
This appendix defines a number of utility procedures and functions that are called by other parts of the Venn Diagrammer application.

```

UNIT Utilities;
INTERFACE
  USES
    Global;

  PROCEDURE DoPlotIcon (myRect: Rect; myIcon: Handle; myWindow: WindowPtr;
                        myMode: Integer);
  PROCEDURE DoOutlineControl (myControl: univ ControlHandle);
  PROCEDURE DoDefaultButton (myDialog: DialogPtr);
  FUNCTION IsDaccWindow (myWindow: WindowPtr): Boolean;
  FUNCTION IsAppWindow (myWindow: WindowPtr): Boolean;
  FUNCTION IsDialogWindow (myWindow: WindowPtr): Boolean;
  PROCEDURE DoPositionWindow (myWindow: WindowPtr);
  PROCEDURE DoSetWindowTitle (myWindow: WindowPtr);
  FUNCTION DoTrackRect (myWindow: WindowPtr; myRect: Rect): Boolean;
  PROCEDURE DoStatusText (myWindow: WindowPtr; myText: Str255);
  PROCEDURE DoStatusMesg (myWindow: WindowPtr; myMessage: Integer);
  PROCEDURE DoBadError (myError: Integer);
  FUNCTION IsFindFolder: Boolean;
  FUNCTION MyRandom (last: Integer): Integer;

IMPLEMENTATION

{DoPlotIcon: plot a piece of an icon in a specified rectangle}
  PROCEDURE DoPlotIcon (myRect: Rect; myIcon: Handle; myWindow: WindowPtr;
                        myMode: Integer);

    VAR
      myBitMap: BitMap;
  BEGIN
    myBitMap.baseAddr := myIcon^;
    myBitMap.rowBytes := 4;
    myBitMap.bounds := myRect;
    CopyBits(myBitMap, myWindow^.portBits, myRect, myRect, myMode, NIL);
  END;

```

Utility Routines

```
{DoOutlineControl: draw bold outline around a control}
PROCEDURE DoOutlineControl (myControl: UNIV ControlHandle);
VAR
    myOval:      Integer;
    myRect:      Rect;
    origPen:     PenState;
    origPort:    GrafPtr;
BEGIN
    IF myControl <> NIL THEN
        BEGIN
            GetPort(origPort);
            SetPort(myControl^^.ctrlOwner);
            GetPenState(origPen);
            PenNormal;

            myRect := myControl^^.ctrlRect;
            InsetRect(myRect, -4, -4);
            myOval := ((myRect.bottom - myRect.top) DIV 2) + 2;

            IF (myControl^^.ctrlHilite = kCntrlActivate) THEN
                PenPat(black)
            ELSE
                PenPat(gray);
            PenSize(3, 3);
            FrameRoundRect(myRect, myOval, myOval);
            SetPenState(origPen);           {restore previous pen state}
            SetPort(origPort);
        END;
    END;
END;

{DoDefaultButton: draw bold outline around default button in a dialog}
{this procedure assumes that the default button is item number 1 (i.e., iOK)}
PROCEDURE DoDefaultButton (myDialog: DialogPtr);
VAR
    myType:      Integer;
    myHand:      Handle;
    myRect:      Rect;
BEGIN
    GetDialogItem(myDialog, iOK, myType, myHand, myRect);
    DoOutlineControl(myHand);
END;

{IsDAccWindow: determine if specified window belongs to a desk accessory}
```

Utility Routines

```

FUNCTION IsDaccWindow (myWindow: WindowPtr): Boolean;
BEGIN
    IF myWindow = NIL THEN
        IsDaccWindow := FALSE
    ELSE
        IsDaccWindow := WindowPeek(myWindow)^.windowKind < 0;
    END;
END;

{IsAppWindow: determine if specified window belongs to my app}
FUNCTION IsAppWindow (myWindow: WindowPtr): Boolean;
BEGIN
    IF myWindow = NIL THEN
        IsAppWindow := FALSE
    ELSE
        IsAppWindow := WindowPeek(myWindow)^.windowKind = userKind;
    END;
END;

{IsDialogWindow: determine if specified window is a dialog}
FUNCTION IsDialogWindow (myWindow: WindowPtr): Boolean;
BEGIN
    IF myWindow = NIL THEN
        IsDialogWindow := FALSE
    ELSE
        IsDialogWindow := WindowPeek(myWindow)^.windowKind = dialogKind;
    END;
END;

{DoPositionWindow: set the position of a new window}
PROCEDURE DoPositionWindow (myWindow: WindowPtr);
BEGIN
    END;

{DoSetWindowTitle: construct a title for a new window}
PROCEDURE DoSetWindowTitle (myWindow: WindowPtr);
    VAR
        myName:      Str255;
        myRank:      Str255;
BEGIN
    GetWTitle(myWindow, myName);
    gNumDocWindows := gNumDocWindows + 1;
    NumToString(gNumDocWindows, myRank);
    myName := concat(myName, ' ', myRank);
    SetWTitle(myWindow, myName);
END;

```

Utility Routines

```
{DoTrackRect: do "TrackBox" for a random rectangle}
{this is used to process clicks in a window tool}
FUNCTION DoTrackRect (myWindow: WindowPtr; myRect: Rect): Boolean;
    VAR
        myIgnore:    LongInt;
        myPoint:     Point;
    BEGIN
        InvertRect(myRect); {invert the rectangle}
        REPEAT
            Delay(kVisualDelay, myIgnore)
        UNTIL NOT StillDown; {keep inversion until mouse is released}
        InvertRect(myRect);

        GetMouse(myPoint); {get mouse location in local coordinates}
        DoTrackRect := PtInRect(myPoint, myRect);
    END;

{DoStatusText: print a message in a window's status area}
PROCEDURE DoStatusText (myWindow: WindowPtr; myText: Str255);
    VAR
        myRect:      Rect;
        origSize:     Integer;
        origFont:     Integer;
        myHandle:     MyDocRecHnd;
    CONST
        kSlop = 4;
        kSize = 9;
        kFont = applFont;
    BEGIN
        IF myWindow <> NIL THEN
            BEGIN
                SetPort(myWindow);
                origSize := myWindow^.txSize; {remember original size and font}
                origFont := myWindow^.txFont;
                TextSize(kSize); {set desired size and font}
                TextFont(kFont);

                SetRect(myRect, kToolWd * kNumTools, 0,
                        myWindow^.portRect.right, kToolHt);
                EraseRect(myRect);
                IF length(myText) > 0 THEN
                    BEGIN
```

Utility Routines

```

        MoveTo(myRect.left + kSlop, myRect.bottom - kSlop);
        DrawString(myText);
    END;

    TextSize(origSize);           {restore original size and font}
    TextFont(origFont);

    {Remember the last message printed in this window.}
    myHandle := MyDocRecHnd(GetWRefCon(myWindow));
    myHandle^.statusText := myText;
END;
END;

{DoStatusMesg: call DoStatusText, getting the text from a resource}
PROCEDURE DoStatusMesg (myWindow: WindowPtr; myMessageID: Integer);
    VAR
        myText:    Str255;
    BEGIN
        GetIndString(myText, rVennD, myMessageID);
        DoStatusText(myWindow, myText);
    END;

{DoBadError: inform the user of fatal errors, then terminate the app}
PROCEDURE DoBadError (myError: Integer);
    VAR
        myItem:    Integer;
        myMessage:  Str255;
    BEGIN
        SetCursor(arrow);           {set arrow cursor}
        GetIndString(myMessage, kErrorStrings, myError);
        ParamText(myMessage, '', '', '');
        myItem := Alert(rErrorAlert, NIL);    {display message}
        ExitToShell;                 {terminate execution}
    END;

{IsFindFolder: is the FindFolder function available?}
FUNCTION IsFindFolder: Boolean;
    VAR
        myResult:  OSErr;
        myFeature:  LongInt;
    BEGIN
        IsFindFolder := FALSE;        {assume it's not available}
        myResult := Gestalt(gestaltFindFolderAttr, myFeature);

```

Utility Routines

```

    IF myResult = noErr THEN
        IsFindFolder := BTST(myFeature, gestaltFindFolderPresent);
    END;

{MyRandom: generate a reasonably random number between 0 and last}
    FUNCTION MyRandom (last: Integer): Integer;
    BEGIN
        MyRandom := ABS(Random) MOD SUCC(last);
    END;
END.
```