

```
Unit ImageFrameUnit;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  
ExtCtrls, Buttons;
```

```
type
```

```
TScallShiftFn = function(Val, Scall, Shift: Extended): Extended of Object;
```

```
TAxisCrossPos = (AtStart, AtEnd, At0);
```

```
TRealPoint = record X, Y: extended end;
```

```
TRealArray = array of Extended;
```

```
PRealArray = ^TRealArray;
```

```
TGraphLine = record
```

```
Color: TColor;
```

```
Title: String;
```

```
TitleLeft: Boolean;
```

```
Data: PRealArray
```

```
end;
```

```
PGraphLine = ^TGraphLine;
```

```
TGraphAxis = class
```

```
private
```

```
Image: TImage;
```

```
StartPos, EndPos, Size: TPoint;
```

```
TextWidthDiv, TextHeightDiv: 0..2;
```

```
NumUnit: Int64;
```

```
Mult: Integer;
```

```
dMult: Extended;
```

```
Direct: Boolean;
```

```
ReCalc: Boolean;
```

```
UseLogScale: Boolean;
```

```
GraphScale, ReScale: TScallShiftFn;
```

```
function LinGraphScale(Val, Scale, Shift: Extended): Extended;
```

```
function LogGraphScale(Val, Scale, Shift: Extended): Extended;
```

```
function LinReScale(Val, Scale, Shift: Extended): Extended;
```

```
function LogReScale(Val, Scale, Shift: Extended): Extended;
```

```
public
```

```
Font: TFont;
```

```
LabelsShift, dLabelsShift: TPoint;
```

```
NumFormat: String;
```

```
Pos0: TPoint;
```

```
Scale: TRealPoint;
```

```
MinValue, MaxValue: Extended;
```

```
LabelsNum: Real;
```

```
Title: TCaption;
```

```
LogScale: Boolean;
```

```
constructor Create;
```

```
destructor Destroy; override;
```

```
constructor Install(OnImage: TImage; AxStart, AxEnd: TPoint;
```

```
StartValue, EndValue: Extended;
```

```
AxLabelIntNum: Real; AxLabelsLength: Integer;
```

```
AxTitle: TCaption);
```

```
procedure SetLimits(StartValue, EndValue: Extended);
```

```
procedure Shift(ToStartPos: TPoint);
```

```
function CheckLimits(Value: Extended): Boolean;
```

```
procedure Draw;
```

```
procedure SavePS;
```

```
function XGraphScale(Value: Extended): Extended;
```

```
function YGraphScale(Value: Extended): Extended;
```

```
function XtoGraph(Value: Extended): Integer;
```

```
function YtoGraph(Value: Extended): Integer;
```

```
function GraphXvalue(X: Extended): Extended;
```

```
function GraphYvalue(Y: Extended): Extended;
```

```
function XlimitPos(Y: Extended; Left: Boolean): Integer;
```

```
end;
```

```
TImageFrame = class(TFrame)
```

```
SpeedBar: TPanel;
```

```
SpeedButton2: TSpeedButton;
```

```
SpeedButton3: TSpeedButton;
```

```
SpeedButton4: TSpeedButton;
```

```
Image: TImage;
```

```
SaveDialog: TSaveDialog;
```

```

PrintDialog: TPrintDialog;
procedure ImageMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure SaveAs(Sender: TObject);
procedure ImagePrint(Sender: TObject);
procedure SavePS(Sender: TObject);
procedure SaveDialogTypeChange(Sender: TObject);
private
{ Private declarations }
LLeftAx, RRightAx: Integer;
Installed: Boolean;
ExtraLines: TImageFrame;
ExDataLines: Set of Byte;
Crossing: Boolean;
CrossPos: TPoint;
protected
procedure PutLnTitle(ALine: TGraphLine);
procedure PutLnTitleAt(ALine: TGraphLine; Y: Integer);
procedure PSHeader(const FileName: String);
procedure ImageToPS;
procedure PSClose;
public
{ Public declarations }
Axis: array of TGraphAxis;
Lines: array of PGraphLine;
PointsCount: Integer;
FixLimits: Boolean;
destructor Destroy; override;
procedure SetGraphFrame(GrLeft, GrUp, GrRight, GrDown: Integer;
    AxLabelsLength: Integer;
    XaxPos: TAxisCrossPos;
    Xstart, Xend: Extended; XaxDirect: Boolean;
    XlabIntNum: Real; Xtitle: String;
    YaxPos: TAxisCrossPos;
    Ystart, Yend: Extended; YaxDirect: Boolean;
    YlabIntNum: Real; Ytitle: String);
procedure SetGraphLine(LColor: TColor;
    LTitle: String; LTitleLeft: Boolean = False;
    LNum: Integer = 0; LinCount: Integer = 0);
procedure SetExDataLine(PDataArr: PRealArray; LColor: TColor;
    LTitle: String; LTitleLeft: Boolean = False;
    LNum: Integer = -1; LinCount: Integer = 0);
procedure SetExLineRec(PLineRec: PGraphLine;
    LNum: Integer = -1; LinCount: Integer = 0);
procedure SetExtraLines(FromImage: TImageFrame);
procedure NewGraph;
procedure AddGrPoint(NewData: array of Extended; ExtraNum: Integer = 0);
procedure PutLineTitles; overload;
procedure PutLineTitles(Ypos: array of Integer); overload;
procedure SavePStoFile(const FileName: String);
function XGraphScale(Value: Extended): Extended;
function YGraphScale(Value: Extended): Extended;
function XtoGraph(Value: Extended): Integer;
function YtoGraph(Value: Extended): Integer;
function GraphXvalue(X: Extended): Extended;
function GraphYvalue(Y: Extended): Extended;
procedure CrossOn; overload;
procedure CrossOn(Pos: TPoint); overload;
procedure CrossOn(X, Y: Integer); overload;
procedure CrossOff;
procedure CrossOnVal(X, Y: Extended);
end;

```

implementation

```
{ $R *.DFM }
```

```
uses JPEG, Math, PrintScalingUnit, ImagePrintUnit;
```

```
var
```

```
PSfile: TextFile;
PXLtoPS, X0PS, Y0PS, FontPXL, PSFontHeight: Integer;
```

```
procedure PointToPS(X, Y: Extended);
```

```
begin
```

```

Write(PSfile, ' ', IntToStr(X0PS + Round(X*PXLtoPS)),
      ' ', IntToStr(Y0PS - Round(Y*PXLtoPS)))
end;

procedure PSmove(X, Y: Extended);
begin PointToPS(X, Y); Write(PSfile, ' m') end;

procedure PSline(X, Y: Extended);
begin PointToPS(X, Y); Write(PSfile, ' l') end;

procedure PSColor(Color: TColor);
begin
  WriteLn(PSfile, ' ', (Color mod $100)/$FF :6:4,
          ((Color mod $10000) div $100)/$FF :7:4,
          ((Color mod $1000000) div $10000)/$FF : 7:4, ' srgb')
end;

procedure PSFont(Font: TFont);
begin
  FontPXL := Abs(Font.Height); PSFontHeight := FontPXL*PXLtoPS;
  WriteLn(PSfile, ' (', Font.Name, ') ff ', PSFontHeight, ' scf sf')
end;

procedure PSText(X, Y: Extended; const Text: String);
begin
  PSMove(X, Y + FontPXL);
  WriteLn(PSfile, ' (', Text, ') sh')
end;

constructor TGraphAxis.Create;
begin
  inherited;
  Font := TFont.Create
end;

destructor TGraphAxis.Destroy;
begin
  Font.Destroy;
  inherited
end;

constructor TGraphAxis.Install(OnImage: TImage; AxStart, AxEnd: TPoint;
  StartValue, EndValue: Extended;
  AxLabelIntNum: Real; AxLabelsLength: Integer;
  AxTitle: TCaption);
begin
  Create;
  Image := OnImage;
  Font.Assign(OnImage.Canvas.Font);
  StartPos := AxStart; EndPos := AxEnd;
  Size := Point(EndPos.x - StartPos.x, EndPos.y - StartPos.y);
  with Size do
  begin
    if x = 0
    then
      begin
        LabelsShift := Point(AxLabelsLength, 0);
        dLabelsShift := Point(AxLabelsLength * 2 div 3, 0);
        TextWidthDiv := Ord(AxLabelsLength < 0);
        TextHeightDiv := 2
      end
    else if y = 0
    then
      begin
        LabelsShift := Point(0, AxLabelsLength);
        dLabelsShift := Point(0, AxLabelsLength * 2 div 3);
        TextWidthDiv := 2;
        TextHeightDiv := Ord(AxLabelsLength < 0)
      end
    else
      begin
        LabelsShift := Point(-y div (Abs(x)+Abs(y))*AxLabelsLength,
                             x div (Abs(x)+Abs(y))*AxLabelsLength);
        dLabelsShift := Point(LabelsShift.X * 2 div 3, LabelsShift.Y * 2 div 3);
        TextWidthDiv := Ord(LabelsShift.x < 0);

```

```

        TextHeightDiv := Ord(LabelsShift.y < 0)
    end
end;
SetLimits(StartValue, EndValue);
LabelsNum := AxLabelIntNum + 1;
Title := AxTitle
end;

procedure TGraphAxis.SetLimits(StartValue, EndValue: Extended);
begin
    Direct := StartValue <= EndValue;
    if StartValue = EndValue
    then
        begin
            MinValue := StartValue; MaxValue := MinValue;
            Scale.X := 0; Scale.Y := 0;
            Pos0 := Point((StartPos.x + EndPos.x) div 2,
                (StartPos.y + EndPos.y) div 2);
            UseLogScale := False; GraphScale := LinGraphScale; ReScale := LinReScale
        end
    else
        begin
            if Direct
            then begin MinValue := StartValue; MaxValue := EndValue end
            else begin MinValue := EndValue; MaxValue := StartValue end;
            if LogScale and (MinValue*MaxValue > 0)
                and (LabelsNum < Abs(log10(MaxValue/MinValue))*7)
            then
                begin
                    if not UseLogScale then
                        begin
                            UseLogScale := True; GraphScale := LogGraphScale; ReScale := LogReScale
                        end;
                            Scale.X := Size.x / ln(EndValue/StartValue);
                            Scale.Y := Size.y / ln(EndValue/StartValue);
                            Pos0 := Point(StartPos.x - Round(Scale.X * ln(Abs(StartValue))),
                                StartPos.y - Round(Scale.Y * ln(Abs(StartValue))))
                        end
                    else
                        begin
                            UseLogScale := False; GraphScale := LinGraphScale; ReScale := LinReScale;
                            Scale.X := Size.x / (EndValue - StartValue);
                            Scale.Y := Size.y / (EndValue - StartValue);
                            Pos0 := Point(StartPos.x - Round(Scale.X * StartValue),
                                StartPos.y - Round(Scale.Y * StartValue))
                        end
                    end;
                end;
                ReCalc := True;
            end;

procedure TGraphAxis.Shift(ToStartPos: TPoint);
begin with ToStartPos do begin
    StartPos.x := x; EndPos.x := x + Size.x;
    StartPos.y := y; EndPos.y := y + Size.y
end end;

function TGraphAxis.CheckLimits(Value: Extended): Boolean;
begin
    if Value > MaxValue
    then
        begin
            if Direct then SetLimits(MinValue, Value) else SetLimits(Value, MaxValue);
            Result := True
        end
    else if Value < MinValue
    then
        begin
            if Direct then SetLimits(Value, MaxValue) else SetLimits(MaxValue, Value);
            Result := True
        end
    else Result := False
end;

procedure TGraphAxis.Draw;
var I, L, N: Integer; D, R, U: Extended; defLabShift: TPoint;

```

```
procedure DrawLabel(LabValue: Extended; Text: String);
var X, Y: Integer;
begin with Image.Canvas do begin
  X := Pos('E', Text);
  if (X > 0) and (X < Length(Text)) and (Text[X+1] in ['-','+', '0'..'9']) then
    begin
      if (Copy(Text, X-3, 3) = '1,0') and (NumFormat = '%.1e')
      then begin Delete(Text, X-2, 2); Dec(X) end
      else Inc(X);
      case Text[X] of
        '-': Inc(X);
        '+': Delete(Text, X, 1);
      end;
      while (Text[X] = '0') and (X < Length(Text)) and (Text[X+1] in ['0'..'9'])
      do Delete(Text, X, 1)
    end;
  X := XtoGraph(LabValue); Y := YtoGraph(LabValue);
  MoveTo(X, Y);
  Inc(X, LabelsShift.X); Inc(Y, LabelsShift.Y); LineTo(X, Y);
  if TextWidthDiv = 0 then Inc(X) else Dec(X, TextWidth(Text) div TextWidthDiv);
  if TextHeightDiv <> 0 then Dec(Y, TextHeight(Text) div TextHeightDiv);
  TextOut(X, Y, Text)
end end;

procedure DrawNumLabel(LabValue: Extended);
begin DrawLabel(LabValue, Format(NumFormat, [LabValue])) end;

procedure DrawTitleLabel(LabValue: Extended);
begin
  case TextWidthDiv of
    0: DrawLabel(LabValue, Format(NumFormat, [LabValue]) + Title);
    1: DrawLabel(LabValue, Title + Format(NumFormat, [LabValue]));
    2: begin
        DrawLabel(LabValue, Format(NumFormat, [LabValue]));
        with Image.Canvas, PenPos do TextOut(x, y, Title)
      end
  end
end;

begin
  if ReCalc then
    begin
      if LabelsNum > 2
      then
        if UseLogScale then
          begin
            R := Abs(Log10(MaxValue/MinValue))/(LabelsNum - 1);
            if R > 1 then
              begin
                Mult := -Round(R); NumUnit := Round(IntPower(10, -Mult));
                NumFormat := '%.1e'
              end
            else
              begin
                Mult := Round(1/R); NumUnit := 10;
                if Mult < 5 then NumFormat := '%.1g' else NumFormat := '%.2g'
              end;
            end;
          end
        if Direct then
          if MinValue > 0 then
            dMult := IntPower(NumUnit, Floor(Log10(MinValue)) div Max(-Mult,1))
          else dMult := -IntPower(NumUnit,
                                Ceil(Log10(Abs(MinValue))) div Max(-Mult,1) - 1)
          else if MaxValue > 0 then
            dMult := IntPower(NumUnit, Ceil(Log10(MaxValue)) div Max(-Mult,1) - 1)
          else dMult := -IntPower(NumUnit,
                                Floor(Log10(Abs(MaxValue))) div Max(-Mult,1));
          end
        end
      else
        begin
          D := (MaxValue - MinValue)/(LabelsNum - 1); R := 0.075 * D;
          NumUnit := 1;
          if R >= 0.1
          then
            begin
              while NumUnit < R do NumUnit := NumUnit * 10;
              case Round(D / NumUnit) of
                0: Mult := NumUnit;
```

```

1..2: Mult := NumUnit*2;
3:   Mult := NumUnit*3;
4..6: Mult := NumUnit*5;
else begin NumUnit := NumUnit*10; Mult := NumUnit end
end;
if Mult = NumUnit then dMult := NumUnit/2 else dMult := NumUnit;
if NumUnit <= 1000
then NumFormat := '%.0f'
else NumFormat := '%.'
    + IntToStr(Ceil(Log10(Max(MaxValue,Abs(MinValue))/NumUnit)))
    + 'g';
NumUnit := 1
end
else if R <> 0
then
begin
N := 0;
while (R * NumUnit < 0.1) and (NumUnit < High(NumUnit) div 10) do
    begin Inc(N); NumUnit := NumUnit * 10 end;
case Round(D*NumUnit) of
1..2: Mult := 2;
3:   Mult := 3;
4..6: Mult := 5;
else begin
    Mult := 1;
    Dec(N); NumUnit := NumUnit div 10
    end
end;
if Mult = 1 then dMult := 0.5/NumUnit else dMult := 1/NumUnit;
if N <= 5
then NumFormat := '%.' + IntToStr(N) + 'f'
else if R * NumUnit >= 1
then NumFormat := '%.'
    + IntToStr(Ceil(Log10(Max(MaxValue,Abs(MinValue))*NumUnit)))
    + 'g'
else NumFormat := '%g'
end
else begin NumUnit := 0; NumFormat := '%g' end
end
else begin NumUnit := 0; NumFormat := '%g' end;
ReCalc := False
end;
Image.Canvas.Font.Assign(Font);
Image.Canvas.Pen.Color := Font.Color;
if NumUnit <> 0
then
if UseLogScale then
if Direct = (MinValue > 0) then
begin
L := Max(-Mult, 1);
R := dMult;
if Direct then
begin
N := Floor(Log10(Abs(MinValue))) div L;
L := Floor(Log10(Abs(MaxValue))) div L;
R := IntPower(NumUnit, N);
// L := Floor(Log10(MaxValue/R)) div Max(-Mult,1);
U := MaxValue/R; D := MinValue/R
end
else
begin
N := Floor(Log10(Abs(MaxValue))) div L;
L := Floor(Log10(Abs(MinValue))) div L;
R := -IntPower(NumUnit, N);
U := MinValue/R; D := MaxValue/R
end;
if Mult <= 1 then
begin
if D > 1 then begin R := R*NumUnit; Inc(N) end;
for N := N+1 to L do begin DrawNumLabel(R); R := R*NumUnit end;
DrawTitleLabel(R)
end
else if (U < 10) {15} or (Mult < 5) and (U < 20)} then
case Mult of
2: begin DrawTitlelabel(10*R); DrawNumLabel(3*R); DrawNumLabel(R) end;

```

```
3: if U >= 10 then
    begin
        DrawTitleLabel(10*R); DrawNumLabel(5*R); DrawNumLabel(2*R);
        if D <= 1 then DrawNumLabel(R)
        end
    end
else begin DrawTitleLabel(5*R); DrawNumLabel(2*R); DrawNumLabel(R) end;
4: case Floor(U) of
    0..4: begin
        DrawTitleLabel(3*R); DrawNumLabel(2*R); DrawNumLabel(R)
        end;
    5..9: begin
        DrawTitleLabel(5*R); DrawNumLabel(3*R); DrawNumLabel(2*R);
        if D <= 1 then DrawNumLabel(R)
        end;
    else begin
        DrawTitleLabel(10*R); DrawNumLabel(5*R); DrawNumLabel(3*R);
        case Ceil(D) of
            0,1: begin DrawNumLabel(2*R); DrawNumLabel(R) end;
            2: DrawNumLabel(2*R)
        end
    end
end
end
else case Floor(U) of
    0..2: begin
        DrawTitleLabel(2*R); DrawNumLabel(1.5*R); DrawNumLabel(R)
        end;
    3..4: begin
        DrawTitleLabel(3*R); DrawNumLabel(2*R); DrawNumLabel(1.5*R);
        if D <= 1 then DrawNumLabel(R)
        end;
    5..6: begin
        DrawTitleLabel(5*R); DrawNumLabel(3*R); DrawNumLabel(2*R);
        case Ceil(2*D) of
            0..2: begin DrawNumLabel(1.5*R); DrawNumLabel(R) end;
            3: DrawNumLabel(1.5*R)
        end
    end;
    7..9: begin
        DrawTitleLabel(7*R); DrawNumLabel(5*R); DrawNumLabel(3*R);
        case Ceil(2*D) of
            0..2: begin
                DrawNumLabel(2*R); DrawNumLabel(1.5*R); DrawNumLabel(R)
                end;
            3: begin DrawNumLabel(2*R); DrawNumLabel(1.5*R) end;
            4: DrawNumLabel(2*R)
        end
    end;
else begin
    DrawTitleLabel(10*R); DrawNumLabel(7*R); DrawNumLabel(5*R);
    case Ceil(2*D) of
        0..2: begin
            DrawNumLabel(3*R); DrawNumLabel(2*R);
            DrawNumLabel(1.5*R); DrawNumLabel(R)
            end;
        3: begin
            DrawNumLabel(3*R); DrawNumLabel(2*R);
            DrawNumLabel(1.5*R)
            end;
        4: begin DrawNumLabel(3*R); DrawNumLabel(2*R) end;
        5,6: DrawNumLabel(3*R)
    end
end
end
end
else
begin
    case Mult of
        2: case Ceil(D) of
            0,1: begin DrawNumLabel(R); DrawNumLabel(3*R) end;
            2,3: DrawNumLabel(3*R);
        end;
        3: case Ceil(D) of
            0,1: begin
                DrawNumLabel(R); DrawNumLabel(2*R); DrawNumLabel(5*R)
                end;
            end;
```

```
2:   begin DrawNumLabel(2*R); DrawNumLabel(5*R) end;
3,5: DrawNumLabel(5*R)
end;
4: case Ceil(D) of
  0,1: begin
    DrawNumLabel(R); DrawNumLabel(2*R);
    DrawNumLabel(3*R); DrawNumLabel(5*R)
    end;
  2:   begin
    DrawNumLabel(2*R); DrawNumLabel(3*R); DrawNumLabel(5*R)
    end;
  3:   begin DrawNumLabel(3*R); DrawNumLabel(5*R) end;
  4,5: DrawNumLabel(5*R)
end;
else case Ceil(2*D) of
  0..2: begin
    DrawNumLabel(R); DrawNumLabel(1.5*R); DrawNumLabel(2*R);
    DrawNumLabel(3*R); DrawNumLabel(5*R); DrawNumLabel(7*R)
    end;
  3:   begin
    DrawNumLabel(1.5*R); DrawNumLabel(2*R); DrawNumLabel(3*R);
    DrawNumLabel(5*R); DrawNumLabel(7*R)
    end;
  4:   begin
    DrawNumLabel(2*R); DrawNumLabel(3*R);
    DrawNumLabel(5*R); DrawNumLabel(7*R)
    end;
  5..6: begin
    DrawNumLabel(3*R); DrawNumLabel(5*R); DrawNumLabel(7*R)
    end;
  7..10: begin DrawNumLabel(5*R); DrawNumLabel(7*R) end;
  11..14: DrawNumLabel(7*R);
end
end;
R := R*NumUnit;
for N := N+2 to L do
begin
  DrawNumLabel(R);
  case Mult of
    2: DrawNumLabel(3*R);
    3: begin DrawNumLabel(2*R); DrawNumLabel(5*R) end;
    4: begin
        DrawNumLabel(2*R); DrawNumLabel(3*R); DrawNumLabel(5*R)
        end;
    else begin
        DrawNumLabel(1.5*R); DrawNumLabel(2*R);
        DrawNumLabel(3*R); DrawNumLabel(5*R); DrawNumLabel(7*R)
        end
    end;
  R := R*NumUnit
end;
if Direct then U := MaxValue/R else U := MinValue/R;
case Floor(U) of
  0: DrawTitleLabel(R);
  1: if (Mult >= 5) and (U >= 1.5) then
    begin DrawNumLabel(R); DrawTitleLabel(1.5*R) end
    else DrawTitleLabel(R);
  2: if Mult < 3 then DrawTitleLabel(R) else
    begin
      DrawNumLabel(R);
      if Mult >= 5 then DrawNumLabel(1.5*R);
      DrawTitleLabel(2*R)
    end;
  3,4: begin
    DrawNumLabel(R);
    case Mult of
      2: DrawTitleLabel(3*R);
      3: DrawTitleLabel(2*R);
      4: begin DrawNumLabel(2*R); DrawTitleLabel(3*R) end;
    else begin
      DrawNumLabel(1.5*R); DrawNumLabel(2*R);
      DrawTitleLabel(3*R)
    end
    end
  end
end;
end;
```



```

else begin
    DrawNumLabel(R);
    case Mult of
        2: DrawTitleLabel(3*R);
        3: begin DrawNumLabel(2*R); DrawTitleLabel(5*R) end;
        4: begin
            DrawNumLabel(2*R); DrawNumLabel(3*R); DrawTitleLabel(5*R)
            end;
        else begin
            DrawNumLabel(1.5*R); DrawNumLabel(2*R); DrawNumLabel(3*R);
            if (U < 7) then DrawTitleLabel(5*R)
            else begin DrawNumLabel(5*R); DrawTitleLabel(7*R) end
            end
        end
    end
end
end
else
begin
    L := Max(-Mult, 1);
    if Direct then
begin
    N := Ceil(Log10(Abs(MinValue))) div L - 1;
    L := Ceil(Log10(Abs(MaxValue))) div L;
    R := -IntPower(NumUnit, N);
    D := MinValue/R; U := MaxValue/R
end
else
begin
    N := Ceil(Log10(Abs(MaxValue))) div L - 1;
    L := Ceil(Log10(Abs(MinValue))) div L;
    R := IntPower(NumUnit, N);
    D := MaxValue/R; U := MinValue/R
end;
if Mult <= 1 then
begin
    if D >= NumUnit then DrawNumLabel(R*NumUnit);
    for N := N-1 downto L do begin DrawNumLabel(R); R := R/NumUnit end;
    DrawTitleLabel(R)
end
else if (U > 0.7) or (Mult < 5) and (U > 0.5) then
case Mult of
    2: begin DrawTitleLabel(R); DrawNumLabel(3*R); DrawNumLabel(10*R) end;
    3: if U <= 1 then
        begin
            if D >= 10 then DrawNumLabel(10*R);
            DrawNumLabel(5*R); DrawNumLabel(2*R); DrawTitleLabel(R)
            end
        else
            begin DrawNumLabel(10*R); DrawNumLabel(5*R); DrawTitleLabel(2*R) end;
    4: case Ceil(U) of
        0,1: begin
            DrawTitleLabel(R); DrawNumLabel(2*R); DrawNumLabel(3*R);
            case Floor(D) of
                5..9: DrawNumLabel(5*R);
                10: begin DrawNumLabel(5*R); DrawNumLabel(10*R) end;
            end;
        end;
        2: begin
            DrawTitleLabel(2*R); DrawNumLabel(3*R); DrawNumLabel(5*R);
            if D >= 10 then DrawNumLabel(10*R)
            end;
        else begin
            DrawTitleLabel(R); DrawNumLabel(2*R); DrawNumLabel(3*R)
            end;
        end
    end
else case Ceil(2*U) of
    0..2: begin
        DrawTitleLabel(R); DrawNumLabel(1.5*R); DrawNumLabel(2*R);
        case Floor(D) of
            3..4: DrawNumLabel(3*R);
            5..6: begin DrawNumLabel(3*R); DrawNumLabel(5*R) end;
            7..9: begin
                DrawNumLabel(3*R); DrawNumLabel(5*R);

```

```
        DrawNumLabel(7*R)
    end;
10: begin
    DrawNumLabel(3*R); DrawNumLabel(5*R);
    DrawNumLabel(7*R); DrawNumLabel(10*R)
end;
end
end;
3: begin
    DrawTitleLabel(1.5*R); DrawNumLabel(2*R); DrawNumLabel(3*R);
    case Floor(D) of
        5..6: DrawNumLabel(5*R);
        7..9: begin DrawNumLabel(5*R); DrawNumLabel(7*R) end;
        10: begin
            DrawNumLabel(5*R); DrawNumLabel(7*R); DrawNumLabel(10*R)
        end;
    end
end;
4..5: begin
    DrawTitleLabel(2*R); DrawNumLabel(3*R); DrawNumLabel(5*R);
    case Floor(D) of
        7..9: DrawNumLabel(7*R);
        10: begin DrawNumLabel(7*R); DrawNumLabel(10*R) end;
    end
end;
6..9: begin
    DrawTitleLabel(3*R); DrawNumLabel(5*R); DrawNumLabel(7*R);
    if D >= 10 then DrawNumLabel(10*R)
end;
else begin
    DrawTitleLabel(5*R); DrawNumLabel(7*R); DrawNumLabel(10*R)
end;
end
end
else
begin
    case Mult of
        2: case Floor(D) of
            3..9: DrawNumLabel(3*R);
            10: begin DrawNumLabel(10*R); DrawNumLabel(3*R) end;
        end;
        3: case Ceil(D) of
            2..4: DrawNumLabel(2*R);
            5..9: begin DrawNumLabel(5*R); DrawNumLabel(2*R) end;
            10: begin
                DrawNumLabel(10*R); DrawNumLabel(5*R); DrawNumLabel(2*R)
            end;
        end;
        4: case Ceil(D) of
            2: DrawNumLabel(2*R);
            3..4: begin DrawNumLabel(3*R); DrawNumLabel(2*R) end;
            5..9: begin
                DrawNumLabel(5*R); DrawNumLabel(3*R); DrawNumLabel(2*R)
            end;
            10: begin
                DrawNumLabel(10*R); DrawNumLabel(5*R);
                DrawNumLabel(3*R); DrawNumLabel(2*R)
            end;
        end;
    end;
else case Floor(2*D) of
    3: DrawNumLabel(1.5*R);
    4..5: begin DrawNumLabel(2*R); DrawNumLabel(1.5*R) end;
    6..9: begin
        DrawNumLabel(3*R); DrawNumLabel(2*R); DrawNumLabel(1.5*R)
    end;
    10..13: begin
        DrawNumLabel(5*R); DrawNumLabel(3*R);
        DrawNumLabel(2*R); DrawNumLabel(1.5*R)
    end;
    14..19: begin
        DrawNumLabel(7*R); DrawNumLabel(5*R); DrawNumLabel(3*R);
        DrawNumLabel(2*R); DrawNumLabel(1.5*R)
    end;
    20: begin
        DrawNumLabel(10*R); DrawNumLabel(7*R); DrawNumLabel(5*R);
```

```
        DrawNumLabel(3*R); DrawNumLabel(2*R); DrawNumLabel(1.5*R)
    end;
end
end;
R := R/NumUnit;
for N := N-1 downto L do
begin
    DrawNumLabel(10*R);
    case Mult of
        2: DrawNumLabel(3*R);
        3: begin DrawNumLabel(5*R); DrawNumLabel(2*R) end;
        4: begin
            DrawNumLabel(5*R); DrawNumLabel(3*R); DrawNumLabel(2*R)
        end;
        else begin
            DrawNumLabel(7*R); DrawNumLabel(5*R);
            DrawNumLabel(3*R); DrawNumLabel(2*R); DrawNumLabel(1.5*R)
        end
    end;
    R := R/NumUnit
end;
if Direct then U := MaxValue/R else U := MinValue/R;
case Ceil(U) of
    0..2: begin
        DrawNumLabel(10*R);
        case Mult of
            2: DrawTitleLabel(3*R);
            3: begin DrawNumLabel(5*R); DrawTitleLabel(2*R) end;
            4: begin
                DrawNumLabel(5*R); DrawNumLabel(3*R); DrawTitleLabel(2*R)
            end;
            else begin
                DrawNumLabel(7*R); DrawNumLabel(5*R); DrawNumLabel(3*R);
                if (U > 1.5) then DrawTitleLabel(2*R)
                else begin DrawNumLabel(2*R); DrawTitleLabel(1.5*R) end
            end
        end
    end;
    3..4: begin
        DrawNumLabel(10*R);
        case Mult of
            2: DrawTitleLabel(3*R);
            3: DrawTitleLabel(5*R);
            4: begin DrawNumLabel(5*R); DrawTitleLabel(3*R) end;
            else begin
                DrawNumLabel(7*R); DrawNumLabel(5*R);
                DrawTitleLabel(3*R)
            end
        end
    end;
    5..6: if Mult < 3 then DrawTitleLabel(10*R) else
        begin
            DrawNumLabel(10*R);
            if Mult >= 5 then DrawNumLabel(7*R);
            DrawTitleLabel(5*R)
        end;
    7..9: if (Mult >= 5) and (U >= 7) then
        begin DrawNumLabel(10*R); DrawTitleLabel(7*R) end
        else DrawTitleLabel(10*R);
    else DrawTitleLabel(10*R);
end
end
end
end
else
begin
    defLabShift := LabelsShift; LabelsShift := dLabelsShift;
    for N := Ceil(MinValue/dMult) to Floor(MaxValue/dMult) do
        DrawLabel(N*dMult, '');
    LabelsShift := defLabShift;
    for N := Ceil(MinValue*NumUnit/Mult) + Ord(not Direct)
    to Floor(MaxValue*NumUnit/Mult) - Ord(Direct) do
        DrawNumLabel(N*Mult/NumUnit);
    if Direct
    then DrawTitleLabel(Floor(MaxValue*NumUnit/Mult)*Mult/NumUnit)
    else DrawTitleLabel(Ceil(MinValue*NumUnit/Mult)*Mult/NumUnit)
```

```
end
else
begin
  if LabelsNum = 2 then
  begin
    if (Scale.X = 0) and (Scale.Y = 0) then Pos0 := StartPos;
    if Direct then DrawNumLabel(MinValue) else DrawNumLabel(MaxValue);
    if (Scale.X = 0) and (Scale.Y = 0) then Pos0 := EndPos
  end;
  if Direct then DrawTitleLabel(MaxValue) else DrawTitleLabel(MinValue);
  if (Scale.X = 0) and (Scale.Y = 0) then
    Pos0 := Point((StartPos.x + EndPos.x) div 2,
                  (StartPos.y + EndPos.y) div 2)
  end;
with Image.Canvas do
  begin MoveTo(StartPos.x, StartPos.y); LineTo(EndPos.x, EndPos.y) end
end;

procedure TGraphAxis.SavePS;
var N: Integer; R: Extended;
procedure DrawLabel(LabValue: Extended; Text: String);
var X, Y: Extended;
begin
  X := Pos0.x + Scale.X*LabValue;
  Y := Pos0.y + Scale.Y*LabValue;
  PSMove(X, Y);
  X := X + LabelsShift.X; Y := Y + LabelsShift.Y; PSLine(X, Y);
  Write(PSfile, ' s');
  with Image.Canvas do
  begin
    if TextWidthDiv <> 0 then
      X := X - TextWidth(Text) div TextWidthDiv;
    if TextHeightDiv <> 0 then
      Y := Y - TextHeight(Text) div TextHeightDiv
    end;
    PSText(X, Y, Text)
  end;
begin
  WriteLn(PSfile);
  WriteLn(PSfile, 'gs');
  WriteLn(PSfile, PXLtoPS * 3 div 4, ' slw');
  with Image.Canvas.Font do
    if (Font.Name <> Name) or (Font.Height <> Height) then
    begin
      PSFont(Font);
      if Font.Color <> Color then PSColor(Font.Color);
      Assign(Font)
    end
    else if Font.Color <> Color then PSColor(Font.Color);
  PSColor(Font.Color);
  if ReCalc then Draw;
  if NumUnit <> 0
  then
  begin
    for N := Ceil(MinValue*NumUnit/Mult) + Ord(not Direct)
    to Floor(MaxValue*NumUnit/Mult) - Ord(Direct) do
    begin
      R := N*Mult/NumUnit;
      DrawLabel(R, Format(NumFormat, [R]))
    end;
    if Direct
    then R := Floor(MaxValue*NumUnit/Mult)*Mult/NumUnit
    else R := Ceil(MinValue*NumUnit/Mult)*Mult/NumUnit;
    case TextWidthDiv of
      0: DrawLabel(R, Format(NumFormat, [R]) + Title);
      1: DrawLabel(R, Title + Format(NumFormat, [R]));
      2: begin
          DrawLabel(R, Format(NumFormat, [R]));
          WriteLn(PSfile, '(', Title, ') sh')
        end
    end
  end
end
else
begin
  if (Scale.X = 0) and (Scale.Y = 0) then Pos0 := EndPos;
```

```

if Direct then R := MaxValue else R := MinValue;
case TextWidthDiv of
  0: DrawLabel(R, Format(NumFormat, [R]) + Title);
  1: DrawLabel(R, Title + Format(NumFormat, [R]));
  2: begin
    DrawLabel(MaxValue, Format(NumFormat, [R]));
    WriteLn(PSfile, '(', Title, ') sh')
  end
end;
if (Scale.X = 0) and (Scale.Y = 0) then Pos0 := StartPos;
if Direct then R := MinValue else R := MaxValue;
DrawLabel(R, Format(NumFormat, [R]))
end;
PSMove(StartPos.x, StartPos.y); PSLine(EndPos.x, EndPos.y);
WriteLn(PSfile, ' ', PXLtoPS, ' slw s gr')
end;

function TGraphAxis.LinGraphScale(Val, Scale, Shift: Extended): Extended;
begin Result := Val*Scale + Shift end;

function TGraphAxis.LogGraphScale(Val, Scale, Shift: Extended): Extended;
begin Result := ln(Abs(Val))*Scale + Shift end;

function TGraphAxis.LinReScale(Val, Scale, Shift: Extended): Extended;
begin Result := (Val - Shift)/Scale end;

function TGraphAxis.LogReScale(Val, Scale, Shift: Extended): Extended;
begin
  Result := exp((Val - Shift)/Scale); if MinValue < 0 then Result := -Result
end;

function TGraphAxis.XGraphScale;
begin Result := GraphScale(Value, Scale.X, Pos0.x) end;

function TGraphAxis.YGraphScale;
begin Result := GraphScale(Value, Scale.Y, Pos0.y) end;

function TGraphAxis.XtoGraph;
begin Result := Round(XGraphScale(Value)) end;

function TGraphAxis.YtoGraph;
begin Result := Round(YGraphScale(Value)) end;

function TGraphAxis.GraphXvalue;
begin
  if Scale.X = 0 then Result := MinValue
  else Result := ReScale(X, Scale.X, Pos0.x)
end;

function TGraphAxis.GraphYvalue;
begin
  if Scale.Y = 0 then Result := MinValue
  else Result := ReScale(Y, Scale.Y, Pos0.y)
end;

function TGraphAxis.XlimitPos(Y: Extended; Left: Boolean): Integer;
begin
  if Left
  then case TextWidthDiv of
    0: if Scale.X = 0
      then Result := StartPos.x - Abs(LabelsShift.x)*2
      else Result := XtoGraph(GraphYvalue(Y)) - Abs(LabelsShift.x)*2;
    1: begin
      Y := GraphYvalue(Y);
      Result := XtoGraph(Y)
        - Image.Canvas.TextWidth(Format(NumFormat, [Y]))
        - Abs(LabelsShift.x)*2
      end;
    else Result := Min(StartPos.x, EndPos.x) - Abs(LabelsShift.y)*2
  end
  else case TextWidthDiv of
    0: begin
      Y := GraphYvalue(Y);
      Result := XtoGraph(Y)
        + Image.Canvas.TextWidth(Format(NumFormat, [Y]))

```

```
        + Abs(LabelsShift.x)*2
    end;
1: if Scale.X = 0
    then Result := StartPos.x + Abs(LabelsShift.x)*2
    else Result := XtoGraph(GraphYvalue(Y)) + Abs(LabelsShift.x)*2;
else Result := Max(StartPos.x, EndPos.x) + Abs(LabelsShift.y)*2
end
end;

destructor TImageFrame.Destroy;
var I: Integer;
begin
    for I := High(Lines) downto 0 do
        if Assigned(Lines[I]) then
            begin
                with Lines[I]^ do
                    if Assigned(Data) and not ((I+128) in ExDataLines) then
                        begin
                            Finalize(Data^);
                            Dispose(Data);
                            Data := Nil
                        end;
                    if not (I in ExDataLines) then
                        Dispose(Lines[I])
                    end;
                end;
            end;
        Finalize(Lines);
    for I := High(Axis) downto 0 do Axis[I].Free;
    Finalize(Axis);
    inherited
end;

procedure TImageFrame.SetGraphFrame(GrLeft, GrUp, GrRight, GrDown: Integer;
    AxLabelsLength: Integer;
    XaxPos: TAxisCrossPos; Xstart, Xend: Extended; XaxDirect: Boolean;
    XlabIntNum: Real; Xtitle: String;
    YaxPos: TAxisCrossPos; Ystart, Yend: Extended; YaxDirect: Boolean;
    YlabIntNum: Real; Ytitle: String);
var I, L, Q, R: Integer;
begin
    Image.Canvas.Font := Font;
    SetLength(Axis, 2);
    if (XaxPos = At0) and (Ystart <> Yend) then
        if Ystart = 0 then XaxPos := AtStart
        else if Yend = 0 then XaxPos := AtEnd;
    case XaxPos of
        AtStart: if YaxDirect
            then begin L := GrDown; R := AxLabelsLength end
            else begin L := GrUp; R := -AxLabelsLength end;
        AtEnd:   if YaxDirect
            then begin L := GrUp; R := -AxLabelsLength end
            else begin L := GrDown; R := AxLabelsLength end;
        At0:     begin
            if Abs(Ystart) = Abs(Yend)
            then L := (GrUp + GrDown) div 2
            else L := GrDown;
            R := AxLabelsLength
        end
    end;
    end;
    if XaxDirect
    then begin I := GrLeft; Q := GrRight end
    else begin I := GrRight; Q := GrLeft end;
    if (XaxPos <> at0) or (YaxPos = at0) or (Ystart = Yend) then
        Axis[0] := TGraphAxis.Install(Image, Point(I, L), Point(Q, L),
            Xstart, Xend, XlabIntNum, R, Xtitle);
    if (YaxPos = At0) and (Xstart <> Xend) then
        if Xstart = 0 then YaxPos := AtStart
        else if Xend = 0 then YaxPos := AtEnd;
    case YaxPos of
        AtStart: if XaxDirect
            then
                begin
                    L := GrLeft; R := -AxLabelsLength;
                    LLeftAx := 1; RRightAx := 0
                end
            else
```

```

        begin
            L := GrRight; R := AxLabelsLength;
            LLeftAx := 0; RRightAx := 1
        end;
AtEnd:   if XaxDirect
        then
            begin
                L := GrRight; R := AxLabelsLength;
                LLeftAx := 0; RRightAx := 1
            end
        else
            begin
                L := GrLeft; R := -AxLabelsLength;
                LLeftAx := 1; RRightAx := 0
            end;
At0:     begin
            L := Axis[0].Pos0.x; R := AxLabelsLength;
            LLeftAx := 0; RRightAx := 0
        end
    end;
    if YaxDirect
    then begin I := GrDown; Q := GrUp end
    else begin I := GrUp; Q := GrDown end;
    Axis[1] := TGraphAxis.Install(Image, Point(L, I), Point(L, Q),
                                Ystart, Yend, YlabIntNum, R, Ytitle);
    Installed := (Axis[0].Scale.x <> 0) or (Axis[1].Scale.y <> 0);
    if (XaxPos = at0) and (Ystart <> Yend) then
        begin
            L := Axis[1].Pos0.y; R := AxLabelsLength;
            if XaxDirect
            then begin I := GrLeft; Q := GrRight end
            else begin I := GrRight; Q := GrLeft end;
            if YaxPos = at0
            then
                begin
                    Axis[0].StartPos := Point(I, L);
                    Axis[0].EndPos := Point(Q, L)
                end
            else Axis[0] := TGraphAxis.Install(Image, Point(I, L), Point(Q, L),
                                Xstart, Xend, XlabIntNum, R, Xtitle)
            end;
    PointsCount := -1;
    NewGraph;
    Show
end;

procedure TImageFrame.SetGraphLine(LColor: TColor;
    LTitle: String; LTitleLeft: Boolean = False;
    LNum: Integer = 0; LinCount: Integer = 0);
var L, P: Integer;
begin
    if LNum <= 0 then LNum := Max(Length(Lines), 1);
    if LinCount < LNum then LinCount := LNum;
    if LinCount > High(Lines) then
        begin
            P := Length(Lines);
            SetLength(Lines, LinCount + 1);
            for L := P to LinCount do begin New(Lines[L]); New(Lines[L]^Data) end
        end;
    with Lines[LNum]^ do
        begin Color := LColor; Title := LTitle; TitleLeft := LTitleLeft end
end;

procedure TImageFrame.SetExDataLine(PDataArr: PRealArray; LColor: TColor;
    LTitle: String; LTitleLeft: Boolean = False;
    LNum: Integer = -1; LinCount: Integer = 0);
var L, P: Integer;
begin
    if LNum < 0 then LNum := Max(Length(Lines), 1);
    if LinCount < LNum then LinCount := LNum;
    if LinCount > High(Lines) then
        begin
            P := Length(Lines);
            SetLength(Lines, LinCount + 1);
            for L := P to LinCount do New(Lines[L]);

```

```

end
else
begin
  with Lines[LNum]^ do
    if Assigned(Data) and not ((LNum+128) in ExDataLines) then
      Dispose(Data);
    LinCount := -1
  end;
with Lines[LNum]^ do
  begin
    Color := LColor; Title := LTitle; TitleLeft := LTitleLeft;
    Data := PDataArr;
    if Assigned(Data) then Include(ExDataLines, LNum+128) else New(Data)
  end;
for L := 1 to LinCount do with Lines[L]^ do
  if not Assigned(Data) then New(Data)
end;

procedure TImageFrame.SetExLineRec(PLineRec: PGraphLine;
  LNum: Integer = -1; LinCount: Integer = 0);
var L: Integer;
begin
  if LNum < 0 then LNum := Max(Length(Lines), 1);
  if LinCount < LNum then LinCount := LNum;
  if LinCount > High(Lines) then SetLength(Lines, LinCount + 1)
else
  begin
    if Assigned(Lines[LNum]) and not (Lnum in ExDataLines) then
      Dispose(Lines[LNum]);
    LinCount := -1
  end;
Lines[LNum] := PLineRec;
if Assigned(Lines[LNum])
then Include(ExDataLines, LNum)
else New(Lines[LNum]);
with Lines[LNum]^ do
  if Assigned(Data) then Include(ExDataLines, LNum+128) else New(Data);
for L := 1 to LinCount do
  begin
    if not Assigned(Lines[L]) then New(Lines[L]);
    with Lines[L]^ do if not Assigned(Data) then New(Data)
  end
end;

procedure TImageFrame.SetExtraLines(FromImage: TImageFrame);
begin
  ExtraLines := FromImage
end;

procedure TImageFrame.PutLnTitle(ALine: TGraphLine);
var X0, X1, Xt, Y: Integer;
begin
  with ALine do if Title <> '' then with Image.Canvas do
  begin
    if TitleLeft
    then
      begin
        X0 := Axis[LLeftAx].XlimitPos(YGraphScale(Data^[0]), True);
        Y := YtoGraph(Data^[0]);
        with Axis[0].LabelsShift do X1 := X0 - (Abs(x)+ Abs(y))*2;
        Xt := X1 - TextWidth(Title)
      end
    else
      begin
        X0 := Axis[RRightAx].XlimitPos(YGraphScale(Data^[PointsCount]), False);
        Y := YtoGraph(Data^[PointsCount]);
        with Axis[0].LabelsShift do X1 := X0 + (Abs(x)+ Abs(y))*2;
        Xt := X1;
      end;
    Pen.Color := Color; Font.Color := Color;
    MoveTo(X0, Y); LineTo(X1, Y);
    TextOut(Xt , Y - TextHeight(Title) div 2, Title)
  end
end;
end;

```



```

procedure TImageFrame.PutLnTitleAt(ALine: TGraphLine; Y: Integer);
var X0, X1, Xt: Integer;
begin
  with ALine do if Title <> '' then with Image.Canvas do
    begin
      if TitleLeft
      then
        begin
          X0 := Axis[LLeftAx].XlimitPos(YGraphScale(Data^[0]), True);
          with Axis[0].LabelsShift do X1 := X0 - (Abs(x)+ Abs(y))*2;
          Xt := X1 - TextWidth(Title)
        end
      else
        begin
          X0 := Axis[RRightAx].XlimitPos(YGraphScale(Data^[PointsCount]), False);
          with Axis[0].LabelsShift do X1 := X0 + (Abs(x)+ Abs(y))*2;
          Xt := X1;
        end;
      Pen.Color := Color; Font.Color := Color;
      MoveTo(X0, Y); LineTo(X1, Y);
      TextOut(Xt, Y - TextHeight(Title) div 2, Title)
    end
  end;

procedure TImageFrame.NewGraph;
var L, P: Integer;
begin
  Image.Canvas.Font.Assign(Font);
  with Image, Canvas do
    begin
      Brush.Color := clWhite; Pen.Color := Font.Color;
      FillRect(Rect(0, 0, Width, Height))
    end;
  for L := 0 to High(Axis) do Axis[L].Draw;
  with Image.Canvas do if PointsCount > 0 then for L := 1 to High(Lines) do
    begin
      Pen.Color := Lines[L]^Color; Font.Color := Lines[L]^Color;
      MoveTo(XtoGraph(Lines[0]^Data^[0]), YtoGraph(Lines[L]^Data^[0]));
      for P := 1 to PointsCount do
        LineTo(XtoGraph(Lines[0]^Data^[P]), YtoGraph(Lines[L]^Data^[P]));
      PutLnTitle(Lines[L]^);
    end;
  if Assigned(ExtraLines) and (ExtraLines.PointsCount > 0) then
    with Image.Canvas do for L := 1 to High(ExtraLines.Lines) do
      begin
        Pen.Color := ExtraLines.Lines[L]^Color;
        Font.Color := ExtraLines.Lines[L]^Color;
        MoveTo(XtoGraph(ExtraLines.Lines[0]^Data^[0]),
          YtoGraph(ExtraLines.Lines[L]^Data^[0]));
        for P := 1 to ExtraLines.PointsCount do
          LineTo(XtoGraph(ExtraLines.Lines[0]^Data^[P]),
            YtoGraph(ExtraLines.Lines[L]^Data^[P]));
        PutLnTitle(ExtraLines.Lines[L]^);
      end
    end;

procedure TImageFrame.PutLineTitles;
var L: Integer;
begin
  for L := 1 to High(Lines) do PutLnTitle(Lines[L]^);
end;

procedure TImageFrame.PutLineTitles(Ypos: array of Integer);
var L, M: Integer;
begin
  M := Min(Length(Ypos), High(Lines));
  for L := 1 to M do PutLnTitleAt(Lines[L]^, Ypos[L-1]);
  for L := M+1 to High(Lines) do PutLnTitle(Lines[L]^);
end;

procedure TImageFrame.AddGrPoint(NewData: array of Extended;
  ExtraNum: Integer = 0);
var L, M: Integer; NewLimits: Boolean;
begin
  Inc(PointsCount);

```

```

if PointsCount + ExtraNum > High(Lines[0]^Data^) then
  begin
    M := PointsCount + ExtraNum + 1;
    for L := 0 to High(Lines) do SetLength(Lines[L]^Data^, M)
  end;
for L := 0 to Min(High(Lines), High(NewData)) do
  Lines[L]^Data^[PointsCount] := NewData[L];
if FixLimits then NewLimits := False
else
  begin
    NewLimits := Axis[0].CheckLimits(NewData[0]);
    for L := 1 to High(NewData) do
      if Axis[1].CheckLimits(NewData[L]) then NewLimits := True
    end;
  if NewLimits
then NewGraph
else if PointsCount > 0 then
  with Image.Canvas do for L := 1 to High(Lines) do
  begin
    Pen.Color := Lines[L]^Color;
    MoveTo(XtoGraph(Lines[0]^Data^[PointsCount-1]),
      YtoGraph(Lines[L]^Data^[PointsCount-1]));
    LineTo(XtoGraph(Lines[0]^Data^[PointsCount]),
      YtoGraph(Lines[L]^Data^[PointsCount]));
  end
end;

procedure TImageFrame.PSHeader(const FileName: String);
begin
  PXLtoPS := 14400 div Screen.Width;
  X0PS := 7200 - Image.Width * PXLtoPS div 2; if X0PS < 0 then X0PS := 0;
  Y0PS := 9000 + Image.Height * PXLtoPS div 2;
  if Y0PS < Image.Height * PXLtoPS then Y0PS := Image.Height * PXLtoPS;
  AssignFile(PSfile, FileName);
  Rewrite(PSfile);
  WriteLn(PSfile, '!PS-Adobe EPSF-2.0');
  WriteLn(PSfile, '%Title: ', ExtractFileName(FileName));
  WriteLn(PSfile, '%CreationDate: ', DateTimeToStr(Now));
  WriteLn(PSfile, '%BoundingBox: ', X0ps div 25,
    ' ', (Y0PS - Image.Height * PXLtoPS) div 25,
    ' ', Ceil((X0PS + Image.Width * PXLtoPS) / 25),
    ' ', Ceil(Y0PS / 25));
  WriteLn(PSfile, '%EndComments');
  WriteLn(PSfile, '/gsdict 12 dict def');
  WriteLn(PSfile, 'gsdict begin');
  WriteLn(PSfile, '/gr {grestore} bind def');
  WriteLn(PSfile, '/gs {gsave} bind def');
  WriteLn(PSfile, '/l {lineto} bind def');
  WriteLn(PSfile, '/m {moveto} bind def');
  WriteLn(PSfile, '/s {stroke} bind def');
  WriteLn(PSfile, '/sc {scale} bind def');
  WriteLn(PSfile, '/sh {show} bind def');
  WriteLn(PSfile, '/slw {setlinewidth} bind def');
  WriteLn(PSfile, '/srgb {setrgbcolor} bind def');
  WriteLn(PSfile, '/ff {findfont} bind def');
  WriteLn(PSfile, '/sf {setfont} bind def');
  WriteLn(PSfile, '/scf {scalefont} bind def');
  WriteLn(PSfile);
  WriteLn(PSfile, '%Page: 1 1');
  WriteLn(PSfile);
  WriteLn(PSfile, '0.04 0.04 sc');
  WriteLn(PSfile, PXLtoPS, ' slw');
  PSFont(Font);
  PSColor(Font.Color);
  WriteLn(PSfile)
end;

procedure TImageFrame.ImageToPS;
var L, P: Integer; Y: Extended;
begin
  for L := 0 to High(Axis) do Axis[L].SavePS;
  Image.Canvas.Font.Assign(Font);
  if PointsCount > 0 then for L := 1 to High(Lines) do
  begin
    WriteLn(PSfile, 'gs');

```

```

PSColor(Lines[L].Color);
PSMove(XGraphScale(Lines[0]^Data^[0]), YGraphScale(Lines[L]^Data^[0]));
for P := 1 to PointsCount do
  PSLine(XGraphScale(Lines[0]^Data^[P]), YGraphScale(Lines[L]^Data^[P]));
with Lines[L]^ do if Title <> '' then
  if TitleLeft
  then
  begin
    Y := YGraphScale(Data^[0]); P := Axis[LLeftAx].XlimitPos(Y, True);
    PSMove(P, Y);
    with Axis[0].LabelsShift do Dec(P, (Abs(x)+ Abs(y))*2);
    PSLine(P, Y);
    PSText(P - Image.Canvas.TextWidth(Title),
            Y - Image.Canvas.TextHeight(Title) div 2,
            Title)
  end
else
  begin
    Y := YGraphScale(Data^[PointsCount]);
    P := Axis[RRightAx].XlimitPos(Y, False);
    PSMove(P, Y);
    with Axis[0].LabelsShift do Inc(P, (Abs(x)+ Abs(y))*2);
    PSLine(P, Y);
    PSText(P, Y - Image.Canvas.TextHeight(Title) div 2, Title)
  end;
WriteLn(PSfile, ' s gr')
end;
if Assigned(ExtraLines) and (ExtraLines.PointsCount > 0) then
for L := 1 to High(ExtraLines.Lines) do
begin
  WriteLn(PSfile, 'gs');
  PSColor(ExtraLines.Lines[L].Color);
  PSMove(XGraphScale(ExtraLines.Lines[0]^Data^[0]),
          YGraphScale(ExtraLines.Lines[L]^Data^[0]));
  for P := 1 to ExtraLines.PointsCount do
    PSLine(XGraphScale(ExtraLines.Lines[0]^Data^[P]),
            YGraphScale(ExtraLines.Lines[L]^Data^[P]));
  with ExtraLines.Lines[L]^ do if Title <> '' then
    if TitleLeft
    then
    begin
      Y := YGraphScale(Data^[0]); P := Axis[LLeftAx].XlimitPos(Y, True);
      PSMove(P, Y);
      with Axis[0].LabelsShift do Dec(P, (Abs(x)+ Abs(y))*2);
      PSLine(P, Y);
      PSText(P - Image.Canvas.TextWidth(Title),
              Y - Image.Canvas.TextHeight(Title) div 2,
              Title)
    end
    else
    begin
      Y := YGraphScale(Data^[PointsCount]);
      P := Axis[RRightAx].XlimitPos(Y, False);
      PSMove(P, Y);
      with Axis[0].LabelsShift do Inc(P, (Abs(x)+ Abs(y))*2);
      PSLine(P, Y);
      PSText(P, Y - Image.Canvas.TextHeight(Title) div 2, Title)
    end;
  WriteLn(PSfile, ' s gr')
end
end;
end;

procedure TImageFrame.PSClose;
begin
  WriteLn(PSfile, 'showpage');
  WriteLn(PSfile, 'end');
  WriteLn(PSfile, 'grestore');
  CloseFile(PSfile)
end;

procedure TImageFrame.SavePStoFile(const FileName: String);
begin
  PSHeader(FileName);
  ImageToPS;
  PSClose
end;

```

```

end;

function TImageFrame.XGraphScale;
begin Result := Axis[0].XGraphScale(Value) end;

function TImageFrame.YGraphScale;
begin Result := Axis[1].YGraphScale(Value) end;

function TImageFrame.XtoGraph;
begin with Axis[0] do begin
  if Value > MaxValue then Result := XtoGraph(MaxValue)
  else if Value < MinValue then Result := XtoGraph(MinValue)
  else Result := XtoGraph(Value)
end end;

function TImageFrame.YtoGraph;
begin with Axis[1] do begin
  if Value > MaxValue then Result := YtoGraph(MaxValue)
  else if Value < MinValue then Result := YtoGraph(MinValue)
  else Result := YtoGraph(Value)
end end;

function TImageFrame.GraphXvalue;
begin Result := Axis[0].GraphXvalue(X) end;

function TImageFrame.GraphYvalue;
begin Result := Axis[1].GraphYvalue(Y) end;

procedure TImageFrame.ImageMouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  if Installed then
    Hint := Format('(%d, %d) : (%.4g, %.4g)',
      [X, Y, GraphXvalue(X), GraphYvalue(Y)])
end;

procedure TImageFrame.SaveAs(Sender: TObject);
var JP: TJPEGImage;
begin
  if SaveDialog.Execute then
    if SaveDialog.FilterIndex = 2 then
      begin
        JP := TJPEGImage.Create;
        try
          with JP do
            begin
              Assign(ImagePrintForm.Image.Picture.Bitmap);
              SaveToFile(SaveDialog.FileName)
            end;
          finally
            jp.Free;
          end
        end
      else ImagePrintForm.Image.Picture.SaveToFile(SaveDialog.FileName);
end;

procedure TImageFrame.ImagePrint(Sender: TObject);
begin
  Application.CreateForm(TImagePrintForm, ImagePrintForm);
  try
    ImagePrintForm.ClientHeight := Image.Height;
    ImagePrintForm.ClientWidth := Image.Width;
    with Image do
      ImagePrintForm.Image.Canvas.CopyRect(Rect(0,0, Width, Height), Canvas,
        Rect(0,0, Width, Height));
    ImagePrintForm.Show;
    if PrintScalingDialog.Execute(Self.Owner as TForm)
      and PrintDialog.Execute
    then
      begin
        ImagePrintForm.PrintScale := (Self.Owner as TForm).PrintScale;
        ImagePrintForm.Print;
        ImagePrintForm.Close;
      end
    finally
  end;
end;

```

```

ImagePrintForm.Free;
ImagePrintForm := Nil
end
end;

procedure TImageFrame.SavePS(Sender: TObject);
var DefFilter, DefExt: String; DefIndex: Integer;
begin
  with SaveDialog do
    begin
      DefIndex := FilterIndex; FilterIndex := 1;
      DefFilter := Filter; Filter := 'Pictrue to PostScript file|*.eps;*.ps';
      DefExt := DefaultExt; DefaultExt := 'eps';
      if FileName <> '' then
        FileName := Copy(FileName, 1,
          Length(FileName) - Length(ExtractFileExt(FileName)))
      end;
    try
      if SaveDialog.Execute then SavePStoFile(SaveDialog.FileName)
    finally
      SaveDialog.Filter := DefFilter;
      SaveDialog.FilterIndex := DefIndex;
      SaveDialog.DefaultExt := DefExt
    end
  end;
end;

procedure TImageFrame.CrossOn;
begin with Image.Canvas, Pen, CrossPos do begin
  Color := clGreen; Mode := pmXor;
  MoveTo(X-5, Y); LineTo(X+5, Y); MoveTo(X, Y-5); LineTo(X, Y+5);
  Crossing := not Crossing
end end;

procedure TImageFrame.CrossOn(Pos: TPoint);
begin
  if Crossing then CrossOn;
  CrossPos := Pos; CrossOn;
end;

procedure TImageFrame.CrossOn(X, Y: Integer);
begin CrossOn(Point(X, Y)) end;

procedure TImageFrame.CrossOff;
begin if Crossing then CrossOn end;

procedure TImageFrame.CrossOnVal(X, Y: Extended);
begin CrossOn(XtoGraph(X), YtoGraph(Y)) end;

procedure TImageFrame.SaveDialogTypeChange(Sender: TObject);
begin
  case SaveDialog.FilterIndex of
    1: SaveDialog.DefaultExt := 'BMP';
    2: SaveDialog.DefaultExt := 'JPG';
  end;
end;
end.

```