LMComponents: object-oriented extension of LMath library

Viatcheslav Nesterov

September 22, 2024

Contents

1 Unit lmPointsVec 4
- 1.1 Description . . . 4
- 1.2 Classes, Interfaces, Objects and Records . . . 4
  - 1.2.1 ERealPointsException Class . . . 4
    - 1.2.1.1 Hierarchy . . . 4
    - 1.2.1.2 Description . . . 4
  - 1.2.2 TPoints Class . . . 4
    - 1.2.2.1 Hierarchy . . . 4
    - 1.2.2.2 Declaration . . . 4
    - 1.2.2.3 Properties . . . 5
    - 1.2.2.4 Fields . . . 5
    - 1.2.2.5 Methods . . . 6

2 Unit lmFilters 9
- 2.1 Description . . . 9
- 2.2 Types . . . 9
  - 2.2.1 TInputFunc . . . 9
  - 2.2.2 TOutputProc . . . 9
- 2.3 Classes, Interfaces, Objects and Records . . . 10
  - 2.3.1 EFilterException Class . . . 10
    - 2.3.1.1 Hierarchy . . . 10
  - 2.3.2 TDigFilter Class . . . 10
    - 2.3.2.1 Hierarchy . . . 10
    - 2.3.2.2 Declaration . . . 10
    - 2.3.2.3 Description . . . 10
    - 2.3.2.4 Events . . . 10
    - 2.3.2.5 Fields . . . 11
    - 2.3.2.6 Methods . . . 11
  - 2.3.3 TOneFreqFilter Class . . . 12
    - 2.3.3.1 Hierarchy . . . 12
    - 2.3.3.2 Declaration . . . 12
    - 2.3.3.3 Description . . . 12
    - 2.3.3.4 Properties . . . 12
    - 2.3.3.5 Methods . . . 12
  - 2.3.4 TFIRFilter Class . . . 12
    - 2.3.4.1 Hierarchy . . . 12
    - 2.3.4.2 Declaration . . . 13
    - 2.3.4.3 Description . . . 13
    - 2.3.4.4 Properties . . . 13
    - 2.3.4.5 Fields . . . 13
    - 2.3.4.6 Methods . . . 13
  - 2.3.5 TMovAvFilter Class . . . 14
    - 2.3.5.1 Hierarchy . . . 14
    - 2.3.5.2 Declaration . . . 14
    - 2.3.5.3 Description . . . 14
    - 2.3.5.4 Properties . . . 14
    - 2.3.5.5 Methods . . . 14
  - 2.3.6 TGaussFilter Class . . . 15

2.3.6.1 Hierarchy . . . 15
2.3.6.2 Declaration . . . 15
2.3.6.3 Description . . . 15
2.3.6.4 Properties . . . 15
2.3.6.5 Methods . . . 17
2.3.7 TMedianFilter Class . . . 17
2.3.7.1 Hierarchy . . . 17
2.3.7.2 Declaration . . . 17
2.3.7.3 Description . . . 17
2.3.7.4 Methods . . . 18
2.4 Functions and Procedures . . . 18
2.4.1 Register . . . 18

3 Unit lmRecursFilters **19**
3.1 Overview . . . 19
3.2 Classes . . . 19
3.2.1 THighPassFilter Class . . . 19
3.2.1.1 Hierarchy . . . 19
3.2.1.2 Declaration . . . 19
3.2.1.3 Description . . . 19
3.2.1.4 Methods . . . 19
3.2.2 TNarrowBandFilter Class . . . 20
3.2.2.1 Hierarchy . . . 20
3.2.2.2 Declaration . . . 20
3.2.2.3 Description . . . 20
3.2.2.4 Properties . . . 20
3.2.2.5 Methods . . . 20
3.2.3 TNotchFilter Class . . . 21
3.2.3.1 Hierarchy . . . 21
3.2.3.2 Declaration . . . 21
3.2.3.3 Description . . . 22
3.2.4 TBandPassFilter Class . . . 22
3.2.4.1 Hierarchy . . . 22
3.2.4.2 Description . . . 22
3.2.5 TChebyshevFilter Class . . . 22
3.2.5.1 Hierarchy . . . 22
3.2.5.2 Declaration . . . 22
3.2.5.3 Description . . . 22
3.2.6 Functions and Procedures . . . 23
3.2.6.1 Register . . . 23

4 Unit lmcoordsys **24**
4.1 Description . . . 24
4.2 Classes . . . 24
4.2.1 TCoordSys Class . . . 24
4.2.1.1 Hierarchy . . . 24
4.2.1.2 Declaration . . . 24
4.2.1.3 Description . . . 26

4.2.1.4 Properties . . . 26
4.2.1.5 Fields . . . 27
4.2.1.6 Methods . . . 28
4.3 Functions and Procedures . . . 31
4.3.1 Register . . . 31
4.4 Constants . . . 31

5 Unit lmNumericEdits **33**
5.1 Classes, Interfaces, Objects and Records . . . 33
5.1.1 TFloatEdit Class . . . 33
5.1.1.1 Declaration . . . 33
5.2 Functions and Procedures . . . 34
5.2.1 Register . . . 34

6 Unit lmNumericInputDialogs **35**
6.1 Functions and Procedures . . . 35
6.1.1 IntervalQuery . . . 35
6.1.2 FloatInputDialog . . . 35

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 of Float; if the package was compiled with -dDebug setting, range check is done 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` with additional range check. 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 digital filters of a signal. They are implemented as non-visual components:
TDigFilter. Ancestor class for all digital filters.
TOneFreqFilter. Ancestor class for all filters with a single corner frequency.
TFIRFilter. Ancestor class for filters with finite impulse responce filters.
TMovAvFilter. Moving average filter
TGaussFilter. Gaussian filter. The only one, which can be used only for filtering ready data, but not real time.
TMedianFilter. Median filter.

Most of them can be used both for filtering of earlier sampled data and for filtering in real time, during data acquisition. To make the components format-independent, data points are fed into filter with user-defined `OnInput` event (see 2.2.1 for type description and 2.3.2.4 for simplest example of implementation) and returned by another user-defined `OnOutput` event (2.2.2 and 2.3.2.4). You can drop a component onto a form and define `OnInput` and `OnOutput` events. Afterwards, set sampling rate and cut-off frequency by call of `SetupFilter` (2.3.3.5) method. For moving average filter (2.3.5), one has an alternative to define averaging window length instead of defining sampling rate and cut-off frequency. For median filter (2.3.7), this is the only possibility, because for this filter cut-off frequency is not defined. After setting up the filter properties, either call `Filter` (2.3.2.6) method for off-line filtering of existing data, or initiate filtering in real time. To do so, call `InitFiltering` (2.3.2.6) once, and then call `NextPoint` (2.3.2.6) method whenever new datapoint is aquired. Method `Filter` calls `OnInput` event when needs a next input value, and `OnOutput` when it is ready to return a next output value. Similarly, `NextPoint` calls `OnInput` and `OnOutput` when is invoked. This technique with `OnInput` and `OnOutput` events makes the components independent of a format of data which are filtered.

Implemented are gaussian filter (2.3.6), moving average filter (2.3.5), which are probably the best "smoothing" filters for time domain, and median filter (2.3.7) which is ideal to remove short spikes preserving sharp edges. One-pole high-pass filter, notch filter and Chebyshev filter are implemented in `lmRecursFilters` unit (3). All of these filters except Gaussian can be used for both real time and off-line filtering. Gaussian filter requires forward and backward filtering, hence, can be used only off line.

2.2 Types

2.2.1 TInputFunc

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

Description Procedural type used by `OnInput` event of `TDigFilter` (2.3.2) and its inheritants.

2.2.2 TOutputProc

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

Description Procedural type used by `OnOutput` event of `TDigFilter` (2.3.2) and its inheritants.

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;
  Index:integer;
public
 procedure Filter(StartIndex, EndIndex:integer); virtual; abstract;
 procedure InitFiltering; virtual;
 procedure NextPoint; 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.

Both OnInput and OnOutput events are called from Filter (2.3.2.6) 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

Index

Declaration `protected Index:Integer;`

Description Integer field for indexing of the data points. Descendants use it in `NextPoint` (2.3.2.6) method.

Methods

Filter

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

Description Receives input signal values calling `OnInput` (2.3.2.4), makes actual filtering and outputs result calling `OnOutput` (2.3.2.4).

InitFiltering

Declaration `procedure InitFiltering; virtual;`

Description Initiates process of online filtering; sets `Index` to zero.

NextPoint

Declaration `procedure NextPoint; virtual; abstract;`

Description Descendants use it for real-time filtering. Calls `OnInput` (2.3.2.4) and passes `Index` (2.3.2.5) value as parameter to get input value, after that calculates filtered output value and calls `onOutput` (2.3.2.4), to which passes `Index` and calculated value as parameters and increments `Index`.

How to use. Define `OnInput` and `OnOutput` events such that `OnInput` returns a newly acquired data value and `OnOutput` returns in `Val` the calculated filtered value to your final (filtered) dataset. Before the beginning of data aquisition, call `InitFiltering`. Value of `Index` field is initially set to zero. During the aquisition cycle, simply call `NextPoint` whenever new data value is acquired.

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, ACornerFreq : Float); virtual;
published
  property SamplingRate : Float;
  property CornerFreq     : Float;
end;
```

Description

Descendant of `TDigFilter` 2.3.2 which defines filters with one cut-off frequency (that is, not pass-band or stop-band with distinct cut-offs for low and high frequencies). Introduces `SamplingRate` and `CornerFreq` properties.

Properties

SamplingRate `published property SamplingRate;`

Description Sampling rate, usually Hz; must have same units as `CornerFreq`.

CornerFreq `published property CornerFreq;`

Description Cut-off (or corner) frequency, usually Hz. Must have same units as `SamplingRate` and be less then $SamplingRate/2$.

Methods

Create

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

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

SetupFilter

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

Description Procedure which sets `SamplingRate` and `CornerFreq`. 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;
  PrevPtr:integer;
  WindowData:TVector;
  procedure SetWinLength(L:integer); virtual;
  procedure InitFiltering; override;
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 PrevPtr:integer;`

PrevPtr `protected FWinLength: integer;`

Description Used internally by `InitFiltering` and `NextPoint`

WindowData `protected WindowData:TVector;`

Description Used internally by `InitFiltering` and `NextPoint`

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, allocates `WindowData`.

2.3.5 TMovAvFilter Class

Hierarchy

TMovAvFilter > TFIRFilter > TOneFreqFilter > TDigFilter > TComponent

Declaration

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

Description

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

Properties

WinLength

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

Description Contains length of the data window for averaging.

Methods

SetWinLength

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

Description Sets WinLength, calculates corresponding CornerFreq.

Filter

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

Description Filters a dataset, sequentially calling OnInput and OnOutput beginning from StartIndex to EndIndex.

SetupFilter

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

Description Sets `SamplingRate` and `CornerFreq`, calculates and sets corresponding `WinLength`. Importantly, if WinLength is set directly, corresponding new value for `CornerFreq` is set.

NextPoint

Declaration procedure NextPoint; override;

Description During real time filtering calculates next data point. See 2.3.2.6.

2.3.6 TGaussFilter Class

Hierarchy

TGaussFilter > `TOneFreqFilter` > `TDigFilter` > TComponent

Declaration

```
TGaussFilter = class(TOneFreqFilter)
public
  constructor Create(AOwner:TComponent); override;
  procedure SetupFilter(ASamplingRate, ACornerFreq: Float); override;
  procedure Filter(StartIndex, EndIndex:integer); override;
published
  property OnInputBackward : TInputFunc;
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

Properties

OnInputBackward

Declaration property OnInputBackward : TInputFunc;

Description `TGaussFilter` filters data in two steps: forward and backward filtering. Backward filtering uses data which were already processed by forward filtering. Hence, if you place filtered data into a new variable, you must define `OnInputBackward` event which should read data from the output variable. If, in contrast, you filter `in situ` such that input data are transformed rather than new data created, defining `OnInputBackward` event is not necessary.

Example 1. Different structures are used for input and output. `OnInputBackward` event is needed.

```
var
  MyGaussFilter : TGaussFilter;
  MyInputVector, MyOutputVector : TVector;
.........
function SomeObject.InputFunction(I:Integer):float;
begin
  Result := MyInputVector[I];
end;

{Note that output goes to a different place than input...}
procedure SomeObject.OutProcedure(Val: Float; I:Integer);
begin
  MyOutputVector[I] := Val;
end;

{...Hence, InputBackward event is needed to read values
which were processed by forward filtering procedure}
function SomeObject.InputBackward(I:Integer):float;
begin
  Result := MyOutputVector[I];
end;
.......
begin
  MyGaussfilter := TGaussFilter.Create(nil);
  MyGaussFilter.OnInput := @MyObject.InputFunction;
  MyGaussFilter.OnInputBackward := @SomeObject.InputBackward;
  MyGaussFilter.OnOutput := @SomeObject.OutProcedure;
  MyGaussFilter.Filter(0,High(MyInputVector));
end;
```

Example 2. Filtering in place. No need for separate `InputBackward` procedure.

```
var
  MyGaussFilter : TGaussFilter;
  MyDataVector  : TVector;
.........
function SomeObject.InputFunction(I:Integer):float;
begin
  Result := MyDataVector[I];
end;

{Note that output goes to the same place where from input was read.}
procedure SomeObject.OutProcedure(Val: Float; I:Integer);
begin
  MyDataVector[I] := Val;
end;
.......
begin
```

```
  MyGaussfilter := TGaussFilter.Create(nil);
  MyGaussFilter.OnInput := @MyObject.InputFunction;
  {Separate OnInputBackward is not needed and is not assigned.}
  MyGaussFilter.OnOutput := @SomeObject.OutProcedure;
  MyGaussFilter.Filter(0,High(MyInputVector));
end;
```

Methods

Create

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

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

SetupFilter

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

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

Filter

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

Description Filters a dataset, sequentially calling `OnInput` and `OnOutput` beginning from `StartIndex` to `EndIndex`.

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
  procedure InitFiltering; override;
  procedure NextPoint; override;
  constructor Create(AOwner:TComponent); override;
  procedure filter(StartIndex, EndIndex:integer); override;
end;
```

Description

Implementation of Median Filter. Use `Filter` method to filter a ready data set, use `InitFiltering` and `NextPoint` methods for real time signal filtering.

Methods

FindMedian

Declaration `protected function FindMedian:Float;`

Description Finds Median of the filtering window. Is called from `Filter`, a user does not need to call it directly.

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.

InitFiltering

Declaration procedure InitFiltering; override;

Description Initiates real time filtering. Call this procedure before start of acquiring and filtering of real time signal.

NextPoint

Declaration procedure NextPoint; override;

Description During real time filtering calculates next data point. See 2.3.2.6.

Filter

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

Description See 2.3.2.6.

2.4 Functions and Procedures

2.4.1 Register

Declaration `procedure Register;`

Chapter 3

Unit lmRecursFilters

3.1 Overview

Several components implementing infinite impulse response filters are defined in this unit:
THighPassFilter
TNarrowBandFilter
TNotchFilter
TBandPassFilter
TChebyshevFilter

3.2 Classes

3.2.1 THighPassFilter Class

Hierarchy

THighPassFilter > TOneFreqFilter > TDigFilter > TComponent

Declaration

```
THighPassFilter = class(TOneFreqFilter)
public
  procedure SetupFilter(ASamplingRate, ACornerFreq : Float); override;
  procedure Filter(StartIndex, EndIndex:integer); override;
  procedure InitFiltering; override;
  procedure NextPoint; override;
end;
```

Description

THighPassFilter component implements a single-pole high-pass filter. Call SetupFilter method to set sampling rate and corner (cut-off) frequency. Procedure Filter can be used to filter an existing data set.

Methods

SetupFilter

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

Description Sets SamplingRate and CornerFreq and calculates filter coefficients.

Filter

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

Description Filters a dataset, sequentially calling OnInput and OnOutput beginning from StartIndex to EndIndex.

InitFiltering

Declaration `public procedure InitFiltering; override;`

Description Initiates real time filtering. Call this procedure before start of acquiring and filtering of real time signal. CornerFreq and SamplingRate, calculates corresponding filter coefficients.

NextPoint

Declaration `public procedure NextPoint; override;`

Description During real time filtering calculates next data point. See 2.3.2.6.

3.2.2 TNarrowBandFilter Class

Hierarchy

TNarrowBandFilter > `TOneFreqFilter` > `TDigFilter` > TComponent

Declaration

```
 TNarrowBandFilter = class(TOneFreqFilter)
 protected
  Old: array[-2..0] of Float;
  New: array[-2..0] of Float;
  NewVal: float;
  procedure SetBandWidth(ABandWidth:float); virtual;
  procedure SetSamplingrate(AValue: Float); override;
  procedure SetCornerFreq(ACornerFreq:float); override;
public
  procedure SetupFilter(ASamplingRate, ACornerFreq : float); override;
  procedure SetupNarrowBandFilter(ASamplingRate, ABandFreq, ABandWidth : float); virt
  procedure Filter(StartIndex: integer; EndIndex:integer); override;
  procedure InitFiltering; override;
  procedure NextPoint; override;
published
  property BandWidth : float;
end;
```

Description

`TNarrowBandFilter` is an ancestor class for one-pole band-reject, called also notch, filter (`TNotchFilter`, 3.2.3) and one-pole band-pass filter (`TBandpassFilter`, 3.2.4). These filters have two properties: central frequency which is completely suppressed (notch filter) or completely passed (band-pass) and band width, which describes how steep is the frequency response of the filter. Central frequency is defined by CornerFreq. For band width, new property is introduced: `BandWidth`.

Properties

BandWidth `published property BandWidth : float;`

Description Defines the width of rejected or passed band between points with energy 0.5, which means amplitude 0.707 of original. Smaller is the value, steeper is the frequency response, but at the expense of larger ripple in time domain. Default value is 5.0.

Methods

SetBandWidth

Declaration `protected procedure SetBandWidth(ABandWidth:float); virtual;`

Description Setter of `BandWidth`. Sets new value and calls `SetupNarrowBandFilter` with new values.

SetSamplingrate

Declaration protected procedure SetSamplingrate(AValue: Float); override;

Description Setter of `SamplingRate`. Sets new value and calls `SetupNarrowBandFilter` with new values.

SetCornerFreq

Declaration `protected procedure SetCornerFreq(ACornerFreq:float); override;`

Description Setter of `CornerFreq`. Sets new value and calls `SetupNarrowBandFilter` with new values.

SetupFilter

Declaration
```
public procedure SetupFilter(ASamplingRate, ABandFreq, ABandWidth
: float); override;
```

Description Calls `SetupNarrowBandFilter` with new values for `Samplingrate`, and `CornerFreq` leaving `BandWidth` unchanged (default value is 5.0) and calculates filter coefficients accordingly. It is more efficient to call `SetupNarrowBandFilter` directly rather than set values of the properties or call `SetupFilter`. However, `SetupFilter` allows to use descendants of `TNarrowBandFilter` in more generic code where variable of `TDigFilter` class is used.

Filter

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

Description See 2.3.2.6.

InitFiltering

Declaration `public procedure InitFiltering; override;`

Description See 2.3.2.6.

NextPoint

Declaration `public procedure NextPoint; override;`

Description See 2.3.2.6.

3.2.3 TNotchFilter Class

Hierarchy

TNotchFilter > TNarrowBandFilter > `TOneFreqFilter` > `TDigFilter` > TComponent

Declaration

```
TNotchFilter = class(TNarrowBandFilter)
  public
  procedure SetupFilter(ASamplingRate, ABandFreq, ABandWidth : float); override;
end;
```

Description

Implements notch (band-reject) filter, which may be used, for example, to eliminate a 50 Hz hum from a signal. See 3.2.2 for methods and properties.

3.2.4 TBandPassFilter Class

Hierarchy

TBandPassFilter > TNarrowBandFilter > `TOneFreqFilter` > `TDigFilter` > TComponent

Description

Implements band-pass filter, which allows to isolate a narrow band around given frequency. See 3.2.2 for methods and properties.

3.2.5 TChebyshevFilter Class

Hierarchy

TChebyshevFilter > `TOneFreqFilter` > `TDigFilter` > TComponent

Declaration

```
TChebyshevFilter = class(TOneFreqFilter)
  procedure Filter(StartIndex, EndIndex:integer); override;
  procedure SetupChebyshevFilter(ASamplingRate: Float; ACornerFreq: Float;
   ANPoles: integer; APRipple: float; AHighPass:boolean);
  procedure SetupFilter(ASamplingRate, ACornerFreq : float);
  procedure InitFiltering;
  procedure NextPoint;
end;
```

Description

`TChebyshevFilter` implements a Chebyshev filter, which may be used both as a high- or low-pass filter. Main characteristic of this filter is a steep frequency response at the expense of a considerable signal distortion in time domain. Another noticeable feature of Chebyshev filters is a ripple in frequency domain; here is implemented type 1 filter where ripple is allowed only in pass-band. Larger ripple comes with a steeper frequency response. Setting ripple to zero converts Chebyshev filter to the Butterworth maximally flat filter. Drawback is a relatively flat frequency response. In most cases, ripple 0.5 % is a good choice, when frequency response is already almost maximally steep, while ripple is nearly invisible. Setting ripple to 0 converts Chebyshev filter to "maximally flat" Butterworth filter. More poles make the frequency response steeper, but calculations are more complicated. With too many poles filter performance can deteriorate due to rounding errors, hence, in this implementation maximal number of poles is limited to 10. Usually, 6 poles is a good choice. Note, that `SetupFilter` is not used for `TChebyshevFilter`. Rather, use `TChebyshevFilter.SetupChebyshevFilter` method.

SetupChebyshevFilter

Declaration
```
public procedure SetupChebyshevFilter(ASamplingRate: Float;
ACornerFreq: Float; ANPoles: integer; APRipple: float;
AHighPass:boolean);
```

Description Similar to `SetupFilter`, defines `SamplingRate` and `CornerFreq`. Additionally, with `APoles` it defines number of poles; APRipple defines amount of ripple in the pass band; finally, AHighPass defines if a high pass (if true) or low pass (if false) filter is requested. `APoles` must be positive even below or equal 10.

Filter

Declaration procedure Filter(StartIndex, EndIndex:integer);

Description Filters a dataset, sequentially calling `OnInput` and `OnOutput` beginning from `StartIndex` to `EndIndex`.

SetupFilter

Declaration procedure SetupFilter(ASamplingRate, ACornerFreq : float);

Description Calls `SetupChebyshevFilter` with given `ASamplingRate` and `ACornerFreq` and leaves other parameters unchanged. Default values after construction are 6 poles, ripple 0.5% and low pass filter. It is more efficient to call `SetupChebyshevFilter` directly rather than use `SetupFilter`. However, this latter methid allows to use Chebyshev filter in a generic code where variable of `TDigFilter` type is used.

InitFiltering

Declaration procedure InitFiltering;

Description Initiates real time filtering. Call this procedure before start of acquiring and filtering of real time signal.

NextPoint

Declaration procedure NextPoint;

Description During real time filtering calculates next data point. See 2.3.2.6.

3.2.6 Functions and Procedures

Register

Declaration `procedure Register;`

Chapter 4

Unit lmcoordsys

4.1 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 lines and user data, as well as numeric format for axis 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.

4.2 Classes

4.2.1 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); virtual;
published
  property XAxisLabel:string read FXAxisLabel write FXAxisLabel;
  property YAxisLabel:string read FYAxisLabel write FYAxisLabel;
  property MinX:Float;
  property MinY:Float;
  property MaxX:Float;
  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;
  property OutputPen:TPen;
  property GridPen:TPen;
  property LeftMargin:integer default 0;
  property RightMargin:integer default 0;
  property LowerMargin:integer default 0;
  property UpperMargin:integer default 0;
  property XGridDist:Float;
  property YGridDist:Float;
  property XGridNumbersPrecision: integer default 5;
  property XGridNumbersDecimals: integer default 2;
  property YGridNumbersPrecision: integer default 9;
  property YGridNumbersDecimals: integer default 4;
  property Canvas;
  property OnDrawData:TNotifyEvent;
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, pixels

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 never must 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, SetYGridDist

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.

4.3 Functions and Procedures

4.3.1 Register

Declaration `procedure Register;`

4.4 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 5

Unit lmNumericEdits

5.1 Classes, Interfaces, Objects and Records

5.1.1 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`.

5.2 Functions and Procedures

5.2.1 Register

Declaration `procedure Register;`

Chapter 6

Unit lmNumericInputDialogs

6.1 Functions and Procedures

6.1.1 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".

6.1.2 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.