```
;;; v19: LINE and 2D/3D polylines always project as connected linework.
;;; v19: preserve curve vertices and split P2A curves at rejected vertices.
;;; v19: context-safe MLeader arrow reader and explicit WCS text anchors.
;;; v19: independent Select all / Deselect all for types and options.
;;; v19: native TEXT / MTEXT / MULTILEADER copies in both directions.
;;; First MLeader arrow is projected; annotation layout and size preserved.
;;; v19: Arabic DCL uses explicit UTF-8 BOM; no legacy encoding fallback.
;;; Set LISPSYS to 1, restart Civil 3D, then APPLOAD this file.
;;; Projection math and compact two-column menu are unchanged.
;;; ================================================================
;;; AlignProfProject_v19_AR.lsp  -  Civil 3D  (النسخة العربية)
;;; Requires AutoCAD/Civil 3D 2021+ with LISPSYS = 1 (Unicode).
;;; Project objects from Alignment (plan) to Profile View and vice versa.
;;;
;;; Commands:
;;;   APMENU  - Menu dialog to run everything (start here)
;;;   A2P     - Selected plan objects  -> Profile View
;;;   A2PPICK - Click locations in plan -> Profile View (works on Xrefs,
;;;             DXF/DWG references, block nodes - anything you can snap to)
;;;   P2A     - Profile View objects -> Plan (on alignment, Z = elevation)
;;;
;;; v12: fixes points near a PI (angle point with no curve).
;;;   Civil 3D's own StationOffset can return a wrong offset near a PI
;;;   (e.g. 26 m = distance to the PREVIOUS PI, while the point is only
;;;   5 cm from the alignment). v12 no longer trusts that call: it
;;;   measures the TRUE shortest distance itself. The alignment is
;;;   sampled densely (every 0.25 m or finer, plus every PI / curve
;;;   start / end), the nearest spot is found, then refined on the real
;;;   alignment. Offset sign: + right, - left. A point near a PI gets
;;;   the PI station and its real (small) offset.
;;;   Menu option: "Exact nearest distance" (on by default). Untick it
;;;   to go back to Civil 3D's StationOffset.
;;;
;;; v11: the three Project buttons at the bottom are GONE.
;;;   * The dialog now has ONE way to run: choose the direction
;;;     (Plan -> Profile View, Plan -> Profile by clicking locations,
;;;     or Profile View -> Plan), tick your options, then press OK.
;;;   * OK closes the dialog and IMMEDIATELY asks you to select the
;;;     objects in the drawing (or to click locations for the pick
;;;     mode). No extra button press.
;;;
;;; v10: every dropdown starts with "<< Pick from drawing... >>".
;;;
;;; v9: fixes "one line goes far off" and shows WHERE offsets are measured.
;;;   * Line / polyline vertices that have no square position (outside a
;;;     bend), no elevation or an elevation far outside the profile view
;;;     grid are LEFT OUT - the line is split there instead of being
;;;     joined into a long vertical spike.
;;;   * Option: lines crossing the alignment -> only the crossing point.
;;;   * Option: draw offset check lines in plan (layer PROJ-A2P-OFFSET)
```

```
;;;     from each object's TRUE position to the alignment point used.
;;;     This shows at once if a point's label sits away from the point.
;;;
;;; v8: APMENU uses named dropdowns for alignment, profile view and
;;;     elevation profile; profile lists follow the selected alignment.
;;;     A surface dropdown also replaces the menu's surface pick prompt.
;;;     Direct A2P / A2PPICK / P2A commands still support drawing picks.
;;;
;;; v7: nothing is skipped silently any more.
;;;   * Points with no square position on the alignment are DRAWN at the
;;;     nearest station on layer PROJ-A2P-CHECK (yellow), label "CHECK".
;;;   * At the end a message box shows how many were projected / skipped
;;;     and WHY (also if the command stopped with an error).
;;;
;;; v6 fix for "two points get the SAME station, one with a big offset":
;;;   Civil 3D's StationOffset gives the NEAREST station when a point has
;;;   no square (perpendicular) position on the alignment - i.e. it lies
;;;   past the alignment's start/end, or on the outside of a sharp bend
;;;   (angle point with no curve). The "offset" is then just the straight
;;;   distance to that end/corner. v6 checks every result:
;;;   * past start/end -> station is extended straight along the end
;;;     tangent (option "Extend alignment straight past its start/end")
;;;   * outside a bend -> skipped with the reason (or kept if
;;;     "Accept nearest station when no square position" is ticked)
;;;
;;; v5 fixes for "point cannot be projected":
;;;   * COGO points with no elevation / Z = 0 now take elevation from a
;;;     profile line or a surface (Elevation source in the menu).
;;;   * Points whose elevation is above/below the profile view grid are
;;;     still drawn (scale is extended) instead of being skipped.
;;;   * Every skipped object is listed WITH THE REASON.
;;;   * New pick mode for points that cannot be selected (e.g. in Xrefs).
;;; Load with APPLOAD, then type APMENU.
;;; ====================================================================

(vl-load-com)

;;; ---------- settings (kept for the drawing session) ------------------
(if (not (boundp 'app:*types*))
  (setq app:*types* '("POINT" "INSERT" "TEXT" "MTEXT" "MULTILEADER" "LINE"
                      "LWPOLYLINE" "POLYLINE" "AECC_COGO_POINT" "AECC_PIPE")))
(if (not (boundp 'app:*annotation-v19*))
  (progn
    (if (not (member "MULTILEADER" app:*types*))
      (setq app:*types* (cons "MULTILEADER" app:*types*)))
    (setq app:*annotation-v19* T)))
(if (not (boundp 'app:*labels*)) (setq app:*labels* T))
(if (not (boundp 'app:*zsrc*)) (setq app:*zsrc* "z_obj"))
(if (not (boundp 'app:*extend*)) (setq app:*extend* T))
(if (not (boundp 'app:*maxoff*)) (setq app:*maxoff* ""))
```

```
(if (not (boundp 'app:*step*)) (setq app:*step* ""))
(if (not (boundp 'app:*th*)) (setq app:*th* "2.5"))
(if (not (boundp 'app:*off*)) (setq app:*off* "0"))
(if (not (boundp 'app:*shift*)) (setq app:*shift* "0"))
(if (not (boundp 'app:*eq*)) (setq app:*eq* nil))
(if (not (boundp 'app:*report*)) (setq app:*report* T))
(if (not (boundp 'app:*extaln*)) (setq app:*extaln* T))
(if (not (boundp 'app:*witness*)) (setq app:*witness* T))
(if (not (boundp 'app:*lineclip*)) (setq app:*lineclip* T))
;; Retire crossing-marker mode, including settings retained from old versions.
(setq app:*cross* nil)
(if (not (boundp 'app:*exactso*)) (setq app:*exactso* T))
(setq app:*samp* nil)
(setq app:*nearest* T app:*skips* nil)
(setq app:*sotype* "ok" app:*nchk* 0)
(setq app:*menu-run* nil app:*prof* nil app:*surf* nil app:*pvref* nil app:*extrap*
nil)
;; which direction OK runs: "A2P" | "PICK" | "P2A"
(if (not (boundp 'app:*run*)) (setq app:*run* "A2P"))

;;; ---------- small helpers -----------------------------------------
(defun app:num (s / v)
  (if (and s (/= (vl-string-trim " " s) "") (setq v (distof s 2))) v))

(defun app:shift () (cond ((app:num app:*shift*)) (0.0)))

(defun app:prop (o p / r)
  (setq r (vl-catch-all-apply 'vlax-get (list o p)))
  (if (vl-catch-all-error-p r) nil r))

(defun app:rep (s)
  (if (or (not app:*menu-run*) app:*report*) (princ (strcat "\n  " s))))

(defun app:skip (et reason)
  (if (null reason) (setq reason "unknown reason"))
  (setq app:*skips* (cons (strcat et ": " reason) app:*skips*))
  (princ (strcat "\n  SKIPPED " et ": " reason)))

;; final message box: counts + first skip reasons
(defun app:summary (title cnt skip / txt n r)
  (setq txt (strcat title "\n\nتم إسقاط: " (itoa cnt) "\nتم تخطي: " (itoa skip)) n 0)
  (if (> app:*nchk* 0)
    (setq txt (strcat txt "\n\n" (itoa app:*nchk*)
                      " drawn on layer PROJ-A2P-CHECK (no square position on the"
                      " alignment - nearest station used, please check them).")))
  (if app:*skips* (setq txt (strcat txt "\n\nReasons:")))
  (foreach r (reverse app:*skips*)
    (if (< n 8) (setq txt (strcat txt "\n- " r)))
    (setq n (1+ n)))
  (if (> n 8) (setq txt (strcat txt "\n... more in the command line (F2)")))
```

```
  (if (and (= cnt 0) (= skip 0))
    (setq txt (strcat txt "\n\nNothing was selected. Check the object type tick
boxes"
                      " in the menu (Points, Blocks, COGO points ...)")))
  (alert txt))

;; Z that means "no elevation": nil, 0, or Civil 3D "<none>" (huge value)
(defun app:noz (z) (or (null z) (equal z 0.0 1e-9) (> (abs z) 1e10)))

(defun app:ztxt (z)
  (cond ((null z) "none") ((> (abs z) 1e10) "none") (T (rtos z 2 3))))

;;; ---------- menu object catalogs ------------------------------------
(if (not (boundp 'app:*aln-h*)) (setq app:*aln-h* nil))
(if (not (boundp 'app:*view-h*)) (setq app:*view-h* nil))
(if (not (boundp 'app:*prof-h*)) (setq app:*prof-h* nil))
(if (not (boundp 'app:*surf-h*)) (setq app:*surf-h* nil))
(setq app:*alignments* nil app:*views* nil app:*profiles* nil app:*surfaces* nil)
;; Catalog entries: (display-label VLA-object handle). Handles preserve
;; choices when catalogs are refreshed. No Civil 3D version ProgID needed.
(defun app:entry (o / nm h)
  (setq nm (app:prop o 'Name) h (app:prop o 'Handle))
  (if (and h (not (vlax-object-released-p o)))
    (list (strcat (if (and (= (type nm) 'STR) (/= nm "")) nm
                      (vla-get-ObjectName o)) " [" h "]") o h)))

(defun app:catalog-sort (items)
  (vl-sort items '(lambda (a b) (< (strcase (car a)) (strcase (car b))))))

(defun app:drawing-catalog (filter / ss i o entry items)
  (if (setq ss (ssget "_X" (list (cons 0 filter))))
    (progn
      (setq i 0)
      (repeat (sslength ss)
        (setq o (vlax-ename->vla-object (ssname ss i)) i (1+ i))
        (if (setq entry (app:entry o)) (setq items (cons entry items))))))
  (app:catalog-sort items))

(defun app:same-alignment (o aln / owner id aid)
  (setq owner (app:prop o 'Alignment))
  (cond
    ((= (type owner) 'VLA-OBJECT)
     (equal (app:prop owner 'Handle) (app:prop aln 'Handle)))
    (T
     (setq id (app:prop o 'AlignmentId) aid (app:prop aln 'ObjectID))
     (and id aid (equal id aid)))))

(defun app:child-catalog (aln property filter / coll count i o entry items fallback)
  ;; Prefer the alignment's own collection: this includes profile objects
  ;; that do not have a selectable representation in the current view.
```

```
  (setq coll (app:prop aln property))
  (if (and (= (type coll) 'VLA-OBJECT)
           (numberp (setq count (app:prop coll 'Count))))
    (progn
      (setq i 0)
      (repeat count
        (setq o (vl-catch-all-apply 'vla-Item (list coll i)) i (1+ i))
        (if (and (not (vl-catch-all-error-p o)) (setq entry (app:entry o)))
          (setq items (cons entry items)))))
    ;; Only include scanned objects with a verified parent; do not guess
    ;; alignment ownership from names or station ranges.
    (foreach fallback (app:drawing-catalog filter)
      (if (app:same-alignment (cadr fallback) aln)
        (setq items (cons fallback items)))))
  (app:catalog-sort items))

;; Index 0 of every dropdown is "<< Pick from drawing... >>" (sentinel
;; handle "PICK"); the real objects follow at index 1..n.
(defun app:popup (key items previous / index n item)
  (start_list key)
  (add_list "<< Pick from drawing... >>")
  (if items (foreach item items (add_list (car item)))
    (add_list "<No matching objects in this drawing>"))
  (end_list)
  (setq index 0 n 1)
  (foreach item items
    (if (equal (caddr item) previous) (setq index n))
    (setq n (1+ n)))
  (set_tile key (itoa index))
  (mode_tile key 0)
  (if (and items (> index 0)) (caddr (nth (1- index) items))))

;; Handle a dropdown change. Index 0 means the user wants to pick the
;; object on screen: close the dialog with a pick code. Otherwise store
;; the chosen object's handle. Returns T when a pick was requested.
(defun app:popup-changed (value items setvar pickcode / idx)
  (setq idx (atoi value))
  (if (or (= idx 0) (>= idx (1+ (length items))))
    (progn (done_dialog pickcode) T)
    (progn (set setvar (caddr (nth (1- idx) items))) nil)))

(defun app:catalog-object (items handle / item result)
  (foreach item items
    (if (equal handle (caddr item)) (setq result (cadr item))))
  (if (and result (not (vlax-object-released-p result))) result))

(defun app:source-controls ()
  (mode_tile "p_prof" (if (and app:*profiles* (member app:*zsrc* '("z_prof0"
"z_prof"))) 0 1))
  (mode_tile "p_surf" (if (and app:*surfaces* (member app:*zsrc* '("z_surf0"
```

```
"z_surf"))) 0 1)))

;; Rebuild the profile view / elevation profile lists for the alignment
;; whose handle is stored in app:*aln-h*.
(defun app:alignment-refresh (/ aln)
  (setq aln (app:catalog-object app:*alignments* app:*aln-h*)
        app:*views* nil app:*profiles* nil)
  (if aln
    (setq app:*views* (app:child-catalog aln 'ProfileViews "AECC_GRAPH_PROFILE")
          app:*profiles* (app:child-catalog aln 'Profiles "AECC_PROFILE")))
  (setq app:*view-h* (app:popup "p_view" app:*views* app:*view-h*)
        app:*prof-h* (app:popup "p_prof" app:*profiles* app:*prof-h*))
  (app:source-controls))

(defun app:alignment-changed (value)
  (if (not (app:popup-changed value app:*alignments* 'app:*aln-h* 10))
    (app:alignment-refresh)))

(defun app:menu-catalogs (/ previous)
  (setq app:*alignments* (app:drawing-catalog "AECC_ALIGNMENT")
        app:*surfaces* (app:drawing-catalog "AECC_TIN_SURFACE,AECC_GRID_SURFACE"))
  (setq app:*aln-h* (app:popup "p_aln" app:*alignments* app:*aln-h*)
        app:*surf-h* (app:popup "p_surf" app:*surfaces* app:*surf-h*))
  (app:alignment-refresh))

;; After the dialog closes for a drawing pick: ask for the object, keep
;; its handle, and refresh the lists so the reopened menu shows it.
(defun app:menu-pick (kind / msg typ obj h)
  (setq msg (cond ((= kind "alignment") "\nانقر المحور على الشاشة: ")
                  ((= kind "view") "\nانقر شبكة القطاع على الشاشة: ")
                  ((= kind "profile") "\nانقر خط البروفايل على الشاشة: ")
                  (T "\nانقر السطح على الشاشة: "))
        typ (cond ((= kind "alignment") "AECC_ALIGNMENT")
                  ((= kind "view") "AECC_GRAPH_PROFILE")
                  ((= kind "profile") "AECC_PROFILE")
                  (T "AECC_TIN_SURFACE,AECC_GRID_SURFACE")))
  (setq obj (vl-catch-all-apply 'app:obj (list msg typ)))
  (if (vl-catch-all-error-p obj)
    (princ (strcat "\nPick cancelled: " (vl-catch-all-error-message obj)))
    (if (setq h (app:prop obj 'Handle))
      (cond ((= kind "alignment") (setq app:*aln-h* h app:*view-h* nil app:*prof-h*
nil))
            ((= kind "view") (setq app:*view-h* h))
            ((= kind "profile") (setq app:*prof-h* h))
            (T (setq app:*surf-h* h))))))

(defun app:chosen (kind prompt object-type / items handle result)
  (if app:*menu-run*
    (progn
      (cond ((= kind "alignment") (setq items app:*alignments* handle app:*aln-h*))
```

```
            ((= kind "view") (setq items app:*views* handle app:*view-h*))
            ((= kind "profile") (setq items app:*profiles* handle app:*prof-h*))
            ((= kind "surface") (setq items app:*surfaces* handle app:*surf-h*)))
      (setq result (app:catalog-object items handle))
      (if result result
        (error (strcat "Selected " kind " is unavailable. Reopen APMENU and choose
it again."))))
    (app:obj prompt object-type)))

;;; ---------- station equations / correction --------------------------
(defun app:eqlist (aln / r eqs i q)
  (setq r (vl-catch-all-apply 'vlax-get (list aln 'StationEquations)))
  (if (not (vl-catch-all-error-p r))
    (progn (setq i 0)
      (repeat (vla-get-Count r)
        (setq q (vla-Item r i))
        (setq eqs (cons (list (vlax-get q 'RawStationBack) (vlax-get q 'StationBack)
                              (vlax-get q 'StationAhead)) eqs) i (1+ i)))))
  (vl-sort eqs '(lambda (a b) (< (car a) (car b)))))

(defun app:raw->sta (aln raw / s q)
  (setq s raw)
  (if app:*eq*
    (foreach q (app:eqlist aln)
      (if (>= raw (car q)) (setq s (+ s (- (caddr q) (cadr q)))))))
  (+ s (app:shift)))

(defun app:sta->raw (aln sta / s q)
  (setq s (- sta (app:shift)))
  (if app:*eq*
    (foreach q (reverse (app:eqlist aln))
      (if (>= s (caddr q)) (setq s (- s (- (caddr q) (cadr q)))))))
  s)

;;; ---------- object picking / drawing --------------------------------
(defun app:obj (msg typ / e o)
  (while (not o)
    (setq e (entsel msg))
    (cond ((null e) (princ "\nلم يتم اختيار شيء."))
          ((wcmatch (vla-get-ObjectName (setq o (vlax-ename->vla-object (car e))))
typ))
          (T (setq o nil) (princ "\nنوع العنصر غير صحيح.")))
  o)

(defun app:layer (name col / lay)
  (if (not (tblsearch "LAYER" name))
    (progn
      (setq lay (vla-add (vla-get-Layers (vla-get-ActiveDocument
(vlax-get-acad-object))) name))
      (vla-put-Color lay col)))
```

```
  name)

(defun app:pline3d (pts lay / ms arr)
  (setq ms (vla-get-ModelSpace (vla-get-ActiveDocument (vlax-get-acad-object))))
  (setq arr (vlax-make-safearray vlax-vbDouble (cons 0 (1- (* 3 (length pts))))))
  (vlax-safearray-fill arr (apply 'append pts))
  (vla-put-Layer (vla-Add3DPoly ms arr) lay))

(defun app:text (pt str h lay)
  (entmakex (list '(0 . "TEXT") (cons 8 lay) (cons 10 pt) (cons 40 h) (cons 1
str))))

(defun app:label (pt str h lay)
  (if app:*labels* (app:text pt str h lay)))

;; Optional generated labels never replace copied annotation content.
(defun app:station-label (pt sta elev lay / h)
  (if app:*labels*
    (progn
      (setq h (cond ((app:num app:*th*)) (2.5)))
      (app:label (list (+ (car pt) h) (cadr pt) (caddr pt))
                 (strcat "STA=" (rtos sta 2 3) " LEVEL=" (rtos elev 2 3)) h lay))))

(defun app:profile-piece (pts pv lay / se)
  (app:pline3d pts lay)
  (if (and app:*labels* (setq se (app:pvse pv (car pts))))
    (app:station-label (car pts) (car se) (cadr se) lay)))

(defun app:mesh-p (e / d flags)
  (setq d (entget e) flags (cdr (assoc 70 d)))
  (and (= (cdr (assoc 0 d)) "POLYLINE") flags (/= 0 (logand flags 80))))

(defun app:point (pt lay) (entmakex (list '(0 . "POINT") (cons 8 lay) (cons 10
pt))))

(defun app:block (pt name lay)
  (entmakex (list '(0 . "INSERT") (cons 2 name) (cons 8 lay) (cons 10 pt))))

(defun app:circle (pt r lay)
  (entmakex (list '(0 . "CIRCLE") (cons 8 lay) (cons 10 pt) (cons 40 r))))

(defun app:3dpt (v) (vlax-safearray->list (vlax-variant-value v)))

(defun app:cogoname (o / n)
  (setq n (app:prop o 'Name))
  (if (or (null n) (= n "")) (setq n (app:prop o 'Number)))
  (cond ((null n) "") ((numberp n) (itoa (fix n))) (T n)))

;; Native curve queries return WCS, including 2D polyline elevation and
;; 3D vertex Z. Merge original vertex distances with regular samples so
```

```
;; short segments and sharp corners are never bypassed by the sample step.
(defun app:vertices-worker (e step / o len d distances pts param finish p next)
  (setq o (vlax-ename->vla-object e)
        finish (vlax-curve-getEndParam o)
        len (vlax-curve-getDistAtParam o finish)
        d 0.0 distances (list 0.0 len))
  (if (> len 1e-9)
    (progn
      (if (<= step 0.0) (setq step (/ len 50.0)))
      (while (< d len)
        (setq distances (cons d distances) next (+ d step)
              d (if (> next d) next len)))
      (setq param (vlax-curve-getStartParam o))
      (while (<= param finish)
        (setq distances (cons (vlax-curve-getDistAtParam o param) distances)
              param (+ param 1.0)))
      (foreach d (vl-sort distances '<)
        (if (setq p (vlax-curve-getPointAtDist o d))
          (setq pts (cons p pts))))
      (reverse pts))))

(defun app:vertices (e step / result)
  (setq result (vl-catch-all-apply 'app:vertices-worker (list e step)))
  ;; Unsupported meshes and degenerate curves yield no usable vertices,
  ;; rather than stopping projection of all the other selected objects.
  (if (vl-catch-all-error-p result) nil result))

;; A rejected profile vertex must end the current piece. Never connect
;; valid vertices across an invalid station / elevation gap.
(defun app:p2a-curve (vertices pv aln off lay / v se p pts pieces bad firstse)
  (setq pieces 0 bad 0)
  (foreach v vertices
    (if (and (setq se (app:pvse pv v))
             (setq p (app:planpt aln (car se) off)))
      (progn
        (if (not pts) (setq firstse se))
        (setq pts (cons (list (car p) (cadr p) (cadr se)) pts)))
      (progn
        (if (> (length pts) 1)
          (progn (app:pline3d (reverse pts) lay)
                 (app:station-label (car (reverse pts)) (car firstse) (cadr firstse)
lay)
                 (setq pieces (1+ pieces))))
        (setq pts nil bad (1+ bad)))))
  (if (> (length pts) 1)
    (progn (app:pline3d (reverse pts) lay)
                 (app:station-label (car (reverse pts)) (car firstse) (cadr firstse)
lay)
                 (setq pieces (1+ pieces))))
  (if (> pieces 0)
```

```
    (progn
      (app:rep (strcat "Curve pieces=" (itoa pieces) " / skipped vertices=" (itoa
bad)))
      T)
    "fewer than 2 consecutive usable curve vertices / unsupported or zero-length
curve"))

;;; ---------- alignment / profile view math ----------------------------
;; raw Civil 3D call: plan XY -> (raw-station offset) or nil
(defun app:c3dso (aln pt / sta off)
  (if (not (vl-catch-all-error-p
             (vl-catch-all-apply 'vlax-invoke-method
               (list aln 'StationOffset (car pt) (cadr pt) 'sta 'off))))
    (list sta off)))


;; ---------- v12: exact nearest station / offset ------------------------
;; cache key: handle + length (rebuilt when the alignment is edited)
(defun app:aln-key (aln / l)
  (setq l (app:prop aln 'Length))
  (strcat (vla-get-handle aln) "|" (rtos (if l l 0.0) 2 4)))

;; start / end station of every alignment entity (PIs, curve ends)
(defun app:entity-stations (aln / ents n i e s r)
  (if (and (setq ents (app:prop aln 'Entities)) (setq n (app:prop ents 'Count)))
    (progn
      (setq i 0)
      (while (< i n)
        (setq e (vl-catch-all-apply 'vlax-invoke-method (list ents 'Item i)))
        (if (not (vl-catch-all-error-p e))
          (progn
            (if (setq s (app:prop e 'StartingStation)) (setq r (cons s r)))
            (if (setq s (app:prop e 'EndingStation)) (setq r (cons s r)))))
        (setq i (1+ i)))))
  r)

;; dense list of (raw-station x y) along the alignment centreline
(defun app:samples (aln / key hit st en len step s lst out p)
  (setq key (app:aln-key aln))
  (if (setq hit (assoc key app:*samp*))
    (cdr hit)
    (progn
      (setq st (app:prop aln 'StartingStation) en (app:prop aln 'EndingStation))
      (if (and st en (> en st))
        (progn
          (setq len (- en st) step (min 0.25 (max 0.02 (/ len 2000.0))) s st)
          (if (> (/ len step) 20000.0) (setq step (/ len 20000.0)))
          (while (< s en) (setq lst (cons s lst) s (+ s step)))
          (setq lst (cons en lst))
          (foreach q (app:entity-stations aln)
```

```
          (if (and (>= q st) (<= q en)) (setq lst (cons q lst))))
        (setq lst (vl-sort lst '<))
        (foreach q lst
          (if (and (or (null out) (> (- q (car (car out))) 1e-6))
                   (setq p (app:ploc aln q 0.0)))
            (setq out (cons (list q (car p) (cadr p)) out))))
        (setq out (reverse out))
        (setq app:*samp* (cons (cons key out) app:*samp*))
        out)))))

;; nearest spot on chord a-b (each (s x y)) -> (distance station)
(defun app:segnear (pt a b / dx dy l2 u x y)
  (setq dx (- (cadr b) (cadr a)) dy (- (caddr b) (caddr a))
        l2 (+ (* dx dx) (* dy dy)))
  (setq u (if (> l2 1e-12)
            (/ (+ (* (- (car pt) (cadr a)) dx) (* (- (cadr pt) (caddr a)) dy)) l2)
            0.0))
  (setq u (max 0.0 (min 1.0 u))
        x (+ (cadr a) (* u dx)) y (+ (caddr a) (* u dy)))
  (list (distance (list (car pt) (cadr pt)) (list x y))
        (+ (car a) (* u (- (car b) (car a))))))

;; plan distance from pt to the alignment point at raw station s
(defun app:pdist (aln s pt / p)
  (if (setq p (app:ploc aln s 0.0))
    (distance (list (car p) (cadr p)) (list (car pt) (cadr pt)))))

;; TRUE shortest distance: plan XY -> (raw-station signed-offset) or nil
(defun app:nearso (aln pt / sm a r best h raw lo hi i m1 m2 d1 d2 p tv off)
  (if (and (setq sm (app:samples aln)) (cdr sm))
    (progn
      (setq a (car sm))
      (foreach b (cdr sm)
        (setq r (app:segnear pt a b))
        (if (or (null best) (< (car r) (car best)))
          (setq best r h (- (car b) (car a))))
        (setq a b))
      ;; refine on the real alignment around the best chord
      (setq raw (cadr best)
            lo (max (car (car sm)) (- raw h))
            hi (min (car (last sm)) (+ raw h))
            i 0)
      (while (< i 40)
        (setq m1 (+ lo (/ (- hi lo) 3.0)) m2 (- hi (/ (- hi lo) 3.0))
              d1 (app:pdist aln m1 pt) d2 (app:pdist aln m2 pt))
        (if (and d1 d2)
          (if (< d1 d2) (setq hi m2) (setq lo m1))
          (setq i 99))
        (setq i (1+ i)))
      (setq m1 (/ (+ lo hi) 2.0))
```

```
      (if (and (setq d1 (app:pdist aln m1 pt)) (< d1 (car best)))
        (setq best (list d1 m1)))
      (setq raw (cadr best) off (car best)
            p (app:ploc aln raw 0.0) tv (app:tan aln raw))
      ;; sign: + right of the alignment, - left
      (if (and p tv
               (< (- (* (cadr tv) (- (car pt) (car p)))
                     (* (car tv) (- (cadr pt) (cadr p)))) 0.0))
        (setq off (- off)))
      (list raw off))))

;; plan XY -> (raw-station offset): exact method, else Civil 3D's call
(defun app:rawso (aln pt)
  (if app:*exactso*
    (cond ((app:nearso aln pt)) ((app:c3dso aln pt)))
    (app:c3dso aln pt)))

;; raw station + offset -> plan XY (no equations / correction)
(defun app:ploc (aln raw off / x y)
  (if (not (vl-catch-all-error-p
             (vl-catch-all-apply 'vlax-invoke-method
               (list aln 'PointLocation raw off 'x 'y))))
    (list x y)))

;; unit direction of the alignment at a raw station, or nil
(defun app:tan (aln raw / s0 s1 a b st en d)
  (setq st (app:prop aln 'StartingStation) en (app:prop aln 'EndingStation))
  (setq s0 (- raw 0.05) s1 (+ raw 0.05))
  (if (and st (< s0 st)) (setq s0 st))
  (if (and en (> s1 en)) (setq s1 en))
  (if (and (> s1 s0) (setq a (app:ploc aln s0 0.0)) (setq b (app:ploc aln s1 0.0))
           (> (setq d (distance a b)) 1e-9))
    (list (/ (- (car b) (car a)) d) (/ (- (cadr b) (cadr a)) d))))

;; plan XY -> (station offset) or nil.
;; Checks that the point is really SQUARE to the alignment at the station.
;; Sets app:*sotype*: "ok" | "ext" (extended past start/end) | "noperp"
(defun app:staoff (aln pt / r raw off b tv vx vy al cr tol st en)
  (setq app:*sotype* "ok")
  (if (setq r (app:rawso aln pt))
    (progn
      (setq raw (car r) off (cadr r)
            st (app:prop aln 'StartingStation) en (app:prop aln 'EndingStation))
      (if (and (setq b (app:ploc aln raw 0.0)) (setq tv (app:tan aln raw)))
        (progn
          (setq vx (- (car pt) (car b)) vy (- (cadr pt) (cadr b))
                al (+ (* vx (car tv)) (* vy (cadr tv)))      ; along the alignment
                cr (- (* (cadr tv) vx) (* (car tv) vy))      ; square to it (+
right)
                tol (max 0.05 (* 0.002 (abs off))))
```

```
          (if (> (abs al) tol)
            (cond
              ((and app:*extaln* en (equal raw en 1e-3) (> al 0.0))
               (setq raw (+ raw al) off cr app:*sotype* "ext"))
              ((and app:*extaln* st (equal raw st 1e-3) (< al 0.0))
               (setq raw (+ raw al) off cr app:*sotype* "ext"))
              ;; exact method: nearest spot inside the alignment (e.g. a PI)
              ;; IS the correct answer -> keep it
              ((and app:*exactso* st en (> raw (+ st 1e-3)) (< raw (- en 1e-3)))
               (setq app:*sotype* "ok"))
              (T (setq app:*sotype* "noperp"))))))
      (list (app:raw->sta aln raw) off))))

;; text explaining a "noperp" result
(defun app:noperp-why (so)
  (strcat "it has no square position on the alignment (past its start/end or outside
a bend). "
          "Civil 3D gave the nearest station " (rtos (car so) 2 3)
          " and straight distance " (rtos (abs (cadr so)) 2 3)
          ". Tick 'Extend alignment past start/end' or 'Accept nearest station'."))

;; T when a staoff result may be used
(defun app:so-ok (so) (and so (or (/= app:*sotype* "noperp") app:*nearest*)))

;; suffix for the report
(defun app:sotag ()
  (cond ((= app:*sotype* "ext") "  (beyond alignment end - station extended)")
        ((= app:*sotype* "noperp") "  (NO square position - nearest station used)")
        (T "")))

;; station offset -> plan XY
(defun app:planpt (aln sta off / x y)
  (if (not (vl-catch-all-error-p
             (vl-catch-all-apply 'vlax-invoke-method
               (list aln 'PointLocation (app:sta->raw aln sta) off 'x 'y))))
    (list x y)))

;; plan check line: object TRUE position -> alignment point at its station
(defun app:witness (aln so pt th / f lay)
  (if (and app:*witness* so pt (setq f (app:planpt aln (car so) 0.0)))
    (progn
      (setq lay (app:layer "PROJ-A2P-OFFSET" 6))
      (entmakex (list '(0 . "LINE") (cons 8 lay)
                      (list 10 (car pt) (cadr pt) 0.0) (list 11 (car f) (cadr f)
0.0)))
      (app:text (list (/ (+ (car pt) (car f)) 2.0) (/ (+ (cadr pt) (cadr f)) 2.0)
0.0)
                (strcat "OFF=" (rtos (cadr so) 2 3) " STA=" (rtos (car so) 2 3)
                        " X=" (rtos (car pt) 2 3) " Y=" (rtos (cadr pt) 2 3))
                (* 0.4 th) lay))))
```

```
;; direct Civil 3D call: station elevation -> profile view XY
(defun app:pvxy (pv sta elev / x y)
  (if (not (vl-catch-all-error-p
             (vl-catch-all-apply 'vlax-invoke-method
               (list pv 'FindXYAtStationAndElevation sta elev 'x 'y))))
    (list x y 0.0)))

;; direct Civil 3D call: profile view XY -> (station elevation)
(defun app:pvsexy (pv pt / sta elev)
  (if (not (vl-catch-all-error-p
             (vl-catch-all-apply 'vlax-invoke-method
               (list pv 'FindStationAndElevationAtXY (car pt) (cadr pt) 'sta
'elev))))
    (list sta elev)))

;; Two reference corners of the profile view grid, used to extend the
;; scale for objects above / below / beyond the grid.
;; Result: (s1 e1 x1 y1 s2 e2 x2 y2) or nil
(defun app:pvref (pv / s1 s2 e1 e2 a b)
  (if (not app:*pvref*)
    (progn
      (setq s1 (app:prop pv 'StationStart) s2 (app:prop pv 'StationEnd)
            e1 (app:prop pv 'ElevationMin) e2 (app:prop pv 'ElevationMax))
      (if (and s1 s2 e1 e2 (/= s1 s2) (/= e1 e2)
               (setq a (app:pvxy pv s1 e1)) (setq b (app:pvxy pv s2 e2)))
        (setq app:*pvref* (list s1 e1 (car a) (cadr a) s2 e2 (car b) (cadr b)))
        (setq app:*pvref* 'none))))
  (if (listp app:*pvref*) app:*pvref*))

;; station elevation -> profile view XY (extends scale when outside grid)
(defun app:pvpt (pv sta elev / p r)
  (setq app:*extrap* nil)
  (cond ((setq p (app:pvxy pv sta elev)) p)
        ((and app:*extend* (setq r (app:pvref pv)))
         (setq app:*extrap* T)
         (list (+ (nth 2 r) (* (- sta (nth 0 r)) (/ (- (nth 6 r) (nth 2 r)) (- (nth
4 r) (nth 0 r)))))
               (+ (nth 3 r) (* (- elev (nth 1 r)) (/ (- (nth 7 r) (nth 3 r)) (- (nth
5 r) (nth 1 r)))))
               0.0))))

;; profile view XY -> (station elevation) (extends scale when outside grid)
(defun app:pvse (pv pt / se r)
  (setq app:*extrap* nil)
  (cond ((setq se (app:pvsexy pv pt)) se)
        ((and app:*extend* (setq r (app:pvref pv)))
         (setq app:*extrap* T)
         (list (+ (nth 0 r) (* (- (car pt) (nth 2 r)) (/ (- (nth 4 r) (nth 0 r)) (-
(nth 6 r) (nth 2 r)))))
```

```
               (+ (nth 1 r) (* (- (cadr pt) (nth 3 r)) (/ (- (nth 5 r) (nth 1 r)) (-
(nth 7 r) (nth 3 r)))))))))

;;; ---------- elevation source ----------------------------------------
(defun app:profelev (prof sta / r)
  (if prof
    (progn
      (setq r (vl-catch-all-apply 'vlax-invoke-method (list prof 'ElevationAt sta)))
      (if (vl-catch-all-error-p r) nil r))))

(defun app:surfelev (surf pt / r)
  (if surf
    (progn
      (setq r (vl-catch-all-apply 'vlax-invoke-method
                (list surf 'FindElevationAtXY (car pt) (cadr pt))))
      (if (or (vl-catch-all-error-p r) (app:noz r)) nil r))))

;; z = object's own Z, sta = station, pt = plan point
(defun app:getz (z sta pt)
  (cond ((= app:*zsrc* "z_prof") (app:profelev app:*prof* sta))
        ((= app:*zsrc* "z_surf") (app:surfelev app:*surf* pt))
        ((not (app:noz z)) z)
        ((= app:*zsrc* "z_prof0") (app:profelev app:*prof* sta))
        ((= app:*zsrc* "z_surf0") (app:surfelev app:*surf* pt))))

(defun app:zsrc-name ()
  (cdr (assoc app:*zsrc* '(("z_obj" . "object Z") ("z_prof0" . "object Z, else
profile")
                           ("z_prof" . "profile line") ("z_surf" . "surface")
                           ("z_surf0" . "object Z, else surface")))))

;; ask (command line only) and select the profile / surface if needed
(defun app:setup-zsrc (/ k)
  (if (not app:*menu-run*)
    (progn
      (initget "Z ZProfile Profile Surface ZSurface")
      (setq k (getkword "\nElevation source [Z/ZProfile/Profile/Surface/ZSurface]
<Z>: "))
      (setq app:*zsrc* (cdr (assoc (if k k "Z")
                                   '(("Z" . "z_obj") ("ZProfile" . "z_prof0")
("Profile" . "z_prof")
                                     ("Surface" . "z_surf") ("ZSurface" .
"z_surf0")))))))
  (setq app:*prof* nil app:*surf* nil)
  (if (member app:*zsrc* '("z_prof0" "z_prof"))
    (setq app:*prof* (app:chosen "profile" "\nSelect PROFILE line (gives the
elevations): " "AeccDbProfile")))
  (if (member app:*zsrc* '("z_surf0" "z_surf"))
    (setq app:*surf* (app:chosen "surface" "\nSelect SURFACE (gives the elevations):
" "AeccDbSurface*"))))
```

```
;; Section markers require their DXF codes AND parent context. MText may
;; contain the same strings; node group 10 is a landing, not an arrow.
;; Use the actual line ID with the native API, never assume IDs start at 0.
(defun app:mleader-anchor (o / item context node inside anchor lineid vertices
result)
  (foreach item (entget (vlax-vla-object->ename o))
    (cond
      ((and (= (car item) 300) (= (cdr item) "CONTEXT_DATA{"))
       (setq context T node nil inside nil))
      ((and (= (car item) 301) (= (cdr item) "}"))
       (setq context nil node nil inside nil))
      ((and context (= (car item) 302) (= (cdr item) "LEADER{"))
       (setq node T inside nil))
      ((and context (= (car item) 303) (= (cdr item) "}"))
       (setq node nil inside nil))
      ((and context node (= (car item) 304) (= (cdr item) "LEADER_LINE{"))
       (setq inside T))
      ((and inside (= (car item) 305) (= (cdr item) "}"))
       (setq inside nil))
      ((and context node inside (= (car item) 91) (not lineid))
       (setq lineid (cdr item)))
      ((and context node inside (= (car item) 10) (not anchor))
       (setq anchor (cdr item)))))
  (if lineid
    (progn
      (setq result (vl-catch-all-apply 'vlax-invoke-method
                     (list o 'GetLeaderLineVertices lineid)))
      (if (not (vl-catch-all-error-p result))
        (progn
          (setq vertices (app:array-list result))
          (if (and vertices (>= (length vertices) 3)
                   (numberp (car vertices)) (numberp (cadr vertices))
                   (numberp (caddr vertices)))
            (setq anchor (list (car vertices) (cadr vertices) (caddr
vertices))))))))
  anchor)

;;; ---------- native annotations (v19) --------------------------------
;; Copy, do not explode: preserve text type, formatting, style and leaders.
;; TEXT/MTEXT use their insertion/alignment point; MLeader uses first arrow.
;; A multi-arrow MLeader is moved rigidly using its first arrow as anchor.
;; Other arrows retain their relative positions, not independent stations.
(defun app:array-list (value)
  (cond ((= (type value) 'VARIANT) (app:array-list (vlax-variant-value value)))
        ((= (type value) 'SAFEARRAY) (vlax-safearray->list value))
        ((listp value) value)))

(defun app:annotation-anchor (o et / e d h v p)
  (cond
```

```
    ((= et "MULTILEADER") (app:mleader-anchor o))
    ((= et "TEXT")
     (setq e (vlax-vla-object->ename o) d (entget e)
           h (cond ((cdr (assoc 72 d))) (0))
           v (cond ((cdr (assoc 73 d))) (0)))
     ;; Baseline-left uses insertion. Justified text uses alignment point
     ;; (the second endpoint for Aligned/Fit). DXF TEXT points are in OCS.
     (setq p (cdr (assoc (if (and (= h 0) (= v 0)) 10 11) d)))
     (if p (trans p e 0)))
    (T
     ;; MTEXT attachment point is WCS; group 11 is a direction, not a point.
     (cdr (assoc 10 (entget (vlax-vla-object->ename o)))))))

(defun app:annotation-copy-worker (o anchor target lay / copied)
  (setq copied (vla-Copy o) app:*annotation-copy* copied)
  (vla-Move copied (vlax-3d-point anchor) (vlax-3d-point target))
  (vla-put-Layer copied lay)
  (vla-Update copied)
  T)

(defun app:annotation-copy (o anchor target lay / result)
  (setq app:*annotation-copy* nil)
  (setq result (vl-catch-all-apply 'app:annotation-copy-worker
                 (list o anchor target lay)))
  (if (vl-catch-all-error-p result)
    (progn
      (if app:*annotation-copy*
        (vl-catch-all-apply 'vla-Delete (list app:*annotation-copy*)))
      (setq app:*annotation-copy* nil)
      (strcat "annotation copy failed: " (vl-catch-all-error-message result)))
    (progn (setq app:*annotation-copy* nil) T)))

(defun app:project-annotation (o et direction aln pv off maxoff lay / anchor so z
target se result)
  (setq result (vl-catch-all-apply 'app:annotation-anchor (list o et)))
  (if (vl-catch-all-error-p result) (setq result nil))
  (setq anchor result)
  (cond
    ((not (and anchor (= (length anchor) 3)))
     "cannot read annotation insertion point / MLeader arrow point")
    ((= direction "A2P")
     (cond
       ((not (setq so (app:staoff aln anchor))) "annotation anchor is outside the
alignment")
       ((not (app:so-ok so)) (app:noperp-why so))
       ((and maxoff (> (abs (cadr so)) maxoff)) "annotation anchor exceeds Max
offset")
       ((not (setq z (app:getz (caddr anchor) (car so) anchor)))
        "annotation anchor has no elevation; choose a Profile or Surface elevation
source")
```

```
       ((not (setq target (app:pvpt pv (car so) z))) "annotation anchor is outside
the profile view")
       (T
        (setq result (app:annotation-copy o anchor target lay))
        (if (= result T)
          (progn
            (app:witness aln so anchor (cond ((app:num app:*th*)) (2.5)))
            (app:station-label target (car so) z lay)
            (app:rep (strcat et " X=" (rtos (car anchor) 2 3)
                             " Y=" (rtos (cadr anchor) 2 3)
                             " STA=" (rtos (car so) 2 3)
                             " OFF=" (rtos (cadr so) 2 3) " Z=" (rtos z 2 3)))))
        result)))
    (T
     (cond
       ((not (setq se (app:pvse pv anchor))) "annotation anchor is outside the
profile view")
       ((not (setq target (app:planpt aln (car se) off))) "annotation anchor station
is outside the alignment")
       (T
        (setq target (list (car target) (cadr target) (cadr se))
              result (app:annotation-copy o anchor target lay))
        (if (= result T)
          (progn
            (app:station-label target (car se) (cadr se) lay)
            (app:rep (strcat et " STA=" (rtos (car se) 2 3) " Z=" (rtos (cadr se) 2
3)))))
        result)))))

;;; ---------- one point-like object plan -> profile --------------------
;; returns T when drawn, otherwise a reason string
(defun app:a2p-point (aln pv et o pt th lay maxoff / so z p nm)
  (cond
    ((not (setq so (app:staoff aln pt)))
     "it is not beside the alignment (before its start or after its end)")
    ((not (app:so-ok so)) (app:noperp-why so))
    ((and maxoff (> (abs (cadr so)) maxoff))
     (strcat "offset " (rtos (abs (cadr so)) 2 3) " is more than Max offset " (rtos
maxoff 2 3)))
    ((not (setq z (app:getz (caddr pt) (car so) pt)))
     (strcat "no elevation at STA " (rtos (car so) 2 3) " (object Z = " (app:ztxt
(caddr pt))
             ", source = " (app:zsrc-name) "). Pick 'Profile' or 'Surface' as
Elevation source."))
    ((not (setq p (app:pvpt pv (car so) z)))
     (strcat "STA " (rtos (car so) 2 3) " / Z " (rtos z 2 3)
             " is outside the profile view. Tick 'Draw objects outside the profile
view grid'."))
    (T
     (setq nm (if (= et "AECC_COGO_POINT") (app:cogoname o) ""))
```

```
     (if (= app:*sotype* "noperp")
       (setq lay (app:layer "PROJ-A2P-CHECK" 2) nm (strcat "CHECK " nm) app:*nchk*
(1+ app:*nchk*)))
     (app:witness aln so pt th)
     (app:rep (strcat et " " nm "  X=" (rtos (car pt) 2 3) " Y=" (rtos (cadr pt) 2
3)
                      "  STA=" (rtos (car so) 2 3) "  OFF=" (rtos (cadr so) 2 3)
                      "  Z=" (rtos z 2 3)
                      (app:sotag)
                      (if app:*extrap* "  (outside view grid - drawn by extending
the scale)" "")))
     (cond ((= et "INSERT") (app:block p (vla-get-EffectiveName o) lay)
                            (app:station-label p (car so) z lay))
           ((member et '("TEXT" "MTEXT")) (app:annotation-copy o pt p lay))
           (T (app:point p lay)
              (app:label (list (+ (car p) th) (cadr p) 0.0)
                         (strcat (if (/= nm "") (strcat nm "  ") "")
                                 "STA=" (rtos (car so) 2 3) " Z=" (rtos z 2 3))
                         th lay)))
     T)))

;;; ---------- A2P : plan -> profile ----------------------------------
(defun c:A2P (/ *error* doc aln pv maxoff step th ss i e et o pt so z p
                pts cnt skip sp ep so1 so2 t1 xp zc dia lay ok why
                segs nuse nbad zmin zmax smin smax bad pso pz pv0 ncr v)
  (setq doc (vla-get-ActiveDocument (vlax-get-acad-object)))
  (defun *error* (m) (vla-EndUndoMark doc) (if m (princ (strcat "\nError: " m)))
(princ))
  (setq aln (app:chosen "alignment" "\nاختر المحور: " "AeccDbAlignment")
        pv  (app:chosen "view" "\nاختر شبكة القطاع الطولي: " "AeccDbGraphProfile")
        app:*pvref* nil)
  (app:setup-zsrc)
  (if app:*menu-run*
    (setq maxoff (app:num app:*maxoff*) step (app:num app:*step*) th (app:num
app:*th*))
    (setq maxoff (getdist "\nMax offset from alignment <no limit>: ")
          step   (getdist "\nPolyline sampling step <auto>: ")
          th     (getdist "\nText height in profile <2.5>: ")))
  (if (null step) (setq step 0.0))
  (if (null th) (setq th 2.5))
  (setq lay (app:layer "PROJ-A2P" 1) cnt 0 skip 0 app:*skips* nil app:*nchk* 0)
  (princ "\nاختر العناصر في المسقط لإسقاطها: ")
  (if (setq ss (ssget (app:selection-filter "A2P")))
    (progn
      (vla-StartUndoMark doc)
      (setq i 0)
      (repeat (sslength ss)
        (setq e (ssname ss i) et (cdr (assoc 0 (entget e))) o
(vlax-ename->vla-object e)
              ok nil why nil)
```

```
        (cond
          ;; ---- pipes: crossing with alignment
          ((= et "AECC_PIPE")
           (setq sp (app:3dpt (vla-get-StartPoint o)) ep (app:3dpt (vla-get-EndPoint
o))
                 so1 (app:staoff aln sp) so2 (app:staoff aln ep))
           (if (and so1 so2 (<= (* (cadr so1) (cadr so2)) 0.0)
                    (/= (cadr so1) (cadr so2)))
             (progn
               (setq t1 (/ (cadr so1) (- (cadr so1) (cadr so2)))
                     xp (+ (car so1) (* t1 (- (car so2) (car so1))))
                     zc (+ (caddr sp) (* t1 (- (caddr ep) (caddr sp))))
                     dia (vlax-get o 'InnerDiameterOrWidth)
                     p (app:pvpt pv xp zc))
               (if p
                 (progn
                   (app:circle p (/ dia 2.0) lay)
                   (app:station-label p xp zc lay)
                   (app:label (list (car p) (+ (cadr p) dia) 0.0)
                              (strcat (vla-get-Name o) " D=" (rtos dia 2 3)
                                      " INV=" (rtos (- zc (/ dia 2.0)) 2 3))
                              th lay)
                   (app:rep (strcat "PIPE " (vla-get-Name o) "  STA=" (rtos xp 2 3)
                                    "  Zc=" (rtos zc 2 3)))
                   (setq ok T))
                 (setq why "crossing is outside the profile view")))
             (setq why (strcat "pipe " (vla-get-Name o) " does not cross the
alignment"))))
          ;; ---- curves
          ((app:mesh-p e)
           (setq why "polygon meshes and polyface meshes are not supported"))
          ((member et '("LINE" "LWPOLYLINE" "POLYLINE"))
           (setq pts nil segs 0 nuse 0 nbad 0 ncr 0 zmin nil zmax nil smin nil smax
nil
                 pso nil pz nil pv0 nil)
           (foreach v (app:vertices e step)
             (setq p nil
                   bad (cond ((not (setq so (app:staoff aln v)))
                              "vertex is not beside the alignment")
                             ((= app:*sotype* "noperp")
                              "vertex has no square position on the alignment
(outside a bend)")
                             ((and maxoff (> (abs (cadr so)) maxoff))
                              "vertex is farther than Max offset")
                             ((not (setq z (app:getz (caddr v) (car so) v)))
                              (strcat "vertex has no elevation (Z = " (app:ztxt
(caddr v)) ")"))
                             ((not (setq p (app:pvpt pv (car so) z)))
                              "vertex is outside the profile view")
                             ((and app:*lineclip* app:*extrap*)
```

```
                                (strcat "vertex Z " (rtos z 2 3)
                                        " is outside the profile view grid"))))
              (if bad
                (progn
                  (setq nbad (1+ nbad) pso nil)
                  (if (not why) (setq why bad))
                  (if (> (length pts) 1)
                    (progn (app:profile-piece (reverse pts) pv lay) (setq segs (1+
segs))))
                  (setq pts nil))
                (progn
                  (setq nuse (1+ nuse)
                        zmin (if zmin (min zmin z) z) zmax (if zmax (max zmax z) z)
                        smin (if smin (min smin (car so)) (car so))
                        smax (if smax (max smax (car so)) (car so)))
                  (setq pts (cons p pts)))))
            (if (> (length pts) 1)
              (progn (app:profile-piece (reverse pts) pv lay) (setq segs (1+ segs))))
            (cond
              ((> segs 0)
               (setq ok T)
               (app:rep (strcat et "  vertices used " (itoa nuse) ", left out " (itoa
nbad)
                                "  STA " (rtos smin 2 3) " to " (rtos smax 2 3)
                                "  Z " (rtos zmin 2 3) " to " (rtos zmax 2 3)
                                (if (> segs 1) (strcat "  drawn in " (itoa segs) "
pieces") "")
                                (if why (strcat "  (left out: " why ")") ""))))
              (T (setq why (strcat "fewer than 2 usable vertices"
                                   (if why (strcat " - " why) ""))))))
            ((member et '("TEXT" "MTEXT" "MULTILEADER"))
             (setq why (app:project-annotation o et "A2P" aln pv 0.0 maxoff lay))
             (if (= why T) (setq ok T why nil)))
            ;; ---- point-like (POINT, COGO point, block)
            (T
             (setq pt (cond ((= et "AECC_COGO_POINT")
                             (list (app:prop o 'Easting) (app:prop o 'Northing)
(app:prop o 'Elevation)))
                            ((= et "POINT") (cdr (assoc 10 (entget e))))
                            (T (app:3dpt (vla-get-InsertionPoint o)))))
             (setq why (app:a2p-point aln pv et o pt th lay maxoff))
             (if (= why T) (setq ok T why nil))))
          (if ok (setq cnt (1+ cnt)) (progn (setq skip (1+ skip)) (app:skip et why)))
          (setq i (1+ i)))
        (vla-EndUndoMark doc)
        (princ (strcat "\nProjected: " (itoa cnt) "   Skipped: " (itoa skip)
                       (if (> skip 0) "  (reasons listed above - press F2)" ""))))))
  (app:summary "Plan -> Profile View" cnt skip)
  (princ))
```

```
;;; ---------- A2PPICK : click locations in plan -> profile -------------
(defun c:A2PPICK (/ *error* doc aln pv th lay pt so z p cnt)
  (setq doc (vla-get-ActiveDocument (vlax-get-acad-object)))
  (defun *error* (m) (vla-EndUndoMark doc) (if m (princ (strcat "\nError: " m)))
(princ))
  (setq aln (app:chosen "alignment" "\nاختر المحور: " "AeccDbAlignment")
        pv  (app:chosen "view" "\nاختر شبكة القطاع الطولي: " "AeccDbGraphProfile")
        app:*pvref* nil)
  (app:setup-zsrc)
  (setq th (if app:*menu-run* (app:num app:*th*) (getdist "\nText height in profile
<2.5>: ")))
  (if (null th) (setq th 2.5))
  (setq lay (app:layer "PROJ-A2P" 1) cnt 0)
  (vla-StartUndoMark doc)
  (while (setq pt (getpoint "\nالتقط موقعاً في المسقط <Enter = إنهاء>: "))
    (setq pt (trans pt 1 0))
    (if (not (setq so (app:staoff aln pt)))
      (princ "\n  This location is not beside the alignment.")
      (if (not (app:so-ok so))
        (princ (strcat "\n  Skipped: " (app:noperp-why so)))
      (progn
        (setq z (app:getz (caddr pt) (car so) pt))
        (if (not z)
          (setq z (getreal (strcat "\n  STA " (rtos (car so) 2 3) "  OFF " (rtos
(cadr so) 2 3)
                                   " - no elevation found. Type elevation <skip>:
"))))
        (cond ((not z) (princ "\n  Skipped."))
              ((not (setq p (app:pvpt pv (car so) z)))
               (princ "\n  Outside the profile view (tick 'Draw outside the
grid')."))
              (T
               (app:point p lay)
               (app:witness aln so pt th)
               (app:label (list (+ (car p) th) (cadr p) 0.0)
                          (strcat "STA=" (rtos (car so) 2 3) " Z=" (rtos z 2 3)) th
lay)
               (princ (strcat "\n  Projected  STA=" (rtos (car so) 2 3) "  OFF="
(rtos (cadr so) 2 3)
                              "  Z=" (rtos z 2 3) (app:sotag)
                              (if app:*extrap* "  (outside grid - scale extended)"
"")))
               (setq cnt (1+ cnt))))))))
  (vla-EndUndoMark doc)
  (princ (strcat "\nProjected: " (itoa cnt)))
  (princ))

;;; ---------- P2A : profile -> plan ----------------------------------
(defun c:P2A (/ *error* doc pv aln off th ss i e et o pt se p pts cnt skip lay step
ok why)
```

```
  (setq doc (vla-get-ActiveDocument (vlax-get-acad-object)))
  (defun *error* (m) (vla-EndUndoMark doc) (if m (princ (strcat "\nError: " m)))
(princ))
  (setq pv  (app:chosen "view" "\nاختر شبكة القطاع الطولي: " "AeccDbGraphProfile")
        aln (app:chosen "alignment" "\nاختر المحور: " "AeccDbAlignment")
        app:*pvref* nil)
  (if app:*menu-run*
    (setq off (app:num app:*off*) step (app:num app:*step*) th (app:num app:*th*))
    (setq off (getreal "\nOffset from alignment (+right / -left) <0>: ")
          step (getdist "\nPolyline sampling step (profile units) <auto>: ")
          th  (getdist "\nText height in plan <2.5>: ")))
  (if (null off) (setq off 0.0))
  (if (null step) (setq step 0.0))
  (if (null th) (setq th 2.5))
  (setq lay (app:layer "PROJ-P2A" 3) cnt 0 skip 0 app:*skips* nil app:*nchk* 0)
  (princ "\nاختر العناصر في القطاع الطولي: ")
  (if (setq ss (ssget (app:selection-filter "P2A")))
    (progn
      (vla-StartUndoMark doc)
      (setq i 0)
      (repeat (sslength ss)
        (setq e (ssname ss i) et (cdr (assoc 0 (entget e))) o
(vlax-ename->vla-object e)
              ok nil why nil)
        (cond
          ((app:mesh-p e)
           (setq why "polygon meshes and polyface meshes are not supported"))
          ((member et '("LINE" "LWPOLYLINE" "POLYLINE"))
           (setq why (app:p2a-curve (app:vertices e step) pv aln off lay))
           (if (= why T) (setq ok T why nil)))
          ((member et '("TEXT" "MTEXT" "MULTILEADER"))
           (setq why (app:project-annotation o et "P2A" aln pv off nil lay))
           (if (= why T) (setq ok T why nil)))
          (T
           (setq pt (if (= et "POINT") (cdr (assoc 10 (entget e))) (app:3dpt
(vla-get-InsertionPoint o))))
           (cond
             ((not (setq se (app:pvse pv pt)))
              (setq why "it is outside the profile view"))
             ((not (setq p (app:planpt aln (car se) off)))
              (setq why (strcat "station " (rtos (car se) 2 3) " is not on the
alignment")))
             (T
              (app:rep (strcat et "  STA=" (rtos (car se) 2 3) "  Z=" (rtos (cadr
se) 2 3)
                               (if app:*extrap* "  (outside view grid - scale
extended)" "")))
              (setq p (list (car p) (cadr p) (cadr se)))
              (cond ((= et "INSERT") (app:block p (vla-get-EffectiveName o) lay)
                                     (app:station-label p (car se) (cadr se) lay))
```

```
                    ((member et '("TEXT" "MTEXT")) (app:annotation-copy o pt p lay))
                    (T (app:point p lay)
                       (app:label (list (+ (car p) th) (cadr p) (caddr p))
                                  (strcat "STA=" (rtos (car se) 2 3) " Z=" (rtos
(cadr se) 2 3))
                                  th lay)))
                (setq ok T)))))
          (if ok (setq cnt (1+ cnt)) (progn (setq skip (1+ skip)) (app:skip et why)))
          (setq i (1+ i)))
        (vla-EndUndoMark doc)
        (princ (strcat "\nProjected: " (itoa cnt) "   Skipped: " (itoa skip)))))
  (app:summary "Profile View -> Plan" cnt skip)
  (princ))

;;; ---------- menu ----------------------------------------------------
(defun app:toggle-type (types value / typ)
  (foreach typ types
    (if (= value "1")
      (if (not (member typ app:*types*))
        (setq app:*types* (cons typ app:*types*)))
      (setq app:*types* (vl-remove typ app:*types*)))))

;;; Bulk actions update saved settings AND visible ticks; set_tile does not
;;; invoke action_tile. Each group is independent of the other settings.
(defun app:bulk-types (value)
  (app:toggle-type '("POINT") value)
  (set_tile "t_point" value)
  (app:toggle-type '("INSERT") value)
  (set_tile "t_block" value)
  (app:toggle-type '("TEXT" "MTEXT") value)
  (set_tile "t_text" value)
  (app:toggle-type '("LINE") value)
  (set_tile "t_line" value)
  (app:toggle-type '("LWPOLYLINE" "POLYLINE") value)
  (set_tile "t_poly" value)
  (app:toggle-type '("MULTILEADER") value)
  (set_tile "t_mleader" value)
  (app:toggle-type '("AECC_COGO_POINT") value)
  (set_tile "t_cogo" value)
  (app:toggle-type '("AECC_PIPE") value)
  (set_tile "t_pipe" value)
  (princ))

(defun app:bulk-options (value)
  (setq app:*extend* (= value "1"))
  (set_tile "t_extend" value)
  (setq app:*labels* (= value "1"))
  (set_tile "t_labels" value)
  (setq app:*eq* (= value "1"))
  (set_tile "t_eq" value)
```

```
  (setq app:*report* (= value "1"))
  (set_tile "t_report" value)
  (setq app:*exactso* (= value "1"))
  (set_tile "t_exactso" value)
  (setq app:*extaln* (= value "1"))
  (set_tile "t_extaln" value)
  (setq app:*nearest* (= value "1"))
  (set_tile "t_nearest" value)
  (setq app:*witness* (= value "1"))
  (set_tile "t_witness" value)
  (setq app:*lineclip* (= value "1"))
  (set_tile "t_lineclip" value)
  (princ))

(defun app:selection-filter (direction / types result typ)
  (setq types (if app:*menu-run* app:*types*
                '("POINT" "INSERT" "TEXT" "MTEXT" "MULTILEADER" "LINE" "LWPOLYLINE"
                  "POLYLINE" "AECC_COGO_POINT" "AECC_PIPE")))
  (if (= direction "P2A")
    (setq types (vl-remove "AECC_PIPE" (vl-remove "AECC_COGO_POINT" types))))
  (foreach typ types
    (setq result (if result (strcat result "," typ) typ)))
  (if result (list (cons 0 result))))

(defun app:menu-validate (/ direction reason)
  (setq direction (if (= app:*run* "P2A") "P2A" "A2P"))
  (cond
    ((not (app:catalog-object app:*alignments* app:*aln-h*))
     (setq reason "Choose an alignment from the dropdown. If the list is empty,
create or load a Civil 3D alignment first."))
    ((not (app:catalog-object app:*views* app:*view-h*))
     (setq reason "No profile view is available for this alignment. Choose another
alignment or create its profile view."))
    ((and (/= app:*run* "P2A") (member app:*zsrc* '("z_prof0" "z_prof"))
          (not (app:catalog-object app:*profiles* app:*prof-h*)))
     (setq reason "Choose an elevation profile for this alignment, or change the
elevation source."))
    ((and (/= app:*run* "P2A") (member app:*zsrc* '("z_surf0" "z_surf"))
          (not (app:catalog-object app:*surfaces* app:*surf-h*)))
     (setq reason "Choose an elevation surface, or change the elevation source."))
    ((and (/= app:*run* "PICK") (not (app:selection-filter direction)))
     (setq reason "Tick at least one supported object type for this direction.")))
  (if reason (alert reason) (done_dialog 1)))

;; Do not silently write Arabic using the Windows legacy code page.
(defun app:unicode-ready (/ value)
  (setq value (vl-catch-all-apply 'getvar (list "LISPSYS")))
  (if (and (not (vl-catch-all-error-p value)) (member value '(1 2)))
    T
    (progn
```

```
      (alert (strcat "Arabic menu requires Civil 3D 2021 or newer.\n\n"
                     "1. Type LISPSYS and set it to 1.\n"
                     "2. Close and restart Civil 3D.\n"
                     "3. APPLOAD AlignProfProject_v19_AR.lsp again.\n"
                     "4. Type APMENU.\n\n"
                     "Changing LISPSYS without restarting is not enough."))
      nil)))

(defun app:menu-write-dcl (f / fh line)
  (setq fh (vl-catch-all-apply 'open (list f "w" "utf8-bom")))
  (if (and (not (vl-catch-all-error-p fh)) fh)
    (progn
      (foreach line
        '("apmenu : dialog {"
          "  label = \"مُسقِط المحور <-> القطاع الطولي  v19 (عربي)\";"
          "  : row {"
          "    : column {"
          "  : boxed_column { label = \"اختيار المحور والقطاع الطولي\";"
          "    : popup_list { key = \"p_aln\"; label = \"المحور:\"; width = 55; }"
          "    : popup_list { key = \"p_view\"; label = \"شبكة القطاع الطولي:\"; width =
55; }"
          "    : popup_list { key = \"p_prof\"; label = \"خط المنسوب (بروفايل):\"; width
= 55; }"
          "    : popup_list { key = \"p_surf\"; label = \"السطح (للمناسيب):\"; width =
55; }"
          "  }"
          "  : boxed_radio_column { key = \"r_src\"; label = \"مصدر المنسوب (مسقط ->
قطاع)\";"
          "    : radio_button { key = \"z_obj\"; label = \"منسوب العنصر Z فقط\"; }"
          "    : radio_button { key = \"z_prof0\"; label = \"منسوب العنصر، وإن كان صفراً فمن
خط البروفايل\"; }"
          "    : radio_button { key = \"z_prof\"; label = \"دائماً منسوب خط البروفايل عند
المحطة\"; }"
          "    : radio_button { key = \"z_surf0\"; label = \"منسوب العنصر، وإن كان صفراً فمن
السطح\"; }"
          "    : radio_button { key = \"z_surf\"; label = \"دائماً منسوب السطح عند النقطة\";
}"
          "  }"
          "  : boxed_column { label = \"قيم (اتركها فارغة للافتراضي)\";"
          "    : edit_box { key = \"e_maxoff\"; label = \"أقصى إزاحة عن المحور (فارغ = بلا
حد):\"; edit_width = 10; }"
          "    : edit_box { key = \"e_step\"; label = \"خطوة تقسيم البولي لاين (فارغ = تلقائي):
\"; edit_width = 10; }"
          "    : edit_box { key = \"e_th\"; label = \"ارتفاع النص:\"; edit_width = 10;
}"
          "    : edit_box { key = \"e_off\"; label = \"إزاحة الإسقاط للمسقط (+يمين / -يسار):\";
edit_width = 10; }"
          "    : edit_box { key = \"e_shift\"; label = \"تصحيح المحطة (+/-):\";
edit_width = 10; }"
          "  }"
```

```
            "    }"
            "    : column {"
            "  : boxed_column { label = \"العناصر المراد إسقاطها - ضع علامة\";"
            "    : row {"
            "      : button { key = \"b_types_all\"; label = \"تحديد الكل\"; }"
            "      : button { key = \"b_types_none\"; label = \"إلغاء تحديد الكل\"; }"
            "    }"
            "    : row {"
            "      : column {"
            "        : toggle { key = \"t_point\"; label = \"نقاط\"; }"
            "        : toggle { key = \"t_block\"; label = \"بلوكات\"; }"
            "        : toggle { key = \"t_text\"; label = \"نصوص TEXT / MTEXT\"; }"
            "        : toggle { key = \"t_line\"; label = \"خطوط\"; }"
            "      }"
            "      : column {"
            "        : toggle { key = \"t_poly\"; label = \"2بولي لاين D / 3D\"; }"
            "        : toggle { key = \"t_mleader\"; label = \"تعليقات بأسهم MLeader\"; }"
            "        : toggle { key = \"t_cogo\"; label = \"نقاط COGO (من المسقط فقط)\"; }"
            "        : toggle { key = \"t_pipe\"; label = \"تقاطعات المواسير (من المسقط فقط)\";
}"
            "      }"
            "    }"
            "  }"
            "  : boxed_column { label = \"خيارات\";"
            "    : row {"
            "      : button { key = \"b_options_all\"; label = \"تحديد الكل\"; }"
            "      : button { key = \"b_options_none\"; label = \"إلغاء تحديد الكل\"; }"
            "    }"
            "    : toggle { key = \"t_extend\"; label = \"رسم العناصر خارج شبكة القطاع (تمديد
المقياس)\"; }"
            "    : toggle { key = \"t_labels\"; label = \"كتابة المحطة والمنسوب عند العناصر
المسقطة\"; }"
            "    : toggle { key = \"t_eq\"; label = \"تطبيق معادلات المحطات للمحور\"; }"
            "    : toggle { key = \"t_report\"; label = \"عرض المحطة / الإزاحة / المنسوب لكل
عنصر\"; }"
            "    : toggle { key = \"t_exactso\"; label = \"أقصر مسافة دقيقة (يصلح النقاط القريبة من
PI)\"; }"
            "    : toggle { key = \"t_extaln\"; label = \"تمديد المحور مستقيماً بعد بدايته / نهايته\";
}"
            "    : toggle { key = \"t_nearest\"; label = \"قبول أقرب محطة عند عدم وجود عمودي
(خارج الانكسار)\"; }"
            "    : toggle { key = \"t_witness\"; label = \"رسم خطوط تحقق الإزاحة في المسقط\";
}"
            "    : toggle { key = \"t_lineclip\"; label = \"الخطوط: استبعاد الرؤوس البعيدة عن
الشبكة\"; }"
            "  }"
            "  : boxed_radio_column { key = \"r_run\"; label = \"التشغيل - اضغط موافق ثم اختر
العناصر\";"
            "    : radio_button { key = \"A2P\"; label = \"مسقط -> قطاع طولي (ثم تختار
العناصر)\"; }"
```

```
          "    : radio_button { key = \"PICK\"; label = \"مسقط - قطاع طولي (ثم تنقر المواقع
يشمل Xref)\"; }"
          "    : radio_button { key = \"P2A\"; label = \"قطاع طولي -> مسقط (ثم تختار
العناصر)\"; }"
          "  }"
          "    }"
          "  }"
          "  : row {"
          "    : button { key = \"b_ok\"; label = \"موافق - ابدأ واختر العناصر\"; is_default
= true; }"
          "    : button { key = \"b_help\"; label = \"مساعدة\"; }"
          "    : button { key = \"cancel\"; label = \"إغلاق\"; is_cancel = true; }"
          "  }"
          "}")
        (write-line line fh))
      (close fh)
      f)
    (progn
      (alert "Cannot write the Arabic UTF-8 menu. Set LISPSYS to 1, restart Civil
3D, then APPLOAD this file again. No legacy encoding fallback was used.")
      nil)))

(defun app:menu-help ()
  (alert (strcat
    "مُسقِط المحور <-> القطاع الطولي  v19\n\n"
    "النصوص والتعليقات: نسخ أصلية مع الحفاظ على التنسيق والحجم.\n"
    "رأس السهم الأول؛ تبقى بقية الأسهم بمواقعها النسبية MLeader يستخدم.\n"
    "الطرف الثاني Aligned/Fit نقطة المحاذاة للنص المبرر؛ TEXT:.\n"
    "نقطة إرفاق النص وليست مركز الكتابة الظاهر MTEXT:.\n"
    "ارتفاع النص يخص التسميات الجديدة فقط، لا النصوص المنسوخة.\n\n"
    "طريقة التشغيل:\n"
    "1. اختر المحور وشبكة القطاع من القوائم المنسدلة.\n"
    "2. ضع علامة على أنواع العناصر والخيارات.\n"
    "3. اختر اتجاه التشغيل في مربع التشغيل.\n"
    "4. اضغط موافق، ثم اختر العناصر واضغط Enter.\n\n"
    "يسمح بالنقر على الشاشة << Pick from drawing >> أول خيار في كل قائمة.\n\n"
    "يتم تخطي النقطة إذا:\n"
    " - اختر مصدر منسوب بروفايل أو سطح -> ليس لها منسوب (Z = 0)\n"
    " - أبعد من أقصى إزاحة -> امسح قيمة أقصى إزاحة\n"
    " - داخل Xref -> استخدم وضع النقر على المواقع\n\n"
    "اضغط F2 لرؤية سبب كل تخطٍ.\n"
    "النتائج: PROJ-A2P (أحمر)، PROJ-P2A (أخضر)."))

(defun app:menu-run (command-name / result)
  (setq app:*menu-run* T)
  (setq result (vl-catch-all-apply command-name nil))
  (setq app:*menu-run* nil)
  (if (vl-catch-all-error-p result)
    (progn
      (princ (strcat "\nProjection stopped: " (vl-catch-all-error-message result)))
```

```
      (alert (strcat "Projection stopped with an error:\n\n"
                     (vl-catch-all-error-message result)
                     "\n\nPlease send this message.")))))

(defun c:APMENU (/ dclfile id go res pair)
  (setq dclfile (vl-filename-mktemp "apmenu_" (getvar "TEMPPREFIX") ".dcl"))
  (if (and (app:unicode-ready)
           (app:menu-write-dcl dclfile)
           (setq id (load_dialog dclfile)) (> id 0))
    (progn
      (setq go T)
      (while go
        (if (not (new_dialog "apmenu" id))
          (progn (princ "\nCould not show menu dialog.") (setq go nil))
          (progn
            (foreach pair '(("t_point" . "POINT") ("t_block" . "INSERT")
                            ("t_text" . "TEXT") ("t_mleader" . "MULTILEADER")
("t_line" . "LINE")
                            ("t_poly" . "LWPOLYLINE")
                            ("t_cogo" . "AECC_COGO_POINT") ("t_pipe" . "AECC_PIPE"))
              (set_tile (car pair) (if (member (cdr pair) app:*types*) "1" "0")))
            (set_tile "r_src" app:*zsrc*)
            (app:menu-catalogs)
            (action_tile "p_aln" "(app:alignment-changed $value)")
            (action_tile "p_view" "(app:popup-changed $value app:*views*
'app:*view-h* 11)")
            (action_tile "p_prof" "(app:popup-changed $value app:*profiles*
'app:*prof-h* 12)")
            (action_tile "p_surf" "(app:popup-changed $value app:*surfaces*
'app:*surf-h* 13)")
            (set_tile "t_extend" (if app:*extend* "1" "0"))
            (set_tile "t_labels" (if app:*labels* "1" "0"))
            (set_tile "t_eq" (if app:*eq* "1" "0"))
            (set_tile "t_report" (if app:*report* "1" "0"))
            (set_tile "t_extaln" (if app:*extaln* "1" "0"))
            (set_tile "t_nearest" (if app:*nearest* "1" "0"))
            (action_tile "t_extaln" "(setq app:*extaln* (= $value \"1\"))")
            (set_tile "t_exactso" (if app:*exactso* "1" "0"))
            (action_tile "t_exactso" "(setq app:*exactso* (= $value \"1\"))")
            (action_tile "t_nearest" "(setq app:*nearest* (= $value \"1\"))")
            (set_tile "t_witness" (if app:*witness* "1" "0"))
            (set_tile "t_lineclip" (if app:*lineclip* "1" "0"))
            (action_tile "t_witness" "(setq app:*witness* (= $value \"1\"))")
            (action_tile "t_lineclip" "(setq app:*lineclip* (= $value \"1\"))")
            (set_tile "e_maxoff" app:*maxoff*) (set_tile "e_step" app:*step*)
            (set_tile "e_th" app:*th*) (set_tile "e_off" app:*off*) (set_tile
"e_shift" app:*shift*)
            (action_tile "r_src" "(setq app:*zsrc* $value)(app:source-controls)")
            (action_tile "t_extend" "(setq app:*extend* (= $value \"1\"))")
            (action_tile "t_eq" "(setq app:*eq* (= $value \"1\"))")
```

```
            (action_tile "t_report" "(setq app:*report* (= $value \"1\"))")
            (action_tile "t_labels" "(setq app:*labels* (= $value \"1\"))")
            (action_tile "e_maxoff" "(setq app:*maxoff* $value)")
            (action_tile "e_step" "(setq app:*step* $value)")
            (action_tile "e_th" "(setq app:*th* $value)")
            (action_tile "e_off" "(setq app:*off* $value)")
            (action_tile "e_shift" "(setq app:*shift* $value)")
            (action_tile "t_point" "(app:toggle-type '(\"POINT\") $value)")
            (action_tile "t_block" "(app:toggle-type '(\"INSERT\") $value)")
            (action_tile "t_text" "(app:toggle-type '(\"TEXT\" \"MTEXT\") $value)")
            (action_tile "t_mleader" "(app:toggle-type '(\"MULTILEADER\") $value)")
            (action_tile "t_line" "(app:toggle-type '(\"LINE\") $value)")
            (action_tile "t_poly" "(app:toggle-type '(\"LWPOLYLINE\" \"POLYLINE\")
$value)")
            (action_tile "t_cogo" "(app:toggle-type '(\"AECC_COGO_POINT\") $value)")
            (action_tile "t_pipe" "(app:toggle-type '(\"AECC_PIPE\") $value)")
            (set_tile "r_run" app:*run*)
            (action_tile "r_run" "(setq app:*run* $value)")
            (action_tile "b_types_all" "(app:bulk-types \"1\")")
            (action_tile "b_types_none" "(app:bulk-types \"0\")")
            (action_tile "b_options_all" "(app:bulk-options \"1\")")
            (action_tile "b_options_none" "(app:bulk-options \"0\")")
            (action_tile "b_ok" "(app:menu-validate)")
            (action_tile "b_help" "(done_dialog 4)")
            (action_tile "cancel" "(done_dialog 0)")
            (setq res (start_dialog))
            (cond ((= res 1) (app:menu-run
                               (cdr (assoc app:*run*
                                     '(("A2P" . c:A2P) ("PICK" . c:A2PPICK) ("P2A" .
c:P2A)))))
                  (princ "\nDone. The menu reopens - change options or press
Close."))
                  ((= res 4) (app:menu-help))
                  ((= res 10) (app:menu-pick "alignment"))
                  ((= res 11) (app:menu-pick "view"))
                  ((= res 12) (app:menu-pick "profile"))
                  ((= res 13) (app:menu-pick "surface"))
                  (T (setq go nil))))))
      (unload_dialog id))
    (princ "\nCould not create or load the menu dialog."))
  (if (and dclfile (findfile dclfile)) (vl-file-delete dclfile))
  (princ))

(princ "\nAlignProfProject v19 loaded. Type APMENU (menu), A2P, A2PPICK or P2A.")
(princ)
```