


# **1 Visual-LISP Editor ab AutoCAD 2000**

## **1.1 Allgemein**

Der Visual-LISP Editor ist eine komfortable Programmierumgebung innerhalb der AutoCAD-Sitzung. LISP-Programme können in die aktuelle Zeichnung geladen und ausgeführt werden. **Diese Dokumentation ist keine vollständige Beschreibung des Visual-LISP-Editors, es werden lediglich sinnvolle Arbeitsweisen mit dem Editor beschrieben.**

Der Visual-LISP Editor wird mit **Befehl:VLISP↵** in der Eingabeaufforderung gestartet.

## **1.2 Fenster**

Das Umschalten zwischen den Fenstern erfolgt über den Button  oder über **→Fenster →...**

In beiden Fällen stehen alle geöffneten Fenster zur Verfügung.

### **1.2.1 Texteditorfenster**

Erzeugen Sie eine neue Datei über **→Datei→neu** oder öffnen Sie eine bereits bestehende Datei über **→Datei→öffnen**. Für jede Datei wird ein eigenes **Texteditorfenster** geöffnet. Die wichtigsten Dateiformate sind **\*.isp** und **\*.dcl**. In den LSP-Dateien wird der Quellcode der LISP-Programme, in den DCL-Dateien der Dialogfenster geschrieben.

### **1.2.2 Visual-LISP Konsole**

Wenn Sie ein Programm laden wird das Fenster **Visual-LISP Konsole** aktiviert. In diesem Fenster wird protokolliert, welche Dateien geladen worden sind. Sie haben die Möglichkeit, den Programmaufruf und LISP-Ausdrücke direkt einzugeben, sie werden nach Bestätigung mit **ENTER** sofort ausgewertet.

### **1.2.3 Ausgabe für Erstellung**

In diesem Fenster werden die Ergebnisse von Fehlerprüfungen ausgegeben.

### **1.2.4 Ablaufverfolgung**


In diesem werden alle Vorgänge bezüglich der Visual-LISP Umgebung protokolliert, es sind keine Eingaben möglich, das Fenster dient der Information.

### 1.3 Programme laden, ausführen, debuggen

Zur Beschreibung der einzelnen Schritte wird der folgende Beispielcode dienen. Der Code ist in einem geöffneten Texteditorfenster vorhanden, die Datei muss nicht gespeichert sein, es **sollte** aber vor Probelaufen gespeichert werden. Die Zeilennummern sind nur zur Orientierung dargestellt.

```
01:      (defun c:JB_schulung ( / a b c faktor)
02:        (setq a 10.0
03:              b 20.0
04:              c 30.0)
05:        (if (setq faktor(getreal "\Bitte geben Sie einen Faktor ein:"))
06:            (progn
07:              (setq a -(+(* a faktor)b)c))
08:              (princ (strcat"\nErgebnis: " (rtos a 2 2)))
09:            )
10:        )
11:      (princ)
12:    )
```

#### 1.3.1 Laden des Codes und Prüfung auf Syntaxfehler

Um einen Code auszuführen muss er geladen sein. Vor dem Laden ist es sinnvoll, den Code über → *Extras* → *Text im Editor prüfen* oder über den Button  prüfen zu lassen.

Beispiel: (Syntax-Fehler in Zeile 0, fehlende Klammer )


```
07:      (setq a -(+(* a faktor)b)c))
```

Das Ergebnis der Fehlerprüfung wird im Fenster **Ausgabe für Erstellung** wie folgt dargestellt:


```
[CHECKING TEXT Lädt JB_schulung.lsp...]
-
; Fehler: Fehlerhafter Variablenname in SETQ: (+ (* A FAKTOR) B)
-
; Fehler: Überzählige rechte Klammer in Eingabe
; Prüfung durchgeführt.
```

Das Ergebnis zeigt, dass durch die fehlende Klammer die Funktion „-“ nicht als Funktion erkannt wird, folglich wird ein unkorrekter Variablenname erkannt.

Des weiteren fehlt die sich öffnenden Klammer im Gesamt-Konstrukt. Es muss zu jeder sich öffnenden auch eine sich schließenden Klammer existieren. Im Beispiel es ist eine überzählige sich schließende Klammer vorhanden.

 Durch einen Doppelklick auf die farblich markierte Fehlermeldung wird das **Texteditorfenster** automatisch wieder das aktuelle Fenster. Der Teil des Codes, der die Fehlermeldung hervorgerufen hat, wird automatisch markiert.


Dieser Vorgang muss solange wiederholt werden, bis die Fehler beseitigt sind.

Jetzt kann der Code geladen werden, entweder über → *Extras* → *Text in Editor laden* oder über den Button . Ist der Code fehlerfrei, so wird die **Visual-LISP Konsole** das aktive Fenster. Es wird angeschrieben, dass ein Formular (Funktion) aus der Datei [Pfad /

Dateiname] geladen ist. Wären immer noch Syntax-Fehler im Code vorhanden, dann würde auch hier eine Fehlermeldung angezeigt werden.

### 1.3.2 Aufrufen der geladenen Funktion

Zum Aufrufen der Funktion kann direkt im aktuellen Fenster (**Visual-LISP-Konsole**) der Funktionsaufruf (`c:JB_schulung`)↵ eingegeben werden. Es ist wichtig, diesen Aufruf in Klammern zu setzen.

Sinnvoller ist es, über den Button  oder über die Taskleiste direkt in die AutoCAD-Umgebung zu wechseln. Der Funktionsaufruf lautet `Befehl:JB_schulung`↵. Wenn die Funktion ohne `c:` definiert worden wäre, so hätte der Funktionsaufruf wieder in Klammern gesetzt werden müssen.

☞ Durch die Cursertaste „nach oben“ in der AutoCAD-Umgebung können die bereits getätigten Eingaben wiederholt werden.

Nach dem Funktionsaufruf erwartet die Funktion die Eingabe des Faktors und gibt als Rückgabewert des Ergebnis der Berechnung zurück.

### 1.3.3 Debuggen

Um logische Fehler aufzuspüren kann die automatische Fehlersuche des Editors nicht weiterhelfen. Hier ist gefragt, wo eine Funktion aufgrund fehlerhafter oder fehlender Argumente nicht korrekt arbeiten kann.

Beispiele:

(/ a b) => wenn a = 1, b = 0; Division durch 0


(/ a b) => wenn a = nil, b = 10; Fehlerhafter Argumenttyp: numberp: nil

Bei komplexen Programmen muss man sich dem Fehler annähern, indem Haltepunkte an Positionen gesetzt werden, wo das gesamte Programm noch fehlerfrei läuft. Von dort aus geht man im Code schrittweise vor und kontrolliert die verwendeten Variablen.

Für die Fehlersuche wird in dem Beispielcode in Zeile 02 der Variablen **a nil** und nicht **10.0** zugewiesen.

☞ Durch die Verwendung von Haltepunkten und des Überwachungsfensters können globale und lokale Variablen überwacht werden.

### 1.3.3.1 Setzen von Haltepunkten


Haltepunkte sollten grundsätzlich immer an eine sich öffnende Klammer gesetzt werden. Vor dem setzen eines Haltepunktes muss der Cursor vor eine sich öffnende Klammer positioniert werden. Dann kann über den Button  oder über **F9** ein Haltepunkt gesetzt werden, wenn bereits einer existiert, dann wird er wieder entfernt. Die Haltepunkte werden als rot hinterlegte Klammern dargestellt.

```
01:      (defun c:JB_schulung ( / a b c faktor)
02:      [setq a nil
03:          b 20.0
04:          c 30.0)
05:      (if (setq faktor(getreal "\Bitte geben Sie einen Faktor ein:"))
06:          (progn
07:              (setq a (-(* a faktor)b)c))
08:              (princ (strcat"\nErgebnis: " (rtos a 2 2))))
09:          )
10:      )
11:      (princ)
12:      )
```

Über **→ Ansicht → Haltepunkt-Fenster** erscheint eine Übersicht über alle gesetzten Haltepunkte. Dieses Fenster ist bei langen unübersichtlichen Codes nützlich, weil über einen Button **Anzeigen** die Möglichkeit gegeben ist, im Code direkt zu den Haltepunkten zu springen.

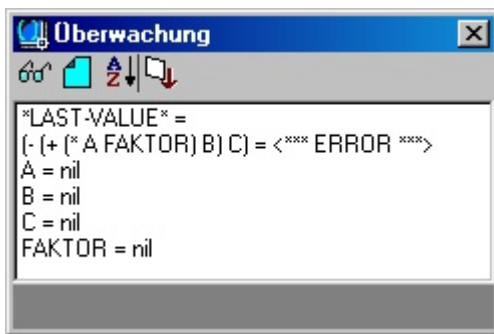
 Um alle Haltepunkte zu entfernen kann **STRG+SHIFT+F9** verwendet werden.

### 1.3.3.2 Variablen überwachen


Falls das Überwachungsfenster noch nicht geöffnet ist, wird es **Ansicht → Überwachungsfenster** oder über den Button  geöffnet. Wenn noch keine Überwachung aufgelistet ist, so wird über **→ Debugging → letzte Auswertung überwachen** die Überwachung „LAST-VALUE“ hinzugefügt. Damit wird immer der letzte Rückgabewert überwacht. Das ist besonders hilfreich, wenn man sich in Einzelschritten durch den Code bewegt.

Als weitere Variablen, die überwacht werden sollen, werden **a**, **b**, **c** und **faktor** wie folgt hinzugefügt: Sie klicken innerhalb des Quelltextes eine Variable mit gedrückter rechter Maustaste an, im Kontextmenü wird **Überwachung hinzufügen...** gewählt und das Folgefenster mit **ENTER** bestätigt. Es ist ebenfalls möglich, ganze Ausdrücke zu überwachen, wenn z.B. in Zeile 07 der Ausdruck `(-(* a faktor)b)c` markiert ist kann dieser ebenfalls über das Kontextmenü der Überwachung hinzugefügt werden.

Wenn alle Überwachungen hinzugefügt sind, sollte das Überwachungsfenster wie folgt aussehen:



Die Überwachung kann nur 20 Werte gleichzeitig überwachen. Bei dem Versuch mehr Überwachungen hinzuzufügen kommt folgende Meldung: „**Bereich für Überwachung überschritten**“

Um Überwachungen zu entfernen wird ein Eintrag markiert und über die rechte Maustaste das Kontextmenü aufgerufen: *Aus Überwachung entfernen*. Um die gesamte Liste zu löschen wird der Button  innerhalb des Überwachungsfensters verwendet.

### 1.3.3.3 Schrittweise durch den Code bewegen



ist der Button für einen **Einzelschritt**

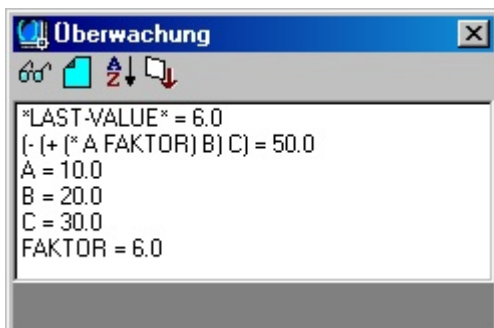


ist der Button für einen **Prozedurschritt**




ist der Button für einen **Prozedurabschluss**

Der Code wird wie unter 1.3.1 beschrieben in der AutoCAD-Umgebung aufgerufen. Sobald der erste Haltepunkt erreicht ist, wird der Visual-LISP Editor das aktive Fenster. Über den Button **Einzelschritt** bewegt man sich in Einzelschritten durch den Code. Dabei ist das Überwachungsfenster zu beachten. Sobald die Zuweisung der Werte für die Variablen **a**, **b**, **c** erfolgt ist werden diese Inhalte in der Überwachung dargestellt. Sobald die Funktion `getreal` in der Zeile 05 erreicht ist wird das AutoCAD-Fenster wieder das aktive Fenster, denn es wird die Benutzereingabe erwartet. Nach der Benutzereingabe ist der eingegebene Wert in der Überwachung „**LAST-VALUE**“ zu sehen, erst nach dem nächsten Einzelschritt wird der Wert an die Variable **faktor** zugewiesen und im Überwachungsfenster dargestellt. In Zeile 07 bei `(* a faktor)` bricht die Funktion ab, weil der Wert der Variablen **a = nil** ist. Wenn der „Fehler“ beseitigt ist, und der Variablen **a** wieder der Wert **10.0** zugewiesen ist sieht das Überwachungsfenster zur Laufzeit folgendermaßen aus:



Die Buttons **Prozedurschritt** wird verwendet, wenn Bereiche, die bereits fehlerfrei sind, übersprungen werden sollen. Über den Button **Prozedurabschluss** wird das Ende der Hauptfunktion erreicht.

Über den Button  wird der Debug-Modus verlassen, alle Werte werden wieder zurückgesetzt.

### 1.3.4 Editor-Funktionen

Innerhalb des Texteditorfensters gibt es viele Funktionalitäten, mit denen der Code effizient geschrieben und bearbeitet werden kann.

#### 1.3.4.1 Markieren bestimmter Bereiche

Durch einen Doppelklick vor eine sich öffnende Klammer oder hinter eine sich schließende Klammer wird der gesamte Ausdruck markiert.

```
(defun c:JB_schulung ( / a b c faktor)
  (setq a 10.0
        b 20.0
        c 30.0)
  (if (setq faktor(getreal "\Bitte geben Sie einen Faktor ein:"))
      (progn
        (setq a (-(* a faktor)b)c))
        (princ (strcat"\nErgebnis: " (rtos a 2 2)))
      )
    )
  (princ)
)
```

Die dargestellte Markierung wird ebenfalls erreicht, wenn der Cursor vor der sich öffnenden oder hinter die sich schließenden Klammer positioniert ist und **STRG+SHIFT+[** oder **STRG+SHIFT+]** verwendet wird.

Um sich an nur an das Ende oder an den Anfang eines Ausdruckes zu bewegen wird ebenso verfahren, nur dass **STRG+[** oder **STRG+]** zu verwenden ist.

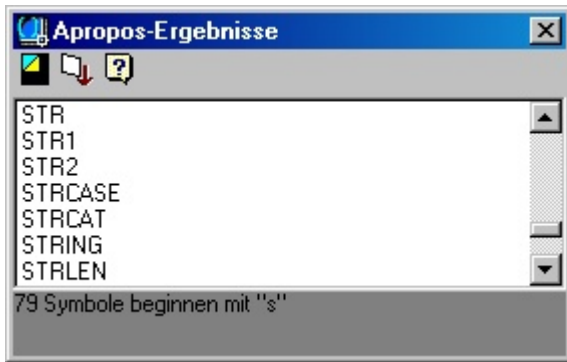
Durch diese Möglichkeiten lassen sich schnell Fehler bezüglich fehlender oder überzähliger Klammern finden.

#### 1.3.4.2 Automatische Ergänzung von Code

Wenn Code geschrieben wird gibt es die Möglichkeit, dass die Zeichenfolge anhand der ersten eingetippten Zeichen automatisch erkannt wird. Es werden alle bereits verwendeten Wörter im Code, deren Beginn der eingetippten Zeichenfolge entspricht in der Reihenfolge der letzten Verwendung vorgeschlagen.

Unter den Beispielcode wird ein „s“ eingegeben, über **STRG+SPACE** wird das Wort in folgender Reihenfolge komplettiert: „**strcat**“ „**setq**“ „**Sie**“ „**s**“. Durch erneutes verwenden von **STRG+SPACE** werden die Wortvorschläge in immer der gleichen Reihenfolge gemacht.

Als weitere Möglichkeit der automatischen Ergänzung kann dieses Wort aus einem Symbolsatz ergänzt werden. Es wird wieder das „s“ am Ende des Codes eingegeben und **STRG+SHIFT+SPACE** verwendet. Es öffnet sich das Fenster **Apropos-Ergebnisse**:





Es werden alle zur Verfügung stehenden Funktionen angezeigt, dem die mit „s“ beginnen. Nach einer Auswahl einer Funktion aus der Liste kann diese über das Kontextmenü kopiert und im Code eingefügt werden.


### 1.3.4.3 Suchfunktionen und Auswahl der Position im Code

Um bestimmte Zeichenfolgen innerhalb des Codes zu suchen, zu ersetzen oder Lesezeichen zu setzen wird der folgende Werkzeugkasten verwendet




 Es wird das **Suchfenster** geöffnet in dem Optionen bezüglich der Suche bestimmter Zeichenfolgen innerhalb des Codes eingegeben werden können.

 Es wird das **Suchen-Ersetzenfenster** geöffnet. Wenn mehrere gleiche Zeichenfolgen innerhalb des Codes durch eine neue Zeichenfolge ersetzt werden müssen sollte dieses Fenster verwendet werden. Dadurch werden größere Ersetzungen schnell und frei von Tippfehlern durchgeführt.


 Es wird die **Schnellsuche** nach dem oberen Eintrag im links liegendem Pulldownmenü innerhalb des Codes gestartet, der Startpunkt der Suche ist die aktuelle Cursor-Position.

 Es wird ein **Lesezeichen** in der Zeile des aktiven Cursors ein- oder ausgeschaltet.

 Sie können sich von Lesezeichen zu Lesezeichen bewegen.

 Es werden alle Lesezeichen gelöscht.

Über **→ Ansicht → Gehe zu Zeile...** oder **STRG+G** kann eine Zeilennummer angesteuert werden.

 Über das Kontextmenü bei einem Rechtsklick mit der Maus an eine beliebige Stelle im Code kann **Zur letzten Arbeitsposition wählen** gewählt werden. Das ist sinnvoll, wenn man sich z.B. aktiv in Zeile 10 befindet. Um den Code zu erweitern soll z.B. eine Textfolge aus Zeile 500 kopiert werden um es in den aktiven Bereich einzufügen. Über **STRG+G** geht man nach Zeile 500 und kopiert den gewünschten Bereich, über das Kontextmenü **Zur letzten Arbeitsposition** wieder zu der ursprünglichen Position in Zeile 10. Dort kann der kopierte Bereich eingefügt werden.

Als letzte Arbeitsposition gilt immer das zuletzt eingegebenen oder geänderte Zeichen im Code.