```
File created:     18-Apr-88 14:00:00  {ERINYES}<LISPUSERS>LYRIC>EQUATIONFORMS.;2

  changes to:     (VARS EQUATIONFORMSCOMS)
                  (FNS EQ.SumGroup EQ.IntegralGroup EQ.MakeNSItem)

previous date:    30-Apr-87 10:52:34  {IE:PARC:XEROX}<LISP>LYRIC>LISPUSERS>EQUATIONFORMS.;1

  Read Table:     INTERLISP

     Package:     INTERLISP

       Format:     XCCS
```

```
(RPAQQ EQUATIONFORMSCOMS
       [

;;; ATTACHEDBOX module: Part 1 of 5

                                                            ; Utility functions to manipulate attached regions

        ;; These functions use two sets of coords: global coords in which positions are given w.r.t.  the lower left corner of a box, and side coords in
        ;; which positions are given w.r.t.  a particular side of the box.  For the side coords, the origin is at the point on the side closest to the l.l.
        ;; corner of the box, the x-axis points along the side toward the other end, and the y-axis points away from the box region

        (FNS AB.RealPosition AB.PointPos AB.SidePosition AB.PlaceRegion AB.AdjustToLL AB.OppositeSide
             AB.RegionToBox AB.BoxToRegion AB.RelativePos AB.BiggerRegion AB.Check AB.PositionRegion
             AB.Position2Regions)


;;; EQGROUP module: Part 2 of 5

                                                            ; group equation functions
        (FNS EQ.Group EQ.GroupCreate EQ.Make.group)
                                                            ; set up data definitions
        [P (EQIO.AddType 'group 'EQ.Group 1 '(objectProps (enclosureKind NIL enclosureSide NIL)
                                                     pieceNames
                                                     ("item")
                                                     wholeEditFn EQ.EnclosureEdit initialPropFn EQ.GroupCreate]


;;; specific enclosure data

        (RECORDS EQ.EnclosureData)
        (FNS EQ.AddEnclosure EQ.GetEnclosureData)
                                                            ; set up data for enclosures
        (P (EQ.AddEnclosure 'angles (FUNCTION EQ.angles)
                  "< angle brackets >")
           (EQ.AddEnclosure 'bars (FUNCTION EQ.bars)
                  "| bars |")
           (EQ.AddEnclosure 'braces (FUNCTION EQ.braces)
                  "{ braces }")
           (EQ.AddEnclosure 'brackets (FUNCTION EQ.brackets)
                  "[ brackets ]")
           (EQ.AddEnclosure 'parentheses (FUNCTION EQ.parentheses)
                  "( parentheses )")
           (EQIO.TypeProp 'group 'defaultEnclosure 'brackets))


;;; general enclosure functions

        (FNS EQ.enclosure EQ.EnclosureCreate EQ.EnclosureEdit EQ.EnclosureKind EQ.EnclosureSide)


;;; enclosure form functions

        (FNS EQ.angles EQ.bars EQ.braces EQ.brackets EQ.parentheses EQ.enclosureForm EQ.enclosureWidth)


;;; enclosure drawing functions

        (FNS EQ.DrawAngles EQ.DrawBars EQ.DrawBraces EQ.DrawBrackets EQ.DrawParentheses)


;;; EQMATRIX module: Part 3 of 5

                                                            ; matrix equation functions
        (FNS EQ.Matrix EQ.Make.matrix EQ.layout EQ.MatrixAdd EQ.MatrixChanged EQ.MatrixCreate EQ.MatrixDelete
             EQ.MatrixEdit EQ.MatrixGetMenu EQ.MatrixSelect)
        (INITVARS (EQ.Matrix.MaxPieces 100))
        (GLOBALVARS EQ.Matrix.MaxPieces)
        [P (EQIO.AddType 'matrix 'EQ.Matrix 1 '(objectProps (rows 1 columns 1 enclosureKind NIL enclosureSide
                                                                        NIL)
                                                       variable? T wholeEditFn EQ.MatrixEdit specialSelectFn
                                                       EQ.MatrixSelect initialPropFn EQ.MatrixCreate changeFn
```

```
                                                              EQ.MatrixChanged]
```

;;; EQNFORMS module: Part 4 of 5

```
                                                                          ; fraction
        (FNS EQ.Fraction EQ.Make.fraction)
```

;;; sum group

```
        (FNS EQ.SumGroup EQ.Make.sum EQ.Make.product EQ.Make.union EQ.Make.intersection)
```

;;; integral group

```
        (FNS EQ.IntegralGroup EQ.Make.integral EQ.Make.lineIntegral)
```

;;; super- and sub- scripts

```
        (FNS EQ.Script EQ.Make.sub/superscripts)
```

;;; max/min/limit etc

```
        (FNS EQ.MaxMin EQ.Make.max/min)
```

;;; utilities

```
        (FNS EQ.StreamSize EQ.UseNS? EQ.MakeNSItem)
        (GLOBALVARS EQ.UseNSChars EQ.NSChars)
                                                                          ; EQ.UseNSChars = NIL to use press fonts for display
        [INITVARS EQ.UseNSChars (EQ.NSChars '(SUM ((CLASSIC 24)
                                                 9814)
                                                PRODUCT
                                                ((CLASSIC 24)
                                                 9811)
                                                SUM
                                                ((MODERN 30)
                                                 61306)
                                                PRODUCT
                                                ((MODERN 30)
                                                 61307)
                                                INTERSECTION
                                                ((MODERN 30)
                                                 61270)
                                                UNION
                                                ((MODERN 30)
                                                 61271)
                                                INTEGRAL
                                                ((MODERN 30)
                                                 61301)
                                                LINEINTEGRAL
                                                ((MODERN 30)
                                                 61302]
        [P [EQIO.AddType 'fraction 'EQ.Fraction 2 '(pieceNames ("numerator" "denominator"]
           [EQIO.AddType 'sum 'EQ.SumGroup 3 '(initialData (-2 -2 0)
                                                  pieceNames
                                                  ("index" "limit" "summand"]
           [EQIO.AddType 'product 'EQ.SumGroup 3 '(initialData (-2 -2 0)
                                                      pieceNames
                                                      ("index" "limit" "factor"]
           [EQIO.AddType 'union 'EQ.SumGroup 3 '(initialData (-2 -2 0)
                                                    pieceNames
                                                    ("index" "limit" "set"]
           [EQIO.AddType 'intersection 'EQ.SumGroup 3 '(initialData (-2 -2 0)
                                                           pieceNames
                                                           ("index" "limit" "set"]
           [EQIO.AddType 'integral 'EQ.IntegralGroup 3 '(initialData (-2 -2 0)
                                                            pieceNames
                                                            ("lower limit" "upper limit" "integrand"]
           (EQIO.AddType 'lineIntegral 'EQ.IntegralGroup 3 '(initialData (-2 -2 0)
                                                                pieceNames
                                                                ("lower limit" "upper limit" "integrand")
                                                                menuLabel "line integral"))
           [EQIO.AddType 'sub/superscripts 'EQ.Script 5 '(initialData (0 -1 -1 -1 -1)
                                                             pieceNames
                                                             ("main value" "right subscript" "right
                                                                    superscript" "left subscript" "left
                                                                    superscript"]
           (EQIO.AddType 'max/min 'EQ.MaxMin 3 '(initialData (0 -2 0)
                                                    pieceNames
                                                    ("function" "index" "value")
                                                    menuLabel "max min limit"]
```

;;; EQROOT module: Part 5 of 5

```
        (FNS EQ.Root EQ.Make.root)
        (FNS EQ.DrawRadicalSign)
        (P (EQIO.AddType 'root 'EQ.Root 2 '(pieceNames ("radicand" "index")
                                               initialData
                                               (0 -1])
```

;;; ATTACHEDBOX module: Part 1 of 5

;; Utility functions to manipulate attached regions

;; These functions use two sets of coords: global coords in which positions are given w.r.t. the lower left corner of a box, and side coords in which
;; positions are given w.r.t. a particular side of the box. For the side coords, the origin is at the point on the side closest to the l.l. corner of the box, the
;; x-axis points along the side toward the other end, and the y-axis points away from the box region

```
(DEFINEQ

(AB.RealPosition
  [LAMBDA (pos box side)                                        (* THH " 8-May-85 11:31")

               (* returns position of pt. relative to l.l. corner of box given position relative to specified side of box)

    (SELECTQ side
        ((top bottom)
            [create POSITION
                   XCOORD _ (fetch XCOORD of pos)
                   YCOORD _ (COND
                               ((EQ side 'top)
                                (PLUS (fetch YSIZE of box)
                                      (fetch YCOORD of pos)))
                               (T (MINUS (fetch YCOORD of pos])
        ((left right)
            [create POSITION
                   YCOORD _ (fetch XCOORD of pos)
                   XCOORD _ (COND
                               ((EQ side 'right)
                                (PLUS (fetch XSIZE of box)
                                      (fetch YCOORD of pos)))
                               (T (MINUS (fetch YCOORD of pos])
        NIL])

(AB.PointPos
  [LAMBDA (box side pt)                                         (* THH " 8-May-85 11:34")

               (* returns position of pt relative to side -- pt is an atom specifying relative position along the side)

    (PROG [(horizSide (OR (EQ side 'top)
                          (EQ side 'bottom]
          (RETURN (create POSITION
                         XCOORD _ (SELECTQ pt
                                      (low                      (* lower end of side)
                                           0)
                                      (display                  (* display pt of side)
                                               (COND
                                                  (horizSide 0)
                                                  (T (fetch YDESC of box))))
                                      (center                   (* center of side)
                                              (IQUOTIENT (COND
                                                            (horizSide (fetch XSIZE of box))
                                                            (T (fetch YSIZE of box)))
                                                     2))
                                      (high                     (* upper end of side)
                                            (COND
                                               (horizSide (fetch XSIZE of box))
                                               (T (fetch YSIZE of box))))
                                      NIL)
                         YCOORD _ 0])

(AB.SidePosition
  [LAMBDA (pos box side)                                        (* THH " 8-May-85 11:32")

               (* returns position of pt rel. to side given position rel. to l.l. corner of box)

    (SELECTQ side
        ((top bottom)
            [create POSITION
                   XCOORD _ (fetch XCOORD of pos)
                   YCOORD _ (COND
                               ((EQ side 'top)
                                (DIFFERENCE (fetch YCOORD of pos)
                                       (fetch YSIZE of box)))
                               (T (MINUS (fetch YCOORD of pos])
        ((left right)
            [create POSITION
                   XCOORD _ (fetch YCOORD of pos)
                   YCOORD _ (COND
                               ((EQ side 'right)
                                (DIFFERENCE (fetch XCOORD of pos)
                                       (fetch XSIZE of box)))
                               (T (MINUS (fetch XCOORD of pos])
        NIL])
```

(AB.PlaceRegion

```
  [LAMBDA (mainBox side mainPt addBox addPt gap shift)          (* THH "10-May-85 16:51")

            (* returns placed region relative to l.l. corner of main box when addBox is placed as specified)

    (PROG ((posMainPt (AB.PointPos mainBox side mainPt))
           (opposite (AB.OppositeSide side))
           posAddPt posSide)
          (SETQ posAddPt (create POSITION
                                 XCOORD _ (PLUS (fetch XCOORD of posMainPt)
                                                shift)
                                 YCOORD _ (PLUS (fetch YCOORD of posMainPt)
                                                gap)))
          (SETQ posSide (AB.PointPos addBox opposite addPt))
          (replace XCOORD of posSide with (DIFFERENCE (fetch XCOORD of posAddPt)
                                                (fetch XCOORD of posSide)))
          (replace YCOORD of posSide with (fetch YCOORD of posAddPt))
          (SETQ posSide (AB.AdjustToLL (AB.RealPosition posSide mainBox side)
                                addBox side))
          (RETURN (create REGION
                          LEFT _ (fetch XCOORD of posSide)
                          BOTTOM _ (fetch YCOORD of posSide)
                          WIDTH _ (fetch XSIZE of addBox)
                          HEIGHT _ (fetch YSIZE of addBox])
```

(AB.AdjustToLL

```
  [LAMBDA (pos addBox side)                                     (* THH "10-May-85 14:58")

            (* gets position of l.l. corner of addBox w.r.t. l.l of main box given pos of side opposite side of main box)

    (SELECTQ side
         ((top right)
             NIL)
         (left (replace XCOORD of pos with (DIFFERENCE (fetch XCOORD of pos)
                                                  (fetch XSIZE of addBox))))
         (bottom (replace YCOORD of pos with (DIFFERENCE (fetch YCOORD of pos)
                                                    (fetch YSIZE of addBox))))
         NIL)
    pos])
```

(AB.OppositeSide

```
  [LAMBDA (side)                                                (* THH " 8-May-85 14:11")
    (SELECTQ side
         (top 'bottom)
         (bottom 'top)
         (left 'right)
         (right 'left)
         NIL])
```

(AB.RegionToBox

```
  [LAMBDA (region displayYPos)                                  (* thh%: "13-May-85 10:16")

            (* returns image box corresponding to region whose y display position, relative to
            (0,0) is given by displayYPos)

    (COND
       ((NOT displayYPos)
        (SETQ displayYPos 0)))
    (create IMAGEBOX
           XSIZE _ (fetch WIDTH of region)
           YSIZE _ (fetch HEIGHT of region)
           YDESC _ (DIFFERENCE displayYPos (fetch BOTTOM of region))
           XKERN _ 0])
```

(AB.BoxToRegion

```
  [LAMBDA (box cornerXPos cornerYPos)                           (* thh%: "13-May-85 09:54")

            (* returns region corresponding to image box whose l.l. corner is positioned at cornerXPos, cornerYPos --
            if cornerYPos is NIL then cornerXPos should be a position)

    [COND
       ((NOT cornerYPos)
        (SETQ cornerYPos (fetch YCOORD of cornerXPos))
        (SETQ cornerXPos (fetch XCOORD of cornerXPos]
    (create REGION
           LEFT _ cornerXPos
           BOTTOM _ cornerYPos
           WIDTH _ (fetch XSIZE of box)
           HEIGHT _ (fetch YSIZE of box])
```

(AB.RelativePos

```
  [LAMBDA (region bigRegion yShift)                    (* thh%: "13-May-85 10:27")
                                                       (* returns relative position of l.l. of region w.r.t.
                                                       l.l. of bigRegion)

    (create POSITION
           XCOORD _ (DIFFERENCE (fetch LEFT of region)
                            (fetch LEFT of bigRegion))
           YCOORD _ (PLUS (DIFFERENCE (fetch BOTTOM of region)
                                  (fetch BOTTOM of bigRegion))
                         yShift])
```

(AB.BiggerRegion

```
  [LAMBDA (region extra)                               (* THH "23-May-85 14:03")
                                                       (* creates a region that has extra space all around)
                                                       (* extra should be nonnegative)

    (COND
       ((OR (NOT extra)
            (ZEROP extra))
        region)
       (T (create REGION
                 LEFT _ (DIFFERENCE (fetch LEFT of region)
                                extra)
                 BOTTOM _ (DIFFERENCE (fetch BOTTOM of region)
                                  extra)
                 WIDTH _ (PLUS (fetch WIDTH of region)
                              (TIMES 2 extra))
                 HEIGHT _ (PLUS (fetch HEIGHT of region)
                               (TIMES 2 extra])
```

(AB.Check

```
  [LAMBDA (region regionList side clear)               (* THH "23-May-85 14:02")
                                                       (* moves region away from side if it overlaps any regions in
                                                       regionList)

    (COND
       ((NOT clear)
        (SETQ clear 0)))
    [COND
       ((GREATERP clear 0)
        (SETQ regionList (for r in regionList collect (AB.BiggerRegion r clear]

          (* must go through list repeatedly since moving region could cause it to overlap previous regions)

    [repeatwhile overlap bind overlap
       do (SETQ overlap NIL)
          (for r in regionList do (COND
                                     ((REGIONSINTERSECTP region r)
                                      (SETQ overlap T)
                                      (SELECTQ side
                                          (top (replace BOTTOM of region with (PLUS (fetch BOTTOM of r)
                                                                                  (fetch HEIGHT of r))))
                                          (bottom (replace BOTTOM of region with (DIFFERENCE (fetch BOTTOM
                                                                                                 of r)
                                                                                        (fetch HEIGHT of region))))
                                          (left (replace LEFT of region with (DIFFERENCE (fetch LEFT of r)
                                                                                      (fetch WIDTH of region))))
                                          (right (replace LEFT of region with (PLUS (fetch LEFT of r)
                                                                                 (fetch WIDTH of r))))
                                          (SHOULDNT]
    region])
```

(AB.PositionRegion

```
  [LAMBDA (mainBox addedRegions side mainPt addBox addPt gap shift clear)
                                                       (* thh%: " 9-Jan-86 10:00")

          (* positions addBox next to side of mainBox so that addPt on the added box is at the specified relative position to the mainPt
          on the main box, then moves addBox away from mainBox if necessary to avoid being within distance clear of added regions
          on other sides as specified in addedRegions. Returns (region newAddedRegions) where region is region of added box w.r.t.
          l.l. corner of mainBox and newAddedRegions includes this new box to prevent future additions from overlapping it)

    (PROG ((place (AB.Check (AB.PlaceRegion mainBox side mainPt addBox addPt gap shift)
                         addedRegions side clear)))
          (RETURN (CONS place addedRegions])
```

(AB.Position2Regions

```
  [LAMBDA (mainBox addedRegions side highBox highPt lowBox lowPt highGap lowGap highShift lowShift clear)
```

```
                                                                  (* thh%: " 9-Jan-86 10:00")

          (* positions highBox and lowBox next to side on mainBox -- moves the boxes apart if they are closer than clear apart, and
          moves them away from mainBox if they overlap previous regions as specfied by addedRegions)

          (* returns (highRegion lowRegion newAddedRegions) where regions are w.r.t.
          l.l. corner of mainBox and newAddedRegions includes the added regions)

    (PROG (placeHigh placeLow)
          (SETQ placeHigh (AB.PlaceRegion mainBox side 'high highBox highPt highGap highShift))
          (SETQ placeLow (AB.PlaceRegion mainBox side 'low lowBox lowPt lowGap lowShift))
          (COND
             ((NOT clear)
              (SETQ clear 0)))
          [COND
             ((REGIONSINTERSECTP placeHigh (AB.BiggerRegion placeLow clear))
                                                                  (* move regions apart)
              (PROG (shift totalSize)                             (* if the two added regions overlap then separate them)
                    (SELECTQ side
                        ((top bottom)
                             (SETQ shift (DIFFERENCE (PLUS (fetch LEFT of placeLow)
                                                           (fetch WIDTH of placeLow)
                                                           clear)
                                                (fetch LEFT of placeHigh)))
                             [COND
                                ((GREATERP shift 0)
                                 (SETQ totalSize (PLUS (fetch WIDTH of placeLow)
                                                       (fetch WIDTH of placeHigh)))
                                 [add (fetch LEFT of placeLow)
                                      (MINUS (FIX (PLUS 0.5 (FQUOTIENT (TIMES shift (fetch WIDTH of placeLow))
                                                                   totalSize]
                                 (add (fetch LEFT of placeHigh)
                                      (FIX (PLUS 0.5 (FQUOTIENT (TIMES shift (fetch WIDTH of placeHigh))
                                                            totalSize])
                        ((left right)
                             (SETQ shift (DIFFERENCE (PLUS (fetch BOTTOM of placeLow)
                                                           (fetch HEIGHT of placeLow)
                                                           clear)
                                                (fetch BOTTOM of placeHigh)))
                             [COND
                                ((GREATERP shift 0)
                                 (SETQ totalSize (PLUS (fetch HEIGHT of placeLow)
                                                       (fetch HEIGHT of placeHigh)))
                                 [add (fetch BOTTOM of placeLow)
                                      (MINUS (FIX (PLUS 0.5 (FQUOTIENT (TIMES shift (fetch HEIGHT of placeLow))
                                                                   totalSize]
                                 (add (fetch BOTTOM of placeHigh)
                                      (FIX (PLUS 0.5 (FQUOTIENT (TIMES shift (fetch HEIGHT of placeHigh))
                                                            totalSize])
                        (SHOULDNT]
          (SETQ placeLow (AB.Check placeLow addedRegions side clear))
          (SETQ placeHigh (AB.Check placeHigh addedRegions side clear))
          (RETURN (CONS placeHigh (CONS placeLow addedRegions])

)
```

;;; EQGROUP module: Part 2 of 5

;; group equation functions

```
(DEFINEQ

(EQ.Group
  [LAMBDA (eqnObj imageStream draw?)                              (* THH "12-Jul-85 08:24")
                                                                  (* form function for group -- one argument with enclosure)
    (LET ((innerBox (FS.Box (EQIO.EqnData eqnObj 1)
                            imageStream))
          enclose pos)
         (SETQ enclose (EQ.enclosure innerBox eqnObj imageStream draw?))
         (SETQ pos (CDR enclose))
         (add (fetch YCOORD of pos)
              (fetch YDESC of innerBox))
         (EQIO.MakeSpec (CAR enclose)
                (LIST (EQIO.MakeDataSpec pos])

(EQ.GroupCreate
  [LAMBDA NIL                                                     (* thh%: " 1-Jul-85 14:57")
    (EQ.EnclosureCreate T])

(EQ.Make.group
  [LAMBDA (data enclosureKind enclosureSide fontSpec)             (* thh%: " 9-Jan-86 10:24")
    (EQN.Make 'group (LIST data)
           fontSpec
           (LIST 'enclosureKind enclosureKind 'enclosureSide enclosureSide])
```

```
)

;; set up data definitions

(EQIO.AddType 'group 'EQ.Group 1 '(objectProps (enclosureKind NIL enclosureSide NIL)
                                        pieceNames
                                        ("item")
                                        wholeEditFn EQ.EnclosureEdit initialPropFn EQ.GroupCreate))


;;; specific enclosure data

(DECLARE%: EVAL@COMPILE

(RECORD EQ.EnclosureData (formFn label))
)

(DEFINEQ

(EQ.AddEnclosure
  [LAMBDA (kind formFn label)                                  (* THH "12-Jul-85 08:18")
                                                               (* adds data for new form of enclosure)
    (LET ((enclosures (EQIO.TypeProp 'group 'enclosures))
          (newValue (create EQ.EnclosureData
                            formFn _ formFn
                            label _ label)))
         [COND
            (enclosures (LISTPUT enclosures kind newValue))
            (T (SETQ enclosures (LIST kind newValue]
         (EQIO.TypeProp 'group 'enclosures enclosures)
         (EQIO.TypeProp 'group 'kindMenu NIL])


(EQ.GetEnclosureData
  [LAMBDA (kind)                                               (* THH "12-Jul-85 08:19")
    (LISTGET (EQIO.TypeProp 'group 'enclosures)
           kind])

)

;; set up data for enclosures

(EQ.AddEnclosure 'angles (FUNCTION EQ.angles)
       "< angle brackets >")

(EQ.AddEnclosure 'bars (FUNCTION EQ.bars)
       "| bars |")

(EQ.AddEnclosure 'braces (FUNCTION EQ.braces)
       "{ braces }")

(EQ.AddEnclosure 'brackets (FUNCTION EQ.brackets)
       "[ brackets ]")

(EQ.AddEnclosure 'parentheses (FUNCTION EQ.parentheses)
       "( parentheses )")

(EQIO.TypeProp 'group 'defaultEnclosure 'brackets)


;;; general enclosure functions

(DEFINEQ

(EQ.enclosure
  [LAMBDA (innerBox eqnObj imageStream draw?)                  (* THH "12-Jul-85 08:32")

             (* returns (outerBox . pos) where outerBox is box with enclosures and pos is position of l.l.
             corner of inner box wrt l.l. corner of outer box)
                                                               (* if draw? is non-NIL, draws the enclosures)
    (LET [[kind (OR (EQIO.EqnProperty eqnObj 'enclosureKind)
                    (EQIO.TypeProp 'group 'defaultEnclosure]
          (which (EQIO.EqnProperty eqnObj 'enclosureSide]
         (COND
            ([NOT (OR (EQ which 'left)
                      (EQ which 'right]
             (SETQ which NIL)))
         (LET [(formFn (fetch (EQ.EnclosureData formFn) of (EQ.GetEnclosureData kind]
              (COND
                 (formFn                                       (* note%: use of kind arg allows same formFn to be used for
                                                               different enclosures)
                        (APPLY* formFn innerBox imageStream draw? which kind))
                 (T                                            (* no enclosure so outer box is same as inner box)
                    (CONS innerBox (create POSITION
                                          XCOORD _ 0
                                          YCOORD _ 0])
```

```
(EQ.EnclosureCreate
  [LAMBDA (getWhich?)                                    (* thh%: " 1-Jul-85 14:46")
                                                         (* returns prop list describing desired enclosure)
                                                         (* if getWhich? is non-NIL also asks for side specification for
                                                         enclosure)
    (LET ((kind (EQ.EnclosureKind))
          which)
         (COND
            (getWhich? (SETQ which (EQ.EnclosureSide))
                   (LIST 'enclosureKind kind 'enclosureSide which))
            (T (LIST 'enclosureKind kind])

(EQ.EnclosureEdit
  [LAMBDA (eqnObj)                                       (* thh%: " 1-Jul-85 14:55")
                                                         (* allows type of enclosure to be changed)
                                                         (* returns non-NIL if object changed)
    (LET ((editMenu (EQIO.TypeProp 'group 'editMenu))
          newValue)
         (COND
            ((NOT (type? MENU editMenu))
             [SETQ editMenu (create MENU
                                   CENTERFLG _ T
                                   TITLE _ "change what?"
                                   ITEMS _ '(("symbol" 'kind)
                                             ("which side" 'side]
             (EQIO.TypeProp 'group 'editMenu editMenu)))
         (SELECTQ (MENU editMenu)
             (kind (COND
                      ((SETQ newValue (EQ.EnclosureKind))
                       (EQIO.EqnProperty eqnObj 'enclosureKind newValue)
                       T)
                      (T NIL)))
             (side (COND
                      ((SETQ newValue (EQ.EnclosureSide))
                       (EQIO.EqnProperty eqnObj 'enclosureSide newValue)
                       T)
                      (T NIL)))
             NIL])

(EQ.EnclosureKind
  [LAMBDA NIL                                            (* THH "12-Jul-85 08:39")
                                                         (* gets desired kind of enclosure)
    (LET [(kindMenu (EQIO.TypeProp 'group 'kindMenu]
         (COND
            ((NOT (type? MENU kindMenu))
             (SETQ kindMenu (create MENU
                                   CENTERFLG _ T
                                   ITEMS _ [LET [(enclosures (EQIO.TypeProp 'group 'enclosures]
                                                (while enclosures bind kind data
                                                   collect (SETQ kind (CAR enclosures))
                                                          (SETQ data (CADR enclosures))
                                                          (SETQ enclosures (CDDR enclosures))
                                                          (LIST (fetch (EQ.EnclosureData label) of data)
                                                                (KWOTE kind]
                                   TITLE _ "enclosures"))
             (EQIO.TypeProp 'group 'kindMenu kindMenu)))
         (MENU kindMenu])

(EQ.EnclosureSide
  [LAMBDA NIL                                            (* thh%: " 1-Jul-85 14:48")
                                                         (* gets desired side for enclosure)
    (LET [(whichMenu (EQIO.TypeProp 'group 'whichMenu]
         (COND
            ((NOT (type? MENU whichMenu))
             (SETQ whichMenu (create MENU
                                    ITEMS _ '(left right both)
                                    TITLE _ "Which side?"))
             (EQIO.TypeProp 'group 'whichMenu whichMenu)))
         (MENU whichMenu])
)

;;; enclosure form functions

(DEFINEQ

(EQ.angles
  [LAMBDA (innerBox imageStream draw? which)             ; Edited 21-Apr-87 09:00 by thh:
    (LET ((size (EQ.StreamSize imageStream))
          Hgap Hex Vgap Vex width height descent spacing overlap)
         (SETQ Hex size)
         (SETQ Vgap size)
         (SETQ Vex Hex)
         (SETQ height (PLUS (fetch YSIZE of innerBox)
                            (TIMES 2 Vgap)))
         (SETQ width (EQ.enclosureWidth size height))
         (SETQ Hgap (MAX (TIMES 3 size)
                         (IQUOTIENT height 5)))
         [SETQ spacing (PLUS (fetch XSIZE of innerBox)
                             (TIMES 2 (PLUS Hgap width]
         (SETQ descent (PLUS (fetch YDESC of innerBox)
                             Vgap))
         (SETQ overlap (MAX (TIMES 2 size)
                            (IQUOTIENT (PLUS Hgap width)
                                   2)))

          (* * draw angle brackets if requested)

         (COND
            (draw? (EQ.DrawAngles height descent spacing Hex width overlap imageStream which)))

          (* * determine outer box and position of inner box)

         (EQ.enclosureForm spacing Hex height Vex descent width NIL Hgap Vgap])
```

```
(EQ.bars
  [LAMBDA (innerBox imageStream draw? which)                     ; Edited 21-Apr-87 09:00 by thh:
    (LET ((size (EQ.StreamSize imageStream))
          Hgap Hex Vgap Vex width height descent spacing)
         (SETQ Hgap (TIMES 2 size))
         (SETQ Hex size)
         (SETQ Vgap Hgap)
         (SETQ Vex Hex)
         (SETQ height (PLUS (fetch YSIZE of innerBox)
                            (TIMES 2 Vgap)))
         (SETQ width (EQ.enclosureWidth size height))
         [SETQ spacing (PLUS (fetch XSIZE of innerBox)
                             (TIMES 2 (PLUS Hgap width]
         (SETQ descent (PLUS (fetch YDESC of innerBox)
                             Vgap))

          (* * draw bars if requested)

         (COND
            (draw? (EQ.DrawBars height descent spacing Hex width imageStream which)))

          (* * determine outer box and position of inner box)

         (EQ.enclosureForm spacing Hex height Vex descent width NIL Hgap Vgap])
```

```
(EQ.braces
  [LAMBDA (innerBox imageStream draw? which)                     ; Edited 21-Apr-87 09:01 by thh:
    (LET ((size (EQ.StreamSize imageStream))
          Hgap Hex Vgap Vex width height descent spacing overlap point space extra)
         (SETQ Vgap (TIMES 3 size))
         (SETQ Hex size)
         (SETQ Vex Hex)
         (SETQ height (PLUS (fetch YSIZE of innerBox)
                            (TIMES 2 Vgap)))
         (SETQ width (EQ.enclosureWidth size height))
         (add height (TIMES 2 width))
         (SETQ descent (PLUS (fetch YDESC of innerBox)
                             Vgap width))
         (SETQ Hgap (MAX (TIMES 4 size)
                         (IQUOTIENT height 5)))
         [SETQ spacing (PLUS (fetch XSIZE of innerBox)
                             (TIMES 2 (PLUS Hgap width]
         (SETQ overlap (PLUS Hgap width))
         (SETQ point (PLUS size (IQUOTIENT overlap 2)))
         (SETQ space (IQUOTIENT overlap 2))
         (SETQ extra space)

          (* * draw braces if requested)

         (COND
            (draw? (EQ.DrawBraces height descent spacing Hex width overlap extra point space imageStream which))
            )

          (* * determine outer box and position of inner box)

         (EQ.enclosureForm spacing Hex height Vex descent width T Hgap Vgap])
```

```
(EQ.brackets
  [LAMBDA (innerBox imageStream draw? which)                     ; Edited 21-Apr-87 09:02 by thh:
    (LET ((size (EQ.StreamSize imageStream))
          Hgap Hex Vgap Vex width height descent spacing overlap)
         (SETQ Hgap (TIMES 3 size))
         (SETQ Hex size)
         (SETQ Vgap Hgap)
         (SETQ Vex Hex)
         (SETQ height (PLUS (fetch YSIZE of innerBox)
                            (TIMES 2 Vgap)))
         (SETQ width (EQ.enclosureWidth size height))
         (add height (TIMES 2 width))
         [SETQ spacing (PLUS (fetch XSIZE of innerBox)
                             (TIMES 2 (PLUS Hgap width]
         (SETQ descent (PLUS (fetch YDESC of innerBox)
                             Vgap width))
         (SETQ overlap (MAX (TIMES 3 size)
                            (PLUS Hgap width)))

          (* * draw brackets if requested)

         (COND
            (draw? (EQ.DrawBrackets height descent spacing Hex width overlap imageStream which)))

          (* * determine outer box and position of inner box)

         (EQ.enclosureForm spacing Hex height Vex descent width T Hgap Vgap])
```

(EQ.parentheses

```
  [LAMBDA (innerBox imageStream draw? which)                    ; Edited 21-Apr-87 09:03 by thh:
    (LET ((size (EQ.StreamSize imageStream))
          Hgap Hex Vgap Vex width height descent spacing overlap)
         (SETQ Vgap (TIMES 2 size))
         (SETQ Hex size)
         (SETQ Vex Hex)
         (SETQ height (PLUS (fetch YSIZE of innerBox)
                            (TIMES 2 Vgap)))
         (SETQ width (EQ.enclosureWidth size height))
         (add height (TIMES 2 width))
         (SETQ descent (PLUS (fetch YDESC of innerBox)
                             Vgap width))
         (SETQ Hgap (MAX (TIMES 2 size)
                         (IQUOTIENT height 8)))
         [SETQ spacing (PLUS (fetch XSIZE of innerBox)
                             (TIMES 2 (PLUS Hgap width]
         (SETQ overlap (MAX (TIMES 3 size)
                            (IQUOTIENT (PLUS Hgap width)
                                   2)))

          (* * draw braces if requested)

         (COND
            (draw? (EQ.DrawParentheses height descent spacing Hex width overlap imageStream which)))

          (* * determine outer box and position of inner box)

         (EQ.enclosureForm spacing Hex height Vex descent width T Hgap Vgap])
```

(EQ.enclosureForm

```
  [LAMBDA (spacing Hex height Vex descent width verticalWidth? Hgap Vgap)
                                                                ; Edited 21-Apr-87 08:59 by thh:

          (* computes outer box and position of inner box from parameters --
          verticalWidth? non-NIL means enclosure wraps under the box)

    (CONS (create IMAGEBOX
                 XSIZE _ (PLUS spacing (TIMES 2 Hex))
                 YSIZE _ (PLUS height (TIMES 2 Vex))
                 YDESC _ (PLUS descent Vex)
                 XKERN _ 0)
          (create POSITION
                 XCOORD _ (PLUS Hgap width Hex)
                 YCOORD _ (PLUS Vgap (COND
                                        (verticalWidth? width)
                                        (T 0))
                                Vex])
```

(EQ.enclosureWidth

```
  [LAMBDA (size height)                                         (* THH "16-Jul-85 09:15")
    (MAX size (IQUOTIENT height 100])
)

;;; enclosure drawing functions

(DEFINEQ
(EQ.DrawAngles
  [LAMBDA (height descent spacing xShift width overlap imageStream which)
                                                                       (* thh%: "19-Aug-85 09:49")
                                                                       (* draws specified angle brackets on imageStream)
    (LET ((halfWidth (IQUOTIENT width 2))
          (halfWidth1 (IQUOTIENT (SUB1 width)
                              2))
          (lowerX (PLUS (DSPXPOSITION NIL imageStream)
                        xShift))
          (lowerY (DIFFERENCE (DSPYPOSITION NIL imageStream)
                         descent))
          top middle left right)
         (SETQ top (PLUS lowerY (SUB1 height)))
         (SETQ middle (PLUS lowerY (IQUOTIENT (SUB1 height)
                                          2)))

          (* * left angle bracket)

         (COND
            ((NOT (EQ which 'right))
             (SETQ left (PLUS lowerX overlap halfWidth))
             (DRAWLINE left lowerY (PLUS lowerX halfWidth)
                    middle width NIL imageStream)
             (DRAWLINE (PLUS lowerX halfWidth)
                    middle left top width NIL imageStream)))

          (* * right angle bracket)

         (COND
            ((NOT (EQ which 'left))
             [SETQ right (PLUS lowerX (SUB1 spacing)
                               (MINUS (PLUS overlap halfWidth1]
             (DRAWLINE right lowerY (PLUS lowerX (SUB1 spacing)
                                          (MINUS halfWidth1))
                    middle width NIL imageStream)
             (DRAWLINE (PLUS lowerX (SUB1 spacing)
                             (MINUS halfWidth1))
                    middle right top width NIL imageStream])

(EQ.DrawBars
  [LAMBDA (height descent spacing xShift width imageStream which)
                                                                       (* thh%: "19-Aug-85 09:53")
                                                                       (* draws specified vertical bar on imageStream)
    (PROG ((halfWidth (IQUOTIENT width 2))
           (halfWidth1 (IQUOTIENT (SUB1 width)
                              2))
           (lowerX (PLUS (DSPXPOSITION NIL imageStream)
                         xShift))
           (lowerY (DIFFERENCE (DSPYPOSITION NIL imageStream)
                          descent)))

          (* * left bar)

          (COND
             ((NOT (EQ which 'right))
              (MOVETO (PLUS lowerX halfWidth1)
                     lowerY imageStream)
              (RELDRAWTO 0 (SUB1 height)
                     width NIL imageStream)))

          (* * right bar)

          (COND
             ((NOT (EQ which 'left))
              (MOVETO (DIFFERENCE (PLUS lowerX spacing)
                             halfWidth)
                     lowerY imageStream)
              (RELDRAWTO 0 (SUB1 height)
                     width NIL imageStream])

(EQ.DrawBraces
  [LAMBDA (height descent spacing xShift width overlap extra point space imageStream which)
                                                                       (* THH "16-Jul-85 09:20")
                                                                       (* draws specified brace on imageStream)
    (LET ((halfWidth (IQUOTIENT width 2))
          (halfWidth1 (IQUOTIENT (SUB1 width)
                              2))
          (lowerX (PLUS (DSPXPOSITION NIL imageStream)
                        xShift))
          (lowerY (DIFFERENCE (DSPYPOSITION NIL imageStream)
                         descent))
          top middle left1 left2 right1 right2)
         (SETQ top (PLUS lowerY (SUB1 height)))
         (SETQ middle (PLUS lowerY (IQUOTIENT (SUB1 height)
                                         2)))

          (* * left brace)

         (COND
            ((NOT (EQ which 'right))
             (SETQ left1 (PLUS lowerX (SUB1 overlap)))
             (SETQ left2 (PLUS lowerX (SUB1 point)))
             (DRAWCURVE (LIST (create POSITION
                                      XCOORD _ left1
                                      YCOORD _ lowerY)
                              (create POSITION
                                      XCOORD _ left2
                                      YCOORD _ (PLUS lowerY extra))
                              (create POSITION
                                      XCOORD _ left2
                                      YCOORD _ (DIFFERENCE middle (SUB1 space)))
                              (create POSITION
                                      XCOORD _ lowerX
                                      YCOORD _ middle))
                    NIL width NIL imageStream)
             (DRAWCURVE (LIST (create POSITION
                                      XCOORD _ lowerX
                                      YCOORD _ middle)
                              (create POSITION
                                      XCOORD _ left2
                                      YCOORD _ (PLUS middle (SUB1 space)))
                              (create POSITION
                                      XCOORD _ left2
                                      YCOORD _ (DIFFERENCE top extra))
                              (create POSITION
                                      XCOORD _ left1
                                      YCOORD _ top))
                    NIL width NIL imageStream)))

          (* * right brace)

         (COND
            ((NOT (EQ which 'left))
             [SETQ right1 (PLUS lowerX (SUB1 spacing)
                                (MINUS (SUB1 overlap]
             [SETQ right2 (PLUS lowerX (SUB1 spacing)
                                (MINUS (SUB1 point]
             (DRAWCURVE (LIST (create POSITION
                                      XCOORD _ right1
                                      YCOORD _ lowerY)
                              (create POSITION
                                      XCOORD _ right2
                                      YCOORD _ (PLUS lowerY extra))
                              (create POSITION
                                      XCOORD _ right2
                                      YCOORD _ (DIFFERENCE middle (SUB1 space)))
                              (create POSITION
                                      XCOORD _ (PLUS lowerX (SUB1 spacing))
                                      YCOORD _ middle))
                    NIL width NIL imageStream)
             (DRAWCURVE (LIST (create POSITION
                                      XCOORD _ (PLUS lowerX (SUB1 spacing))
                                      YCOORD _ middle)
                              (create POSITION
                                      XCOORD _ right2
                                      YCOORD _ (PLUS middle (SUB1 space)))
                              (create POSITION
                                      XCOORD _ right2
                                      YCOORD _ (DIFFERENCE top extra))
                              (create POSITION
                                      XCOORD _ right1
                                      YCOORD _ top))
                    NIL width NIL imageStream])
```

(**EQ.DrawBrackets**

```
  [LAMBDA (height descent spacing xShift width overlap imageStream which)
                                                                        (* THH "12-Jul-85 09:06")
                                                                        (* draws specified bracket on imageStream)
    (PROG ((halfWidth (IQUOTIENT width 2))
           (halfWidth1 (IQUOTIENT (SUB1 width)
                              2))
           (lowerX (PLUS (DSPXPOSITION NIL imageStream)
                         xShift))
           (lowerY (DIFFERENCE (DSPYPOSITION NIL imageStream)
                          descent)))

           (* * left bracket)

           (COND
              ((NOT (EQ which 'right))
              (MOVETO (PLUS lowerX halfWidth1)
                     lowerY imageStream)
              (RELDRAWTO 0 (SUB1 height)
                     width NIL imageStream)
              (RELMOVETO 0 (MINUS halfWidth)
                     imageStream)
              (RELDRAWTO overlap 0 width NIL imageStream)
              (MOVETO (PLUS lowerX (SUB1 width))
                     (PLUS lowerY halfWidth1)
                     imageStream)
              (RELDRAWTO overlap 0 width NIL imageStream)))

          (* * right bracket)

          (COND
             ((NOT (EQ which 'left))
              (MOVETO (DIFFERENCE (PLUS lowerX spacing)
                             (PLUS overlap width))
                     (PLUS lowerY halfWidth1)
                     imageStream)
              (RELDRAWTO overlap 0 width NIL imageStream)
              (RELMOVETO halfWidth1 (MINUS halfWidth1)
                     imageStream)
              (RELDRAWTO 0 (SUB1 height)
                     width NIL imageStream)
              (RELMOVETO 0 (MINUS halfWidth)
                     imageStream)
              (RELDRAWTO (MINUS overlap)
                     0 width NIL imageStream])
```

(**EQ.DrawParentheses**

```
  [LAMBDA (height descent spacing xShift width overlap imageStream which)
                                                                         (* THH "12-Jul-85 09:06")
                                                                         (* draws specified parenthesis on imageStream)
    (LET ((halfWidth (IQUOTIENT width 2))
          (halfWidth1 (IQUOTIENT (SUB1 width)
                             2))
          (lowerX (PLUS (DSPXPOSITION NIL imageStream)
                        xShift))
          (lowerY (DIFFERENCE (DSPYPOSITION NIL imageStream)
                         descent))
          top middle left right)
         (SETQ top (PLUS lowerY (SUB1 height)))
         (SETQ middle (PLUS lowerY (IQUOTIENT (SUB1 height)
                                          2)))

          (* * left parenthesis)

         (COND
            ((NOT (EQ which 'right))
             (SETQ left (PLUS lowerX overlap halfWidth))
             (DRAWCURVE (LIST (create POSITION
                                     XCOORD _ left
                                     YCOORD _ lowerY)
                              (create POSITION
                                     XCOORD _ (PLUS lowerX halfWidth)
                                     YCOORD _ middle)
                              (create POSITION
                                     XCOORD _ left
                                     YCOORD _ top))
                    NIL width NIL imageStream)))

          (* * right parenthesis)

         (COND
            ((NOT (EQ which 'left))
             [SETQ right (PLUS lowerX (SUB1 spacing)
                               (MINUS (PLUS overlap halfWidth1]
             (DRAWCURVE (LIST (create POSITION
                                     XCOORD _ right
                                     YCOORD _ lowerY)
                              (create POSITION
                                     XCOORD _ (PLUS lowerX (SUB1 spacing)
                                                    (MINUS halfWidth1))
                                     YCOORD _ middle)
                              (create POSITION
                                     XCOORD _ right
                                     YCOORD _ top))
                    NIL width NIL imageStream])
)
```

;;; EQMATRIX module: Part 3 of 5

;; matrix equation functions

```
(DEFINEQ

(EQ.Matrix
  [LAMBDA (eqnObj imageStream draw?)                                          (* THH "12-Jul-85 09:50")
                                                                              (* form function for matrix)
                                                                              (* this equation form allows a variable number of parts arranged
    and stored in rows)
    (LET ((layout (EQ.layout eqnObj imageStream))
          enclose specs)
         (SETQ enclose (EQ.enclosure (EQIO.GetBox layout)
                              eqnObj imageStream draw?))

          (* * enclose of form (outerBox . pos) -- must now shift positions of individual pieces in layout
          (assumes no selection region in layout -- it would also need to be shifted))

         (SETQ specs (EQIO.GetDataSpecList layout))
         (for dataSpec in specs bind pos (xShift _ (fetch (POSITION XCOORD) of (CDR enclose)))
                                      (yShift _ (fetch (POSITION YCOORD) of (CDR enclose)))
            do (SETQ pos (EQIO.GetDataPosition dataSpec))
               (add (fetch (POSITION XCOORD) of pos)
                    xShift)
               (add (fetch (POSITION YCOORD) of pos)
                    yShift))
         (EQIO.MakeSpec (CAR enclose)
                specs])

(EQ.Make.matrix
  [LAMBDA (rows columns dataList enclosureKind enclosureSide fontSpec)
                                                                              (* thh%: " 9-Jan-86 10:23")

          (* * may want to call initial prop fn???)

    (LET ((numPieces (TIMES rows columns)))
         (EQN.Make 'matrix dataList fontSpec (LIST 'numPieces numPieces 'rows rows 'columns columns
                                                   'enclosureKind enclosureKind 'enclosureSide enclosureSide])

(EQ.layout
  [LAMBDA (eqnObj imageStream)                                                (* THH "12-Jul-85 10:26")
                                                                              (* form function for table of parts)
                                                                              (* this equation form allows a variable number of parts arranged
    and stored in rows)

          (* layout defined by rowGap -- distance between rows, colGap --
          distance between columns, shift -- distance of center above baseline)

    (LET ((size (EQ.StreamSize imageStream))
          (columns (EQIO.EqnProperty eqnObj 'columns))
          (rows (EQIO.EqnProperty eqnObj 'rows))
          (fontSpec (FONTCREATE (COND
                                   ((EQIO.EqnProperty eqnObj 'fontSpec))
                                   (T DEFAULTFONT))
                           NIL NIL NIL imageStream))
          layoutBox boxList colData rowData colGap rowGap shift)

          (* * set quantities that define layout)

         (SETQ shift (IQUOTIENT (FONTPROP fontSpec 'ASCENT)
                            2))
         (SETQ colGap (STRINGWIDTH "  " fontSpec))
         (SETQ rowGap colGap)

          (* * determine overall box)

         [COND
            ((OR (ILEQ columns 0)
                 (ILEQ rows 0))
             (SETQ layoutBox (create IMAGEBOX
                                     XSIZE _ 0
                                     YSIZE _ 0)))
            (T (SETQ boxList (for piece in (EQIO.EqnDataList eqnObj) collect (FS.Box piece imageStream)))
               (SETQ colData (ARRAY columns 'FIXP 0))
               (SETQ rowData (ARRAY rows))
               (for row from 1 to rows bind (boxes _ boxList)
                                            rAscent rDesc
                  do (SETQ rAscent 0)
                     (SETQ rDesc 0)
                     [for col from 1 to columns bind b do (SETQ b (CAR boxes))
                                                          (SETQ boxes (CDR boxes))
                                                          [SETQ rAscent (MAX rAscent (DIFFERENCE (fetch YSIZE
                                                                                                        of b)
                                                                                                 (fetch YDESC
                                                                                                    of b]
                                                          (SETQ rDesc (MAX rDesc (fetch YDESC of b)))
                                                          (SETA colData col (MAX (ELT colData col)
                                                                                 (fetch XSIZE of b]
                    (SETA rowData row (CONS (PLUS rAscent rDesc)
                                            rDesc)))
              (SETQ layoutBox (create IMAGEBOX
                                      XSIZE _ (PLUS (for col from 1 to columns sum (ELT colData col))
                                                    (TIMES (SUB1 columns)
                                                           colGap))
                                      YSIZE _ (PLUS (for row from 1 to rows sum (CAR (ELT rowData row)))
                                                    (TIMES (SUB1 rows)
                                                           rowGap]
          (replace YDESC of layoutBox with (DIFFERENCE (IQUOTIENT (fetch YSIZE of layoutBox)
                                                                  2)
                                                       shift))
          (replace XKERN of layoutBox with 0)

           (* * return overall box and individual positions of all the parts)

          (LET [(xLow 0)
                (yHigh (fetch YSIZE of layoutBox))
                (colPos (ARRAY columns 'FIXP]                          (* (xLow,yHigh) is position of upper left corner of block of matrix
               parts)
               [COND
                  ((IGREATERP columns 0)                                (* set colPos to horizontal position of left edge of each column)
                   (SETA colPos 1 xLow)
                   (for col from 2 to columns do (SETA colPos col (PLUS (ELT colPos (SUB1 col))
                                                                         colGap
                                                                         (ELT colData (SUB1 col]
               (EQIO.MakeSpec layoutBox
                      (COND
                         [(AND (IGREATERP columns 0)
                               (IGREATERP rows 0))
                          (for row from 1 to rows bind rowValue (boxes _ boxList)
                                                       (rowPos _ (PLUS yHigh rowGap))
                             join [SETQ rowPos (DIFFERENCE rowPos (PLUS rowGap (CAR (SETQ rowValue
                                                                                      (ELT rowData row]
                                                                    (* rowPos is vertical position of bottom of row)
                                   (for col from 1 to columns bind b
                                      collect (SETQ b (CAR boxes))
                                              (SETQ boxes (CDR boxes))
                                              (EQIO.MakeDataSpec (create POSITION
                                                                        XCOORD _
                                                                        (PLUS (ELT colPos col)
                                                                              (IQUOTIENT (DIFFERENCE (ELT colData col
                                                                                                          )
                                                                                                     (fetch XSIZE
                                                                                                        of b))
                                                                                         2))
                                                                        YCOORD _ (PLUS rowPos (CDR rowValue]
                         (T                                         (* no pieces in this layout)
                            NIL])


(EQ.MatrixAdd
  [LAMBDA (eqnObj which place window)                               (* thh%: " 3-Jun-85 10:19")
                                                                    (* adds new row/column to matrix)
                                                                    (* currently copies lists when adding new data instead of
                                                                    modifying the original lists)
    (PROG ((rows (EQIO.EqnProperty eqnObj 'rows))
           (columns (EQIO.EqnProperty eqnObj 'columns))
           (dataList (EQIO.EqnDataList eqnObj))
           newData tempData firstPiece continueFlg)
          (SELECTQ which
               (row (SETQ newData (EQN.DefaultData (EQIO.EqnType eqnObj)
                                         (EQIO.EqnProperty eqnObj 'fontSpec)
                                         columns))                  (* insert newData after the place*columns piece)
                    (SETQ tempData (for d in dataList as i from 1 to (TIMES place columns)
                                      collect (SETQ dataList (CDR dataList))
                                              d))
                    (EQIO.SetDataList eqnObj (APPEND tempData newData dataList))
                    (add rows 1)
                    (EQIO.EqnProperty eqnObj 'rows rows)
                    (SETQ firstPiece (ADD1 (TIMES columns place)))
                    [SETQ continueFlg (CONS 1 (TIMES columns (ADD1 place])
               (column (SETQ newData (EQN.DefaultData (EQIO.EqnType eqnObj)
                                            (EQIO.EqnProperty eqnObj 'fontSpec)
                                            rows))
                       [EQIO.SetDataList eqnObj
                              (for i from 1 to rows bind newD
                                 join (SETQ newD (CAR newData))
                                      (SETQ newData (CDR newData))
                                      (APPEND (for j from 1 to place bind d collect (SETQ d (CAR dataList))
                                                                                    (SETQ dataList (CDR dataList))
                                                                                    d)
                                             (LIST newD)
                                             (for j from (ADD1 place) to columns bind d collect (SETQ d (CAR dataList
                                                                                                            ))
                                                                                                (SETQ dataList
                                                                                                 (CDR dataList))
```

```
                                                                                                   d]
                      (add columns 1)
                      (EQIO.EqnProperty eqnObj 'columns columns)
                      (SETQ firstPiece (ADD1 place))
                      (SETQ continueFlg (LIST columns)))
              (ERROR "EQ.MatrixAdd: invalid arg for which = " which))
          (EQIO.NumPieces eqnObj (TIMES rows columns))
          (EQN.StartEdit eqnObj window firstPiece continueFlg 'PENDINGDEL])
```

(EQ.MatrixChanged

```
  [LAMBDA (eqnObj)                                                    (* thh%: "31-May-85 14:21")
                                                                      (* called when number of pieces in matrix changed)

    (EQIO.EqnProperty eqnObj 'rowMenu NIL)
    (EQIO.EqnProperty eqnObj 'colMenu NIL])
```

(EQ.MatrixCreate

```
  [LAMBDA NIL                                                         (* THH "12-Jul-85 10:04")
                                                                      (* allows user to specify initial number of rows and columns in a
    new matrix)
    (LET (rows columns numPieces retry)
         (repeatwhile (GREATERP numPieces EQ.Matrix.MaxPieces)
            do (COND
                  (retry (CLRPROMPT)
                         (PROMPTPRINT "Too many matrix elements: " numPieces " [max allowed is "
                                EQ.Matrix.MaxPieces "]")))
               (SETQ rows (MAX (RNUMBER "How many rows for matrix?")
                               0))
               (SETQ columns (MAX (RNUMBER "How many columns for matrix?")
                                  0))
               (SETQ numPieces (TIMES rows columns))
               (SETQ retry T))
         (LIST 'numPieces numPieces 'rows rows 'columns columns])
```

(EQ.MatrixDelete

```
  [LAMBDA (eqnObj which place)                                        (* thh%: " 3-Jun-85 11:56")
                                                                      (* deletes row/column from matrix)
    (COND
       ((IGREATERP place 0)
        (PROG ((rows (EQIO.EqnProperty eqnObj 'rows))
               (columns (EQIO.EqnProperty eqnObj 'columns))
               (dataList (EQIO.EqnDataList eqnObj)))
              (SELECTQ which
                  (row [EQIO.SetDataList eqnObj (for d in dataList as i from 1
                                                   bind (start _ (TIMES (SUB1 place)
                                                                        columns))
                                                        (stop _ (TIMES place columns))
                                                   collect d unless (AND (IGREATERP i start)
                                                                         (ILEQ i stop]
                       (add rows -1)
                       (EQIO.EqnProperty eqnObj 'rows rows))
                  (column (EQIO.SetDataList eqnObj (for d in dataList bind (j _ 0)
                                                      eachtime (COND
                                                                  ((IGEQ j columns)
                                                                   (SETQ j 0)))
                                                               (add j 1)
                                                      collect d unless (IEQP j place)))
                          (add columns -1)
                          (EQIO.EqnProperty eqnObj 'columns columns))
                  (ERROR "EQ.MatrixDelete: invalid arg for which = " which))
              (EQIO.NumPieces eqnObj (TIMES rows columns])
```

(EQ.MatrixEdit

```
  [LAMBDA (eqnObj window button)                                      (* THH "12-Jul-85 10:08")
                                                                      (* adds and removes rows and columns from matrix)
                                                                      (* returns non-NIL if eqnObj modified)
    (COND
       [(EQ (EQIO.EqnType eqnObj)
            'matrix)
        (LET ((editMenu (EQIO.TypeProp 'matrix 'editMenu))
              action place)
             (COND
                ((NOT (type? MENU editMenu))
                 [SETQ editMenu (create MENU
                                       CENTERFLG _ T
                                       ITEMS _ '(("add before col" 'beforeCol)
                                                 ("add before row" 'beforeRow)
                                                 ("add after col" 'afterCol)
                                                 ("add after row" 'afterRow)
                                                 ("DELETE column" 'delCol)
                                                 ("DELETE row" 'delRow)
                                                 ("[change enclosure]" 'enclosure]
                 (EQIO.TypeProp 'matrix 'editMenu editMenu)))
             (SETQ action (MENU editMenu))
             (COND
                ((EQ action 'enclosure)
                 (EQ.EnclosureEdit eqnObj))
                (T                                                      (* edit layout)
                   (SETQ place (SELECTQ action
                                   ((beforeCol afterCol delCol)
                                        (COND
                                           [(IGREATERP (EQIO.EqnProperty eqnObj 'columns)
                                                   0)
                                            (MENU (EQ.MatrixGetMenu eqnObj 'column]
                                           ((EQ action 'beforeCol)
                                            1)
                                           (T 0)))
                                   ((beforeRow afterRow delRow)
                                        (COND
                                           [(IGREATERP (EQIO.EqnProperty eqnObj 'rows)
                                                   0)
                                            (MENU (EQ.MatrixGetMenu eqnObj 'row]
                                           ((EQ action 'beforeRow)
                                            1)
                                           (T 0)))
                                   NIL))
                   (COND
                      (place [COND
                                ((EQ action 'beforeCol)
                                 (add place -1)
                                 (SETQ action 'afterCol))
                                ((EQ action 'beforeRow)
                                 (add place -1)
                                 (SETQ action 'afterRow]
                             (SELECTQ action
                                 (afterCol (EQ.MatrixAdd eqnObj 'column place window)
                                           T)
                                 (afterRow (EQ.MatrixAdd eqnObj 'row place window)
                                           T)
                                 (delCol (EQ.MatrixDelete eqnObj 'column place)
                                         T)
                                 (delRow (EQ.MatrixDelete eqnObj 'row place)
                                         T)
                                 NIL]
       (T                                                               (* eqnObj not a matrix)
          NIL])
```

(**EQ.MatrixGetMenu**

```
  [LAMBDA (eqnObj which)                                               (* thh%: " 3-Jun-85 09:35")
                                                                       (* gets menu for eqnObj, which is either row or column)
    (SELECTQ which
        (row (PROG [(rowMenu (EQIO.EqnProperty eqnObj 'rowMenu]
                   (COND
                      ((NOT (type? MENU rowMenu))
                       (SETQ rowMenu (create MENU
                                            TITLE _ "row #"
                                            ITEMS _ (for i from 1 to (EQIO.EqnProperty eqnObj 'rows) collect i)))
                       (EQIO.EqnProperty eqnObj 'rowMenu rowMenu)))
                   (RETURN rowMenu)))
        (column (PROG [(colMenu (EQIO.EqnProperty eqnObj 'colMenu]
                      (COND
                         ((NOT (type? MENU colMenu))
                          (SETQ colMenu (create MENU
                                               TITLE _ "col #"
                                               MENUROWS _ 1
                                               ITEMS _ (for i from 1 to (EQIO.EqnProperty eqnObj 'columns)
                                                           collect i)))
                          (EQIO.EqnProperty eqnObj 'colMenu colMenu)))
                      (RETURN colMenu)))
        NIL])
```

(**EQ.MatrixSelect**

```
  [LAMBDA (eqnObj)                                                     (* thh%: " 3-Jun-85 09:20")
                                                                       (* selects piece to edit by row and column number)
    (COND
       [(EQ (EQIO.EqnType eqnObj)
            'matrix)
        (PROG (row col)
              (COND
                 ((IGREATERP (EQIO.NumPieces eqnObj)
                          0)
                  [SETQ row (MENU (EQ.MatrixGetMenu eqnObj 'row]
                  (COND
                     (row [SETQ col (MENU (EQ.MatrixGetMenu eqnObj 'column]
                          (COND
                             (col (RETURN (PLUS (TIMES (EQIO.EqnProperty eqnObj 'columns)
                                                       (SUB1 row))
                                                col]
       (T                                                              (* eqnObj is not a matrix)
          NIL])

)

(RPAQ? EQ.Matrix.MaxPieces 100)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS EQ.Matrix.MaxPieces)
)

(EQIO.AddType 'matrix 'EQ.Matrix 1 '(objectProps (rows 1 columns 1 enclosureKind NIL enclosureSide NIL)
                                           variable? T wholeEditFn EQ.MatrixEdit specialSelectFn EQ.MatrixSelect
                                           initialPropFn EQ.MatrixCreate changeFn EQ.MatrixChanged))
```

;;; EQNFORMS module: Part 4 of 5

;; fraction

```
(DEFINEQ

(EQ.Fraction
  [LAMBDA (eqnObj imageStream draw?)                                     (* THH "21-May-85 12:14")
                                                                         (* form function for fraction)

          (* fraction defined by shift -- distance of line above base, gap --
          space above and below the line, width -- width of the line and extend --
          extra length of line on either side)
                                                                         (* these parameters could be specified by props to allow user to
    change them)
    (PROG ((size (EQ.StreamSize imageStream))
           shift gap width extend nBox dBox pos box pNumer pDenom)
          (SETQ shift (IQUOTIENT (FONTPROP (FONTCREATE (COND
                                                          ((EQIO.EqnProperty eqnObj 'fontSpec))
                                                          (T DEFAULTFONT))
                                                    NIL NIL NIL imageStream)
                                         'ASCENT)
                             2))
          (SETQ gap size)
          (SETQ width size)
          (SETQ extend size)
          (SETQ nBox (FS.Box (EQIO.EqnData eqnObj 1)
                             imageStream))
          (SETQ dBox (FS.Box (EQIO.EqnData eqnObj 2)
                             imageStream))
          (SETQ pos (PLUS (fetch YSIZE of dBox)
                          (TIMES 2 gap)
                          width))                                        (* pos is distance from bottom of box to bottom of numerator)
          (SETQ box (create IMAGEBOX
                            XSIZE _ (PLUS (MAX (fetch XSIZE of nBox)
                                               (fetch XSIZE of dBox))
                                          (TIMES 2 extend))
                            YSIZE _ (PLUS (fetch YSIZE of nBox)
                                          pos)
                            YDESC _ (PLUS gap (fetch YSIZE of dBox)
                                          (MINUS shift))
                            XKERN _ 0))
          [SETQ pNumer (create POSITION
                               XCOORD _ (IQUOTIENT (DIFFERENCE (fetch XSIZE of box)
                                                               (fetch XSIZE of nBox))
                                                   2)
                               YCOORD _ (PLUS pos (fetch YDESC of nBox]
          (SETQ pDenom (create POSITION
                               XCOORD _ (IQUOTIENT (DIFFERENCE (fetch XSIZE of box)
                                                               (fetch XSIZE of dBox))
                                                   2)
                               YCOORD _ (fetch YDESC of dBox)))
          (COND
             (draw?                                                      (* note that original x,y pos not preserved)
                    (RELMOVETO 0 shift imageStream)
                    (RELDRAWTO (fetch XSIZE of box)
                           0 width NIL imageStream)))
          [RETURN (EQIO.MakeSpec box (LIST (EQIO.MakeDataSpec pNumer
                                                   (create REGION
                                                          LEFT _ 0
                                                          BOTTOM _ pos
                                                          WIDTH _ (fetch XSIZE of box)
                                                          HEIGHT _ (fetch YSIZE of nBox)))
                                           (EQIO.MakeDataSpec pDenom
                                                   (create REGION
                                                          LEFT _ 0
                                                          BOTTOM _ 0
                                                          WIDTH _ (fetch XSIZE of box)
                                                          HEIGHT _ (fetch YSIZE of dBox]
                                                                         (* selection region for numerator (denominator) is entire top
                                                                         (bottom) half of fraction)
      ])
```

```
(EQ.Make.fraction
  [LAMBDA (numerator denominator fontSpec)                      (* thh%: " 9-Jan-86 10:14")
    (EQN.Make 'fraction (LIST numerator denominator)
           fontSpec])

)


;;; sum group

(DEFINEQ

(EQ.SumGroup
  [LAMBDA (eqnObj imageStream draw?)                            ; Edited 18-Apr-88 13:30 by thh:
                                                                ; form function for forms drawn like a summation

    ;; form defined by gap -- distance between symbol and index (below) and limits (above), and vGap -- distance between symbol and value

    (PROG ((size (EQ.StreamSize imageStream))
           [char (LIST (COND
                          ((EQ.UseNS? imageStream)
                           (EQ.MakeNSItem (EQIO.EqnType eqnObj)))
                          (T                                    ; use press fonts
                             (FS.MakeItem '(Sigma 20)
                                    (SELECTQ (EQIO.EqnType eqnObj)
                                        (sum '"M")
                                        (product '"P")
                                        (union '"U")
                                        (intersection '"I")
                                        (SHOULDNT "eqn not a known sum group"]
           extraShift charBox indexBox limitBox valueBox indexRegion limitRegion valueRegion charRegion
           boxRegion box temp gap vGap)
          (SETQ gap size)
          (SETQ vGap (TIMES 2 size))                            ; extraShift compensates for extra space around the NS chars
          (SETQ extraShift (COND
                              ((EQ.UseNS? imageStream)
                               (TIMES 4 size))
                              (T 0)))
          (SETQ charBox (FS.Box char imageStream))
          (SETQ indexBox (FS.Box (EQIO.EqnData eqnObj 1)
                                imageStream))
          (SETQ limitBox (FS.Box (EQIO.EqnData eqnObj 2)
                                imageStream))
          (SETQ valueBox (FS.Box (EQIO.EqnData eqnObj 3)
                                imageStream))
          (SETQ temp (AB.PositionRegion charBox NIL 'bottom 'center indexBox 'center (DIFFERENCE gap extraShift)
                            0 size))
          (SETQ indexRegion (CAR temp))
          (SETQ temp (AB.PositionRegion charBox temp 'top 'center limitBox 'center (TIMES -2 gap)
                            0 size))                            ; -2*gap used instead of gap since symbol doesn't fill its box
          (SETQ limitRegion (CAR temp))
          (SETQ temp (AB.PositionRegion charBox temp 'right 'display valueBox 'display vGap extraShift size))
          (SETQ valueRegion (CAR temp))
          (SETQ charRegion (AB.BoxToRegion charBox 0 0))
          (SETQ boxRegion (UNIONREGIONS charRegion indexRegion limitRegion valueRegion))
          (SETQ box (AB.RegionToBox boxRegion (PLUS (fetch YDESC of charBox)
                                                    extraShift)))
          (COND
             (draw?                                             ; note that original x, y position not preserved
                    (RELMOVETO (MINUS (fetch LEFT of boxRegion))
                           (MINUS extraShift)
                           imageStream)
                    (FS.Display char imageStream)))
          (RETURN (EQIO.MakeSpec box (LIST (EQIO.MakeDataSpec (AB.RelativePos indexRegion boxRegion
                                                                     (fetch YDESC of indexBox)))
                                           (EQIO.MakeDataSpec (AB.RelativePos limitRegion boxRegion
                                                                     (fetch YDESC of limitBox)))
                                           (EQIO.MakeDataSpec (AB.RelativePos valueRegion boxRegion
                                                                     (fetch YDESC of valueBox])

(EQ.Make.sum
  [LAMBDA (lowerIndex upperLimit summand fontSpec)              (* thh%: " 9-Jan-86 10:16")
    (EQN.Make 'sum (LIST lowerIndex upperLimit summand)
           fontSpec])

(EQ.Make.product
  [LAMBDA (lowerIndex upperLimit factor fontSpec)               (* thh%: " 9-Jan-86 10:17")
    (EQN.Make 'product (LIST lowerIndex upperLimit factor)
           fontSpec])

(EQ.Make.union
  [LAMBDA (lowerIndex upperLimit set fontSpec)                  (* thh%: " 9-Jan-86 10:18")
    (EQN.Make 'union (LIST lowerIndex upperLimit set)
           fontSpec])

(EQ.Make.intersection
  [LAMBDA (lowerIndex upperLimit set fontSpec)                (* thh%: " 9-Jan-86 10:18")
    (EQN.Make 'intersection (LIST lowerIndex upperLimit set)
           fontSpec])

)


;;; integral group

(DEFINEQ

(EQ.IntegralGroup
  [LAMBDA (eqnObj imageStream draw?)                          ; Edited 18-Apr-88 13:32 by thh:
                                                              ; form function for forms drawn like an integral

    ;; form defined by gap -- distance between symbol and limits and vGap -- distance between symbol and value

    (PROG ((size (EQ.StreamSize imageStream))
           [char (LIST (COND
                          ((EQ.UseNS? imageStream)
                           (EQ.MakeNSItem (EQIO.EqnType eqnObj)))
                          (T                                  ; use press fonts
                             (FS.MakeItem '(Sigma 20)
                                    (SELECTQ (EQIO.EqnType eqnObj)
                                        (integral '"S")
                                        (lineIntegral '"C")
                                        (SHOULDNT "eqn not a known integral group"]
           charBox lowerBox upperBox valueBox lowerRegion upperRegion valueRegion charRegion boxRegion box temp
           gap vGap)
          (SETQ gap size)
          (SETQ vGap (TIMES 5 size))
          (SETQ charBox (FS.Box char imageStream))
          (SETQ lowerBox (FS.Box (EQIO.EqnData eqnObj 1)
                                 imageStream))
          (SETQ upperBox (FS.Box (EQIO.EqnData eqnObj 2)
                                 imageStream))
          (SETQ valueBox (FS.Box (EQIO.EqnData eqnObj 3)
                                 imageStream))
          (SETQ temp (AB.Position2Regions charBox NIL 'right upperBox 'center lowerBox 'center (MINUS gap)
                             (TIMES -6 gap)
                             0 0 size))                       ; negative gaps since symbol doesn't fill its box
          (SETQ upperRegion (CAR temp))
          (SETQ lowerRegion (CADR temp))
          (SETQ temp (AB.PositionRegion charBox temp 'right 'display valueBox 'display vGap 0 size))
          (SETQ valueRegion (CAR temp))
          (SETQ charRegion (AB.BoxToRegion charBox 0 0))
          (SETQ boxRegion (UNIONREGIONS charRegion upperRegion lowerRegion valueRegion))
          (SETQ box (AB.RegionToBox boxRegion (fetch YDESC of charBox)))
          (COND
             (draw?                                           ; note that original x, y position not preserved
                    (RELMOVETO (MINUS (fetch LEFT of boxRegion))
                           0 imageStream)
                    (FS.Display char imageStream)))
          (RETURN (EQIO.MakeSpec box (LIST (EQIO.MakeDataSpec (AB.RelativePos lowerRegion boxRegion
                                                                         (fetch YDESC of lowerBox)))
                                            (EQIO.MakeDataSpec (AB.RelativePos upperRegion boxRegion
                                                                         (fetch YDESC of upperBox)))
                                            (EQIO.MakeDataSpec (AB.RelativePos valueRegion boxRegion
                                                                         (fetch YDESC of valueBox])

(EQ.Make.integral
  [LAMBDA (lowerLimit upperLimit integrand fontSpec)          (* thh%: " 9-Jan-86 10:19")
    (EQN.Make 'integral (LIST lowerLimit upperLimit integrand)
           fontSpec])

(EQ.Make.lineIntegral
  [LAMBDA (lowerLimit upperLimit integrand fontSpec)          (* thh%: " 9-Jan-86 10:20")
    (EQN.Make 'lineIntegral (LIST lowerLimit upperLimit integrand)
           fontSpec])

)


;;; super- and sub- scripts

(DEFINEQ

(EQ.Script
  [LAMBDA (eqnObj imageStream draw?)                          (* THH "23-May-85 14:07")
                                                              (* form function for forms with sub and superscripts)

          (* form defined by gap -- horizontal distance between main symbol and sub/superscripts, %, and shift --
          vertical distance of centers of sub/superscripts below/above corner of main symbol)
    (PROG ((size (EQ.StreamSize imageStream))
           mainBox super1Box sub1Box super2Box sub2Box super1Region sub1Region super2Region sub2Region
           mainRegion boxRegion box temp gap shift)
          (SETQ gap size)
          (SETQ shift size)
          (SETQ mainBox (FS.Box (EQIO.EqnData eqnObj 1)
                               imageStream))
          (SETQ sub1Box (FS.Box (EQIO.EqnData eqnObj 2)
                               imageStream))
          (SETQ super1Box (FS.Box (EQIO.EqnData eqnObj 3)
                                 imageStream))
          (SETQ sub2Box (FS.Box (EQIO.EqnData eqnObj 4)
                               imageStream))
          (SETQ super2Box (FS.Box (EQIO.EqnData eqnObj 5)
                                 imageStream))
          (SETQ temp (AB.Position2Regions mainBox NIL 'right super1Box 'center sub1Box 'center gap gap shift
                            (MINUS shift)
                            size))
          (SETQ super1Region (CAR temp))
          (SETQ sub1Region (CADR temp))
          (SETQ temp (AB.Position2Regions mainBox (CADDR temp)
                            'left super2Box 'center sub2Box 'center gap gap shift (MINUS shift)
                            size))
          (SETQ super2Region (CAR temp))
          (SETQ sub2Region (CADR temp))
          (SETQ mainRegion (AB.BoxToRegion mainBox 0 0))
          (SETQ boxRegion (UNIONREGIONS mainRegion super1Region sub1Region super2Region sub2Region))
          (SETQ box (AB.RegionToBox boxRegion (fetch YDESC of mainBox)))

          (* * this form has nothing extra to draw)

          (RETURN (EQIO.MakeSpec box (LIST (EQIO.MakeDataSpec (AB.RelativePos mainRegion boxRegion
                                                                     (fetch YDESC of mainBox)))
                                           (EQIO.MakeDataSpec (AB.RelativePos sub1Region boxRegion
                                                                     (fetch YDESC of sub1Box)))
                                           (EQIO.MakeDataSpec (AB.RelativePos super1Region boxRegion
                                                                     (fetch YDESC of super1Box)))
                                           (EQIO.MakeDataSpec (AB.RelativePos sub2Region boxRegion
                                                                     (fetch YDESC of sub2Box)))
                                           (EQIO.MakeDataSpec (AB.RelativePos super2Region boxRegion
                                                                     (fetch YDESC of super2Box])

(EQ.Make.sub/superscripts
  [LAMBDA (mainValue lowerRight upperRight lowerLeft upperLeft fontSpec)
                                                                    (* thh%: " 9-Jan-86 10:21")
    (EQN.Make 'sub/superscripts (LIST mainValue lowerRight upperRight lowerLeft upperLeft)
           fontSpec])
)

;;; max/min/limit etc

(DEFINEQ

(EQ.MaxMin
  [LAMBDA (eqnObj imageStream draw?)                                (* thh%: " 9-Jan-86 10:02")
                                                                    (* form function for max/min/limit etc)
    (PROG ((size (EQ.StreamSize imageStream))
           functionBox indexBox valueBox box functionRegion indexRegion valueRegion boxRegion gap vGap temp)
          (SETQ gap (TIMES 2 size))
          (SETQ vGap size)
          (SETQ functionBox (FS.Box (EQIO.EqnData eqnObj 1)
                                   imageStream))
          (SETQ indexBox (FS.Box (EQIO.EqnData eqnObj 2)
                                imageStream))
          (SETQ valueBox (FS.Box (EQIO.EqnData eqnObj 3)
                                imageStream))
          (SETQ temp (AB.PositionRegion functionBox NIL 'bottom 'center indexBox 'center vGap 0 size))
          (SETQ indexRegion (CAR temp))
          (SETQ temp (AB.PositionRegion functionBox temp 'right 'display valueBox 'display gap 0 size))
          (SETQ valueRegion (CAR temp))
          (SETQ functionRegion (AB.BoxToRegion functionBox 0 0))
          (SETQ boxRegion (UNIONREGIONS functionRegion indexRegion valueRegion))
          (SETQ box (AB.RegionToBox boxRegion (fetch YDESC of functionBox)))

          (* * nothing extra to draw)

          (RETURN (EQIO.MakeSpec box (LIST (EQIO.MakeDataSpec (AB.RelativePos functionRegion boxRegion
                                                                     (fetch YDESC of functionBox)))
                                           (EQIO.MakeDataSpec (AB.RelativePos indexRegion boxRegion
                                                                     (fetch YDESC of indexBox)))
                                           (EQIO.MakeDataSpec (AB.RelativePos valueRegion boxRegion
                                                                     (fetch YDESC of valueBox])
```

```
(EQ.Make.max/min
  [LAMBDA (function index value fontSpec)                              (* thh%: " 9-Jan-86 10:21")
    (EQN.Make 'max/min (LIST function index value)
           fontSpec])

)


;;; utilities

(DEFINEQ

(EQ.StreamSize
  [LAMBDA (imageStream)                                                (* thh%: " 3-May-85 08:37")
                                                                       (* standard size factor for the stream)
    (MAX 1 (IQUOTIENT (STRINGWIDTH "A" (FONTCREATE DEFAULTFONT NIL NIL NIL imageStream))
                  5])


(EQ.UseNS?
  [LAMBDA (imageStream)                                                (* thh%: " 9-Jan-86 10:29")

            (* * returns non-NIL if NS characters should be used for special symbols on imageStream)

    (SELECTQ (IMAGESTREAMTYPE imageStream)
         (DISPLAY EQ.UseNSChars)
         (PRESS NIL)
         (INTERPRESS T)
         EQ.UseNSChars])


(EQ.MakeNSItem
  [LAMBDA (type)                                                       ; Edited 18-Apr-88 13:29 by thh:
    (LET [(value (LISTGET EQ.NSChars (U-CASE type]
         (OR value (SHOULDNT "no display character for eqn"))
         (FS.MakeItem (CAR value)
                (MKSTRING (CHARACTER (CADR value])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS EQ.UseNSChars EQ.NSChars)
)

;; EQ.UseNSChars = NIL to use press fonts for display

(RPAQ? EQ.UseNSChars NIL)

(RPAQ? EQ.NSChars
       '(SUM ((CLASSIC 24)
              9814)
             PRODUCT
             ((CLASSIC 24)
              9811)
             SUM
             ((MODERN 30)
              61306)
             PRODUCT
             ((MODERN 30)
              61307)
             INTERSECTION
             ((MODERN 30)
              61270)
             UNION
             ((MODERN 30)
              61271)
             INTEGRAL
             ((MODERN 30)
              61301)
             LINEINTEGRAL
             ((MODERN 30)
              61302)))

[EQIO.AddType 'fraction 'EQ.Fraction 2 '(pieceNames ("numerator" "denominator"]

[EQIO.AddType 'sum 'EQ.SumGroup 3 '(initialData (-2 -2 0)
                                          pieceNames
                                          ("index" "limit" "summand"]

[EQIO.AddType 'product 'EQ.SumGroup 3 '(initialData (-2 -2 0)
                                              pieceNames
                                              ("index" "limit" "factor"]

[EQIO.AddType 'union 'EQ.SumGroup 3 '(initialData (-2 -2 0)
                                            pieceNames
                                            ("index" "limit" "set"]

[EQIO.AddType 'intersection 'EQ.SumGroup 3 '(initialData (-2 -2 0)
                                                   pieceNames
                                                   ("index" "limit" "set"]

[EQIO.AddType 'integral 'EQ.IntegralGroup 3 '(initialData (-2 -2 0)
                                                    pieceNames
                                                    ("lower limit" "upper limit" "integrand"]

(EQIO.AddType 'lineIntegral 'EQ.IntegralGroup 3 '(initialData (-2 -2 0)
                                                        pieceNames
                                                        ("lower limit" "upper limit" "integrand")
                                                        menuLabel "line integral"))

[EQIO.AddType 'sub/superscripts 'EQ.Script 5 '(initialData (0 -1 -1 -1 -1)
                                                     pieceNames
                                                     ("main value" "right subscript" "right superscript"
                                                            "left subscript" "left superscript"]

(EQIO.AddType 'max/min 'EQ.MaxMin 3 '(initialData (0 -2 0)
                                            pieceNames
                                            ("function" "index" "value")
                                            menuLabel "max min limit"))
```

;;; EQROOT module: Part 5 of 5

```
(DEFINEQ

(EQ.Root
  [LAMBDA (eqnObj imageStream draw?)                             (* thh%: "18-Mar-86 14:47")
                                                                 (* form function for roots)

          (* layout defined by vGap -- extra vertical space on either side of radicand, gap --
          extra horizontal space on either side of radicand, ivGap -- vertical space between index and radical sign, igap --
          extra horizontal extent of sign around index)

    (PROG ((size (EQ.StreamSize imageStream))
           rBox iBox gap vGap ivGap igap baseLen barLen height rise desc toBottomLen toTopLen width box)
          (SETQ gap (TIMES 2 size))
          (SETQ vGap (TIMES 2 size))
          (SETQ ivGap size)
          (SETQ igap 0)
          (SETQ width size)
          (SETQ rBox (FS.Box (EQIO.EqnData eqnObj 1)
                             imageStream))
          (SETQ iBox (FS.Box (EQIO.EqnData eqnObj 2)
                             imageStream))

          (* * determine size of parts of radical sign and draw it if requested)

          [SETQ baseLen (MAX (TIMES 4 size)
                             (PLUS (TIMES 2 igap)
                                   (fetch XSIZE of iBox]
          (SETQ barLen (PLUS (TIMES 2 gap)
                             (fetch XSIZE of rBox)))
          (SETQ height (PLUS (TIMES 2 vGap)
                             (fetch YSIZE of rBox)
                             (QUOTIENT width 2)))
          (SETQ rise (QUOTIENT height 2))
          (SETQ desc (PLUS (fetch YDESC of rBox)
                           vGap))
          (SETQ toBottomLen (TIMES 2 size))
          (SETQ toTopLen (TIMES 4 toBottomLen))
          (COND
             (draw? (EQ.DrawRadicalSign height rise desc baseLen toBottomLen toTopLen barLen width imageStream)))

          (* * get size and position values)

          (SETQ box (create IMAGEBOX
                           XSIZE _ (PLUS baseLen toBottomLen toTopLen barLen)
                           YSIZE _ (MAX (PLUS height (DIFFERENCE width (QUOTIENT width 2)))
                                        (PLUS rise ivGap (fetch YSIZE of iBox)))
                           YDESC _ desc
                           XKERN _ 0))
          (RETURN (EQIO.MakeSpec box (LIST (EQIO.MakeDataSpec (create POSITION
                                                                      XCOORD _ (PLUS baseLen toBottomLen toTopLen
                                                                                     gap)
                                                                      YCOORD _ desc))
                                           (EQIO.MakeDataSpec (create POSITION
                                                                      XCOORD _
                                                                      (QUOTIENT (DIFFERENCE baseLen
                                                                                       (fetch XSIZE of iBox))
                                                                             2)
                                                                      YCOORD _ (PLUS rise ivGap
                                                                                     (fetch YDESC of iBox])
```

```
(EQ.Make.root
  [LAMBDA (radicand index fontSpec)                              (* thh%: " 9-Jan-86 10:22")
    (EQN.Make 'root (LIST radicand index)
           fontSpec])

)

(DEFINEQ

(EQ.DrawRadicalSign
  [LAMBDA (height rise desc baseLen toBottomLen toTopLen barLen width imageStream)
                                                                 (* thh%: "28-Jun-85 13:04")
                                                                 (* draws specified radical sign on imageStream)
    (RELMOVETO 0 (DIFFERENCE rise desc)
           imageStream)
    (RELDRAWTO baseLen 0 width NIL imageStream)
    (RELDRAWTO toBottomLen (MINUS rise)
           (TIMES 2 width)
           NIL imageStream)
    (RELDRAWTO toTopLen height width NIL imageStream)
    (RELDRAWTO barLen 0 width NIL imageStream])

)

[EQIO.AddType 'root 'EQ.Root 2 '(pieceNames ("radicand" "index")
                                       initialData
                                       (0 -1]

(PUTPROPS EQUATIONFORMS COPYRIGHT ("Xerox Corporation" 1986 1987 1988))
```

{MEDLEY}<lispusers>EQUATIONFORMS.;1 9-Oct-2024 02:37:44
-- Listed on 9-Oct-2024 02:41:20 --

FUNCTION INDEX

VARIABLE INDEX

RECORD INDEX