LMComponents: object-oriented extension of LMath library

Viatcheslav Nesterov

July 16, 2020

Contents

Chapter 1

Unit lmPointsVec

1.1 Description

`TPoints` is a class wrapper around `TRealPointVector` (see LMath.pdf, Chapter 2). Its properties X[index] and Y[index] provide access to X and Y fields as to separate arrays `Float`; if the package was compiled with -dDebug setting, range check is completed for Index. However, use of `X` and `Y` properties has a considerable penalty on performance. Direct access to underlying array `TPoints.Points` is provided for use in time-critical code sections, but in general it is safer to use `X` and `Y` properties.

If `Append` procedure is used for adding new points, `Count` is adjusted automatically; by exceeding current `Capacity`, memory is automatically reallocated.

In addition, many utility methods and properties are provided, such as `MaxX, MaxY, MinX, MinY`; `RemovePoints` allows to remove subarray from an arbitrary index; `constructor Combine` allows to create `TPoints` from two vectors of Float; opposite to it, `Extract` is a mean to extract `X` or `Y` as vector of Float; `SortX` and `SortY` sort `Points` as names suggest.

1.2 Classes, Interfaces, Objects and Records

1.2.1 ERealPointsException Class

Hierarchy

ERealPointsException > Exception

Description

Exception which flags invalid operation with `TPoints`.

1.2.2 TPoints Class

Hierarchy

TPoints > TObject

Declaration

```
TPoints = class
protected
  function GetBuffer(I: integer): pointer;
  function GetX(ind:integer):Float;
  function GetY(ind:integer):Float;
  procedure SetX(ind:integer; value:Float);
  procedure SetY(ind:integer; value:Float);
  function GetPoint(ind:integer):TRealPoint;
  procedure SetPoint(ind:integer; Value:TRealPoint);
public
  Points:TRealPointVector;
  Capacity:integer;
  Count:integer;
  Index:integer;
  constructor Create(ACapacity:integer);
  constructor Combine(XVector,YVector:TVector; Lb, Ub:integer);
```

```
  destructor Destroy; override;
  procedure Append(APoint:TRealPoint);
  function RemovePoints(Ind: integer; ACount:integer):integer;
  function Reallocate(Step:integer):integer;
  procedure FreePoints; virtual;
  procedure AllocatePoints(ACapacity:integer);
  procedure SortX(descending:boolean);
  function MaxX: Float; virtual;
  function MaxY: Float; virtual;
  function MinX: Float; virtual;
  function MinY: Float; virtual;
  function Range: Float; virtual;
  function RangeY: Float; virtual;
  procedure SortY(descending:boolean);
  procedure ExtractX(var AXVector:TVector; Lb, Ub: integer);
  procedure ExtractY(var AYVector:TVector; Lb, Ub: integer);
  property X[I:integer]:Float read GetX write SetX;
  property Y[I:integer]:Float read GetY write SetY;
  property ThePoints[I:integer]:TRealPoint read GetPoint write SetPoint; default;
  property DataBuffer[I:integer]:pointer read GetBuffer;
end;
```

Properties

X `public property X[I:integer]: Float;`

Description Shortcut to `Points[index].X`. By *assigning* X[index], if `Index` > `Count-1`, Count is increased to `Index+1`. If `index` > `Capacity` and -dDebug is set, `ERealPointException` is raised.

By reading the field, if -dDebug and `index` > `Count`, then exception is raised.

Use of this field has, however, penalty on performance; use direct access to `Points` array in time-critical procedures.

Y `public property Y[I:integer]: Float;`

Description Shortcut to `Points[index].Y`. See X property above for description of its behaviour.

ThePoints `public property ThePoints[I:integer]: TRealPoint;`

Description Access to `Points` with range-check; behaviour is similar to X and Y properties, as well as performance penalty.

DataBuffer `public property DataBuffer[I:integer]: pointer;`

Pointer to Points[I]. Useful for fast low-level filling of the data.

Fields

Points `public Points:TRealPointVector;`

Description This is actual array of data. Public access to it is provided for direct operations in time-critical program sections. Don't forget to use and adjust `Count` and `Capacity` fields! Outside time-critical sections, use `X`, `Y` and `ThePoints` properties.

Capacity `public Capacity:integer;`

Description This is currently *allocated* `Points` length. It should not be confused with `Count` which shows number of currently *assigned* points (or, to be exact, highest index of assigned element). If new points are added using `Append` method, `Count` is updated and memory is automatically reallocated as needed.

Index `public Index:integer;`

Description General use pointer. After call of MinY, MaxY, MinX, MaxX functions it points to the first element which has corresponding value.

Count `public Count:integer;`

Description Highest index of assigned element in `Points`. It is automatically adjusted when `X[index]`, `Y[index]` or `ThePoints[index]` properties are assigned, when `RemovePoints` is used or when `Append` procedure is used to add a new point. `Append` adds always to `Points[Count]` position. Attempt to read beyond `Count` raises exception if -dDebug was used. If low-level access to `Points` array was used, `Count` should be adjusted manually.

Methods

Create

Declaration `public constructor Create(ACapacity:integer);`

Description Creates `TPoints` object and allocates memory for `Points` with `Capacity` elements.

Combine

Declaration `public constructor Combine(XVector,YVector:TVector; Lb, Ub:integer);`

Description Creates TPoints object with Points array combined from two arrays of float: `XVector` is used to fill `X` fields in `Points` array; `YVector` is for `Y` fields. `Lb, Ub` are low and upper indexes of the vectors to use. `Count` and `Capacity` of resulting `TPoints` is set to $Ub - Lb$.

Destroy

Declaration `public destructor Destroy; override;`

Append

Declaration `public procedure Append(APoint:TRealPoint);`

Description Appends a point to the end (Count position) and increases Count. If Capacity is exceeded, automatically reallocates more space.

RemovePoints

Declaration `public function RemovePoints(Ind: integer; ACount:integer):integer;`

Description removes `min(ACount, Count-Ind)` points starting from `Ind`, moves rest to left. Returns number of actually removed points. Adjusts `Count` to new value.

Reallocate

Declaration `public function Reallocate(Step:integer):integer;`

Description `Reallocate(Step:integer)` increases `Capacity` by `Step`.

FreePoints

Declaration `public procedure FreePoints; virtual;`

Description Frees `Points`, sets `Count` and `Capacity` to zero.

AllocatePoints

Declaration `public procedure AllocatePoints(ACapacity:integer);`

Description AllocatePoints(ACapacity:integer) allocates given capacity; unlike `Reallocate` does not take into account preexisting `Capacity`.

MinX, MaxX, MinY, MaxY

Declaration
```
function MinX: Float; virtual;
function MaxX: Float; virtual;
function MinY: Float; virtual;
function MaxY: float virtual;
```

Description Return maximal and minimal X and Y values, according to the names. After the call, Index field points to the first found element with this value. These functions use simple linear search. If structure of your data allows more efficient algorithms, override these functions, but don't forget to update Index field.

Range

Declaration `function Range: Float; virtual;`

Description $Range = MaxX - MinX$

RangeY

Declaration `function RangeY: Float;`

Description $RangeY = MaxY - MinY$

SortX, SortY

Declaration `public procedure SortX(descending:boolean); public procedure SortY(descending:boolean);`

Description Sort `Points` by `X` or `Y`, accordingly; if descending then in descending order, otherwise in ascending.

ExtractX, ExtractY

Declaration `public procedure ExtractX(var AXVector:TVector; Lb, Ub: integer); public procedure ExtractY(var AYVector:TVector; Lb, Ub: integer);`

Description Extract all X from [Lb..Ub] interval as TVector. If length of AXVector or AYVector is insufficient, it is reallocated.

Chapter 2

Unit lmFilters

2.1 Description

Unit lmFilters includes several off-line digital filters of a signal which are implemented as non-visual components. You can drop a component on your form, define sampling rate and cut-off frequency, define `OnInput` and `OnOutput` events and call `Filter` method. `OnInput` event is called from `Filter` method whenever the filtering procedure needs next input value, and `OnOutput` when it is ready to return a next value. This technique makes the components independent of a format of data which are filtered.

Currently, implemented are gaussian filter, moving average filter, which are probably the best "smoothing" filters for time domain, and median filter which is ideal to remove short spikes preserving sharp edges.

2.2 Types

2.2.1 TInputFunc

Declaration `TInputFunc = function(Index:integer):Float of Object;`

2.2.2 TOutputProc

Declaration `TOutputproc = procedure(Val:Float; Index:integer) of Object;`

2.3 Classes, Interfaces, Objects and Records

2.3.1 EFilterException Class

Hierarchy

EFilterException > exception

2.3.2 TDigFilter Class

Hierarchy

TDigFilter > TComponent

Declaration

```
TDigFilter = class(TComponent)
protected
  FOnInput:TInputFunc;
  FOnOutput:TOutputProc;
public
  procedure Filter(StartIndex, EndIndex:integer); virtual; abstract;
published
  property OnInput:TInputFunc read FOnInput write FOnInput;
  property OnOutput:TOutputProc read FOnOutput write FOnOutput;
end;
```

Description

TDigFilter is an abstract ancestor class for all digital filters. Itself it is never instantiated, but introduces important common behaviour.

Events

OnInput `published property OnInput: TInputFunc read FOnInput write FOnInput;`

`function(Index:integer):Float of object;` must provide a value of input signal at index `Index`. It is called from `Filter` method.

OnOutput `published property OnOutput: TOutputProc read FOnOutput write FOnOutput;`

`procedure(Val:Float; Index:integer) of object` receives a value of filtered signal at `Index` and can do with it what a user needs. It is called from `Filter` method.

Both OnInput and OnOutput events are called from Filter method, to get next value from the data stream been filtered. This technique makes the filter independent from an actual data format.

The most simple implementation of these events may be following:

```
uses uTypes, lmFilters;
var
  DataArr:TVector;
{.....}
function Main.MyFilterInputFunc(Index:integer):Float;
begin
  Result := DataArr[Index];
end;

procedure Main.MyFilterOutputProc(Val:Float; Index:integer);
begin
  DataArr[Index] := Val;
end;
```

Fields

FOnInput `protected FOnInput:TInputFunc;`

FOnOutput `protected FOnOutput:TOutputProc;`

Methods

Filter

Declaration `public procedure Filter(StartIndex, EndIndex:integer); virtual; abstract;`

Description Receives inpus signal values calling `OnInput`, makes actual filtering and outputs result calling `OnOutput`.

2.3.3 TOneFreqFilter Class

Hierarchy

TOneFreqFilter > `TDigFilter` > TComponent

Declaration

```
TOneFreqFilter = class(TDigFilter)
private
  FSamplingRate : Float;
  FCutFreq1     : Float;
public
  constructor Create(AOwner:TComponent); override;
  procedure SetupFilter(ASamplingRate, ACutFreq1 : Float); virtual;
published
  property SamplingRate : Float read FSamplingRate;
  property Cutfreq1     : Float read FCutFreq1;
end;
```

Description

Descendant of `TDigFilter` which describes lowpass or highpass filters (but not pass- or stopband) with Infinite Impulse Response. Introduces `SamplingRate` and `CutFreq1` properties.

Properties

SamplingRate `published property SamplingRate : Float read FSamplingRate;`

Description Sampling rate, usually Hz.

CutFreq1 `published property CutFreq1 : Float read FCutFreq1;`

Description Cut (or corner) frequency, usually Hz. Must be less then `SamplingRate`.

Methods

Create

Declaration `public constructor Create(AOwner:TComponent); override;`

Description Calls `inherited Create`, sets `SamplingRate` to 14400 and `CutFreq1` at 4000.

SetupFilter

Declaration `public procedure SetupFilter(ASamplingRate, ACutFreq1 : Float); virtual;`

Description Procedure which sets `SamplingRate` and `CutFrequency`. It must be called before first call of `Filter` method.

2.3.4 TFIRFilter Class

Hierarchy

TFIRFilter > `TOneFreqFilter` > `TDigFilter` > TComponent

Declaration

```
TFIRFilter = class(TOneFreqFilter)
protected
  FWinLength : integer;
  procedure SetWinLength(L:integer); virtual;
public
  constructor Create(AOwner:TComponent); override;
published
  property WinLength : integer read FWinLength write SetWinLength;
end;
```

Description

TFIRFilter: Finite Impulse response filter. Abstract class, descendant of `TOneFreqFilter` which introduces `WinLength` property for the filter window length (or length of the Impulse Response).

Properties

WinLength `published property WinLength : integer read FWinLength write SetWinLength;`

Fields

FWinLength `protected FWinLength: integer;`

Methods

Create

Declaration `public constructor Create(AOwner:TComponent); override;`

Description Calls `inherited Create`, sets `WinLength` to 5.

SetWinLength

Declaration `protected procedure SetWinLength(L:integer); virtual;`

Description Sets WinLength to L.

2.3.5 TMovAvFilter Class

Hierarchy

TMovAvFilter > `TFIRFilter` > `TOneFreqFilter` > `TDigFilter` > TComponent

Declaration

```
TMovAvFilter = class(TFIRFilter)
protected
  procedure SetWinLength(L:integer); override;
public
  procedure Filter(StartIndex, EndIndex:integer); override;
  procedure SetupFilter(ASamplingRate, ACutFreq:Float); override;
published
  property WinLength : integer read FWinLength write SetWinLength;
end;
```

Description

Implements moving average filter. It is possible to use SetupFilter procedure to set `CutFreq1` and `SamplingRate` fields or directly set WinLength. In the first case, needed `WinLength` is automatically calculated; in the second case, resulting `CutFreq1` is automatically found, provided that `SamplingRate` was previously set. So, these approaches are mutually exclusive.

Properties

WinLength `published property WinLength : integer read FWinLength write SetWinLength;`

Methods

SetWinLength

Declaration `protected procedure SetWinLength(L:integer); override;`

Description Sets WinLength, calculates corresponding `CutFreq1`.

Filter

Declaration `public procedure Filter(StartIndex, EndIndex:integer); override;`

SetupFilter

Declaration `public procedure SetupFilter(ASamplingRate, ACutFreq:Float); override;`

2.3.6 TGaussFilter Class

Hierarchy

TGaussFilter > `TOneFreqFilter` > `TDigFilter` > TComponent

Declaration

```
TGaussFilter = class(TOneFreqFilter)
public
  constructor Create(AOwner:TComponent); override;
  procedure SetupFilter(ASamplingRate, ACutFreq1: Float); override;
  procedure Filter(StartIndex, EndIndex:integer); override;
end;
```

Description

Implements gaussian filter with the algorithm described in:
Young I.T., L.J. van Vliet. Recursive implementation of the Gaussian Filter. // Signal Processing, 44 (1995) 139-151

Methods

Create

Declaration `public constructor Create(AOwner:TComponent); override;`

Description Calls `inherited Create`, calculates all necessary filter coefficients.

SetupFilter

Declaration `public procedure SetupFilter(ASamplingRate, ACutFreq1: Float); override;`

Description Sets `SamplingRate` and `CutFreq1`, calculates corresponding filter coefficients.

Filter

Declaration `public procedure Filter(StartIndex, EndIndex:integer); override;`

2.3.7 TMedianFilter Class

Hierarchy

TMedianFilter > `TFIRFilter` > `TOneFreqFilter` > `TDigFilter` > TComponent

Declaration

```
TMedianFilter = class(TFIRFilter)
protected
  function FindMedian:Float;
  procedure SetWinLength(L:integer); override;
public
  constructor Create(AOwner:TComponent); override;
  procedure filter(StartIndex, EndIndex:integer); override;
end;
```

Description

Implementation of Median Filter

Methods

FindMedian

Declaration `protected function FindMedian:Float;`

Description Finds Median of the filtering window. Is called from `Filter`.

SetWinLength

Declaration `protected procedure SetWinLength(L:integer); override;`

Description Sets WinLength property, internally allocates buffer for median search. WinLength must be ≥ 3 and odd.

Create

Declaration `public constructor Create(AOwner:TComponent); override;`

Description Calls `inherited Create`, setting window length to 5, allocates corresponding buffer for median search.

Filter

Declaration `public procedure Filter(StartIndex, EndIndex:integer); override;`

2.4 Functions and Procedures

2.4.1 GaussCascadeFreq

Declaration `function GaussCascadeFreq(Freq1, Freq2:Float):Float;`

Description Finds effective cutoff frequency of cascade of 2 gaussian filters.

2.4.2 GaussRiseTime

Declaration `function GaussRiseTime(Freq:Float):Float;`

Description Finds risetime (10–90%) of a gaussian filter with given cut-off frequency.

2.4.3 MovAvRiseTime

Declaration `function MovAvRiseTime(SamplingRate:Float; WLength:integer):Float;`

Description Risetime of moving average filter (0–100%).

2.4.4 MoveAvCutOffFreq

Declaration `function MoveAvCutOffFreq(SamplingRate:Float; WLength:integer):Float;`

Description Cut-off frequency of a moving average filter, given sampling rate and window length.

2.4.5 MoveAvFindWindow

Declaration `function MoveAvFindWindow(SamplingRate, CutOffFreq:Float):Integer;`

Description Finds required window length from desired cut-off frequency and given sampling rate.

2.4.6 Register

Declaration `procedure Register;`

Chapter 3

Unit lmcoordsys

3.1 Description

TCoordSys component implemented in this unit is relatively simple Cartesian coordinate plane for drawing points, lines, graphical primitives and mathematical functions in user's coordinates. Component is derived from TPanel, so, you can place other components, for example, scale edits, on top of it.

Usage: place the component on your form, in the Object Inspector define positions of axes, distance between grid lines (in user space) as well as coordinate limits (MinX, MaxX, MinY, MaxY). Define TPen properties which are used for drawing axes, grid lies and user data as well as numeric format for axes numbering.

For drawing of user's data define OnDrawdata event; all your drawing must occur within it.

Coordinates are converted between user space and screen coordinates with UserToScreen, ScreenToUser, XUserToScreen, YUserToScreen, XScreenToUser and YScreenToUser functions, but you seldom need to call them directly. Procedures PutLine, GoToXY, LineTo, Circle, Aim, FillRect are provided for drawing graphical primitives and data in user's space. Procedure FastDraw serves for fast drawing of arrays of TPoint sorted for X coordinate; DrawSpline and DrawFunc provide plotting of data and mathematical functions.

Canvas property is published, allowing to define easily own drawing procedures.

All drawing of user's data must occur in OnDrawData event, which is called from Paint procedure.

3.2 Classes

3.2.1 TCoordSys Class

Hierarchy

TCoordSys > TPanel

Declaration

```
TCoordSys = class(TPanel)
protected
  function GetFont:TFont;
  procedure SetXAxisLabel(const ALabel:String);virtual;
  function GetXAxisLabel:string; virtual;
  procedure SetYAxisLabel(const ALabel:String);virtual;
  function GetYAxisLabel:string; virtual;
  procedure SetMinX(AMinX:Float); virtual;
  procedure SetMaxX(AMaxX:Float); virtual;
  procedure SetMinY(AMinY:Float); virtual;
  procedure SetMaxY(AMaxY:Float); virtual;
  procedure DrawAxis; virtual;
  procedure DrawGridLines; virtual;
  procedure DrawAxisLabels; virtual; abstract;
  procedure SetLeftMargin(AMargin:integer); virtual;
  procedure SetRightMargin(AMargin:integer); virtual;
```

```
  procedure SetLowerMargin(AMargin:integer); virtual;
  procedure SetUpperMargin(AMargin:integer); virtual;
  procedure SetXPos(AXPos:Float); virtual;
  procedure SetYPos(AYPos:Float); virtual;
  procedure SetXGridDist(AXGridDist:Float); virtual;
  procedure SetYGridDist(AYGridDist:Float); virtual;
  procedure SetAxisPen(APen:TPen); virtual;
  procedure SetGridPen(APen:TPen); virtual;
  procedure SetOutputPen(APen:TPen); virtual;
  procedure SetPenPos(APenPos:TRealPoint); virtual;
  procedure SetGridDir; virtual; abstract;
public
  ScaleX, ScaleY:Float;
  property PenPos: TRealPoint read FPenPos write SetPenPos;
  constructor Create(AOwner:TComponent); override;
  destructor Destroy; override;
  procedure Paint; override;
  procedure LineTo(APoint:TRealPoint); overload;
  procedure LineTo(X,Y:Float); overload;
  procedure NewLimits(AMinX,AMinY,AMaxX,AMaxY:Float); virtual;
  procedure XScrollTo(AX:Float); virtual;
  procedure YScrollTo(AY:Float); virtual;
  procedure PutLine(P1,P2:TRealPoint); overload;
  procedure PutLine(X1,Y1,X2,Y2:Float); overload;
  function UserToScreen(UP:TRealPoint):TPoint;virtual;
  function  XUserToScreen(X:Float):integer;virtual;
  function  YUserToScreen(Y:Float):integer;virtual;
  function  XScreenToUser(X:integer):Float; virtual;
  function  YScreenToUser(Y:integer):Float; virtual;
  procedure Circle(Center:TRealPoint; R:integer); virtual;
  procedure Aim(Center: TRealPoint; R: integer); virtual;
  procedure FillRect(X1,Y1,X2,Y2:Float); overload;
  procedure Fillrect(P1,P2:TRealPoint); overload;
  procedure GoToXY(X,Y:Float);
  function ScreenToUser(SP:TPoint):TRealPoint; virtual;
  procedure ReScale(CoeffX, CoeffY:Float);
  procedure FastDraw(APoints:TRealPointVector; Lb, Ub: integer);
  procedure DrawSpline(APoints:TPoints; Lb, Ub: integer);
  procedure DrawFunc(AFunc:TParamFunc; Params:Pointer; LeftX, RightX : Float); virtua
published
  property XAxisLabel:string read FXAxisLabel write FXAxisLabel;
  property YAxisLabel:string read FYAxisLabel write FYAxisLabel;
  property MinX:Float read FMinX write SetMinX;
  property MinY:Float read FMinY write SetMinY;
  property MaxX:Float read FMaxX write SetMaxX;
  property MaxY:Float read FMaxY write SetMaxY;
  property XPos:Float read FXPos write SetXPos;
  property YPos:Float read FYPos write SetYPos;
```

```
  property Font:TFont read GetFont;
  property AxisPen:TPen read FAxisPen write SetAxisPen;
  property OutputPen:TPen read FOutputPen write SetOutputPen;
  property GridPen:TPen read FGridPen write SetGridPen;
  property LeftMargin:integer read FLeftMargin write SetLeftMargin default 0;
  property RightMargin:integer read FRightMargin write SetRightMargin default 0;
  property LowerMargin:integer read FLowerMargin write SetLowerMargin default 0;
  property UpperMargin:integer read FUpperMargin write SetUpperMargin default 0;
  property XGridDist:Float read FXGridDist write SetXGridDist;
  property YGridDist:Float read FYGridDist write SetYGridDist;
  property XGridNumbersPrecision: integer read FXGridNumbersPrecision write FXGridNum
  property XGridNumbersDecimals: integer read FXGridNumbersDecimals write FXGridNumbe
  property YGridNumbersPrecision: integer read FYGridNumbersPrecision write FYGridNum
  property YGridNumbersDecimals: integer read FYGridNumbersDecimals write FYGridNumbe
  property Canvas;
  property OnDrawData:TNotifyEvent read FOnPaint write FOnPaint;
end;
```

Description

TCoordSys

Properties

PenPos `public property PenPos: TRealPoint;`

Starting position for LineTo. It can be set with GoToXY method, or assigned directly. Difference is that for direct assignment coordinates must be represented as `TRealPoint`, while for `GoToXY` as separate `X,Y:Float`.

XAxisLabel `published property XAxisLabel: string;`

Label of X axis.

YAxisLabel `published property YAxisLabel: string read FYAxisLabel write FYAxisLabel;`

Label of Y axis.

MinX,MinY,MaxX,MaxY

```
published property MinX: Float;
published property MinY: Float;
published property MaxX: Float;
published property MaxY: Float;
```

MinX,MinY,MaxX,MaxY define window bounds in user coordinate space.

XPos `published property XPos: Float;`

Description Position of X-axis in Y-coordinate. Default is 0 as well as for YPos, such that axes cross at (0,0) point.

YPos `published property YPos: Float;`

Description Position of Y-axis in X-coordinate. Default is 0 as well as for XPos, such that axes cross at (0,0) point.

Font `published property Font: TFont;`

AxisPen `published property AxisPen: TPen;`

Pen to draw axis.

OutputPen `published property OutputPen: TPen;`

Pen to draw user's output (from OnDrawData event).

GridPen `published property GridPen: TPen;`

Pen to draw gridlines.

LeftMargin, RightMargin

Upper Margin

LowerMargin
```
published property LeftMargin: integer; default 0;
published property RightMargin: integer; default 0;
published property LowerMargin: integer; default 0;
published property UpperMargin: integer; default 0;
```

Width of margins, in pixel

XGridDist `published property XGridDist: Float;`

Distance between grid lines or ticks on X axis in user space coordinates.

YGridDist `published property YGridDist: Float;`

Distance between grid lines or ticks on Y axis in user space coordinates.

Axis numbering
```
published property XGridNumbersPrecision:integer; default 5;
published property XGridNumbersDecimals:integer; default 2;
```

`Precision` and `Decimal` parameters for `FloatToStrF` call for X axis numbering.
```
  published property YGridNumbersPrecision: integer; default 9;
published property YGridNumbersDecimals: integer; default 4;
```

`Precision` and `Decimal` parameters for `FloatToStrF` call for Y axis numbering.

Canvas `published property Canvas;`

OnDrawData `published property OnDrawData: TNotifyEvent;`

All drawing of user data (like drawfunction, fastdraw, all user-defined drawing etc.) must be done in this event.

Fields

ScaleX, ScaleY
```
public ScaleX:Float;
public ScaleY:Float;
```

Pixels per user unit. You must never set these values manually; they are automatically recalculated by window risizing or changes of MinX, MaxX, MinY, MaxY.

Methods

Create

Declaration `public constructor Create(AOwner:TComponent); override;`

Description Calls `inherited Create`, then creates AxisPen, GridPen and OutputPen.

Destroy

Declaration `public destructor Destroy; override;`

Paint

Declaration `public procedure Paint; override;`

Description Calls inherited (TPanel) Paint, then draws axes using `AxisPen`, after it draws grid-lines and ticks using `GridPen` and, finally, sets `OutputPen` as active pen and calls OnDrawData, where all user-defined data drawing must occur.

NewLimits

Declaration
```
public procedure NewLimits(AMinX,AMinY,AMaxX,AMaxY:Float);
virtual;
```

Description Sets new window bounds in user coordinate space (MinX, MinY, MaxX,MaxY), calls RedrawCoordSys to reflect changes.

ReScale

Declaration `public procedure ReScale(CoeffX, CoeffY:Float);`

Description Multiplies all coordinates, axes and grid positions by a factors `CoeffX` and `CoeffY`. This may be useful for conversion of units of user space, fro example between metric and imperial systems.

XScrollTo, YScrollTo

Declaration
```
public procedure XScrollTo(AX:Float); virtual;
public procedure YScrollTo(AY:Float); virtual;
```

Description Procedures for scroll in `X` or `Y` direction: set `MinX` to `AX` or `MinY` to `AY`, modify `MaxX` or `MaxY` accordingly such that scale is preserved. Redraw the coordinate system and user data.

UserToScreen, XUserToScreen, YUserToScreen

Declaration
```
public function UserToScreen(UP:TRealPoint):TPoint; virtual;
public function XUserToScreen(X:Float):integer; virtual;
public function YUserToScreen(Y:Float):integer; virtual;
```

Description Convert user space coordinates to screen coordinates.

ScreenToUser, XScreenToUser, YScreenToUser

Declaration
```
public function ScreenToUser(SP:TPoint):TRealPoint; virtual;
public function XScreenToUser(X:integer):Float; virtual;
public function YScreenToUser(Y:integer):Float; virtual;
```

Description Convert screen coordinates to user space coordinates.

DrawAxis

Declaration
```
protected procedure DrawAxis; virtual;
```

Description Procedure which draws axes. Is automatically called from `Paint` method, normally user does not call it manually.

DrawGridLines

Declaration
```
protected procedure DrawGridLines; virtual;
```

Description Procedure for drawing ticks and grids. Called from `Paint`.

GoToXY

Declaration
```
public procedure GoToXY(X,Y:Float);
```

Description Sets PenPos property to (X,Y) point. This property is used by LineTo procedure.

LineTo

Declaration
```
public procedure LineTo(APoint:TRealPoint); overload;
public procedure LineTo(X,Y:Float); overload;
```

Description Draws line from PenPos to `APoint` or (X,Y) and updates `PenPos`.

PutLine

Declaration
```
public procedure PutLine(P1,P2:TRealPoint); overload;
public procedure PutLine(X1,Y1,X2,Y2:Float); overload;
```

Description Puts line of OutputColor from (X1,Y1) to (X2,Y2) or from P1 to P2. Unlike `LineTo`, does not use or modify `PenPos`.

Circle

Declaration
```
public procedure Circle(Center:TRealPoint; R:integer); virtual;
```

Description draws a circle with the `Center` in user space coordinates and radius `R` in screen pixels.

Aim

Declaration
```
public procedure Aim(Center: TRealPoint; R: integer); virtual;
```

Description draws circle with cross. `Center` in user space coordinate and radius `R` in pixels.

FillRect

Declaration `public procedure FillRect(X1,Y1,X2,Y2:Float); overload; public procedure Fillrect(P1,P2:TRealPoint); overload;`

Description Draws filled rectangle in user space coordinates.

FastDraw

Declaration `public procedure FastDraw(APoints:TRealPointVector; Lb, Ub: integer);`

Description Fast optimized drawing of large (>10000) arrays of `TRealPoint`. Only if "X" is sorted in ascending order

DrawSpline

Declaration `public procedure DrawSpline(APoints:TPoints; Lb, Ub: integer);`

Description Draws spline through the points Apoints[Lb]..APoints[Ub]

DrawFunc

Declaration `public procedure DrawFunc(AFunc:TParamFunc; Params:Pointer; LeftX, RightX : Float); virtual;`

Description Draws `TParamFunc (function(X:Float; Params:Pointer):Float` from `LeftX` to `RightX`. If they are outside MinX..MaxX they are cropped.

SetMinX, SetMinY, SetMaxX, SetMaxY

Declaration
```
protected procedure SetMinX(AMinX:Float); virtual;
protected procedure SetMaxX(AMaxX:Float); virtual;
protected procedure SetMinY(AMinY:Float); virtual;
protected procedure SetMaxY(AMaxY:Float); virtual;
```

Description Methods to set MinX, `MaxX`, `MinY`, `MaxY` properties. Change limits of drawn user coordinates and rescale the picture.

SetRightMargin, SetLeftMargin, SetLowerMargin, SetUpperMargin

Declaration
```
protected procedure SetRightMargin(AMargin:integer); virtual;
protected procedure SetLeftMargin(AMargin:integer); virtual;
protected procedure SetLowerMargin(AMargin:integer); virtual;
protected procedure SetUpperMargin(AMargin:integer); virtual;
```

Description These procedures set margins which are not used for drawing of data. These are methods to set corresponding properties.

SetXGridDist, SetYDist

Declaration
```
protected procedure SetXGridDist(AXGridDist:Float); virtual;
protected procedure SetYGridDist(AYGridDist:Float); virtual;
```

Description Methods to set XGridDist and `YGridDist` properties.

SetPenPos

Declaration `protected procedure SetPenPos(APenPos:TRealPoint); virtual;`

Description Method to set PenPos property (Alternatively, use GoToXY procedure).

SetXPos, SetYPos

Declaration
```
protected procedure SetXPos(AXPos:Float); virtual;
protected procedure SetYPos(AYPos:Float); virtual;
```

Description Methods to set XPos and YPos properties.

SetAxisPen, SetGridPen, SetOutputPen

Declaration
```
protected procedure SetAxisPen(APen:TPen); virtual;
protected procedure SetGridPen(APen:TPen); virtual;
protected procedure SetOutputPen(APen:TPen); virtual;
```

Declaration Methods to set AxisPen, GridPen and OutputPen properties, used for drawing the coordinate system and user's output.

3.3 Functions and Procedures

3.3.1 Register

Declaration `procedure Register;`

3.4 Constants

ColorAxis

Declaration `ColorAxis = clBlack;`

Description Axis color, black.

ColorBack

Declaration `ColorBack = clSilver;`

Description Background color, Light Gray.

ColorText

Declaration `ColorText = clRed;`

Description Text color, red.

ColorGridLines

Declaration `ColorGridLines = clWhite;`

Description Grid lines color, white.

ColorOutput

Declaration `ColorOutput = clBlue;`

Description Blue. Default color of user data, put by PutPoint, PutLine, LineTo. May be changed by SetOutputColor

UpperLimitForFixedFormat

Declaration `UpperLimitForFixedFormat = 1E7;`

Description everything outside [LowerLimitForFixedFormat..UpperLimitForFixedFormat] is written in ingeneer notation (e.g.1.0E9)

LowerLimitForFixedFormat

Declaration `LowerLimitForFixedFormat = 1E-4;`

Chapter 4

Unit lmNumericEdits

4.1 Classes, Interfaces, Objects and Records

4.1.1 TFloatEdit Class

Hierarchy

TFloatEdit > TEdit

Declaration

```
TFloatEdit = class(TEdit)
protected
  procedure TextChanged; override;
  procedure SetDecimals(ADecimals: Integer); virtual;
  procedure SetValue(const AValue: Float); virtual;
  procedure SetValueEmpty(const AValue: Boolean); virtual;
  procedure KeyPress(var Key: char); override;
public
  constructor Create(TheOwner: TComponent); override;
  procedure PasteFromClipboard; override;
  function ValueToStr(const AValue: Float): String; virtual;
published
  property DecimalPlaces: Integer read FDecimals write SetDecimals default 2;
  property Value: Float read FValue write SetValue;
  property ValueEmpty: Boolean read FValueEmpty write SetValueEmpty default False;
end;
```

Description

TFloatEdit

Properties class defines an Edit component for float numbers. Usage: Drop the component on a form; set and read `Value` property.

DecimalPlaces `published property DecimalPlaces: Integer default 2;`

Value `published property Value: Float;`

ValueEmpty `published property ValueEmpty: Boolean default False;`

Description Is `true` if `Text` contains invalid or empty string. If a user sets `ValueEmpty := true`, `Text` becomes empty string.

Methods

TextChanged

Declaration `protected procedure TextChanged; override;`

Description Tries to convert `Text` to Float. If successful, assigns result of convertion to `Value` and sets `ValueEmpty` to False. Otherwise, sets `ValueEmpty` to True.

KeyPress

Declaration `protected procedure KeyPress(var Key: char); override;`

Description Filters out all symbols except decimal digits, “+”,“-”,“E”,“.” and “,”. “.” and “,” are automatically converted to valid locale-dependent decimal separator.

Create

Declaration `public constructor Create(TheOwner: TComponent); override;`

ValueToStr

Declaration `public function ValueToStr(const AValue: Float): String; virtual;`

Description Converts `Value` to `String` according to `DecimalPlaces`.

4.2 Functions and Procedures

4.2.1 Register

Declaration `procedure Register;`

Chapter 5

Unit lmnumericinputdialogs

5.1 Functions and Procedures

5.1.1 IntervalQuery

Declaration `function IntervalQuery(ACaption, APrompt1, APrompt2 : string; var AInterval:TInterval):boolean;`

Description input dialog with two float edits. Sets TInterval; one edit is for Low, other for High. Returns True if was closed with OK, false otherwise. If one or both edits do not contain valid values, the dialog cannot be closed with "OK".

5.1.2 FloatInputDialog

Declaration `function FloatInputDialog(const InputCaption, InputPrompt : String; var AValue : Float) : Boolean;`

Description Input dialog for Float input. True if was closed with OK and EditBox contains valid value, false otherwise.