14.15.6 LISTE

Listet alle Einträge in der aktuellen Bibliothek auf. Dieser Befehl hat keine Argumente. Die Liste umfaßt Landschaftseinträge der Form '("NAME" "TEX-MAP" "OP-MAP" "AUSRICHTEN").

(c:lsbibl "LIST")

Beispiel (Beispielausgabe):

```
(("Bush #1" "8bush02l.tga" "8bush02o.tga" 0)
("Cactus" "8plnt15l.tga" "8plnt15o.tga" 0)
("Dawn Redwood" "8tree39l.tga" "8tree39o.tga" 0))
```

14.16 LSNEU

Erzeugen von Landschaftsobjekten

(c:lsneu Modus ***[Optionen]*****)**

Der Befehl LSNEU dient dazu, Instanzen von Landschaftsobjekten in der Zeichnung zu erzeugen.

(c:lsneu ***Objekttyp Höhe Position Ausrichtung*****)**

Die Argumente des Befehls LSNEU werden in der folgenden Tabelle beschrieben:

LSNEU-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Objekttyp	STR	Name des Eintrags in der Landschaftsbibliothek	Keiner
Höhe	REAL	Höhe des Objektes in Zeicheneinheiten	Keine
Position	LIST (von realen Zahlen)	Die Position der Basis des Objektes	Keine
Ausrichtung	INT	Bestimmt die Geometrie und die Ausrichtung des Eintrags: 0 - Einzelne Fläche, auf die Kamera ausgerichtet 1 - Einzelne Fläche, nicht auf die Kamera ausgerichtet 2 - Sich kreuzende Flächen, nicht auf die Kamera ausgerichtet 3 - Sich kreuzende Flächen, auf die Kamera ausgerichtet	Keine

```
(c:lsneu "Maple tree" 25.0 '(0.0 1.0 3.0) 1)
```

Erzeugt eine neue Version von "Maple tree", 25 Einheiten hoch, am Punkt 0, 1, 3 mit einer einzelnen Fläche, nicht auf die Kamera ausgerichtet.

Extern definierte Funktion *render* ARX-Anwendung

14.17 MATBIBL

Verwaltet Materialbibliotheken

`(c:matbibl` *Modus Name [Datei]*`)`

Das Argument `Modus`, das in der folgenden Tabelle beschrieben wird, ist eine Zeichenkette, welche die Operation festlegt, die diese Funktion ausführt. Das Argument `Name` ist eine Zeichenkette, die den Namen des einzulesenden, zu erstellenden oder zu löschenden Materials angibt. Das fakultative Argument `Datei` ist eine Zeichenkette, die den Namen der Materialbibliotheksdatei angibt. Das Argument `Datei` muß die Erweiterung *.mli* aufweisen.

MATBIBL-Funktionsmodi

Modus	Beschreibung
I	Liest ein Material aus einer Bibliothek ein
E	Exportiert ein Material zu einer Bibliothek
D	Löscht ein Material aus der Zeichnung
C	Löscht nicht zugewiesene Materialien aus der Zeichnung
L	Listet Materialien auf

Beispiel:

```
(c:matbibl "I" "messing" "render.mli")
```

Liest das Material MESSING aus der vorgegebenen AutoCAD-Render-Materialbibliothek *render.mli* ein.

Das Argument `Datei` wird im Löschmodus nicht benutzt:

```
(c:matbibl "D" "Stahl")
```

Extern definierte Funktion *render* ARX-Anwendung

14.18 3DSPIEGELN

Spiegelt ausgewählte Objekte an einer benutzerdefinierten Ebene

(3dspiegeln *arg1 arg2 ...***)**

Reihenfolge, Anzahl und Typ der Argumente für die Funktion **3dspiegeln** entsprechen genau der Eingabe in der Eingabeaufforderung. Um eine leere Antwort zu kennzeichnen (RETURN), verwenden Sie `nil` oder eine leere Zeichenkette (`""`). Bei erfolgreicher Durchführung gibt **3dspiegeln** den Wert `T` zurück, andernfalls wird `nil` zurückgegeben.

Das folgende Beispiel spiegelt ausgewählte Objekte an der *XY*- Ebene, die durch den Punkt 0,0,5 hindurchgeht, und löscht dann die ursprünglichen Objekte.

```
(setq As (ssget))
(3dspiegeln As "XY" '(0 0 5) "Y")
```

Anmerkung AutoLISP-Unterstützung für die Funktion **3dspiegeln** wird durch die Benutzung der Bibliothek SAGET implementiert.

Extern definierte Funktion *geom3d* ARX-Anwendung

14.19 PSZIEH

Steuert das Aussehen eines importierten PostScript-Bilds, während es durch den Befehl PSIN an seine Position gezogen wird

(c:pszieh *Modus***)**

Das Argument *Modus* ist eine Ganzzahl, die entweder 0 oder 1 sein sollte. Der aktuelle Wert von PSZIEH beeinflußt die interaktive Verwendung des Befehls PSIN. Wenn PSZIEH 1 ist, erzeugt PSIN das PostScript-Bild, sobald es der Benutzer zieht, um es zu skalieren. Wenn PSZIEH 0 ist, erzeugt und zeichnet PSIN nur den Begrenzungsrahmen des Bildes. Bei erfolgreicher Durchführung gibt die Funktion **c:pszieh** den alten Wert von PSZIEH zurück. Tritt ein Fehler auf, wird `nil` zurückgegeben.

Der folgende Code schaltet PSZIEH ein, indem er ihn auf 1 setzt. Der nächste interaktive Aufruf von PSIN erzeugt das PostScript-Bild, sobald der Benutzer es während der Skalierung zieht.

```
(c:pszieh 1)
```

Extern definierte Funktion *acadps* ARX-Anwendung

14.20 PSFÜLL

Füllt einen zweidimensionalen Polylinienumriß mit einem PostScript-Füllmuster

(c:psfüll *Elem Muster [arg1 [arg2]] ...* **)**

Das Argument *Elem* ist der Name der Polylinie. Das Argument *Muster* ist eine Zeichenkette, die den Namen des Füllmusters angibt. Die Zeichenkette *Muster* muß mit dem Namen des Füllmusters übereinstimmen, das in der aktuellen Datei *acad.psf* definiert ist. Die *Argumente* sind Argumente für die interne PostScript-Füll-Prozedur: Ihre Anzahl und ihr Typ entspricht den Argumenten, die für *Muster* erforderlich sind, wie in *acad.psf* definiert. Jedes Argument ist entweder eine Ganzzahl oder eine reale Zahl. Pro Muster können zwischen 0 und 25 Argumente vorhanden sein. Wenn im Aufruf weniger Argumente angegeben sind, als das Muster vorschreibt, werden für die restlichen Argumente die Vorgabewerte benutzt. Bei erfolgreicher Durchführung gibt **c:psfüll** den Wert `T` zurück, andernfalls wird *nil* zurückgegeben.

Das Grauskala-Füllmuster hat ein einziges Argument. Im folgenden Aufruf wird das vorgegebene Grauskala-Argument von 50 Prozent benutzt:

```
(c:psfüll ename "Grauskala")
```

Mit diesem Aufruf wird eine Grauskala von 10 Prozent angegeben:

```
(c:psfüll ename "Grauskala" 10)
```

PostScript-Füllungen sind in Zeichnungsdateien als erweiterte Daten gespeichert und werden durch den Anwendungsnamen `AUTOCAD_POSTSCRIPT_FIGURE` identifiziert.

Extern definierte Funktion *acadps* ARX-Anwendung

14.21 PSIN

Importiert eine PostScript-Datei

(c:psin ***Dateiname Position Maßstab*****)**

Das Argument `Dateiname` ist eine Zeichenkette, die den Namen des PostScript-Bilds enthält. Sie brauchen die Dateierweiterung *.eps* nicht anzugeben. Das Argument `Position` ist ein Punkt, der den Einfügepunkt des (anonymen) PostScript-Blocks angibt. Das Argument `Maßstab` ist eine reale Zahl, die den Skalierfaktor angibt. Bei erfolgreicher Durchführung gibt **c:psin** den Namen des neu erstellten Objektes zurück, wenn ein Fehler auftritt, wird `nil` zurückgegeben.

Der folgende Code importiert eine PostScript-Datei mit dem Namen *beispiel.eps*, fügt sie bei (24,19) ein und skaliert sie mit einem Faktor von 25:

```
(c:psin "beispiel" '(24 19) 25)
```

PostScript-Bilder sind in Zeichnungsdateien als erweiterte Daten gespeichert und werden durch den Anwendungsnamen `AUTOCAD_POSTSCRIPT_FIGURE` identifiziert.

Extern definierte Funktion *acadps* ARX-Anwendung

14.22 RENDER

Erzeugt ein realistisch schattiertes Bild eines dreidimensionalen (3D) Drahtmodells unter Verwendung von Geometrie-, Beleuchtungs- und Oberflächenfinish-Information

(c:render ***[Dateiname|Punkt1 Punkt2]*****)**

Die Argumente des Befehls RENDER werden in der folgenden Tabelle beschrieben:

RENDER-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Dateiname	STR	Name der Renderingdatei	Keiner
Punkt1	LIST (von realen Zahlen)	Erster Punkt des Zuschneidefenster	Keiner
Punkt2	LIST (von realen Zahlen)	Zweiter Punkt des Zuschneidefensters	Keiner

Wenn das Argument `Dateiname` vorhanden ist, wird das Rendering in eine Datei dieses Namens geschrieben. Wenn ein Treiber für das Rendering in eine Datei nicht konfiguriert ist, wird das Argument `Dateiname` ignoriert. Die aktuelle Konfiguration muß das Rendering in eine Datei festlegen. Das Rendering wird durch die aktuellen Einstellungen gesteuert; stellen Sie diese mit der Funktion **c:REINST** ein. Beispiel:

```
(c:reinst "Toggle" "Zuschneidefenster" "Ein")
```

Anmerkung Wenn in den aktuellen Renderingvoreinstellungen Abfrage für Auswahleinstellungen angegeben ist, und die Systemvariable PICKFIRST eingeschaltet ist, wird, wenn beim Aufruf von **c:render** eine Auswahl eingestellt ist, das Rendering der Objekte im Satz ohne weitere Eingabeaufforderung durchgeführt.

14.22.1 Einstellen der Renderfunktion auf Dateioptionen

Stellt die Renderfunktion auf Dateioptionen für das Rendern ein.

(c:rfileopt ***fileformat xres yres aratio***
<mode-specific options>**)**

In der folgenden Tabelle werden die Argumente von **c:rfileopt** beschrieben.

RFILEOPT-Argumente

Argument	Datentyp	Beschreibung
Dateiformat	STR	Bezeichner für das gewünschte Format:

		TGA = Targa-Format PCX = Z-Soft-Bitmap-Format BMP = Microsoft Windows-Format PS = PostScript TIFF = Tagged Image File-Format
xres	INT	X-Auflösung der Ausgabedatei (gültige Werte liegen zwischen 1 und 4096)
yres	INT	Y-Auflösung der Ausgabedatei (gültige Werte liegen zwischen 1 und 4096)
aratio	REAL	Pixel aspect ratio

In der folgenden Tabelle werden die gültigen Werte für das Argument `Colormode` aufgelistet. Jedes Dateiformat akzeptiert eine Untergruppe dieser Werte.

RFILEOPT-Farbmodi

Modus	Beschreibung
MONO	Monochrom
G8	256 Graustufen
C8	256 Farben
C16	16 Bit Farbe
C24	24 Bit Farbe
C32	24 Bit Farbe mit 8 Bit Alpha

Extern definierte Funktion *render* ARX-Anwendung

14.22.1.1 TGA

Gibt das Targa-Format an.

`(c:rfileopt "TGA"` *xres yres aratio colormode interlace compress bottomup*`)`

TGA-Format-Argumente

Argument	Datentyp	Beschreibung
Farbmode	STR	Farbmodus: G8, C8, C24 oder C32
interlace	INT	Interlace-Modus: 1 = kein Interlace 2 = 2:1 Interlace 4 = 4:1 Interlace
compress	STR	Komprimierung (Vorgabe = "COMP"): COMP = Komprimierung aktiviert nil = Keine Komprimierung
bottomup	STR	Unterseite oben (Vorgabe = "UP"): UP = Unterseite oben nil = Oberseite unten

```
(C:RFILEOPT "TGA" 640 480 1.0 "C32" 1 "COMP" "UP")
```

14.22.1.2 PCX

Gibt das Z-Soft Bitmap-Format an.

`(c:rfileopt "PCX"` *xres yres aratio colormode*`)`

PCX-Format-Argumente

Argument	Datentyp	Beschreibung
Farbmodus	STR	Farbmodus: MONO, G8 oder C8

```
(C:RFILEOPT "PCX" 640 480 1.0 "G8")
```

14.22.1.3 BMP

Gibt das Microsoft Windows-Bitmap-Format an.

(c:rfileopt "BMP" ***xres yres aratio colormode*****)**

BMP-Format-Argumente

Argument	Datentyp	Beschreibung
Farbmodus	STR	Farbmodus: MONO, G8 oder C8

```
(C:RFILEOPT "BMP" 640 480 1.0 "C8")
```

14.22.1.4 PS

Gibt das PostScript-Format an.

(c:rfileopt "PS" ***xres yres aratio colormode portrait imagesize [size]*****)**

PS-Format-Argumente

Argument	Datentyp	Beschreibung
Farbmodus	STR	Farbmodus: MONO, G8, C8 oder C24
Längsformat	STR	Quer- oder Hochformat (Vorgabe = "L"): P = Hochformat L = Querformat
Bildgröße	STR	Typ (Vorgabe = "A") A = Auto B = Bild C = Benutzerdefiniert
Größe	INT	Größe des Bildes

```
(C:RFILEOPT "PS" 640 480 1.0 "C24" "P" "C" 640)
```

14.22.1.5 TIFF

Gibt das Tagged Image File-Format an.

(c:rfileopt "TIFF" ***xres yres aratio colormode compress*****)**

TIFF-Format-Argumente

Argument	Datentyp	Beschreibung
Farbmodus	STR	Farbmodus: MONO, G8, C8 oder C24
compress	STR	Komprimierung (Vorgabe = "COMP"): COMP = Komprimierung aktiviert nil = Keine Komprimierung

```
(C:RFILEOPT "TIFF" 640 480 1.0 "C24" nil)
```

14.23 RENDERUPDATE

Regeneriert die Datei ent2face im nächsten Rendering

(c:renderupdate [*RU_Wert*])

Verwenden Sie den Befehl RENDERUPDATE ohne Argumente, um im nächsten Rendering die Datei *en2face* zu regenerieren. Das Argument "ALWAYS" bewirkt, daß die Datei *en2face* bei jedem Rendering regeneriert wird. Mit dem Argument "OFF" wird "ALWAYS" ausgeschaltet.

RU_Wert-Modi

Modus	Beschreibung
ALWAYS	Erzeugt eine neue Geometrie-Datei für jedes Rendering
OFF	Schaltet Render zurück auf den normalen Geometrie-Zwischenspeicher-Modus.

Extern definierte Funktion *render* ARX-Anwendung

14.24 WIEDERGABE

Zeigt ein BMP-, TGA- oder TIFF-Bild an

(c:wiedergabe *Dateiname Typ [xAb yAb xGröße yGröße]*)

Mit dem Befehl WIEDERGABE können Sie BMP-, TGA- oder TIFF-Dateien auf dem AutoCAD-Renderingbildschirm anzeigen. Benutzen Sie die Funktionen dieses Befehls, um die Bilddatei in verschiedenen Abständen zu versetzen und sie in unterschiedlichen Größen wiederzugeben. Die Funktionsargumente werden in der folgenden Tabelle beschrieben.

WIEDERGABE-Funktionsargumente

Argument	Datentyp	Beschreibung	Vorgabe
Dateiname	STR	Bilddateiname	Keiner
Typ	STR	Gültige Werte sind: BMP, TGA oder TIFF	Keiner
xAb	INT	X-Abstand des Bildes in Pixeln	0
yAb	INT	Y-Abstand des Bildes in Pixeln	0
xGröße	INT	*X-Größe des Bildes in Pixeln*	Tatsächliche *X* Größe
yGröße	INT	*Y*-Größe des Bildes in Pixeln	Tatsächliche *Y* Größe

Mit diesem Aufruf wird das Bild *test.tga* wiedergegeben, wobei Pixel vom linken unteren Bereich des Bildes (Nullabstand) bis zu 500 Pixel Breite und 400 Pixel Höhe angezeigt werden. Beispiel:

```
(c:wiedergabe "TEST" "TGA" 0 0 500 400)
```

Extern definierte Funktion *render* ARX-Anwendung

14.25 MAT

Erstellt und bearbeitet Renderingmaterialien, weist sie zu und entfernt sie

(c:mat *Modus Optionen*)

Diese Funktion hat sechs Modi, die in der folgenden Tabelle beschrieben werden. Jeder Modus wird durch ein Zeichenkettenargument festgelegt.

MAT-Funktionsmodi

Modus	Beschreibung
A	Weist Material zu
C	Kopiert Material
D	Löst Material
L	Listet alle Materialien in der Zeichnung auf oder gibt eine Definition des angegebenen Materials zurück
M	Modifiziert Material
N	Neues Material

Extern definierte Funktion *render* ARX-Anwendung

14.25.1 A - Material zuweisen

Im Modus "A" kann ein Material ausgewählten Objekten oder einem ACI-Wert (AutoCAD-Farbindex) zugewiesen werden, je nachdem, ob das dritte Argument (Layername) eine Ganzzahl oder ein Auswahlsatz ist.

`(c:mat "A"` *Name [Aci | Asatz | Layername]*`)`

In der folgenden Tabelle werden die Argumente für die Zuweisung beschrieben.

Argumente für die Zuweisung

Argument	Datentyp	Beschreibung
Name	STR	Name des zuzuweisenden Materials
Aci	INT	ACI-Nummer zwischen 0 und 255
Asatz	INT	Auswahlsatz, der die zuzuweisenden Elemente enthält
Layername	STR	Layername

Beispiel:

```
(c:mat "A" "PURPURTIGER" 1)
```

Weist das Material PURPURTIGER dem ACI 1 (rot) zu.

Wenn Sie das dritte Argument weglassen, gibt der Modus "A" eine Liste mit drei Elementen zurück:

- Eine Liste der Layernamen, denen das Material zugewiesen ist
- Eine Liste von ACIs, denen das Material zugewiesen ist
- Ein Auswahlsatz, der die Objekte enthält, denen das Material zugewiesen ist

Beispiel:

```
Befehl: (c:mat "a" "tholz")
Objekte werden gesammelt... 1 gefunden
                ACIs mit Layernamen
(("erstes" "zweites")(135) <Auswahlsatz 12>))
```

Ein Material-Indexwert im Bereich 1-255 ist eine ACI-Nummer, ein Index größer als 255 zeigt ein AutoCAD-Rendermaterial an, daß nicht durch ACI zugewiesen ist.

14.25.2 C - Material kopieren

Erstellt ein neues Material durch Kopieren eines in der Zeichnung bereits vorhandenen Materials.

`(c:rmat "C"` *alt_Name neu_Name*`)`

Das Argument `alt_Name` ist eine Zeichenkette, die den Namen des zu kopierenden Materials angibt. Das Argument `neu_Name` ist eine Zeichenkette, die den Namen des neuen Materials angibt. Sie können anschließend das neue (oder das alte) Material modifizieren, um seine Definition zu ändern.

```
(c:mat "C" "ROT" "ROT2")
```

14.25.3 D - Material lösen

Im Modus "D" kann ein Material von ausgewählten Objekten, einem ACI-Wert (AutoCAD-Farbindex) oder Layern gelöst werden, je nachdem, ob das zweite Argument (Auswahlsatz) eine Ganzzahl, ein Auswahlsatz oder eine Zeichenkette ist.

(c:mat "D" ***Name [Aci | Asatz | Layername]*****)**

In der folgenden Tabelle werden die Argumente für Lösen beschrieben.

Argumente für Lösen

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name des zu lösenden Materials	Keiner
Aci	INT	ACI-Nummer zwischen 0 und 255	Keine
Asatz	INT	Auswahlsatz, der die zu lösenden Elemente enthält	Keiner
Layername	STR	Layername	Keiner

Zum Beispiel fordert

```
(c:mat "D" (ssget))
```

den Benutzer auf, Objekte auszuwählen, und trennt dann die einzelnen Objekte von ihrem Material.

14.25.4 L - Listet Material auf

Listet Definitionen von Material aus der Zeichnung auf.

(c:mat "L" ***[Name]*****)**

Das fakultative Argument *Name* ist eine Zeichenkette, welche die aufzulistenden Materialdefinitionen angibt. Wenn das Argument *Name* weggelassen wird, listet **c:mat** alle Materialien in der Zeichnung auf. Beispiel:

Befehl: **(c:mat "L")**
("*GLOBAL*" "BLAUES GLAS" "WEISSES PLASTIK" "THOLZ" "BEIGE MATTE")

Die erste Zeichenkette in der Liste bezeichnet das globale Vorgabematerial, *GLOBAL*. Sie können diese Zeichenkette an `(c:mat)` genauso übergeben, wie Sie die Namen von durch Bibliotheken oder Benutzer definierten Materialien übergeben können:

Befehl: **(c:mat "L" "*GLOBAL*")**
("*GLOBAL*" (-1.0 -1.0 -1.0) 1.0 ("" 0.0 0 (1.0 1.0) (0.0 0.0) 0.0) (0.0 0.0 0.0)
1.0 (1.0 1.0 1.0) 1.0 ("" 0.0 0) 0.5 0.0 ("" 0.0 0 (1.0 1.0) (0.0 0.0) 0.0)
1.0 ("" 0.0 0 (1.0 1.0) (0.0 0.0) 0.0))

Die Listenelemente in einer Materialdefinition entsprechen den Argumenten in den Modi Modifizieren und Neu.

14.25.5 M - Modifiziert Material

Die Optionen für den Modus "M" (Modifizieren) entsprechen den Optionen im Modus "N" (Neu). Wenn ein Argument `nil` ist - oder am Ende der Argumentliste nicht eingegeben wurde - behält es seinen aktuellen Wert.

Zum Beispiel ändert der folgende Aufruf das Material BLUE MARBLE so, daß es eine mittelblaue Steinfarbe (Matrix) mit schwarzen Adern aufweist:

```
(c:mat "M" "BLAUER MARMOR" "marmor" '(0.5 0.5 1.0) '(0.0 0.0 0.0))
```

14.25.6 N - Neues Material

Im Modus "N" (Neu) wird ein neues Material erstellt. Die Argumente dieser Funktion hängen nicht nur vom Modus, sondern auch vom zu erstellenden Materialtyp ab. Prozedurale Materialien: Marmor, Granit und Holz verfügen jeweils über einen einzigartigen Satz von Argumenten, der sich von den Argumenten der vorgegebenen Materialien unterscheidet.

In der folgenden Tabelle werden die neuen Argumente beschrieben.

Neue Argumente

Argument	**Datentyp**	**Beschreibung**	**Vorgabe**
Name	STR	Name des zu erstellenden Materials	Keiner
Materialtyp	STR	Neuer Materialtyp. Optionen: STANDARD - Materialtyp "Vorgabe" MARBLE - Materialtyp "Marmor" GRANITE - Materialtyp "Granit" WOOD - Materialtyp "Holz"	Keiner
Beschreibung	(unterschiedlich)	Die Argumente hängen vom zu erstellenden Materialtyp ab	(unterschiedlich)
Asatz	INT	Auswahlsatz, der die zu lösenden Elemente enthält	Keiner
Layername	STR	Layername	Keiner

Außerdem sind die Argumente aller Arten von Bitmaps in einer Subliste angegeben, siehe "Bitmap-Argumente."

14.25.7 N - Neues Material

14.25.7.1 STANDARD

Die Zeichenkette für den Materialtyp "STANDARD" beinhaltet, daß Sie ein neues Vorgabematerial erstellen.

`(c:mat "N"` ***Name*** `"STANDARD"` ***[Farb-Gew [Muster [Umgebung [Umg-Gew [Refl [Refl-Gew [Refl-Map [Rauheit [Transparenz[Opacity-Map [Brechung [Bump-Map]]]]]]]]]]]]***`)`

In der folgenden Tabelle werden die Vorgabeargumente beschrieben.

Vorgabeargumente

Argument	Datentyp	Beschreibung	Vorgabe
Farbe	LIST (oder reale Zahlen)	Die Materialfarbe wird als RGB-Tripel angegeben; (-1.0 -1.0 -1.0) bedeutet, daß die Farbe vom ACI eines Objekts abgeleitet wird (diffuse Farbe)	(-1.0 -1.0 -1.0) - Nach ACI
Farb-Gew	REAL	Gewichtfaktor (Farbwert) - das Maß an diffuser Farbe	0.7
Muster	LIST	Argumente für Muster bzw. Texture-Map	Keine
Umgebung	LIST (oder reale Zahlen)	Umgebungs-(Schatten-) Farbe als RGB-Tripel	(-1.0 -1.0 -1.0) - Nach ACI
Refl-Gew	REAL	Gewichtfaktor (Farbwert) - das Maß an Spiegelfarbe	0.1
Refl	LIST (oder reale Zahlen)	Reflexions-(Spiegel-) Farbe als RGB-Tripel	(-1.0 -1.0 -1.0) - Nach ACI
Refl-Gew	REAL	Gewichtfaktor (Reflexionswert) - das Maß an Spiegelfarbe	0.2
Refl-Map	LIST	Argumente für Reflection-/Environment-Map	Keine
Rauheit	REAL	Rauheit - die Größe eines Spiegelglanzpunkts	0.5
Transparenz	REAL	Transparenz des Materials	0.0
Opacity-Map	LIST	Argumente für Opacity-Map	Keine
Brechung	REAL	Index der Brechung	1.0
Bump-Map	LIST	Argumente für Bump-Map	Keine

Dieser Aufruf erstellt beispielsweise ein leuchtend rotes Material mit einer Muster-Map:

```
(c:mat "N" "RED LACQUER" "STANDARD" ; Name und Typ
'(1.0 0.0 0.0) (1.0) ; Farbe (rot), Gewicht und Texture-Map
'("INLAY.TGA 0.75 0 (0.5 0.5) (0.3 0.3) 0.0)
'(1.0 0.0 0.0) 1.0   ; Umgebungsfarbe und ihr Gewicht (genau wie diffus)
'(1.0 0.0 0.0) 1.0   ; Reflexionsfarbe (weiß) und ihr Gewicht
nil                  ; Keine Reflection-Map
0.2                  ; Rauheit (niedrig)
0.0                  ; Transparenz (keine)
nil                  ; Keine Opacity-Map
0.0                  ; Brechung (keine)
nil                  ; Keine Bump-Map
```

Mit dem nächsten Aufruf wird das Material MAPS erstellt, das mehrere Bitmaps enthält:

```
(c:mat "N" "MAPS" "STANDARD"
'(1.0 0.0 0.0) (1.0) '("weave.tga" 1.0 0)
'(1.0 0.0 0.0) 1.0
'(1.0 0.0 0.0) 1.0 '("room.tga" 0.75)
0.5
0.0
'("hole.tga")
1.0
'("ridges.tga")
```

Mit dem folgenden Aufruf wird ein Material ohne Bitmaps und Vorgabewerte erstellt, dessen Reflexionen beim Raytracing mit Photo Raytrace oder mit einer Environment-Map mit Photo Real erzeugt werden,

```
(c:mat "N" "SHINE" "STANDARD" nil nil nil nil nil nil nil ' (nil nil 1))
```

14.25.8 N - Neues Material

14.25.8.1 Marmor

Die Zeichenkette für den Materialtyp "MARBLE" beinhaltet, daß Sie ein neues Marmormaterial erstellen.

**`(c:mat "N"` *Name* `"MARBLE"` *[Steinfarbe [Aderfarbe{XXxc7}*
[Refl [Refl-Gew [Refl-Map [Rauheit [Turbulenzfaktor
*[Schärfefaktor [Skalierung [Bump-Map]]]]]]]]]])**

In der folgenden Tabelle werden die Marmor-Argumente beschrieben.

MAT - Marmor-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Steinfarbe	LIST (von realen Zahlen)	Der RGB-Wert gibt die Steinfarbe der Marmormatrix an	(-1.0 -1.0 -1.0) - weiß
Aderfarbe	LIST (von realen Zahlen)	Der RGB-Wert gibt die Aderfarbe des Marmors an	(-1.0 -1.0 -1.0) - schwarz
Refl	LIST (von realen Zahlen)	Reflexions-(Spiegel-) Farbe als RGB-Wert	(-1.0 -1.0 -1.0) - Nach ACI
Refl-Gew	REAL	Gewichtfaktor (Reflexionswert) - das Maß an Spiegelfarbe	0.2
Refl-Map	LIST	Argumente für Reflection-/Environment-Map	Keine
Rauheit	REAL	Rauheit - die Größe eines Spiegelglanzpunkts	0.5
Turbulenzfaktor	INT	Turbulenzfaktor - Ausgeprägtheit der Adern	3
Schärfefaktor	REAL	Schärfefaktor - das Maß der Verschwommenheit	1.0
Skalierung	REAL	Gesamtskalierfaktor	0.16
Bump-Map	LIST	Argumente der Bump-Map	Keine

Mit diesem Aufruf wird beispielsweise eine rosa Matrix mit schwarzen Adern erstellt:

```
(c:mat "N" "PINK MARBLE" "MARBLE" '(1.0 0.34 0.79))
```

14.25.8.2 Granit

Die Zeichenkette für den Materialtyp "GRANITE" beinhaltet, daß Sie ein neues Granitmaterial erstellen.

**`(c:mat "N"` *Name* `"GRANITE"` *[erste-Farbe [Gewichtfaktor1*
[zweite-Farbe [Gewichtfaktor2 [dritte-Farbe [Gewichtfaktor3
[vierte-Farbe [Gewichtfaktor4 [Refl [Refl-Gew
[Refl-Map [Rauheit [Schärfefaktor [Skalierung
*[Bump-Map]]]]]]]]]]]]]])**

In der folgenden Tabelle werden die Granit-Argumente beschrieben:

MAT - Granit-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
erste-Farbe	LIST (von realen Zahlen)	RGB-Wert	(-1.0 -1.0 -1.0) - weiß

Refl-Gew1	REAL	Gewichtfaktor (Farbwert) für erste Farbe	1.0
zweite-Farbe	LIST (von realen Zahlen)	RGB-Wert	(0.5 0.5 0.5) - dunkelgrau
Refl-Gew2	REAL	Gewichtfaktor (Farbwert) für zweite Farbe	1.0
dritte-Farbe	LIST (von realen Zahlen)	RGB-Wert	(0.0 0.0 0.0) - schwarz
Refl-Gew3	REAL	Gewichtfaktor (Farbwert) für dritte Farbe	1.0
vierte-Farbe	LIST (von realen Zahlen)	RGB-Wert	(0.7 0.7 0.7) - hellgrau
Refl-Gew4	REAL	Gewichtfaktor (Farbwert) für vierte Farbe	1.0
Refl	LIST (von realen Zahlen)	Reflexions-(Spiegel-) Farbe als RGB-Wert	(-1.0 -1.0 -1.0) - Nach ACI
Refl-Gew	REAL	Gewichtfaktor (Reflexionswert) - das Maß an Spiegelfarbe	0.2
Refl-Map	LIST	Argumente für Reflection-/Environment-Map	Keine
Rauheit	REAL	Rauheit - die Größe eines Spiegelglanzpunkts	0.5
Schärfefaktor	REAL	Schärfefaktor - das Maß der Verschwommenheit	1.0
Skalierung	REAL	Gesamtskalierfaktor	0.16
Bump-Map	LIST	Argumente der Bump-Map	Keine

Mit diesem Aufruf wird beispielsweise Granit ohne Dunkelgrau mit mehr Schwarz und mit Gelb anstatt Hellgrau erstellt:

```
(c:rmap "N" "YELLOW GRANITE"
 nil 0.5 nil 0.0 nil 0.85 '(1.0 1.0 0.0) 0.6)
```

14.25.8.3 Holz

Die Zeichenkette für den Materialtyp "WOOD" beinhaltet, daß Sie ein neues Holzmaterial erstellen.

`(c:mat "N"` ***Name*** **`"WOOD"`** ***[Steinfarbe [helle Farbe***
[dunkle Farbe [Refl [Refl-Gew [Refl-Map [Rauheit
[Verhältnis [Speicherdichte [Breite [Symbol [Bump-Map]]]]]]]]]])

In der folgenden Tabelle werden die Holz-Argumente beschrieben:

MAT - Holz-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
helle Farbe	LIST (von realen Zahlen)	Der RGB-Wert gibt die Farbe der hellen Ringe an	(0.6 0.4 0.3)
dunkle Farbe	LIST (von realen Zahlen)	Der RGB-Wert gibt die Farbe der dunklen Ringe an	(0.3 0.2 0.2) - schwarz
Refl	LIST (von realen Zahlen)	Reflexions-(Spiegel-) Farbe als RGB-Wert	(-1.0 -1.0 -1.0) - Nach ACI
Refl-Gew	REAL	Gewichtfaktor (Reflexionswert) - das Maß an Spiegelfarbe	0.2
Refl-Map	LIST	Argumente für Reflection-/ Environment-Map	Keine
Rauheit	REAL	Rauheit - die Größe eines Spiegelglanzpunkts	0.5
Verhältnis	REAL	Verhältnis der hellen zu den dunklen Ringen	0.5
Speicher-dichte	REAL	Dichte der Ringe	6.0
Breite	REAL	Variation der Ringbreite	0.2
Symbol	REAL	Variation der Ringform	0.2
Skalierung	REAL	Gesamtskalierfaktor	0.16
Bump-Map	LIST	Argumente der Bump-Map	Keine

Mit diesem Aufruf wird beispielsweise ein Holz mit einer unregelmäßigen Maserung erstellt:

```
(c:mat "N" "CRYPTO" "WOOD" nil nil nil nil nil nil nil nil nil 0.56)
```

14.25.8.4 Bitmap-Argumente

Die Argumente, mit denen eine Bitmap angegeben wird, werden an eine Liste weitergeleitet, die als Subliste im Aufruf (c: mat) eingeschlossen werden kann (dies ist die Form, die jeweils zu Beginn der folgenden Abschnitte aufgeführt ist) oder einem Symbol vor einem Aufruf zugewiesen werden kann (c:mat).

Muster/Textur

'(*Name* [*Überblendung* [*Wiederholen* [*Skalierung* [*Abstand*]]]])

In der folgenden Tabelle werden die Muster-/Textur-Argumente beschrieben:

Muster-/Textur-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name der Bitmap-Datei.	Keiner
Überblendung	REAL	Maß der zu verwendenden Map-Farbe	1.0
Wiederholen	INT	Bitmap nebeneinander anordnen: 0 - keine Anordnung nebeneinander (Zuschneiden) 1 - Anordnung nebeneinander (Muster wiederholen)	0
Skalierung	LIST (von realen Zahlen)	U- und V-Skalierfaktoren	(1.0 1.0)
Abstand	LIST (von realen Zahlen)	U- und V-Abstände	(0.0 0.0)

Reflexion/Umgebung

`'(`***Name [Überblendung*** `[`***Raytrace***`]])`

In der folgenden Tabelle werden die Reflexions-/Umgebungs-Argumente beschrieben:

Reflexions-/Umgebungs-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name der Bitmap-Datei.	Keine
Überblendung	REAL	Maß der zu verwendenden Map-Farbe	1.0
Spiegeln	REAL	Spiegelreflexionen erzeugen: 0 - keine Spiegelung 1 - Spiegelung mit Spiegeln werden Raytrace-Reflexionen erzeugt; mit Scanline werden Environment-Maps für Reflexionen verwendet	0

Opacity

`'(`***Name*** `[`***Überblendung*** `[`***Wiederholen*** `[`***Skalierung*** `[`***Abstand***`]]]])`

In der folgenden Tabelle werden die Opacity-Argumente beschrieben.

Opacity-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name der Bitmap-Datei.	Keiner
Überblendung	REAL	Maß der zu verwendenden Map-Farbe	1.0
Wiederholen	INT	Bitmap nebeneinander anordnen: 0 - keine Anordnung nebeneinander (Zuschneiden) 1 - Anordnung nebeneinander (Muster wiederholen)	0
Skalierung	LIST (von realen Zahlen)	U- und V-Skalierfaktoren	(1.0 1.0)
Abstand	LIST (von realen Zahlen)	U- und V-Abstände	(0.0 0.0)

Bump-Map

'(*Name* [*Amplitude* [*Wiederholen* [*Skalierung* [*Abstand*]]]])

In der folgenden Tabelle werden die Relief-Argumente beschrieben.

Relief-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name der Bitmap-Datei.	Keiner
Amplitude	REAL	Grad des Reliefs	1.0
Wiederholen	INT	Bitmap nebeneinander anordnen: 0 - keine Anordnung nebeneinander (Zuschneiden) 1 - Anordnung nebeneinander (Muster wiederholen)	0
Skalierung	LIST (von realen Zahlen)	U- und V-Skalierfaktoren	(1.0 1.0)
Abstand	LIST (von realen Zahlen)	U- und V-Abstände	(0.0 0.0)

14.26 3DDREHEN

Dreht ein Objekt um eine beliebige dreidimensionale (3D-)Achse

(3ddrehen *arg1 arg2 ...*)

Reihenfolge, Anzahl und Typ der Argumente für die Funktion **3ddrehen** entsprechen genau der Eingabe in der Eingabeaufforderung. Um eine leere Antwort zu kennzeichnen (RETURN), verwenden Sie `nil` oder eine leere Zeichenkette (`""`). Bei erfolgreicher Durchführung gibt **3ddrehen** den Wert `T` zurück, andernfalls wird `nil` zurückgegeben.

Das folgende Beispiel dreht die ausgewählten Objekte um 30 Grad um die durch die Punkte *p1* und *p2* angegebene Achse.

```
(setq As (ssget))
(3ddrehen As p1 p2 30)
```

Anmerkung AutoLISP-Unterstützung für die Funktion **3ddrehen** wird durch die Benutzung der Bibliothek SAGET implementiert.

Extern definierte Funktion *geom3d* ARX-Anwendung

14.27 REINST

Legt Rendereinstellungen fest

(c:reinst ***Modus Option [Einstellung]*****)**

REINST legt fest, welche Renderingparameter verwendet werden und welches Renderingverhalten die Vorgabe ist. Diese Funktion enthält fünf Modi, die in der folgenden Tabelle beschrieben werden. Jeder Modus wird durch ein Zeichenkettenargument festgelegt. Die übrigen Argumente sind vom Modus abhängig.

REINST-Funktionsmodi

Modus	Beschreibung
DEST	Ziel des Ansichtsfensters, Renderfensters oder der Datei
ICON	Skalierung der Symbolblöcke Licht und Material
ROPT	Weitere Renderingoptionen
SELECT	Eingabeaufforderung für Objektauswahl
STYPE	Renderingtyp von Render, Photo Real oder Photo Raytrace
TOGGLE	Renderingoptionen

Extern definierte Funktion *render* ARX-Anwendung

See Also

"Einstellen der Renderfunktion auf Dateioptionen."

14.27.1 DEST - Ziel-Voreinstellung

Wählt das zu verwendende Ausgabegerät.

(c:reinst "DEST" ***Option*****)**

Das Argument `Option` ist eine Zeichenkette, welche das Renderingziel angibt. Die möglichen Werte für `Option` werden in der folgenden Tabelle beschrieben:

Optionen von REINST - "DEST"

Wert	Beschreibung
FRAMEBUFFER	Rendering auf Bildschirm
HARDCOPY	Rendering in Renderfenster
FILE	Rendering in Datei

Dieser Aufruf gibt beispielsweise an, daß in eine Datei gerendert werden soll:

```
(c:reinst "DEST" "FILE")
```

14.27.2 ICON - Symbol-Voreinstellung

Legt die Größe des Symbolblocks für Licht oder Material in einer Zeichnung fest.

(c:reinst "ICON" ***Option*****)**

Das Argument `Option` ist eine reale Zahl, welche die Größe des Symbolblocks festlegt (Vorgabewert ist 1.00). Folgender Aufruf ändert zum Beispiel die Symbolskalierung auf 50 Prozent:

```
(c:reinst "ICON" 0.5)
```

14.27.3 STYPE - Renderingtyp

Gibt an, welcher Renderer verwendet wird.

(c:reinst "STYPE" ***Option*****)**

Das Argument *Option* ist eine Zeichenkette, welche den Renderingtyp angibt. Die möglichen Werte für *Option* werden in der folgenden Tabelle beschrieben:

Optionen von REINST - "STYPE"

Wert	Beschreibung
ARENDER	Grundlegendes Rendering
ASCAN	Photo Real-Rendering
ARAY	Photo Raytrace-Rendering

Der folgende Code legt zum Beispiel fest, daß das nächste Rendering vom grundlegenden AutoCAD-Renderer erzeugt wird.

```
(c:reinst "STYPE" "CRENDER")
```

14.27.4 SELECT - Auswahl-Voreinstellungen

Legt fest, ob Sie vor dem Rendering zur Objektauswahl aufgefordert werden.

(c:reinst "SELECT" ***Option*****)**

Das Argument *Option* ist eine Zeichenkette, welche die Eingabeaufforderung angibt. Die möglichen Werte für *Option* werden in der folgenden Tabelle beschrieben:

Optionen von REINST - "SELECT"

Wert	Beschreibung
ALL	Gesamte Szene rendern
ASK	Aufforderung zur Objektauswahl

Der folgende Aufruf zum Beispiel legt fest, daß Sie beim Rendern zur Objektauswahl aufgefordert werden:

```
(c:reinst "SELECT" "ASK")
```

14.27.5 TOGGLE - Umschalten von Voreinstellungen

Steuert verschiedene Renderingoptionen.

(c:reinst "TOGGLE" ***Option Einstellung*****)**

Das Argument *Option* ist eine Zeichenkette, welche die Eingabeaufforderung angibt. Die möglichen Werte für *Option* werden in der folgenden Tabelle beschrieben. Das Argument *Einstellung* ist eine Zeichenkette, die den Status des Schalters angibt. Mögliche Werte für *Einstellung* sind "ON" und "OFF".

Optionen von REINST - "TOGGLE"

Wert	Beschreibung
CACHE	Rendering in eine Zwischenspeicherdatei
SHADOW	Rendern mit Schattierung
SMOOTH	Rendern mit Glättung
MERGE	Objekte mit Hintergrund verschmelzen
FINISH	Material auftragen
SKIPRDLG	Dialogfeld Render nicht anzeigen

Anmerkung Die Option CACHE gibt an, daß die Renderinformationen in eine Zwischenspeicherdatei auf der Festplatte geschrieben werden. Solange die Zeichnungsgeometrie oder die Sicht nicht verändert wird, wird die Datei im

Zwischenspeicher für weitere Renderings verwendet, damit keine erneute Tessellierung erforderlich ist. Dadurch kann das Rendering verkürzt werden (insbesondere bei Volumenkörpern).

Die folgenden Aufrufe zum Beispiel schalten Rendering mit Verschmelzen aus und Schattierung ein:

```
(c:reinst "TOGGLE" "MERGE" "OFF")
(c:reinst "TOGGLE" "SMOOTH" "ON")
```

14.28 BILDSICH

Speichert ein gerendertes Bild in eine Datei im Format BMP, TGA oder TIFF

(c:bildsich ***Dateiname Typ [Ausschnitt] [xAb yAb xGröße yGröße] [Komprimierung]*****)**

Wenn AutoCAD für das Rendern auf einem getrennten Bildschirm konfiguriert ist, sollte das Argument Ausschnitt nicht benutzt werden. Sie können eine Größe und einen Abstand für das Bild angeben, und bei TGA- und TIFF-Dateien können Sie eine Komprimierungsart angeben. Die Argumente werden in der folgenden Tabelle beschrieben.

BILDSICH-Funktionsargumente

Argument	Datentyp	Beschreibung	Vorgabe
Dateiname	STR	Bilddateiname	Keiner
Typ	STR	Dateityp: BMP, TGA oder TIFF	Keiner
Ausschnitt	STR	Teil des Bildschirms, der gespeichert werden soll: A - aktives Ansichtsfenster D - Zeichenbereich F - Gesamtbildschirm	"A"
xAb	INT	*X* Abstand in Pixeln	0
yAb	INT	*Y* Abstand in Pixeln	0
xGröße	INT	*X* Größe in Pixeln	Tatsächliche *X*-Größe
yGröße	INT	*Y* Größe in Pixeln	Tatsächliche *Y*-Größe
Komprimierung	STR	Komprimierungsart KEINE PACK (nur TIFF-Dateien) RLE (nur TGA-Dateien)	Keine

Anmerkung Das Argument Ausschnitt wird jetzt ignoriert, jedoch für die Kompatibilität des Scripts zur Verfügung gestellt.

In diesem Beispiel wird ein Ganzbildschirm-TIFF-Bild mit dem Namen *test.tif* ohne Komprimierung gespeichert:

```
(c:bildsich "TEST" "TIF" "KEINE")
```

Extern definierte Funktion *render* ARX-Anwendung

14.29 SZENE

Erstellt neue Szenen und modifiziert oder löscht vorhandene Szenen nur im Papierbereich

(c:szene ***Modus [Optionen]*****)**

Diese Funktion enthält sechs Modi, die in der folgenden Tabelle beschrieben werden. Jeder Modus wird durch ein Zeichenkettenargument festgelegt. Die übrigen Argumente sind vom Modus abhängig.

SZENE - Funktionsmodi

Modus	Beschreibung
D	Löscht eine vorhandene Szene

L	Listet alle Szenen in der Zeichnung auf oder gibt eine Definition der angegebenen Szene zurück.
M	Modifiziert eine vorhandene Szene
N	Erstellt eine neue Szene
R	Benennt eine vorhandene Szene um
S	Stellt die aktuelle Szene ein

Extern definierte Funktion *render* ARX-Anwendung

14.29.1 D - Löscht Szene

Löscht eine vorhandene Szene.

(c:szene "D" ***Name*****)**

Das Argument *Name* ist eine Zeichenkette, die den Namen der zu löschenden Szene angibt. Falls die aktuelle Szene gelöscht wird, wird *KEINE* zur aktuellen Szene.

```
(c:szene "D" "PLANSICHT")
```

14.29.2 L - Listet Szenen auf

Listet alle Szenen in der Zeichnung auf oder gibt eine Definition der angegebenen Szene zurück.

(c:szene "L" ***[Name]*****)**

Das Argument *Name* ist eine Zeichenkette, die den Namen der zu aufzulistenden Szene angibt. Wenn das Argument *Name* weggelassen wird, gibt **c:szene** eine Liste aller in der Zeichnung definierten Szenen. Wenn Sie *Name* angeben, gibt **c:szene** die Definition der genannten Szene.

Die folgende Anweisung gibt eine Liste der in der Zeichnung definierten Szenennamen zurück.

```
(c:szene "L")                    ergibt:
("" "SZENE1" "SZENE2" "SZENE3")
```

Die leere Zeichenkette (`""`) ist die voreingestellte Szene *KEINE*, die nicht modifiziert werden kann. Der Aufruf

```
(c:szene "L" "SZENE2")
```

gibt eine Definition der genannten Szene zurück. Hierfür einige Beispiele:

```
(T T)                            Die *aktuelle* Sicht mit *allen* Lichtquellen
("SICHT2 nil)                    Eine benannte Sicht ohne Lichtquellen
("SICHT2" ("LICHT1" "LICHT2"))   Eine benannte Sicht mit zwei Lichtquellen
```

14.29.3 M - Modifiziert Szenen

Modifiziert eine vorhandene Szene.

(c:szene "M" ***Name [Sicht [Lichter]]*****)**

Die Optionen für den Modus Modifizieren entsprechen den Optionen im Modus Neu, außer daß Sie *Sicht* als `nil` angeben können, um nur die Lichter zu modifizieren.

Anmerkung Sie müssen das Argument *Lichter* als Liste angeben, auch wenn Sie nur ein Licht festlegen.

Zum Beispiel modifiziert der folgende Aufruf eine Szene mit dem Namen SZENE1 so, daß die benannte Sicht VORNE und alle Lichter in der Zeichnung verwendet werden:

```
(c:szene "M" "SZENE1" "VORNE" (C:LICHT "L"))
```

Der folgende Aufruf modifiziert SZENE1 so, daß die benannte Sicht HINTEN und nur die Lichter P1 und P2 verwendet werden:

```
(c:szene "M" "SZENE1" "HINTEN" '("P1" "P2"))
```

14.29.4 N - Neue Szene

Erstellt eine neue Szene.

(c:szene "N" ***Name [Sicht [Lichter]]*****)**

Die Argumente des Modus NEU werden in der folgenden Tabelle beschrieben.

SZENE - "N" Modus-Argumente

Argument	Datentyp	Beschreibung	Vorgabe
Name	STR	Name der neuen Szene	Keiner
Sicht	STR	Name einer benannten AutoCAD-Sicht	Keiner
	T (SYM)	Sicht *AKTUELL*	
Lichter	LIST (von Zeichenketten)	Liste von Lichtnamen	*ALLE* Lichter in der Zeichnung
	T (SYM)	*ALLE* Lichter in der Zeichnung verwenden	
	nil	Keine Lichter in der Zeichnung benutzen	Ein entferntes Licht "über die Schulter"

Anmerkung Sie müssen das Argument `Lichter` als Liste angeben, auch wenn Sie nur ein Licht festlegen.

So erzeugen Sie eine neue Szene mit dem Namen VORGABE unter Verwendung der Sicht *AKTUELL* und der Lichter *ALLE* :

```
(c:szene "N" "VORGABE")
```

So erzeugen Sie eine neue Szene mit dem Namen DÜSTER unter Verwendung der Sicht *AKTUELL* und des Vorgabelichts "über die Schulter":

```
(c:szene "N" "DÜSTER" T nil)
```

So erzeugen Sie eine neue Szene mit dem Namen SPEZIAL unter Verwendung der benannten Sicht MEINE_SICHT und der Lichter SUN, LAMP und SPOT:

```
(c:szene "N" "SPEZIAL" "MEINE_SICHT" '("SONNE" "LAMPE" "SPOT"))
```

14.29.5 R - Umbenennen von Szenen

Benennt eine Szene um.

(c:szene "R" ***alt_Name neu_Name*****)**

Das Argument `alt_Name` ist eine Zeichenkette, die den Namen der Originalszene angibt, `neu_Name` legt den neuen Namen für die Szene fest.

Beispiel:

```
(c:szene "R" "SPEZIAL" "HELL")
```

14.29.6 S - Szene einstellen

Stellt die aktuelle Szene ein.

(c:szene "S" ***[Name]*****)**

Das Argument `Name` ist eine Zeichenkette, die den Namen der Szene angibt, die zur aktuellen Szene gemacht werden soll. Wenn das Argument `Name` weggelassen wird, gibt **`c:szene`** den Namen der aktuell gewählten Szene. Beispiel:

```
(c:szene "S")
"PLAN"
```

Wenn keine aktuelle Szene vorhanden ist, gibt `c:szene` eine leere Zeichenkette (`""`) zurück.

So machen Sie zum Beispiel SZENE3 zur aktuellen Szene:

```
(c:szene "S" "SZENE3")
```

14.30 MAPPING

Weist ausgewählten Objekten Materialzuordnungs-Koordinaten zu. Die Funktion hat zwei Modi, die durch ein Zeichenkettenargument festgelegt werden.

Mit dem Befehl MAPPING können Sie ausgewählten Objekten Materialzuordnungs-Koordinaten zuweisen. Die Funktion hat zwei Modi, die durch ein Zeichenkettenargument festgelegt werden.

MAPPING-Modi

Modus	Beschreibung
A	Weist dem Auswahlsatz die UV-Darstellung zu.
D	Löst die UV-Darstellung vom Auswahlsatz.

Extern definierte Funktion render ARX-Anwendung

14.30.1 A - Zuweisen

Der Modus "A" (Zuweisen) weist Zuordnungskoordinaten zu. Seine Argumente hängen davon ab, ob Sie Projektions- oder Volumenkörper-Darstellung angeben. Die Argumente für den Modus "Zuweisen" bei der Projektions-Darstellung werden in der folgenden Tabelle beschrieben:

Argumente des Befehls MAPPING - "A" für Projektions-Darstellung

Argument	Datentyp	Beschreibung	Vorgabe
Asname	PICKSET	Der Auswahlsatz, der die Elemente enthält, denen Sie die Zuordnungskoordinaten zuweisen wollen	Keiner
Mapping Typ	STR	Art der Projektions-Darstellung: P - planar D - zylindrisch F - sphärisch	Keine
pt1, pt2, pt3	LIST	Drei Punkte, die die Mapping-Geometrie definieren: Planar - Ecke unten links; Ecke unten rechts; Ecke oben links Zylindrisch - Mitte unten; Mitte oben; in Richtung Naht Sphärisch - Mitte der Kugel; Radius (Norden); in Richtung Naht	Keine
Wied	INT	Kacheln: 0 - keine gekachelte Anordnung (Zuschneiden) 1 - gekachelte Anordnung (Muster wiederholen)	1
Skalierung	LIST (von realen Zahlen)	Die U- und V-Skalierfaktoren	(1.0 1.0)
Abstand	LIST (von realen Zahlen)	Die U- und V-Abstände	(0.0 0.0)

Bei Volumenkörper-Darstellung werden durch die Options-Argumente nur die Zuordnungspunkte angegeben. Diese definieren implizit die Skalierung der UVW-Bemaßung. Die Argumente für den Modus "Zuweisen" bei Volumenkörper-Darstellung werden in der folgenden Tabelle beschrieben:

Argumente des Befehls MAPPING - "A" für Volumenkörper-Darstellung

Argument	Datentyp	Beschreibung	Vorgabe
Asname	PICKSET	Der Auswahlsatz, der die Objekte enthält, denen Sie die Zuordnungskoordinaten zuweisen wollen	Keiner
Mapping Typ	STR	R - Volumenkörper	Keiner
pt1	LIST	Punkt zur Definition des Ursprungs	Keiner
pt1	LIST	Punkt zur Definition der U-Achse	Keiner
pt1	LIST	Punkt zur Definition der V-Achse	Keiner
pt1	LIST	Punkt zur Definition der W-Achse	Keiner

Zum Beispiel werden durch den folgenden Aufruf zylindrische Zuordnungskoordinaten einem Objekt zugeordnet, das der Benutzer auswählt, wobei eine gekachelte Anordnung und die Voreinstellungen für Skalierung und Abstand gewählt wurden:

```
(c:mapping "A" (ssget) "C" '(5.0 5.0 5.0) '(5.0 5.0 10.0)
  '(10.0 0.0 0.0) 1)
```

14.30.2 D - Lösen

Der Modus "D" (Lösen) löst die UV-Darstellung, die den Objekten im Auswahlsatz zugeordnet waren. Diese Objekte werden nun mit den Vorgabewerten für die Zuordnungskoordinaten dargestellt, bis Sie erneut Zuordnungskoordinaten zuweisen. Die Argumente der Funktion Lösen werden in der folgenden Tabelle beschrieben.

Argumente des Befehls MAPPING - "D"

Argument	Datentyp	Beschreibung	Vorgabe
Asname	PICKSET	Der Auswahlsatz, der die Objekte enthält, von denen Sie die Zuordnungskoordinaten lösen wollen	Keine

Dieser Aufruf fordert zum Beispiel den Benutzer auf, Elemente einzugeben, die von ihren Zuordnungskoordinaten gelöst werden sollen:

```
(c:mapping "D" (ssget))
```

14.31 ZEIGMAT

Listet den Materialtyp und das Zuordnungsverfahren für ein gewähltes Objekt auf

(c:zeigmat *arg1*)

Diese Funktion listet den Materialtyp und das Zuordnungsverfahren auf der Basis von `arg1` auf. Das Argument `arg1` kann ein Elementname, eine Ganzzahl, die einen ACI-Wert darstellt, oder der Name eines Layers (eine Zeichenkette) sein.

Extern definierte Funktion *render* ARX-Anwendung

14.32 SOLPROFIL

Erstellt Profildarstellungen von dreidimensionalen Volumenkörpern

(c:solprofil *arg1 arg2 ...*)

Reihenfolge, Anzahl und Typ der Argumente entsprechen genau denen, die Sie bei der Eingabeaufforderung eingeben würden.

Extern definierte Funktion *solids* ARX-Anwendung

14.33 STAT

Zeigt statistische Daten zum letzten Rendering an

(c:stat ***[Dateiname |nil]*****)**

Der Befehl STAT gibt Informationen über das letzte Rendering.

Wenn Sie einen Dateinamen angeben, speichert (c:stat) die Renderinformation in der Datei und zeigt das Dialogfeld Statistik nicht an. Wenn Sie den Dateinamen weglassen, zeigt (c:stat) das Dialogfeld Statistik an.

Der folgende Befehl schreibt zum Beispiel eine Statistik von Ihrem letzten Rendering in die Datei *figures.txt*:

```
(c:stat "figures.txt")
```

Falls die Datei bereits existiert, wird die Statistik angehängt.

Der folgende Befehl:

```
(c:stat "stats.txt")
```

speichert die mit dem letzten Rendering zusammenhängenden Informationen in der Datei stats.txt und speichert außerdem so lange die mit den nächsten Renderingvorgängen zusammenhängenden Informationen in dieser Datei, bis Sie folgendes eingeben:

```
(c:stat nil)
```

Der obengenannte Befehl bewirkt, daß beim Rendering die Statistik nicht mehr gespeichert wird.

Extern definierte Funktion *render* ARX-Anwendung

15 Speicherverwaltung

AutoCAD ist so konzipiert, daß die Speichernutzung den Bedürfnissen des Programms angepaßt wird. Bei Systemen mit begrenztem Speicherplatz kann allerdings eine Feinabstimmung erforderlich werden. Dieses Kapitel enthält Informationen darüber, wie solche Systeme verwaltet werden können.

Alle Symbole, benutzerdefinierte Funktionen und die in diesem Handbuch beschriebenen Standardfunktionen werden während einer AutoCAD-Arbeitssitzung im Speicher gehalten. Wenn AutoLISP gestartet wird, belegt es zwei große Bereiche des Speichers. Der erste Bereich, der als *Heap* bezeichnet wird, ist der Bereich, in dem Funktionen und Symbole (auch als *Knoten* (Nodes) bezeichnet) gespeichert werden; je mehr Symbole und Funktionen Sie haben (und je komplexer Ihre Funktionen sind), desto mehr Platz wird im Heapraum benötigt. Im zweiten Bereich, der als *Stack* bezeichnet wird, werden Funktionsargumente und Teilergebnisse abgelegt. Je tiefer Sie Funktionen verschachteln oder je mehr Rekursionen Ihre Funktionen ausführen, desto mehr Platz wird im Stackraum benötigt.

15.1 Knotenraum

Ein *Knoten* ist eine Speicherstruktur, die alle Datentypen von AutoLISP darstellen kann. Zur Zeit benutzt AutoLISP 12-Byte-Knoten. Um eine Zerstückelung des Speichers und zu großen Heapverwaltungsaufwand zu vermeiden, werden Knoten vom Heap aus in Gruppen angeordnet, die als *Segmente* bezeichnet werden. Vorgabemäßig besteht ein Segment aus 514 Knoten (6168 Byte).

AutoLISP führt eine Liste freier Knoten (Knoten, die noch nicht mit einem Symbol verbunden sind). Wenn ein Knoten zum Speichern eines Symbols oder eines Werts benötigt wird, durchsucht AutoLISP die Liste freier Knoten nach einem verfügbaren Knoten. Sollten keine vorhanden sein, wird eine automatische Abfallsammlung durchgeführt, die alle Knoten, die nicht mehr mit einem Symbol verbunden sind, in diese Liste freier Knoten einträgt. Danach wird ein Knoten ausgewählt, um die Anforderung zu erfüllen.

Wenn sich aus der Abfallsammlung zu wenig freie Knoten ergeben, fordert AutoLISP ein zusätzliches Segment vom Heap an. Wenn die Anforderung erfüllt wird, werden die neuen Knoten in die freie Liste übernommen und einer davon ausgewählt, um die ursprüngliche Anforderung zu erfüllen. Wenn keine zusätzlichen Segmente zur Verfügung stehen, wird ein virtuelles Funktionspaging gestartet, um durch Auslagern der am längsten unbenutzten Funktion Knoten freizumachen; andernfalls meldet das System, daß nicht genug Knotenraum zur Verfügung steht, und die Anforderung von Knotenraum wird abgebrochen. Beachten Sie, daß Knotenraum erst dann an den Heap zurückgegeben wird, wenn Sie AutoCAD beenden.

Es ist möglich, mit Hilfe der Funktion **gc** eine Abfallsammlung zu erzwingen:

```
(gc)
```

Allerdings ist die Abfallsammlung eine sehr zeitaufwendige Operation, die nicht unnötigerweise durchgeführt werden sollte. Dies sollte am besten dem automatischen Mechanismus von AutoLISP überlassen werden, der eine Abfallsammlung nur durchführt, wenn dies unbedingt erforderlich ist.

15.1.1 Wiederherstellen von Knotenraum

Wenn Sie Funktionen und Symbole erstellen, die Sie nur für kurze Zeit brauchen, *entdefinieren* Sie sie, wenn sie nicht mehr gebraucht werden, indem Sie ihnen *nil* zuweisen. Wenn Sie zum Beispiel eine Funktion mit Namen **setup** geladen und benutzt haben, die Sie nun nicht mehr brauchen, können Sie sie mit Hilfe von folgendem Code entfernen:

```
(setq setup nil)
```

Die Zuweisung von *nil* stellt den Knotenraum, der von der Funktion belegt wurde, wieder her. Andere Funktionen und Symbole können ihn dann benutzen.

Wenn Sie den Knotenraum freimachen wollen, der von einem Symbol vom Typ FILE (wie von der Funktion **open** zurückgegeben) belegt wird, müssen Sie es zuerst schließen, bevor Sie es auf nil setzen können. Eine nicht geschlossene Datei nimmt ihren Platz weiter in Anspruch und schränkt so die Anzahl neuer Dateien ein, die geöffnet werden können.

Anmerkung Vor Release 12 unterhielt AutoCAD eine `Atomliste` - eine Liste aller definierten Funktionen und Symbole. Diese Liste wird inzwischen nicht mehr auf diese Weise geführt, daher kann die Methode des "Bereinigens der Atomliste" nicht mehr zum Entfernen von Funktionen und Symbolen benutzt werden. Sie können aber immer noch mit Hilfe der Funktion **atoms-family** eine Liste der definierten Funktionen und Symbole erhalten:

```
(atoms-family 0)
```

Mit Hilfe dieser Funktion können Sie die vollständige Liste oder lediglich ausgewählte Symbole aufrufen. Weitere Information finden Sie unter **"atoms-family"** in Kapitel 13.

15.2 Technische Hinweise

Die folgenden Informationen wenden sich an erfahrene LISP-Benutzer. Sie beschreiben interne AutoLISP-Mechanismen, die möglicherweise bereits im nächsten Release geändert werden.

15.2.1 Zeichenkettenspeicher

Der Zeichenkettenspeicher entstammt demselben Heap wie die Knotensegmente. Wenn Ihr Programm die Funktionen zur manuellen Zuordnung (wird später beschrieben) aufruft, um den gesamten verfügbaren Speicherplatz als Knoten zuzuordnen, werden Sie wahrscheinlich eine Meldung über zu wenig Zeichenkettenspeicherplatz erhalten. Wir empfehlen Ihnen, die Speicherzuordnung durch die automatische Zuordnung der Knoten ausführen zu lassen. Zeichenkettenspeicherplatz wird für viele verschiedene Dinge benötigt, beispielsweise für Symbolnamen, Eingabeaufforderungen und Namen von Menüoptionen, die an AutoLISP zur Bewertung übergeben werden. Wenn Sie lange Menüoptionen mit AutoLISP-Ausdrücken zur Bewertung benutzen, benötigen diese eine beträchtliche Menge an zusammenhängendem Zeichenkettenspeicherplatz aus dem Heap.

15.2.2 Symbolspeicher

Die Speicherdarstellung, die in AutoLISP verwendet wird, macht intensiven Gebrauch von Zeigern. Knoten werden überall verwendet, und alles wird als eine Knotenstruktur dargestellt. Das einfache Gleichsetzen eines Symbols mit einem langen Namen mit einem Wert erfordert zwei Knoten: einen zur Speicherung des Symbolnamens und einen zur Speicherung seines Werts.

```
(setq longsym 3.1415)
```

Wenn der Symbolname sechs Zeichen oder weniger umfaßt, wird er direkt im Knoten für den Symbolnamen gespeichert; andernfalls wird Zeichenkettenspeicherplatz aus dem Heap zur Speicherung des Namens zugeordnet, und der Symbolnamenknoten zeigt dann auf diese Zeichenkette. Kurz gesagt, die Verwendung kurzer Symbolnamen (sechs oder weniger Zeichen) verringert den erforderlichen Zeichenkettenspeicher und vermeidet Heapzerstückelung.

15.2.3 Manuelle Zuordnung

Mit Hilfe der Funktionen **alloc** und **expand** können Sie die Zuordnung von Knoten- und Zeichenkettenspeicher manuell steuern. Wenn Sie diese Ausdrücke am Anfang Ihrer Datei *acad.lsp* einsetzen, können Sie die Knotenzuordnung vorgeben und auch Zeichenkettenspeicherplatz sparen. Dadurch reduziert sich die Anzahl der erforderlichen Abfallsammlungen, was die Laufleistung Ihrer Anwendung verbessert.

Sie können mit Hilfe der Funktion **alloc** die Größe der zukünftigen Segmentanforderungen auf einen anderen Wert als 514 Knoten pro Segment einstellen.

```
(alloc Zahl)
```

Die Funktion **alloc** stellt die Segmentgröße auf den Wert von *Zahl* ein und gibt die vorherige Einstellung zurück.

Mit Hilfe der Funktion **expand** können Sie manuell Knotenraum durch Anforderung einer bestimmten Anzahl von Segmenten zuordnen.

```
(expand Zahl)
```

wobei *Zahl* die Anzahl der Segmente ist, die Sie zuordnen wollen.

Die Funktion **expand** gibt die Anzahl der Segmente zurück, die sie aus dem Heap bekommen konnte; diese Zahl kann weit geringer sein als die angeforderte Zahl, je nachdem wieviel Heapraum noch verfügbar war.

Beispiel

```
(alloc 1024)
```

Setzt die Segmentgröße auf 1024 Knoten, wobei 12.288 Byte Heapraum für jedes Segment benötigt werden.

```
(alloc 100)
```

Setzt die Segmentgröße auf 100, wobei 1200 Byte für jedes Segment benötigt werden.

Wenn die Segmentgröße einen Vorgabewert von 514 Knoten hat, benötigt ein Aufruf der folgenden Anfragen 10 Segmente (10x12x514=61.680 Byte).

```
(expand 10)
```

Sie können mehr Segmente anfordern als verfügbar sind.

Ein praktisches Beispiel:

`(alloc 3000)` *Setzt die Segmentgröße für Knoten auf 3000 Knoten*
`(expand 1)` *Erhält 3000 freie Knoten (ein Segment)*
`(alloc 10000)` *Stellt eine große Segmentgröße ein, um das Hinzufügen weiterer Segmente zu vermeiden*

Dieses Schema benötigt 36.000 Byte für den Knotenraum, wobei der restliche Heap für Zeichenkettenspeicher verwendet werden kann. Die Knotenzuordnung wird vorgegeben und in der freien Liste plaziert. Eine Abfallsammlung ist erst notwendig, wenn Sie alle 3000 Knoten verbraucht haben. Wenn Sie diese Knoten verbraucht haben, werden Abfallsammlungen aufgerufen, um weitere Knotenanforderungen zu erfüllen. Durch (`alloc 10000`) wird vermieden, daß zusätzliche Segmente aus dem Heap zugeordnet werden, wobei dieser Raum als Zeichenkettenspeicher *reserviert* ist.

Wenn Sie die andere Strategie verfolgen und schrittweise die Segmentgröße verringern, bis die Anfrage nach einem Knoten nicht erfolgreich ist (d.h. `(alloc 1) (expand 1)` gibt 0 zurück), verbrauchen Sie den *gesamten* Heap für Knotenraum, wodurch die Meldung "Unzureichender Zeichenkettenspeicher" ausgegeben wird. Dieses Vorgehen wird jedoch nicht empfohlen, da AutoLISP ohne verfügbaren Zeichenkettenspeicher nutzlosist.

15.2.4 Virtuelles Funktionspaging

Virtuelles Funktionspaging kann erst eingesetzt werden, wenn alle anderen Arten des virtuellen Speichers erschöpft sind; dies geschieht auf den meisten Plattformen nur selten. Die Funktion **vmon** wird aus Kompatibilitätsgründen mit früheren Releases von AutoLISP unterstützt.

Wenn Ihre AutoLISP-Anwendung für den verfügbaren Knotenraum zu groß wird, können Sie den *virtuellen Funktionspager* von AutoLISP aktivieren, um Ihrem Programm das weitere Anwachsen zu ermöglichen. Hierzu führen Sie die Funktion **vmon** vor dem ersten **defun** in Ihrem Programm aus:

```
(vmon)
```

Dadurch wird das virtuelle Funktionspaging für den Rest der AutoCAD-Arbeitssitzung aktiviert. Wenn das Funktionspaging einmal aktiviert wurde, kann es nicht mehr ausgeschaltet werden. Nur solche Funktionen, die mit **defun** nach dem Aufruf der Funktion **vmon** erstellt wurden, sind für eine Auslagerung auswählbar. Wenn Sie also Funktionen vor dem Aufruf von **vmon** definieren, werden diese nicht ausgelagert und können immer noch einen Abbruch wegen unzureichendem Knotenraum verursachen.

Nach Ausführung der Funktion **vmon** lagert AutoLISP selten benutzte Funktionen immer dann aus, wenn der Knotenraum knapp wird, und liest sie automatisch wieder ein, wenn sie gebraucht werden. Die Funktionen werden in eine temporäre Datei ausgelagert, die durch den Datei-Pager von AutoCAD verwaltet wird.

Das virtuelle Speichersystem lagert nur *Funktionen* aus; Sie müssen immer noch genügend Knotenraum zur Verfügung haben, um alle Datenlisten, Funktionsnamen und Variablennamen ablegen zu können, die Ihr Programm benutzt.

Die Funktion **mem** zeigt zwei weitere Felder an, wenn **vmon** aktiviert ist.
Swap-ins ist die Anzahl der Funktionen, die aus der Auslagerungsdatei für das virtuelle Speichersystem *eingelagert* wurden. Dies sind ausgelagerte Funktionen, die aus der Auslagerungsdatei geholt werden, wenn sie in einem einfachen Funktionsaufruf angefordert werden. Der Eintrag *Auslagerungsdatei* enthält die Größe der für die ausgelagerten Funktionen erstellten Auslagerungsdatei.

Nach Ausführung der Funktion **vmon** fügen alle **defuns** einen neuen Knoten mit Namen *Auslagerungstabelle* am Anfang jeder Funktionsliste ein. Dieser Knoten wird vor der Liste, welche die formalen Argumente enthält, hinzugefügt. Diese Knoten mit den Auslagerungstabellen sind ausschließlich für den Gebrauch durch den Pager bestimmt und sollten in keiner Weise von Benutzerprogrammen manipuliert werden. Die Funktion **type** gibt `PAGETB` für diese Knoten zurück.

Wenn der Knotenraum knapp wird, wird die am seltensten benutzte Funktion ausgelagert. AutoLISP schreibt sie in die Auslagerungsdatei, nimmt die Adresse der Auslagerungsdatei in die Auslagerungstabelle auf und gibt alle Knoten der Funktion gemäß der Auslagerungstabelle frei. Die Auslagerungstabelle wird markiert, um anzuzeigen, daß die Funktion ausgelagert wurde. Wenn eine ausgelagerte Funktion ausgewertet wird, wird sie wieder aus der Auslagerungsdatei eingelesen (eventuell unter Auslagerung anderer Funktionen), bevor sie ausgeführt wird. Wenn eine Funktion einmal in die Auslagerungsdatei geschrieben wurde, werden bei späteren Auslagerungen lediglich ihre Knoten freigegeben; es besteht keine Notwendigkeit, die Funktion wieder in die Auslagerungsdatei zu schreiben, da sie bereits darin enthalten ist.

In AutoLISP sind Funktionen, die mit **defun** erstellt werden, Listen, die manipuliert werden können. Programme, die dies tun, müssen die Arbeitsweise des Pagers kennen (andernfalls dürfen sie **vmon** nicht benutzen). Mit **defun** erstellte Funktionen haben einen Knoten für die Auslagerungstabelle am Anfang; daher sollten Sie sie überspringen, wenn Sie die

Funktion scannen. Wenn Sie eine Funktion als Liste erstellen (und **defun** umgehen), wird sie problemlos arbeiten, ist aber nicht für eine Auslagerung geeignet, was zu Speicherknappheit führen kann. Umgekehrt können Sie eine Funktion im Speicher festschreiben, indem Sie sie ohne ihre Auslagerungstabelle neu definieren. Um zum Beispiel eine Funktion mit Namen `zorp` im Speicher festzuschreiben, können Sie mit dem folgenden Code die erste Auslagerungstabelle löschen:

```
(setq zorp (cdr zorp))
```

Eine Auslagerungstabelle wird als Leerzeichen ausgegeben, wenn Sie die Funktion drucken. Sie können feststellen, ob eine Funktion auslagerbar ist, indem Sie prüfen, ob nach der ersten öffnenden Klammer ein Leerzeichen folgt; wenn ja, ist die Funktion auslagerbar.

Wenn Sie versuchen, die Funktion als Daten einzulesen, während sie ausgelagert ist, erscheint nur die Auslagerungstabelle in der Funktionsliste. Durch einen Zugriff allein wird die Funktion nicht wieder eingelagert - dies geschieht nur, wenn sie *ausgewertet* wird. Wenn Sie Funktionen erstellen und sie aus dem Stegreif modifizieren, erstellen Sie sie entweder als Listen anstatt mit Hilfe von **defun**, oder benutzen Sie die eben gezeigte Methode, um die Funktion im Speicher festzuschreiben.

16 AutoLISP-Fehlercodes und Fehlermeldungen

Dieses Kapitel erläutert die AutoLISP-Fehlercodes und -Fehlermeldungen.

16.1 Fehlercodes

Die folgende Tabelle enthält die Werte der von AutoLISP generierten Fehlercodes. Wenn ein AutoLISP-Funktionsaufruf einen Fehler verursacht, den AutoCAD erkennt, nimmt die Systemvariable ERRNO einen der Werte in der Tabelle an. AutoLISP-Anwendungen können den aktuellen Wert der Variablen ERRNO mit **`(getvar "errno")`** überprüfen.

Die Systemvariable ERRNO wird nicht immer auf Null zurückgesetzt. Wenn Sie den Wert der Variablen nicht unmittelbar überprüfen, nachdem eine AutoLISP-Funktion einen Fehler gemeldet hat, können die angezeigten Werte zu falschen Schlüssen über die Ursache des Fehlers führen. Die Variable ERRNO wird immer dann zurückgesetzt, wenn Sie eine neue Zeichnung beginnen oder eine vorhandene öffnen.

Anmerkung Die Werte und Bedeutungen der Variablen ERRNO können sich in zukünftigen AutoCAD-Releases ändern

Online-Fehlercodes

Wert	Bedeutung
0	Kein Fehler
1	Ungültiger Symboltabellenname
2	Ungültiger Objekt- oder Auswahlsatzname
3	Maximale Anzahl der Auswahlsätze überschritten
4	Ungültiger Auswahlsatz
5	Unzulässige Anwendung einer Blockdefinition
6	Unzulässige Anwendung des Befehls xref
7	Objektwahl: Auswahl fehlgeschlagen
8	Ende der Objektdatei
9	Ende der Blockdefinitionsdatei
10	Letztes Objekt nicht gefunden
11	Unzulässiger Versuch, Ansichtsfensterelement zu löschen
12	Operation bei Befehl PLINIE unzulässig
13	Unzulässige Referenz
14	Referenzen nicht aktiviert
15	Ungültige Argumente in Anforderung für Koordinatensystem-Konvertierung
16	Unzulässiges Leerzeichen in Anforderung für Koordinatensystem-Konvertierung
17	Unzulässige Verwendung eines gelöschten Objektes
18	Ungültiger Tabellenname
19	Ungültiges Argument in Tabellenfunktion
20	Versuch, eine schreibgeschützte Variable zu definieren
21	Wert Null nicht zulässig
22	Wert liegt nicht im zulässigen Bereich
23	Komplexer REGEN-Prozeß wird ausgeführt
24	Versuch, Elementtyp zu ändern
25	Unzulässiger Layername

26	Unzulässiger Linientypname
27	Unzulässiger Farbname
28	Unzulässiger Textstilname
29	Unzulässiger Symbolname
30	Unzulässiges Feld für Elementtyp
31	Versuch, gelöschtes Element zu ändern
32	Versuch, Subelement seqend zu ändern
33	Versuch, Referenz zu ändern
34	Versuch, Sichtbarkeit des Ansichtsfensters zu ändern
35	Element auf gesperrtem Layer
36	Unzulässiger Elementtyp
37	Unzulässiges Polylinienelement
38	Block enthält unvollständiges komplexes Element
39	Ungültiges Feld für Blockname
40	Doppelte Blockflagfelder
41	Doppelte Blocknamenfelder
42	Unzulässiger Vektor für Normale
43	Fehlender Blockname
44	Fehlendes Blockflag
45	Ungültiger unbenannter Block
46	Ungültige Blockdefinition
47	Obligatorisches Feld fehlt
48	Unbekannter Typ von Extended Data (XDATA)
49	Liste in XDATA falsch verschachtelt
50	APPID-Feld hat falsche Position
51	Maximalwert für XDATA überschritten
52	Elementwahl: Leerantwort
53	APPID duplizieren
54	Versuch, Ansichtsfensterelement zu erzeugen oder zu ändern
55	Versuch, eine Xref, Xdef oder Xdep zu erzeugen oder zu ändern
56	`ssget`-Filter: unerwartetes Listenende
57	`ssget`-Filter: fehlender Testoperand
58	`ssget`-Filter: ungültiger Opcode (-4)-Zeichenfolge
59	`ssget`-Filter: falsche Verschachtelung oder leere Bedingung
60	`ssget`-Filter: Anfang und Ende der Bedingung fehlerhaft
61	`ssget`-Filter: Bedingung enthält falsche Anzahl von Argumenten (für NOT oder XOR)
62	`ssget`-Filter: Verschachtelungslimit überschritten
63	`ssget`-Filter: ungültiger Gruppencode

64	`ssget`-Filter: ungültiger Zeichenfolgentest
65	`ssget`-Filter: ungültiger Vektortest
66	`ssget`-Filter: ungültiger Test für reelle Zahlen
67	`ssget`-Filter: ungültiger Ganzzahltest
68	Digitalisierer ist kein Tablett
69	Tablett nicht kalibriert
70	Ungültige Tablettargumente
71	ADS-Fehler: Kann neuen Ergebnispuffer nicht zuweisen
72	ADS-Fehler: Nullzeiger ermittelt
73	Kann ausführbare Datei nicht öffnen
74	Anwendung bereits geladen
75	Zahl der Anwendungen übersteigt Grenzwert
76	Kann Anwendung nicht ausführen
77	Versionsnummern nicht kompatibel
78	Kann verschachtelte Anwendungen nicht beenden
79	Kann Anwendung nicht beenden
80	Anwendung zur Zeit nicht geladen
81	Nicht genügend Speicher, um Anwendung zu laden
82	ADS-Fehler: ungültige Transformationsmatrix
83	ADS-Fehler: ungültiger Symbolname
84	ADS-Fehler: ungültiger Symbolwert
85	AutoLISP/ADS-Operation wurde durch geöffnetes Dialogfeld behindert

16.2 Fehlermeldungen

Erkennt AutoLISP einen Fehler, wird die aktuelle Funktion abgebrochen, die benutzerdefinierte Funktion **`*error*`** aufgerufen und eine Meldung angezeigt, die auf die Art des Fehlers hinweist. Ist keine benutzerdefinierte Funktion **`*error*`** vorhanden (oder ist **`*error*`** auf Null gesetzt), wird die Fehlermeldung von der Standardfehlerbehandlung in folgender Form angezeigt:

Fehler: *Meldung*

Nach der Meldung wird die Funktion zurückverfolgt. Existiert eine benutzerdefinierte **`*error*`**-Funktion, wird diese nicht zurückverfolgt. Statt dessen wird die Benutzerfunktion mit *Meldung* als einzigem Argument aufgerufen.

16.2.1 Fehler in benutzerdefinierten Programmen

Dieser Abschnitt erläutert Fehlermeldungen, die auftreten können, wenn Sie AutoLISP-Funktionen schreiben oder auf Fehler überprüfen. Die meisten Meldungen weisen auf AutoLISP-Programmierfehler der folgenden Art hin:

- Schreibfehler in Funktions- oder Symbolnamen
- Falsche Anzahl oder falscher Typ von Funktionsargumenten
- Falsch plazierte oder fehlende Klammern
- Falsch plazierte oder fehlende Anführungszeichen (Zeichenkette nicht geschlossen)
- Korrekte Ausführung einer Funktion wurde vor dem Versuch, das Ergebnis der Funktion zu verwenden, nicht überprüft

Obwohl diese Meldungen in der Regel auf Programmierfehler durch den Benutzer hinweisen, können sie auch von Programmfehlern in AutoLISP verursacht werden. Wenn Sie in Ihrem Programm keine Fehler finden können, durch die

die Meldung verursacht worden sein könnte, schicken Sie bitte einen Fehlerbericht an Autodesk, zusammen mit einer Kopie Ihres Codes (vorzugsweise auf einer Diskette).

AutoCAD hat diese Funktion zurückgewiesen

Die Argumente, die an eine AutoCAD-Funktion übergeben wurden, sind ungültig (z. B. **`setvar`** bei einer schreibgeschützten Systemvariablen oder **`tblnext`** mit ungültigem Tabellennamen), oder die Funktion selbst kann im aktuellen Kontext nicht verwendet werden. Beispielsweise können Sie eine Benutzereingabefunktion vom Typ **`get`*`xxx`*** nicht innerhalb der Funktion **`command`** benutzen.

AutoLISP-Stapel-Überlauf

Der AutoLISP-Stapelspeicher ist überfüllt. Dies kann durch besonders häufiges Wiederholen von Funktionen oder durch besonders lange Listen mit Funktionsargumenten verursacht werden.

Basispunkt erforderlich

Die Funktion **`getcorner`** wurde ohne das erforderliche Basispunktargument aufgerufen.

Boole arg 1 > 0 < 15

Das erste Argument der Boole Funktion muß eine Ganzzahl zwischen 0 und 15 sein.

Datei gelesen - unzureichender Platz für Zeichenfolge

Zeichenkettenspeicher wurde erschöpft, während AutoLISP aus einer Datei las. Siehe Kapitel 15, "Speicherverwaltung".

Datei ist nicht offen

Der Dateideskriptor für die E/A-Operation ist nicht für eine geöffnete Datei.

defun-Argumente dürfen nicht den gleichen Namen haben

Diese Fehlermeldung wird angezeigt, wenn für eine Funktion mehrere Argumente mit gleichem Namen definiert wurden und sie dadurch fehlschlägt.

Division durch Null

Es ist nicht erlaubt, durch Null zu dividieren.

Divisionsüberlauf

Die Division durch einen sehr kleinen Wert führte zu einem ungültigen Quotienten.

Eingabe abgebrochen

Die Dateieingabe wurde beendet, weil ein Fehler oder eine vorzeitige Dateiendebedingung erkannt wurde.

Endpunkt fehlt in grvecs

In der an **`grvecs`** übergebenen Vektorliste fehlt ein Endpunkt.

Falsche Anzahl Argumente

Die Funktion **`quote`** erwartet genau ein Argument, es wurde jedoch eine andere Anzahl Argumente übergeben.

Falsche Anzahl Argumente an eine Funktion

Die Anzahl Argumente an die benutzerdefinierte Funktion entspricht nicht der Anzahl formeller Argumente, die in der Funktion defun festgelegt wurden.

Falscher Argumenttyp

Einer Funktion wurde ein falscher Argumententyp übergeben. (Beispielsweise können Sie die Funktion **`strlen`** nicht für eine Ganzzahl benutzen.)

Fehlerhaft gebildete Liste

Eine Liste, die aus einer Datei gelesen wurde, hat vorzeitig geendet. Dies geschieht in den meisten Fällen durch falsch plazierte oder fehlende öffnende oder schließende Klammern oder Anführungszeichen.

Fehlerhafte Anfrage für Befehlslistendaten

Eine Befehlsfunktion wurde gefunden, kann jedoch wegen einer anderen aktiven Funktion nicht ausgeführt werden, oder das Befehls-Interpretierprogramm ist nicht vollständig initialisiert. Dies kann durch einen Aufruf der Funktion **`command`** in *acad.lsp*, *acadr13.lsp* oder einer Datei *.mnl* geschehen.

Fehlerhafte formelle Argumentliste

Bei der Auswertung dieser Funktion hat AutoLISP eine ungültige formelle Argumentliste ermittelt. Vielleicht handelt es sich nicht um eine Funktion, sondern um eine Datenliste.

Fehlerhafte Liste

Eine fehlerhaft gebildete Liste wurde an eine Funktion übertragen. Dieser Fehler kann auftreten, wenn eine Ganzzahl mit einem Dezimalpunkt beginnt. Sie müssen in diesem Fall dem Dezimalpunkt eine Null voranstellen.

Fehlerhafte Zeichenfolge

Eine Zeichenkette, die aus einer Datei gelesen wird, hat vorzeitig geendet.

Fehlerhafter Wert in ENTMOD-Liste

Eine der Sublisten der an die Funktion **`entmod`** übergebenen Assoziationsliste enthält einen unzulässigen Wert.

Fehlerhafter Wert in grvecs-Liste

Eine **`grvecs`**-Liste enthält Werte, die keine 2D- oder 3D-Punkte beschreiben.

Fehlerhafter Wert in ssget-Liste

Eine der Sublisten der an die Funktion (**`ssget`** "**`X`**") übergebenen Assoziationsliste enthält einen unzulässigen Wert.

Funktion abgebrochen

Der Benutzer hat CTRL+C oder ESC (Abbruch) als Antwort auf eine Eingabeaufforderung eingegeben.

Funktion für Argument nicht definiert

Das an **`log`** oder **`sqrt`** übergebene Argument liegt außerhalb des Wertebereichs.

Funktion nicht definiert für Realzahl

Es wurde eine reelle Zahl als Argument an eine Funktion übergeben, die eine Ganzzahl erfordert (z.B. **`lsh val 1.2)`**.

Funktion ungültig

Das erste Element in der Liste ist kein gültiger Funktionsname. Vielleicht handelt es sich um einen Variablennamen oder eine Zahl. Diese Meldung kann auch darauf hinweisen, daß die genannte Funktion fehlerhaft definiert wurde. (Beachten Sie die erforderliche formelle Argumentliste.)

Gleitkomma Ausnahme

Nur bei UNIX-Systemen: Das Betriebssystem hat einen Fehler in der Arithmetik des Gleitkommas gefunden.

Illegal type in Left

Die Datei *.lsp* ist keine reine ASCII-Datei, sie wurde aber in einem Textverarbeitungsprogramm gespeichert und enthält Formatierungscodes.

Kann Datei nicht für Eingabe öffnen -- LOAD-Fehler

Die in der Funktion **`load`** genannte Datei konnte nicht gefunden werden, oder der Benutzer hat kein Zugriffsrecht für die Datei.

Kann den Ausdruck nicht prüfen

Ein Dezimalpunkt wurde falsch plaziert oder ein anderer Ausdruck fehlerhaft festgelegt.

Kann nicht in AutoLISP zurückkehren

Der AutoCAD/AutoLISP Kommunikationspuffer wird von einer aktiven Funktion benutzt. Es kann keine neue Funktion aufgerufen werden, bis die aktive Funktion beendet ist.

Leere Funktion

Es wurde versucht, eine Funktion mit dem Wert nil zu bewerten.

LISPSTACK übergelaufen

Der AutoLISP-Stapelspeicher ist überfüllt. Dies kann durch besonders häufiges Wiederholen von Funktionen oder durch besonders lange Listen mit Funktionsargumenten verursacht werden.

Max. Dateigröße überschritten

Eine Datei hat die maximale Dateigröße des Betriebssystems überschritten.

Maximale Zeichenfolgenlänge überschritten

Eine Zeichenkette mit mehr als 132 Zeichen wurde an eine Funktion übergeben.

Punkt an falscher Stelle

Eine reelle Zahl beginnt mit einem Dezimalpunkt. Sie müssen in diesem Fall dem Dezimalpunkt eine Null voranstellen.

Quit/beenden abbrechen

Die Funktion **`quit`** oder **`exit`** ist aufgerufen worden.

Tastatur-Abbruch

Der Benutzer hat CTRL+C eingegeben, während eine Funktion ausgeführt wurde.

Überzählige rechte Klammer

Es wurde mindestens eine zusätzliche rechte Klammer gefunden.

Ungenügend Knotenraum

Es gibt nicht genügend Heapraum, um die angefragte Operation auszuführen. Siehe Kapitel 15, "Speicherverwaltung".

Ungenügend Zeichenkettenspeicher

Es gibt nicht genügend Heapraum, um die festgelegte Zeichenkette auszuführen. Siehe Kapitel 15, "Speicherverwaltung".

Ungültige Argumentliste

Eine fehlerhafte Argumentliste wurde an eine Funktion übergeben.

Ungültige Assoziationsliste

Die Liste, die der Funktion **`assoc`** übergeben wurde, besteht nicht aus Schlüsselwertlisten.

Ungültige ENTMOD-Liste

Der Funktion **`entmod`** wurde ein Argument übergeben, das keine korrekte Elementdatenliste ist (wie von **`entget`** zurückgegeben wurde).

Ungültige ssget-Liste

An die Funktion (**`ssget`** "**`X`**") wurde ein Argument übergeben, das keine korrekte Elementdatenliste ist (wie von **`entget`** zurückgegeben wurde).

Ungültiger Node

Die Funktion **`type`** hat einen ungültigen Elementtyp gefunden.

Ungültiger Ganzzahlwert

Es wurde eine Zahl erkannt, die unter der kleinsten bzw. über der größten Ganzzahl liegt.

Ungültiges Argument

Falscher Argumenttyp oder Argument außerhalb des Wertebereichs.

Unzulässige Option für ssget

Dieser Fehler wird verursacht, wenn eine ungültige Zeichenkette im Argument *Modus* an **ssget** übergeben wurde.

Unzulässige Punktliste

Eine leere Liste oder eine Liste, die keine Punkte enthält, wird mit einer F-, KP- oder WP-Anfrage verschickt. Verwendet von **ssget** und **grvecs**.

Unzulässige reelle Zahl entdeckt

Es wurde versucht, eine ungültige reelle Zahl von AutoLISP an AutoCAD zu übergeben.

Unzulässige Xdata-Liste

Dieser Fehler wird verursacht, wenn eine erweiterte Elementdatenliste, die falsch erstellt wurde, an **xdsize, ssget, entmod, entmake** oder **txtbox** übertragen wurde.

Unzulässiger Funktionscode

Dem Befehl **tablet** wurde ein unzulässiger Funktionsbezeichner übergeben.

Unzulässiger grvecs Matrixwert

An **grvecs** wurde eine Matrix übertragen, die fehlerhaft erstellt wurde oder den falschen Datentyp enthält (z.B. STR, SYM usw.).

Unzulässiger Konversionscode

Der Funktion **trans** wurde ein ungültiger Bereichsbezeichner übergeben.

Unzulässiger Nodetyp in Liste

Die Funktion **foreach** hat eine ungültigen Elementtyp gefunden.

Unzulässiges Argument

Das Argument zu **gcd** ist negativ oder null.

Unzulässiges Objekt in Funktion

Der **vmon**-Funktionspager hat eine fehlerhaft formulierte Funktion erkannt.

Unzulässiges gepunktetes Paar

Ein gepunktetes Paar ist eine Liste mit zwei Elementen, die durch die Konstruktion *Leerzeichen-Punkt-Leerzeichen* getrennt ist. Diese Fehlermeldung kann auftreten, wenn Sie eine reelle Zahl mit einem Dezimalpunkt beginnen. Sie müssen in diesem Fall dem Dezimalpunkt eine Null voranstellen.

Unzulässiges Punktargument unzulässiger Punkt

Ein falsch definierter Punkt (eine Liste aus zwei reellen Zahlen) wurde an eine Funktion übertragen, die einen Punkt erwartet hat. Dies kann beispielsweise geschehen, wenn eine reelle Zahl mit einem Dezimalpunkt beginnt. Sie müssen in diesem Fall dem Dezimalpunkt eine Null voranstellen.

Unzulässiges Zeichen

Ein Ausdruck enthält ein unzulässiges Zeichen.

Zeichenfolge zu lang

Der Funktion **setvar** wurde eine zu lange Zeichenkette übergeben.

Zu viele Argumente

Einer systemeigenen Funktion wurden zu viele Argumente übergeben.

Zu wenige Argumente

Einer systemeigenen Funktion wurden zu wenig Argumente übergeben.

Zu wenige Argumente für grvecs

Der Funktion **grvecs** wurden zu wenig Argumente übergeben.

16.2.2 Interne Fehler

Die folgenden Fehlermeldungen weisen auf interne AutoLISP-Programmfehler hin. Bitte lassen Sie Autodesk einen Fehlerbericht zukommen.

Bad argument to system call (Unzulässiges Argument für Systemaufruf)

Nur bei UNIX-Systemen: Das Betriebssystem hat einen unzulässigen Systemaufruf gefunden, der von AutoLISP erzeugt wurde.

Bus error (Bus-Fehler)

Nur bei UNIX-Systemen: Das Betriebssystem hat einen Busfehler erkannt.

Hangup (Maschinenstopp)

Nur bei UNIX-Systemen: Das Betriebssystem hat ein *Maschinenstopp*-Signal erkannt.

Illegal instruction (Unzulässige Anweisung)

Nur bei UNIX-Systemen: Das Betriebssystem hat einen ungültigen Maschinenbefehl erhalten.

Segmentation violation (Segmentierung verletzt)

Nur bei UNIX-Systemen: Das Betriebssystem hat einen Versuch erkannt, einen Bereich außerhalb des Speicherbereichs für diesen Prozeß zu adressieren.

Unexpected signal *nnn* (Unerwartetes Signal *nnn)*

Nur bei UNIX-Systemen: Vom Betriebssystem wurde ein unerwartetes Signal empfangen.