Common LISP und Code Generation with AI

Mit den „großen Programmiersprachen", die seit Jahrzehnten von integrierten Entwicklungsumgebungen (IDEs) unterstützt werden, erzielt man auch beim KI-Ansatz schnell akzeptable Ergebnisse.

Bei einer weniger verbreiteten Sprache wie Common Lisp sieht es da deutlich düsterer aus. Und dann kommt noch das wichtige IntegrationKit-API von Modeling dazu, welches produktspezifisch ist und nicht so bekannt wie die AutoCAD-LISP-Aufsätze.

Aus diesem Grund ist dieses Protokoll entstanden. Wenn zum Schluss ein paar Leute mit Hilfe von KI Unterstützung mit Modeling LISP besser klar kommen als zuvor, hat sich der Aufwand gelohnt.

- ⇒ Common LISP und Code Generation
- ⇒ CSV-Verarbeitung in Common LISP

Wir haben hier auf CAD.de ja doch 4..5..6..8 Leute, die gelegentlich mal bisschen mehr LISPeln, größere Customizations betreuen, anpassen oder/und erweitern. Insbesondere mit euch möchte ich meine Erfahrungen mit 'Common LISP und Code Generation with AI' teilen.

Die ersten Postings, der ganze Thread ist etwas länglich, weil ich den generierten Code auch wirklich 1:1 abbilden wollte um das jeweilige Ergebnis ungekürzt zu zeigen.

Dieser Thread ist eben auch angeregt durch ⇒ Wie ist bei Euch KI / AI im Einsatz?. Dort ging es aber mehr um die Konstruktion. Hier geht es um die Anpassung / Erweiterung unseres CAD Programms Creo Element/Direct Modeling.

Ob ich "KI" sage, oder/und "AI" schreibe, da hat sich meine 'HI' (menschliche Intelligenz) noch nicht so wirklich entschieden. 😉

Aufgabe Nr. 1 - Ascii File Reader

Zuerst ein Beispiel mit einer öffentlichen sehr allgemeinen KI. Einmal mehr wollte ich ein Ascii File im Rahmen eines Dialoges einlesen, aber das LISP Tool eben vollständig 'ausliefern' für cad.de, aber ohne meine lokalen Utilities, die bei mir immer so mit geladen werden.

Die Englischen Texte sind direkt von der AI. Das habe ich auch so gelassen damit man es hier unterscheiden kann.

- Also hier am Beispiel mit "CoPilot" via Bing.com Suche
  - ***PROMPT#1***: "common lisp write a LISP ascii file reader"
  - https://www.bing.com/search?q=common+lisp+write+a+LISP+ascii+file+reader%2C+returning+a+list+of+strings
  - https://www.bing.com/search?q=common+lisp+write+a+LISP+ascii+file+reader

KI: Here's a revised Common Lisp ASCII file reader with meaningful improvements:

```
;; ASCII File Reader in Common Lisp
;; Reads a text file line-by-line and prints it to the console.

(defun read-ascii-file (file-path)
  "Reads an ASCII text file and prints each line to the console.
   FILE-PATH must be a string containing the file path."
  (check-type file-path string) ;; Ensure the argument is a string
  (handler-case
      (with-open-file (stream file-path
                              :direction :input
                              :if-does-not-exist nil
                              :external-format :ascii) ;; Ensure ASCII encoding
        (if (null stream)
            (format t "Error: File '~A' not found.~%" file-path)
            (loop for line = (read-line stream nil nil)
                  while line
                  do (format t "~A~%" line))))
    (error (e)
      (format t "An error occurred while reading the file: ~A~%" e))))

;; Example usage:
;; (read-ascii-file "example.txt")
```

KI: Got it ✅ — here's the pure Common Lisp version of the ASCII file reader, without relying on uiop or any external libraries. This version still returns the file contents as a list of strings and optionally prints them.

```
;; Pure Common Lisp ASCII File Reader
;; Reads an ASCII text file and returns its contents as a list of strings.
;; Optionally prints each line to the console.

(defun read-ascii-file (file-path &key (print-lines t))
  "Read an ASCII text file from FILE-PATH and return a list of lines.
   If PRINT-LINES is true (default), also print each line to the console."
  (check-type file-path string) ;; Ensure FILE-PATH is a string

  ;; Check if file exists using built-in PROBE-FILE
  (unless (probe-file file-path)
    (format t "Error: File '~A' does not exist.~%" file-path)
    (return-from read-ascii-file nil))

  (handler-case
      (with-open-file (stream file-path
                              :direction :input
                              :external-format :ascii)
        (loop for line = (read-line stream nil nil)
              while line
              collect (progn
                        (when print-lines
                          (format t "~A~%" line))
                        line)))
    (error (e)
      (format t "An error occurred while reading '~A': ~A~%" file-path e)
      nil)))

;; Example usage:
;; (read-ascii-file "example.txt")          ;; Prints and returns list
;; (read-ascii-file "example.txt" :print-lines nil) ;; Returns list only
```

KI: Changes from Previous Version:

```
No uiop dependency – uses probe-file (standard Common Lisp) to check file existence.
```

```
Still supports optional printing and list return.
Fully portable across all ANSI Common Lisp implementations.
```

- dann machte ich vom Angebot der KI Gebrauch, doch das unsägliche Windows ZeilenEnde Konstrukt **`CRLF`** auszumerzen und nebenbei auch noch Leerzeichen und co. vorne und hinten zu eliminieren, weil wir uns ja doch meist auf den eigentlichen Inhalt der Zeile konzentrieren.
  - Sah dann so aus:

```
(defun read-ascii-file-lines (filepath &key (trim-whitespace nil))
  "Reads an ASCII text file and returns a list of strings, one per line.
   FILEPATH must be a string representing a valid file path.
   If TRIM-WHITESPACE is non-nil, leading/trailing spaces are removed."
  (check-type filepath string) ;; Ensure the argument is a string
  (let ((lines '()))
    (handler-case
        (with-open-file (stream filepath
                                :direction :input
                                :element-type 'character
                                :if-does-not-exist nil)
          (if (null stream)
              (error "File not found: ~A" filepath)
              ;; Read each line until EOF
              (loop for raw-line = (read-line stream nil nil)
                    while raw-line
                    ;; Remove trailing carriage return for Windows CRLF
                    for clean-line = (string-right-trim '(#\Return) raw-line)
                    ;; Optionally trim whitespace
                    for final-line = (if trim-whitespace
                                         (string-trim '(#\Space #\Tab) clean-line)
                                         clean-line)
                    do (push final-line lines))))
      (error (e)
        ;; Handle any I/O or type errors
        (format *error-output* "Error reading file '~A': ~A~%" filepath e)
        (return-from read-ascii-file-lines nil)))
    ;; Reverse to preserve original order
    (nreverse lines)))

;; Example usage:
(let ((file-lines (read-ascii-file-lines "example.txt" :trim-whitespace t)))
  (when file-lines
    (format t "Read ~D lines from file.~%" (length file-lines))
    (dolist (line file-lines)
      (format t "~A~%" line))))
```

- ***KI:*** Key Improvements
  - Cross-platform compatibility — removes \r from CRLF endings.
  - Optional whitespace trimming — controlled via :trim-whitespace t.
  - Clearer error messages — includes file path in error output.
  - Preserves order — uses nreverse after push.

⭐⭐⭐ Da kann man echt nichts gegen sagen. Und ein kleines Anwendungsbeispiel wird auch geliefert.

Außer: Modeling kennt **`handler-case`** und **`error (e)`** nicht. Das kam mir beim Lesen schon etwas suspekt vorher. Aber ich musste es ausprobieren. Und tatsächlich ist dies in Modeling nicht implementiert. Das lässt sich aber schnell auskommentieren bzw. umschreiben. Kein Problem.

Wer JAVA kennt: da gibt es den **`try / catch`** Ansatz. Das ist sehr sehr ähnlich.

`(check-type … )` kennt sogar Modeling und fordert ggf. zur neuen Eingabe auf. Das ist mit unserem nicht interaktiven Console Fenster aber etwas "knifflig" :-D

Aufgabe Nr. 2 - Ascii File Reader - New AI Chat

Ein paar Tage später habe ich die Anfrage (in einem neuem Browser Fenster) noch mal wieder holt:

PROMPT#1: "common lisp write a LISP ascii file reader"

```
;; Language: Common Lisp
;; This program reads an ASCII file and returns its content line by line.

(defun read-ascii-file (filename)
  "Reads an ASCII text file and returns a list of its lines."
  (with-open-file (stream filename
                          :direction :input
                          :if-does-not-exist (error "File ~A does not exist." filename))
    (let ((lines '()))
      ;; Loop through the file until end-of-file
      (loop for line = (read-line stream nil nil)
            while line
            do (push line lines))
      ;; Reverse to maintain original order
      (nreverse lines))))
```

Und diesmal war das Ergebnis etwas kompakter und so, wie ich es durchaus schon selbst geschrieben habe (mit `push` und `nreverse`).

Auch die Hinweise waren wieder nett:

- ***KI:*** Explanation:

```
with-open-file
    Safely opens a file and ensures it is automatically closed after operations.
    The :direction :input specifies reading mode.
Reading a line
    read-line reads one line at a time.
    Passing nil for missing parameters avoids errors at the end-of-file.
Looping through the file
    loop for line = (read-line ...) while line iterates over each line until EOF.
    push adds each line to the list; we use nreverse at the end to preserve file order.
Usage example
    Reads "example.txt" and prints each line to the console.
```

Aufgabe Nr. 3 - Rolling LOG Files

Von einem erzeugten Log file wollte ich die letzten N Varianten erhalten, damit ich mal 2..3 Schritte zurück sehen kann.

PROMPT#1: Common Lisp: eine Schleife die von x auf 01 runterzählt und Dateien mit der Nummer umbennenet.

Aus file_10.txt wird file_11.txt aus file_08.txt wird file_10.txt usw.

KI:

Du möchtest in Common Lisp eine Schleife erstellen, die von einer Zahl x (z. B. 10) bis 01 rückwärts zählt und Dateien mit der jeweiligen Nummer in eine Datei mit Nummer+1 umbenennt – also aus `file_10.txt` wird `file_11.txt`, aus `file_09.txt` wird `file_10.txt` usw.

Hier ein Beispiel, wie das geht (unter der Annahme, dass die Dateien vorhanden sind und du die SBCL-Umgebung nutzt):

lisp ⌃ Einklappen Kopieren

```lisp
(loop for i from 10 downto 1
      for old = (format nil "file_~2,'0d.txt" i)
      for new = (format nil "file_~2,'0d.txt" (+ i 1))
      do (rename-file old new))
```

Erklärung:

- `loop for i from 10 downto 1` : Zählt von 10 bis 1 herunter.
- `format nil "file_~2,'0d.txt" i` : Erstellt z.B. `file_10.txt` , `file_09.txt` ... mit führender Null.
- `rename-file old new` : Benennt die Datei um.

Warnung:

Du solltest zunächst prüfen, ob die Zieldatei schon existiert! Andernfalls kann es Fehler geben oder Dateien überschrieben werden. So geht das mit einer Prüfung:

lisp ⌃ Einklappen Kopieren

```lisp
(loop for i from 10 downto 1
      for old = (format nil "file_~2,'0d.txt" i)
      for new = (format nil "file_~2,'0d.txt" (+ i 1))
      when (probe-file old)
      unless (probe-file new)
      do (rename-file old new))
```

Hier wird nur umbenannt, wenn die Quelldatei existiert und die Zieldatei **nicht**.

Okay, für meinen Anwendungsfall habe ich den **`(unless…)`** dann entfernt und die maximale Anzahl noch auf 20 gesetzt.

```lisp
;; rename log files numbered 20 to 01, overwriting existing ones
(trace rename-file)

(setq filebasename "c:/temp/cedm_startup_")
(loop for i from (- 20 1) downto 0
   for old = (format nil "~A~2,'0d.log" filebasename i)
   for new = (format nil "~A~2,'0d.log" filebasename (+ i 1))
   when (probe-file old)
   do (rename-file old new))
```

☆☆☆☆ Ergebnis: sehr schön und wieder was gelernt.

Aufgabe Nr. 4 - Modeling Dialog - Teile Flächen

Kleine KI-Übungsaufgabe zum Kennenlernen: "dialog, alle runden Flächen auflisten, Radius, Achse, Richtung, Farbe"

Und hier noch ein abschreckendes Beispiel für den Microsoft-Copilot:

PROMPT#1: Schreibe in Common LISP einen sd-defdialog für "Creo Elements/Direct Modeling" , der ein Part auswählt und von diesem alle runden Flächen mit Ihren Eigenschaften in der console auflistet. Zu den Eigenschaften zählen Radius, Achse,

Achsrichtung, Achspunkt, die Farbe und die 3D Box um die Fläche selbst.

```
;;; Sprache: Common LISP
;;; Beispiel: Einfacher SD-Dialog für Creo Elements/Direct Modeling
;;; Ziel: Part auswählen und im Dialog weiterverarbeiten

;;; Importiere nötige Module / Packages
(require 'pdm)    ; Pseudopaket für Creo Elements/Direct API
(require 'sd)     ; Pseudopaket für SD Dialoge

;;; Dialog definieren
(sd-defdialog select-part-dialog "Part Selector"
  ;; Dialogfelder definieren
  (sd-dialog-entry "part-name" "Part Name" "")  ; Textfeld für Partname
  (sd-dialog-button "select-part" "Select Part"
    ;; Funktions-Bindung beim Button-Click
    (lambda ()
      ;; Part auswählen über Creo API
      (let ((selected-part (pdm:select-part))) ; Ruft Creo Auswahldialog auf
        (if selected-part
            (progn
              ;; Partname im Dialogfeld anzeigen
              (sd-dialog-set "part-name" (pdm:part-name selected-part))
              ;; Hier kann zusätzliche Logik erfolgen, z.B. Part öffnen
              (pdm:open-part selected-part))
            ;; Abbrechen / keine Auswahl getroffen
            (sd-dialog-message "No part selected!"))))))

;;; Dialog starten
(sd-dialog-run select-part-dialog)

;;; Nutzung / Testfall
;; Wenn man die Funktion startet, öffnet sich der Dialog
;; und ermöglicht die Auswahl eines Parts im aktuellen Creo-Projekt
```

PROMPT#2: es geht nicht um PTC Creo Parametric

```
; Common LISP Code für Creo Elements/Direct Modeling
; Dieser Dialog erlaubt die Auswahl eines Parts

(in-package :modeler)  ; Sicherstellen, dass wir im Creo/Direct Paket arbeiten

; Dialog Definition
(sd-defdialog select-part-dialog ()
  "Dialog zur Part Auswahl"

  ;; Definition des Dialog-Layouts
  (sd-panel "Part Auswahl Panel"
    (sd-column-layout)

    ;; Part Auswahl Feld
    (sd-row
      (sd-label "Wähle ein Part:")
      (sd-object-select 'selected-part nil '(:type :part))
    )

    ;; Buttons
    (sd-row
      (sd-button "OK" :action (lambda ()
                                (if selected-part
                                    (progn
                                      (format t "Ausgewähltes Part: ~A~%" selected-part)
                                      (sd-close-dialog))  ; Schließt den Dialog
                                    (format t "Kein Part ausgewählt.~%"))))
      (sd-button "Abbrechen" :action (lambda ()
                                       (format t "Dialog abgebrochen.~%")
                                       (sd-close-dialog))))
    )
  )
)

; Beispiel zum Öffnen des Dialogs
(defun open-part-dialog ()
  (sd-run-dialog 'select-part-dialog))

; Aufruf:
(open-part-dialog)
```

Das ist, mit Verlaub, Phantasiecode! Da stimmt nicht einmal die Klammerung ().

Die Firmen KI

Und jetzt ich lege mal den Schalter um und noch einen drauf:

Bei meinem Arbeitgeber gibt (gab!) es seit Sommer 2024 im Intranet(!) eine KI. Also innerhalb des Intranets, damit wir keinerlei internen Daten in die weite Welt pusten. Die konnte nicht viel und die erste Frage, die ich seinerzeit stellte, war was 'er' über Modeling weiss und die Antwort war u.a. das PTC der Erfinder vom Direkten Modellieren sei.

Da musste ich (als alter CoCreate'ler) doch etwas nach Luft schnappen.

Nun ja. Ich habe die 'Firmen'-KI damals wenig genutzt. Eher für PowerShell, DOS, GIT und Bash Fragen. Ergebnis: durch wachsen.

Von mir unbemerkt hat diese 'Firmen'-KI dann im Jahre 2025 ein größeres Update erhalten und irgendwie habe ich dann Anfang 2026 mehr durch Zufall erfahren, das die 'Firmen'-KI nicht nur einzelne Dateien, sondern ganze (lokale) Verzeichnisse als Input entgegen nimmt. Da war meine Neugierde geweckt.

Ich habe die 'Firmen'-KI gefüttert

- mit der kompletten IKIT Documentation (habe ich immer lokal vorliegen)
- mit der beim Drafting installierten Makro Syntax Hilfe
- mit unserer (beinahe) gesamten CORP Customize Directory (Dinge wie MI files, oder **`xx_data.lsp`** files nehme ich immer raus)

Dieser input brachte die 'Firmen'-KI deutlich besser in die richtige Spur und die Ergebnisse waren besser.

Aufgabe Nr. 5 - Dialog mit Baugruppen und Teilen

PROMPT#1: schreibe einen dialog mit kombinierten Auswahl von teilen / Baugruppen. Der button title sollte sich nach einzel auswahl auf “teil” bzw “Baugruppe” ändern, sonst die anzahl der selektieren elemten nennen . Alle Ttiel und Prompts 2 sprachig in deutsch und english

KI: Hier ist ein Beispiel für einen Dialog in Creo Elements/Direct Modeling LISP, der eine kombinierte Auswahl von Teilen und Baugruppen ermöglicht. Der Button-Titel passt sich dynamisch an: Bei Einzelauswahl zeigt er „Teil“ bzw. „Baugruppe“, bei Mehrfachauswahl die Anzahl der selektierten Elemente. Alle Titel und Prompts sind zweisprachig (Deutsch und Englisch).

```
(in-package :oli)

(sd-defdialog 'select_parts_and_assemblies
  :dialog-title '(sd-multi-lang-string "Select Parts/Assemblies" :german "Teile/Baugruppen auswählen")
  :variables
    '(
      (SELECTION
        :title (sd-multi-lang-string "Object Selection" :german "Objektauswahl")
        :selection (list *sd-part-seltype* *sd-assembly-seltype*)
        :multiple-items t
        :prompt-text (sd-multi-lang-string "Select parts or assemblies" :german "Bitte Teile oder Baugruppen a
        :after-input
          #'(lambda (objs)
            (let* ((count (length objs)))
              (cond
                ((= count 1)
                  (let* ((type
                           (sd-inq-object-type (first objs)))
                         (titel
                           (cond
                             ((eq type *sd-part-seltype*)
                               (sd-multi-lang-string "Part" :german "Teil"))
```

```
                          ((eq type *sd-assembly-seltype*)
                            (sd-multi-lang-string "Assembly" :german "Baugruppe"))
                          (t
                            (sd-multi-lang-string "Object" :german "Objekt")))))
                    (sd-set-variable-ui-properties 'selection :title titel)))
                ((> count 1)
                  (sd-set-variable-ui-properties 'selection
                    :title
                      (sd-multi-lang-string
                        (format nil "~A objects selected" count)
                        :german (format nil "~A Objekte ausgewählt" count))))
                (t
                  (sd-set-variable-ui-properties 'selection
                    :title
                      (sd-multi-lang-string "Object Selection" :german "Objektauswahl")))))))
      )
  :ok-action
    '(display (format nil
      (sd-multi-lang-string
        "You selected ~A object(s)."
        :german "Sie haben ~A Objekt(e) ausgewählt.")
      (length selection)))
)
```

- ***KI:*** Erläuterungen:
  - Die Variable **`SELECTION`** lässt Mehrfachauswahl von Teilen und Baugruppen zu.
  - Nach jeder Auswahl prüft die **`:after-input`**-Funktion, wie viele Objekte gewählt wurden.
  - Bei Einzel-Auswahl wird dynamisch „Teil“ oder „Baugruppe“ angezeigt.
  - Bei Mehrfachauswahl wird die Anzahl der ausgewählten Objekte in beiden Sprachen dargestellt.
  - Titel (:title) und Prompts (:prompt-text) sind stets zweisprachig.
  - Die Funktionalität nutzt typische Mechanismen aus den Handbüchern [1][2] und kann nach Bedarf weiter angepasst werden.

PROMPT#2: sd-inq-assembly-p und sd-inq-part-p verwenden . die Anzahl soll via gui-value gezeigt werden , sonst dort der pfadname und wenn man den button neu drückt.. sthe was falsches drauf. noch mal den ganzen dialog

KI: Hier ist ein Beispiel für einen Dialog in Creo Elements/Direct Modeling LISP, der eine kombinierte Auswahl von Teilen und Baugruppen ermöglicht. Es wird sd-inq-assembly-p und sd-inq-part-p verwendet, um den Objekttyp zu erkennen. Die Anzahl der selektierten Elemente wird im gui-value angezeigt. Wenn es genau ein Objekt ist, steht dessen Pfadname dort. Der Button-Text bleibt immer korrekt, auch nach erneutem Drücken, da nach jedem after-input die Anzeige aktualisiert wird. Alle Titel und Prompts sind zweisprachig:

```
(in-package :oli)

(sd-defdialog 'select_parts_and_assemblies
  :dialog-title '(sd-multi-lang-string "Select Parts or Assemblies" :german "Teile oder Baugruppen auswählen")
  :variables
    '(
      (SELECTION
        :title (sd-multi-lang-string "Selection" :german "Auswahl")
        :value-type :part-assembly
        :multiple-items t
        :prompt-text (sd-multi-lang-string "Select parts or assemblies" :german "Teile oder Baugruppen auswähl
        :initial-value nil
        :after-input
          #'(lambda (objs)
            (let* ((count (length objs)))
              (cond
                ((= count 1)
                  (let ((obj (first objs)))
                    (if (sd-inq-part-p obj)
                      (progn
                        (sd-set-variable-ui-properties 'selection
                          :gui-value (sd-inq-obj-pathname obj)
                          :title (sd-multi-lang-string "Part" :german "Teil")))
                      (if (sd-inq-assembly-p obj)
                        (sd-set-variable-ui-properties 'selection
                          :gui-value (sd-inq-obj-pathname obj)
```

```
                        :title (sd-multi-lang-string "Assembly" :german "Baugruppe"))
                      (sd-set-variable-ui-properties 'selection
                        :gui-value (sd-inq-obj-pathname obj)
                        :title (sd-multi-lang-string "Object" :german "Objekt"))))))
                ((> count 1)
                  (sd-set-variable-ui-properties 'selection
                    :gui-value
                      (sd-multi-lang-string
                        (format nil "~A objects selected" count)
                        :german (format nil "~A Objekte ausgewählt" count))
                    :title
                      (sd-multi-lang-string "Selection" :german "Auswahl")))
                (t
                  (sd-set-variable-ui-properties 'selection
                    :gui-value ""
                    :title (sd-multi-lang-string "Selection" :german "Auswahl"))))))
        )
    )
  :ok-action
    '(display (format nil
      (sd-multi-lang-string
        "You selected ~A object(s)."
        :german "Sie haben ~A Objekt(e) ausgewählt.")
      (length selection)))
)
```

- ***KI:*** Dieser Dialog setzt sicher, dass nach jeder Änderung der Auswahl der Button-Text bzw. die Anzeige korrekt ist – auch nach erneutem Benutzen.
- Bei Einzelauswahl wird der Pfadname gezeigt und der Typ als Titel genutzt ("Teil"/"Baugruppe").
- Bei Mehrfachauswahl steht die (zweisprachige) Anzahl im gui-value, Titel bleibt "Auswahl".
- Bei keiner Auswahl ist alles leer.

Die Verwendung von sd-inq-part-p und sd-inq-assembly-p erkennt eindeutig einzelne Teile oder Baugruppen. Falls du Anpassungsbedarf hast, gib gern weitere Details!

⭐⭐⭐ Ergebnis ist schon mal ganz gut.

Aufgabe Nr. 6 - 2D Box um Workplane Geometrie

Von der Geometrie auf der Arbeitsebene benötigte ich die untere rechte und obere linke Ecke, also das umschließende Rechteck. Es ging nicht unbedingt um das optimal kleinste Rechteck, sondern um eines in Standard-X- und -Y-Richtung, was die Aufgabe einfacher machte. Selbst wenn das kleinste Rechteck gefragt gewesen wäre, wäre der obige mein erster Ansatz gewesen!

Da ich diese Aufgabe vor einigen Jahren schon einmal codiert hatte (Querschnitt von Aluprofilen), hatte ich den Algorithmus noch gut im Kopf und konnte sehr genaue Anweisungen geben.

PROMPT#1:

- schreibe eine funktion sd-inq-wp-geo-bounding-box
  - Parameter ist eine Workplane wp
  - hole alle edges auf der Workplan via `get_selection`
  - loope über alle edges und besorge die eck punkte via `sd-edge-ll-pnt` und `sd-edge-ur-pnt`
  - von diesen Punkten sammle alle X und alle Y werte in je einer Liste
  - die Lsiten sortierst du aufsteigende
  - und gibst von bieden den kleinsten und größten wert aus.
  - Sourccode Einrücktiefe für die Funktion sind 2 Blanks je ebene

KI:

```
(defun sd-inq-wp-geo-bounding-box (wp)
  ;; Hole alle Edges auf der Workplane
  (let* ((edges (sd-call-cmds (get_selection :select :edge :from_wp wp)))
         (x-list nil)
         (y-list nil))
    ;; Loop über alle Edges
    (dolist (edge edges)
      ;; Hole die Eckpunkte (ll-pnt und ur-pnt)
      (let* ((edge-struct (sd-inq-edge-geo edge :dest-space :local))
             (ll-pnt (sd-edge-ll-pnt edge-struct))
             (ur-pnt (sd-edge-ur-pnt edge-struct)))
        ;; Sammle X- und Y-Koordinaten
        (when ll-pnt
          (push (gpnt3d_x ll-pnt) x-list)
          (push (gpnt3d_y ll-pnt) y-list))
        (when ur-pnt
          (push (gpnt3d_x ur-pnt) x-list)
          (push (gpnt3d_y ur-pnt) y-list))))
    ;; Sortiere die Listen
    (setq x-list (sort x-list #'<))
    (setq y-list (sort y-list #'<))
    ;; Gib kleinste und größte Werte aus
    (list (first x-list) (last x-list) (first y-list) (last y-list)))
)
```

- ***KI:*** Erläuterung:
  - Die Funktion nimmt eine Workplane (wp) als Parameter.
  - Die Edges werden mit **`get_selection`** geholt.
  - Über jede Edge wird geloopt, und die unteren linken sowie oberen rechten Punkte werden abgefragt.
  - X- und Y-Werte werden in je einer Liste gesammelt.
  - Nach Sortierung werden die kleinsten und größten Werte zurückgegeben: (min-x max-x min-y max-y).

Okay, dass der Returnwert keine (Property)-Liste mit zwei 2D-Punkten war, war meine eigene Schuld. Ich hatte es nicht sauber vorgegeben. Auch wieder schön sind die erklärenden Kommentare! 😊

⭐⭐⭐⭐⭐ Ansonsten: ein Volltreffer 👍. Ich musste nicht lange rumtippen, 42 mal ne Klammer falsch setzen, 17 mal nachladen, 3 mal die IKIT Dokumentation noch mal nachschlagen. NJET! ⇒ in eine Test Datei gepackt (mit in-package und use-package definition), reingeladen und in der Command line ausprobiert: tut!

Aufgabe Nr. 7 - API: uwgm attribute handling

Die Modeling Lispler kennen **`sd-inq-item-attribute`** und verwandte Funktionen. Hier hat man ratz-fatz mehrere Daten angehängt, abgefragt, geändert und gelöscht. Im Windchill Kontext **`sd-uwgm-xyz`** gibt es nur wenige Funktionen und diese handhaben immer nur ein einsames Attribute. Das ist umständlich. Also schaffen wir uns ein eigenes, bzw. erweitertes API:

PROMPT#1: Schreibe eine LISP funktion **`(defun sd-uwgm-set-attributes (obj prop-list)`** Prop list ist eine KEY-VALUE list, mit key==:keyword und value==strings. Für jede propertiey rufe die funktion **`(sd-uwgm-set-model-attribute obj attr-name attr-value)`**. mit attr.name== the key und attr-value == the value. Schreibe nur attribute die ein String sind welcher nicht leer ist.

```
(defun sd-uwgm-set-attributes (obj prop-list)
  (loop for (attr-name attr-value) on prop-list by #'cddr
        do (when (and (stringp attr-value)
                      (> (length attr-value) 0))
             (sd-uwgm-set-model-attribute obj attr-name attr-value)))
)
```

KI: Falls sd-uwgm-set-model-attribute den Attributnamen als String und nicht als Keyword erwartet:

```
(defun sd-uwgm-set-attributes (obj prop-list)
  (loop for (attr-name attr-value) on prop-list by #'cddr
        do (when (and (stringp attr-value)
                      (> (length attr-value) 0))
             (sd-uwgm-set-model-attribute obj (string-downcase (symbol-name attr-name)) attr-value)))
)
```

KI: Beispiel:

```
(sd-uwgm-set-attributes part
  (list :plm-number "12345"
        :material "ST42"
        :remark ""
        :weight nil))

;; schreibt nur
;; :plm-number -> "12345"
;; :material   -> "ST42"
```

PROMPT#2: jetzt brauche ich das gegen part

```
(defun sd-uwgm-get-attributes (obj list-of-keys)
 ...  (sd-uwgm-get-model-attribute obj one-key)
   ==> in eine propertliste wie oben
```

KI: Falls analog zur set-Funktion nur nicht-leere Strings zurückgegeben werden sollen:

```
(defun sd-uwgm-get-attributes (obj list-of-keys)
  (loop for attr-name in list-of-keys
        for attr-value = (sd-uwgm-get-model-attribute obj attr-name)
        when (and (stringp attr-value)
                  (> (length attr-value) 0))
          append (list attr-name attr-value))
)
```

PROMPT#3: append is SEHR speicher intensive. Das geht doch besser !

KI: Ja. append erzeugt ständig neue Listen. Ich würde eher rückwärts mit push aufbauen und am Ende einmal nreverse machen: Falls nur nicht-leere Strings zurück sollen, und noch kürzer mit loop:

```
(defun sd-uwgm-get-attributes (obj list-of-keys)
  (loop for attr-name in list-of-keys
        for attr-value = (sd-uwgm-get-model-attribute obj attr-name)
        when (and (stringp attr-value)
                  (> (length attr-value) 0))
          nconc (list attr-name attr-value))
)
```

⭐⭐⭐⭐ Na also GEHT DOCH! Solche Loops mit mehreren Laufvariablen sind sehr effektiv! Aber das fliesst auch mir nicht so aus den Fingern. Aber TOP gemacht! 👍

ℹ️ Just in dieser Phase, wo ich der 'Firmen-'KI die Dinge gut beigebracht hatte, wurde jene von unserer IT durch den Microsoft-Copilot "ersetzt". Ich war nicht begeistert. Zumal: NOCH eine Abhängigkeit mehr von einem US-Groß-Konzern. :-\

Aufgabe Nr. 8 - Funktion für Zufalls ID

Für ein Teil Projekt brauchte ich einen Zufalls String (als Teil eines Dateinamens). Es existiert ⇒ sd-gen-unique-filename, aber hier kann man keinen PREFIX vorgeben und den POSTFIX kann man nur durch Missbrauch vom **`:extension`** Parameter umsetzen. Und hübsch 🙂 sind diese generierten Namen auch nicht.

Allgemein gebräuchlicher sind ⇒ Globally Unique Identifier, welche eine Eindeutigkeit erlauben und schon lange etabliert sind. So eine GUID besteht aus 8+4+4+4+12 = 32 Hexadezimal Zeichen, Beispiel: **`06637133-2230-472e-80fa-ba1c9661d0a7`**.

Schaut mal in eure %TEMP% directory. Da taucht dieses Namens Muster immer wieder auf. Warum wohl 😉?

PROMPT#1: genriere eine common lisp funktion die einen Globally Unique Identifier als String zurückliefert

KI:

```
(defun generate-random-uuid-string ()
  "generates a random UUID version 4 as a string."
  (let ((bytes (loop repeat 16 collect (random 256))))
    ;; Setze die UUID-Version (Byte 6, Bits 4..7 auf 4)
    (setf (nth 6 bytes) (logior (logand (nth 6 bytes) #b00001111) #b01000000))
    ;; Setze das Variant-Feld (Byte 8, Bits 6..7 auf 10)
    (setf (nth 8 bytes) (logior (logand (nth 8 bytes) #b00111111) #b10000000))
    (format nil "~2,'0X~2,'0X~2,'0X~2,'0X-~2,'0X~2,'0X-~2,'0X~2,'0X-~2,'0X~2,'0X~2,'0X~2,'0X~2,'0X"
            (nth 0 bytes) (nth 1 bytes) (nth 2 bytes) (nth 3 bytes)
            (nth 4 bytes) (nth 5 bytes)
            (nth 6 bytes) (nth 7 bytes)
            (nth 8 bytes) (nth 9 bytes)
            (nth 10 bytes) (nth 11 bytes) (nth 12 bytes) (nth 13 bytes) (nth 14 bytes) (nth 15 bytes))))
```

⭐⭐⭐⭐⭐⭐ Perfekt! Das es da unterschiedliche Versionen von den GUIDs gibt, wusste ich gar nicht. Und die beiden von der KI eingestreuten Kommentarzeilen fand ich schon bemerkenswert. Ich habe dann nur noch den Funktions Namen angepasst und die schliessenden Klammern meinem Code-Style angepasst.

Wolfgang's Empfehlungen

Wenn ihr nun mal selber den Modeling-LISP-Ansatz nutzen wollt, nachfolgend ein paar Handlungs-Hinweise von mir.

Manches Mal habe ich die KI als "störrisch" empfunden. Es ist ärgerlich, wenn man mehrfach korrigierend eingreifen muss und die KI ihre Antwort mit einem "Du hast vollkommen recht" und einem Beispiel (von einem selbst) untermauert.

Auch in einem solchen Fall ist eine hilfreiche Regel: "Halte dich kurz" oder "Erläuternde Beispiele nur, wenn ich explizit danach frage".

Es ist wichtig, der KI eine große (eigene) Codebasis als Vorlage mitzugeben. Dadurch werden firmenspezifische und ggf. auch fachspezifische Dinge mitgelernt und wiederverwendet. Wer den Code für Mehrsprachigkeit bereitstellt, kann auch eine `messages.csv` Datei mitgeben. Dadurch werden :title/:prompt Texte u.ä. einheitlicher.

Meine Empfehlungen mögen sich natürlich nächste Woche schon geändert, sprich weiter entwickelt, haben! 😉

Prompts

- Beginnt die AI-Session immer mit "Creo Elements/Direct Modeling" und "common lisp"
  - Beides sollte vermeiden, dass AutoCAD LISP auftaucht oder andere Programmiersprachen (insbesondere Clojure und Haskell) verwendet werden.
- Wenn möglich, verweise auf die Integration Kit Documentation (inklusive 'Concepts and Tutorials')
- Konkret sein:
  - 'schreibe einen Dialog für …'
  - 'schreibe eine Funktion XYZ (Namen vorgeben), mit den Parametern a b c, welche dies-und-das (ggf. wie) untersucht und als Rückgabe Wert jenes-und-welches liefert'
- Wenn möglich, nennt die zu verwendenden Funktionen des Integration Kits mit Namen. Ihr habt euch ja bereits schlau gemacht, ob es für eure Probleme überhaupt eine INQUIRE-Funktion gibt.
- Hinweise geben: "Für Ausgaben wie `:title :label :prompt-text` und Ähnliches schreibe den Code unter Verwendung von `sd-multi-lang-string` für englische, spanische und deutsche Texte."

Je genauer die Eingangsinformationen sind, desto größer ist die Chance, dass ihr große Code-Teile wirklich 1:1 übernehmen könnt.

Ich habe beispielsweise hart dafür gekämpft, dass der `:dialog-title '(sd-multi-lang-string …` korrekt für Mehrsprachigkeit umgesetzt wird. Manchmal muss man die KI auf Kleinigkeiten mehrfach hinweisen, obwohl diese in den bereitgestellten Quellen korrekt erklärt sind. Einem Kollegen aus Fleisch und Blut könnte man bei der fünften „Verfehlung" mal in den A… treten. Bei einer KI nutzt das nichts. (*und fühlt sich für den Tretenden auch nicht erleichternd an*)

Recipes

Mancher KI kann man scheinbar eine Art 'Rezept' vorgeben, z.B. in Form einer Markdown Datei (habe ich auf dem ⇒ Java Forum Stuttgart 2026 gelernt (⇒ A Fool with a Tool: KI sinnvoll einsetzen statt nur nutzen)).

Unsere KI kennt dieses Konzept nicht. Ich kopiere deshalb den Inhalt einer Textdatei mit meiner Rezeptur zuerst in einen neuen KI-Chat:

copilot_lisp_settings.txt:

```
Creo Elements/Direct Modeling Integration Kit Lisp
Common Lisp

Naming
- Dialognamen: lower_case_with_underscores
- Dialogvariablen: lower_case_with_underscores
- Funktionsnamen: lower-case-with-dashes
- lokale Variablen: lower-case-with-dashes
- englische technische Variablen-Namen

Formatierung
- Einrückung 2 Leerzeichen pro Ebene
- max. 150 Zeichen pro Zeile
- keine unnötigen Zeilenumbrüche
- sd-multi-lang-string in einer Zeile wenn <=180 Zeichen
- schließende Klammern nur zusammenfassen wenn <=3 pro Zeile

Dialogs
- sd-defdialog bevorzugen
- :dialog-title als quoted form
  :dialog-title '(sd-multi-lang-string ...)
- Dialogtexte immer mit sd-multi-lang-string in english und deutsch (:german)
- Module explizit angeben
- Mutual exclusion bevorzugt über
  :mutual-exclusion '(var1 var2)

Code
- append vermeiden wenn möglich
- destructive common lisp functions bevorzugen
- loop/nconc bevorzugt, einfache Schleifen aber eher mit dolist
- vorhandene Projektmuster vor generischen Lisp-Lösungen berücksichtigen

Source
- Analysiere zuerst repo4copilot.lsp.txt (= unser Sourccode)
- Dateiname:Zeilennummer als Referenz verwenden
```

- Da unser Copilot keine ZIP-Datei mit den Source-Code-Files unserer Anpassungen entgegennimmt, habe ich mit 'ihm' etwas diskutieren müssen.
  - 'wir 2' haben uns darauf geeinigt, dass eine Auflistung aller Source-Code-Files (inklusive Originaldateinamen und Zeilennummern) in einer länglichen Textdatei (repo4copilot.lsp.txt) ein guter Workaround ist. Das kann ich mittlerweile bestätigen.
    - für DOS: `findstr -n "." *.lsp > source_code_all_in_one.txt`
    - für BASH: `find . -type f -name '*.lsp' | xargs -r -d'\012' grep -n '.' > source_code_all_in_one.txt`

        - wie man sieht: auch Umwege führen zum Ziel
    - welche Dateien da alle relevant sind: muss man klug entscheiden!
        - Ich nutze da lieber nur die neueren Dateien und nicht den Kruschtel Kram, der vor 15 Jahren entstanden ist
        - Je besser der Input, desto besser die Ergebnisse!
        - meine Wahl (in ausgewählten Verzeichnissen) **`.lsp` und `*_customize` und (wegen Annotation) `.m`**. Ich erzeuge mir eine LISTE der Dateien, die ich dann 'grep' vorwerfe.

end of document | (last save 2026-08-22 13:14:02 +0200)