Matrix unit

!!! ENTER DESCRIPTION HERE

Components

Types

glindex
glmbynp
glmvector
glnpbym
glnpbynp
glnpvarray
glnpvector

Routines

Constants

glnpvector type

Unit
Matrix

Declaration
```
glnpvector = array[0..cMAXVARS] of float;
```

Description
Used internally by matrix routines. This is also the type for the first parameter in the Predict function. The user will need to declare a variable of type glnpvector in order to use the method.

Maxdata unit

This unit is provided to allow the user to modify the floating point type and largest dataset which can be loaded into the regression buffer.

This is included for 16-bit and otherwise memory-limited systems.

Types
float

Constants
cMAXDATA

float type

Unit
Maxdata

Declaration
```
float = double;
```

Description
The float type is used throughout the MLRegress component set. The user can define the type to represent floating point arithmetic in all calculations. This allows the user to modify output precision, and the size of the dataset which can be loaded, based on memory availability.

The default value is double, however the user can define the float type to:

float = extended;
float = double;
float = real;
float = single;

The ability to redefine the float type is particularly useful in the 16-bit version where limited resources may be available.

Mlrquery unit

The Mlrquery unit contains the complete definition of the TMLRQuery component.

Components
TMLRQuery

Types
See Mlrtypes unit
See MaxData unit

Routines
See Mlrtypes unit

Constants
See MaxData unit

TMLRQuery Component

Properties Methods

Unit

MIrquery

Description

The TMLRQuery component is descendent from TQuery and includes several additional properties and methods which enable it to perform multiple linear and curvilinear regression on data stored in the tables accessed by the query.

Use the TMLRQuery component with SQL to define the dataset on which to perform the regression. In addition to the Properties and Methods linked above, TMLRQuery has **all properties**, **methods**, and **events** of the TQuery component.

Differences in the use of TMLRTable and TMLRQuery primarily include the fact that data from multiple tables can be used to define a regression. In addition, it can be easier to filter the records used in the dataset based on non-indexed tables.

Example (assumes defined MLRQuery1: TMLRQuery;)

```
begin
  with MLRQuery1 do
    begin
        Close;
        SQL.Clear;
        SQL.Add('Select * From "PATIENTS.DB"');
        SQL.Add('Where Diagnosis = "Glaucoma"');
        Open
    end
end;
```

The user can now run regression on various fields of this dataset, knowing that only patients with Glaucoma will be included in the regression.

Properties

Run-time only

Key properties

About	DegreesOfFreedom	SSR
b	Description	SST
ControlVar01	e	syx
ControlVar02	F	x
ControlVar03	n	xBar
ControlVar04	nControls	xMax
ControlVar05	Regressed	xMin
ControlVar06	ReportMessage	xStdErr
ControlVar07	ResponseVarY	y
ControlVar08	RSquared	yBar
ControlVar09	ShowLoadProgress	yHat
ControlVar10	ShowWarnings	yMax
corr	SSE	yMin
covar	sSquared	yStdErr
cVariable		

Methods

Key methods

Execute	Output	ReLoad
LoadValues	Predict	ShowPlots

About property

Applies to

TMLRQuery, TMLRTable

Declaration

```
property About: TAbout;
```

Description

The About property contains version and registration information about the associated component. The dialog box displayed when clicking the TAbout ellipses displays current registration information, or fields for entering information.

b property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property b[Index: word]: double;
```

Description
The property b accesses the coefficient vector for the resulting regression. b[0] is the constant in the estimate of the regression model equation

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n + \varepsilon$$

b[1],...,b[n] represent the remaining regression coefficient estimates.

ControlVar01 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar01: TIndepVar;
```

Description
This is the first of the independent variable properties for the given MLRQuery or MLRTable component. For methods and usage, see the TIndepVar class topic in this help file.

All ControlVarxx properties are identical in capabilities.

ControlVar02 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar02: TIndepVar;
```

Description
See ControlVar01.

ControlVar03 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar03: TIndepVar;
```

Description
See ControlVar01.

ControlVar04 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar04: TIndepVar;
```

Description
See ControlVar01.

ControlVar05 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar05: TIndepVar;
```

Description
See ControlVar01.

ControlVar06 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar06: TIndepVar;
```

Description
See ControlVar01.

ControlVar07 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar07: TIndepVar;
```

Description
See ControlVar01.

ControlVar08 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar08: TIndepVar;
```

Description
See ControlVar01.

ControlVar09 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar09: TIndepVar;
```

Description
See ControlVar01.

ControlVar10 property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ControlVar10: TIndepVar;
```

Description
See ControlVar01.

corr property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property corr[Index1,Index2: word]: double;
```

Description
This property allows the user to access the correlation matrix of b for the completed regression. The correlation matrix is (n+1)x(n+1), with *corr* having indexes ranging from 0,...,n.

covar property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property covar[Index1,Index2: word]: double;
```

Description
This property allows the user to access the variance-covariance matrix of b for the completed regression. The variance-covariance matrix is (n+1)x(n+1), with the property *covar* having indexes ranging from 0,...,n.

Example Usage (assuming defined MLRTable1:TMLRTable;)
```
var i,j: word;   covarStr:string;
begin
  covarStr := 'Variance-Covariance Matrix';
  with MLRTable1 do
    for i := 0 to nControls do
      begin
        covarStr := covarStr + #10#13;
        for j := 0 to nControls do
          covarStr := covarStr +'      '+ floatToStr(covar[i,j])
      end ;
  MessageDlg(covarStr,mtInformation,[mbOK],0)
end;
```

cVariable property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property cVariable[Index: word]: TIndepVar;
```

Description
This property allows access to the independent variables defined in the regression model as a contiguous group. This is useful since any of the MLRTable or MLRQuery ControlVarxx properties might be inactive. The regression model might have, for a number of modeling reasons, a situation where, e.g.,

ControlVar01.Field1 = 'Age'
ControlVar01.Transformation = 'None'
and
ControlVar10.Field1 = 'Experience'
ControlVar10.Transformation = 'None'

and where ControlVar02.Transformation = ... = ControlVar09.Transformation = 'Inactive'.

This is a *valid* model with two independent variables. In this case, nControls would be equal to 2 and ***all*** properties of ControlVar01 would be accessible through cVariable[1] and ***all*** properties of ControlVar10 would be accessible through cVariable[2]. The inactive properties ControlVar02 through ControlVar09 would not be accessible through cVariable. So for the situation described above, we would have

cVariable[1].OutputTerm = 'Age'
cVariable[2].OutputTerm = 'Experience'

This property is used primarily for reporting purposes. Note that the user could also access the ControlVarxx properties directly to obtain output; however, (s)he would first have to examine each of the available control variables to determine which ones are active in order to view valid output.

NOTE:
cVariable ***cannot*** be used in the modeling process. It is a read-only property.

DegreesOfFreedom property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property DegreesOfFreedom: word;
```

Description
This is the degrees of freedom for the *Residuals.*

DegreesOfFreedom = n - (m + 1),

where
n = number of observations in the regression model
m = number of independent variables

Description property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property Description: string;
```

Description
This is a design-time and run-time property where the user can define the nature of the regression problem. This description will be shown on the default Output Report.

e property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property e[Index: word]:;
```

Description
The e property accesses the individual residual values resulting from the regression. The residuals can be plotted against any relevant measure, e.g.,

input order
any of the independent variables
dependent variable
fitted value of the dependent variable

The user can use any plotting or charting software to graph the residuals, or the included Plot function can be used.

F property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property F: double;
```

Description
This property is read-only, and returns the value of the F* statistic with (m,n) degrees of freedom,

where

m = number of independent variables (regression DOF)
n = number of observations - m - 1 (residual DOF)

n property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property n: word;
```

Description
This is a read property which indicates the number of records *successfully* read into the regression buffer. The value of n will differ from the value of the RecordCount property if there are any records with null field values in a regression-defined datafield, or if any record violates a transformation.

nControls property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property nControls: word;
```

Description
This is a read property which returns the number of active control (independent) variables in a regression model.

Regressed property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property Regressed: Boolean;
```

Description
The Regressed property simply returns True if the Execute method has been run. Otherwise, its value is False.

ReportMessage property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ReportMessage: string;
```

Description
The ReportMessage appears on the default Output Report form provided by MLRTable and MLRQuery. The user can enter his/her company, department, project, or product name here.

ResponseVarY property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ResponseVarY: TDependentVar;
```

Description
The ResponseVarY property contains the name and transformation, if any, on the datafield in the dataset defined as the dependent variable.

RSquared property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property RSquared: double;
```

Description
This is the multiple correlation coefficient squared, calculated from the regression.

ShowLoadProgress property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ShowLoadProgress: boolean;
```

Description
Set the ShowLoadProgress property to True to enable the dialog and gauge which shows the progress of data being loaded into the regression buffer. This is useful during loading of large datasets.

ShowWarnings property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property ShowWarnings: boolean;
```

Description
The setting of this property determines whether or not Warning and Informational dialogs will be shown when there is an anomaly with the loading or execution of the regression.

An example of an informational message would be a warning of dataset larger than the maximum buffer size, or zero degrees of freedom for the regression. Warning messages are automatically disabled and reenabled during the use of Optigress(tm) optimal regression modeling unit.

Fatal Error messages are NOT disabled by the setting of ShowWarnings.

SSE property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property SSE: double;
```

Description
The property SSE is a read property to access the sum of squares for the residuals.

In multiple regression,

SSE = **(y - xb)'(y - xb)**

where

y = vector of response variable values,
x = matrix of control variable values,
b = vector of regression coefficients,

and

(y-xb)' is the transpose of **(y-xb)**

sSquared property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property sSquared: double;
```

Description
The property sSquared, based on the regression, is the best estimate of the variance σ^2.

sSquared = SSE/(n - p)

where
n = number of observations
p = number of parameters in the model (number of independent variables + 1)

SSR property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property SSR: double;
```

Description
The property SSR accesses the sum-of-squares for the regression. Mathematically, in vector terms,

SSR = **b'x'y** - n(yBar)^2

where

y = vector of response variable observations,
x = matrix of control variable observations,
x' = transpose of x
b = vector of regression coefficients, **b'** = transpose of **b**
n = number of observations,
yBar = mean value of the response variable observations

Informal note: understanding the mathematical details of the derivation of SSR is not necessary for successful and effective use of the MLR components.

SST property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property SST: double;
```

Description
The property SST is the total the sum-of-squares. Of course,

SST = SSE + SSR

Mathematically

SST = **y'y** - n(yBar)^2

where

y = vector of response variable observations,
n = number of observations,
yBar = mean value of the response variable observations

syx property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property syx: double;
```

Description
The property syx gives the conditional value of the sample std. deviation of y given the associated values of x.

syx = *sqrt*(sSquared)
= *sqrt(*SSE/(n - p))

x property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property x[Index1,Index2: word]: double;
```

Description
Property x gives access to the x matrix of independent variable values. The value of *Index1* ranges from 1 to n, and *Index2* ranges from 1 to c,

where

n = number of observations
c = number of independent variables in the model

xBar property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property xBar[Index: word]: double;
```

Description
The property xBar provides the sample mean value of the independent variables in the model referenced by Index. Index ranges from 1 to c,

where

c = number of independent variables in the model.

Example
```
begin
   Edit1.Text := floatToStr(MLRTable1.xBar[2])
end;
```

{The above code would show the mean value of control variable 2 of MLRTable1}

xMax property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property xMax[Index: word]: double;
```

Description
The property xMax accesses the maximum value of each of the the independent variables in the model referenced by *Index*. Index ranges from 1 to c,

where

c = number of independent variables in the model.

xMin property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property xMin[Index: word]: double;
```

Description
The property xMin accesses the minimum value of each of the the independent variables in the model referenced by *Index*. Index ranges from 1 to c,

where

c = number of independent variables in the model.

xStdErr property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property xStdErr[Index: word]: double;
```

Description
The property xStdErr, for each control variable in the model, returns the computed standard error based on the values and number of observations.

for each j = 1,..., c (the number of control variables)

$$xStdErr[j] = [1/(n - 1)]Sqrt(Sum((x[i,j] - xBar[j])^2))$$

where the above sum is from i = 1,...,n (the number of observations).

y property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property y[Index: word]: double;
```

Description
The value of property y is the ith value of the response variable for the regression when y[i] is accessed. *Index* ranges from 1,...,n

where
n = number of observations

yBar property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property yBar: double;
```

Description
The property yBar is the sample mean of all observations of the response variable y.

yHat property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property yHat[Index: word]: double;
```

Description
The ith value of property yHat is the fitted (predicted) value of y, based upon the ith value(s) of the control variables for the regression.

There are 1,...,n such fitted values when n is the number of observations.

yMax property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property yMax: double;
```

Description
The property yMax accesses the maximum value among the response variable observations.

yMin property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property yMin: double;
```

Description
The yMin property provides the minimum value of all observations of response variable y.

yStdErr property

Applies to
TMLRQuery, TMLRTable

Declaration
```
property yStdErr: double;
```

Description
The property yStdErr, for the response variable in the model, returns the computed standard error based on the values and number of observations.

yStdErr = [1/(n - 1)]*Sqrt*(*Sum*((y[i] - yBar)^2))

where the above sum is from i = 1,...,n (the number of observations).

Execute method

Applies To
TMLRQuery, TMLRTable

Declaration
```
procedure Execute;
```

Description
The Execute procedure is used to run the regression after successful load of the dataset. The Execute method should only be implemented after the LoadValues method has returned True. If LoadValues has returned false, floating point exceptions are almost certain to occur.

Example (assumes defined MLRQuery1: TMLRQuery;)
```
begin
  with MLRQuery1 do
      if LoadValues then
          Execute
end;
```

LoadValues method

Applies To
TMLRQuery, TMLRTable

Declaration
```
function LoadValues: boolean;
```

Description
The LoadValues method will load all data from the defined dataset, using the datafields and transformations defined in the regression model. A regression ***cannot*** be attempted without first calling the LoadValues method. A successful load will exclude any records with null field values if that field is defined in the regression and will also exclude any records that would violate the defined transformations.

The LoadValues method will check for obvious data patterns that would result in a singular regression matrix, or insufficient degrees of freedom.

Returns True if no problems with the dataset are found during loading, False if there are problems. Problem datasets also provide Error message dialogs.

Output method

Applies To
TMLRQuery, TMLRTable

Declaration
```
procedure Output;
```

Description
The Output procedure opens the default Report form showing the output of the successful regression.

Predict method

Applies To
TMLRQuery, TMLRTable

Declaration
```
function Predict(var x: glnpvector; np: integer): double;
```

Description
Use the Predict function to view the projected value of an input vector processed through the regression model.

NOTE:
The input values to the vector x above must consider any transformation of the input datafields in the original model. For example, if the model transformation uses Population/1000, the value for the corresponding element of x should be scaled ***before*** input. Thus, a prediction based on data for New York City should have population value input as 8000 as opposed to 8,000,000.

See Also
Process Method

ReLoad method

Applies To
TMLRQuery, TMLRTable

Declaration
```
function ReLoad: boolean;
```

Description
Calling LoadValues has no effect if the regression buffer has already been loaded from the dataset. The function ReLoad actually *forces* the MLRQuery or MLRTable to perform a fresh load of the data into the regression buffer. This is obviously useful if, for example, new data has been added to the underlying table since the last regression, or the model has been changed programatically, and an updated regression is desired.

The ReLoad function is used strategically by the Optigress(tm), optimal regression modeling unit (available separately) which performs exhaustive combinatorial search to find the best regression model.

ShowPlots method

Applies To
TMLRQuery, TMLRTable

Declaration
```
procedure ShowPlots;
```

Description
This procedure opens the plot form, allowing the user to view scatter plots of response variable vs. any control variable, residuals vs. response variable, residuals vs. any control variable, residuals vs. input order, or the cumulative distribution function plot.

MIrtable unit

The MIrtable contains the complete definition of the TMLRTable component.

Components
TMLRTable

Types
See MIrtypes unit
See MaxData unit

Routines
See MIrtypes unit

Constants
See MaxData unit

TMLRTable Component

Properties Methods

Unit

MlrtableDescription

The TMLRTable component is descendent from TTable and includes several additional properties and methods which enable it to perform multiple linear and curvilinear regression on data stored in the tables to which it is attached.

In addition to the Properties and Methods linked above, TMLRTable has **all properties**, **methods**, and **events** of the TTable component.

Properties

Run-time only
Key properties

About
b
ControlVar01
ControlVar02
ControlVar03
ControlVar04
ControlVar05
ControlVar06
ControlVar07
ControlVar08
ControlVar09
ControlVar10
corr
covar
cVariable
DegreesOfFreedom
Description
e
F
n
nControls
Regressed
ReportMessage
ResponseVarY
RSquared
ShowLoadProgress
ShowWarnings
SSE
sSquared
SSR
SST
syx
x
xBar
xMax
xMin
xStdErr
y
yBar
yHat
yMax
yMin
yStdErr

Methods

- Key methods

Execute	Output	ReLoad
LoadValues	Predict	ShowPlots

MlrtypesMlrtypes unit

Welcome to the MLRegress series components for Multivariate Regression Analysis.

The Mlrtypes unit is used as the help Contents and introduction to the regression components help file, since Mlrtypes contains all of the supporting class and type declarations for the properties used by the TMLRTable and TMLRQuery components.

Classes
TAbout
TDependentVar
TIndepVar

Types
TDatasetPtr

Routines
NumericField

Constants
No user interactive constants in Mlrtypes.

TAbout Class

Methods

Unit

Mlrtypes

Description

The TAbout class defines the parameters for the About property of the MLRQuery and MLRTable components. The declaration is:

```
TAbout = class(TStringList);
```

The user name and registration code properties are stored as elements of the string list. The elements are set, modified, and accessed through the following (public) methods:

```
procedure SetName(Value: string);
procedure SetCode(Value: string);

function Name: string;
function Code: string;
```

Methods

- Key methods

  Code ▸ SetCode ▸ SetName
  Name

Code method

Applies To
TAbout

Declaration
```
function Code: string;
```

Description
The Code method returns the current value of the registration code for the current instance of the component with the corresponding About property.

Used internally by the MLRTable and MLRQuery components. Is NULL if no code has been assigned. If no valid code has been assigned, the components are IDE Only, trial versions.

Name method

Applies To
TAbout

Declaration
```
function Name: string;
```

Description
Name returns the registered user's name.

SetCode method

Applies To
TAbout

Declaration
```
procedure SetCode(Value: string);
```

Description
Use SetCode to enter the registration code **(exactly as)** supplied by Applied Analytic Systems when the component set is registered. Without a valid registration code, the component can be used only in a trial (Delphi IDE) mode.

This procedure is needed only when components are created and used programmatically. In design mode, the registration code can be entered on the *About* form.

SetName method

Applies To
TAbout

Declaration
```
procedure SetName(Value: string);
```

Description
Use SetName to enter the user name (exactly as) supplied by Applied Analytic Systems when the component set is registered. Without a valid user name/registration code combination, the component can be used only in a trial (Delphi IDE) mode.

This procedure is needed only when components are created and used programmatically. In design mode, the user name can be entered on the *About* form.

TDependentVar Class

Methods

Unit

Mlrtypes

Description

The TDependentVar class defines the capabilities and parameters for the response variable property of the MLRQuery and MLRTable components. The declaration is:

```
TDependentVar = class(TStringList);
```

All properties are stored as elements of the string list. The elements are set, modified, and accessed through the following (public) methods:

Modeling Methods:

```
procedure SetTransformation(const parValue: string);
procedure SetField(const parValue: string);
procedure SetScaleFactor(const parValue: string);

function Process(Dataset: TDatasetPtr): double;
```

Output Methods:

```
function Transformation: string;
function FieldName: string;
function ScaleFactor: string;
function OutputTerm: string;
```

Typically, the user will have need for the Output Methods for regression reporting. The modeling methods need to be used only when creating and modeling components programmatically. Otherwise, design-time properties are set in the property editors (highly recommended).

Methods

Key methods

- FieldName
- OutputTerm
- Process
- ScaleFactor
- SetField
- SetParameters
- SetScaleFactor
- SetTransformation
- Transformation

FieldName method

Applies To
TDependentVar

Declaration
```
function FieldName: string;
```

Description
The FieldName method accesses the *datafield* value defined in a ResponseVarY transformation. This the actual name of the field in the underlying table. This value will always be valid for any defined ResponseVar.

OutputTerm method

Applies To
TDependentVar, TIndepVar

Declaration
```
function OutputTerm: string;
```

Description
The OutputTerm function returns the appropriate expression for the given independent or dependent variable, based upon the definition of the transformation used to define the variable and the name of the datasource datafield name.

Example OutputTerms
Population/1000
ln(Pressure)
Price*Quantity

Example Usage (assumes var X, Y, b0, b1: string;)
```
begin
  with MLRTable1 do
    begin
      b0 := floatToStr(b[0]);
      b1 := floatToStr(b[1]);
      Y := ResponseVarY.OutputTerm;
      X := ControlVar01.OutputTerm;
      Edit1.Text := Y+' = '+b0+'   +   '+b1+'*('+X+')'
    end
end;
```

Possible Edit1.Text Value after above code:

Sales/100   = 1.520 +   2.491*(Population/1000)

Process method

Applies To
TDependentVar, TIndepVar

Declaration
```
function Process(Dataset: TDatasetPtr): double;
```

Description
This function is used internally by the MLRTable and MLRQuery components to compute the *value* of the response and control variables as the data is loaded into the regression buffers. Since the control and response variables allow transformation of the data, mathematical operations are performed based on one (or possibly two) dataset datafield(s) and a *transformation* function.

Process is also useful to the programmer when using the Predict function on records from any TTable or TQuery containing identically named fields having the same type of numeric data. This is the situation shown in the example below.

Using Process is necessary whenever transformations have been employed, or if there is any uncertainty about whether the regression has been performed using transformed datafields.

Example Usage (assumes var Query1:TQuery;
MLRTable1:TMlrTable;
w:glnpvector; k,m:word;
y: array[1..100] of double;)

{Assumes that a successful regression has been
performed using MLRTable1. }

```
begin
  with Query1 do
    begin
      Close;
      SQL.Clear;
      SQL.Add('Select * From "SALES.DB"');
      SQL.Add('Where Store = "Memphis"');
      Open
    end;
  with MLRTable1 do
    begin
      m := nControls;
      k := 1;
      Query1.First;
      while not Query1.EOF and (k <= 100) do
          begin
            for i := 1 to m do
              w[i] := cVariable[i].Process(@Query1);
            y[k] := Predict(w,m);
            k := k + 1;
            Query1.Next       { "Predict" Sales for 100 Memphis store transactions }
          end
    end
end;
```

ScaleFactor method

Applies To
TDependentVar, TIndepVar

Declaration
```
function ScaleFactor: string;
```

Description
Scalefactor returns the string representation of the value of the divisor defined by a transformation of type 'Scaled'.

SetField method

Applies To
TDependentVar

Declaration
```
procedure SetField(const parValue: string);
```

Description
The procedure Setfield is used to set the ResponseVarY field to the datafield name in the MLRTable or MLRQuery which will be used for the dependent variable.

This procedure is used internally by the response variable property editor. When creating and setting components programmatically, the parValue parameter would be set to the name of the datafield in the underlying dataset. parValue must match **exactly**, the name of an existing numeric type datafield.

Example
```
begin
  MLRQuery1.ResponseVarY.SetTransformation('None')
  MLRQuery1.ResponseVarY.SetField('Crime Rate')
end;
```

SetParameters method

Applies To
TDependentVar, TIndepVar

Declaration
```
procedure SetParameters(const parType: integer; const parValue: string);
```

Description
This procedure is used internally only.

SetScaleFactor method

Applies To
TDependentVar, TIndepVar

Declaration
```
procedure SetScaleFactor(const parValue: string);
```

Description
The SetScaleFactor procedure is used under the *Scaled* transformation to set the value by which to divide DataField1.

This procedure is used internally by the control variable property editor. When creating and setting components programmatically, the parValue parameter would be set to the string representation of the floating point number by which divide DataField1. The number represented by parValue **must be non-zero**.

Example
```
begin
  with MLRTable2 do
    begin
       ResponseVarY.SetTransformation('Scaled');
       ResponseVarY.SetField('Lottery Ticket Sales');
       ResponseVarY.SetScaleFactor('1000');
       ControlVar01.SetTransformation('Scaled');
       ControlVar01.SetField1('Population');
       ControlVar01.SetScaleFactor('1000');
     end
end;
```

NOTE:
The above example can be implemented without code, using the ControlVar and ResponseVar property editors.

SetTransformation method

Applies To
TDependentVar, TIndepVar

Declaration
```
procedure SetTransformation(const parValue: string);
```

Description
The SetTransformation procedure is used to define the transformation to apply to the control variable and/or the response variable.

This procedure is used internally by the control variable and response variable property editors. When creating and setting components programmatically, the parValue parameter would be set to the string representation of the desired transformation.

For the ResponseVarY property, four (4) (string-valued) transformations are defined:

Transformation	**Meaning and Conditions**
'Inverse':	ResponseVarY = 1/datafield (datafield **nonzero for all records**)
'Log':	ResponseVarY = ln(datafield) (datafield **positive for all records**)
'None':	ResponseVarY = datafield
'Scaled':	ResponseVarY = datafield/scalefactor (scalefactor **nonzero**)

For the ControlVarxx property, eight (8) (string-valued) transformations are defined:

Transformation	**Meaning and Conditions**
'Inactive':	ControlVarxx not in current model
'Inverse':	ControlVarxx = 1/datafield1 (datafield1 **nonzero for all records**)
'Log':	ControlVarxx = ln(datafield1) (datafield1 **positive for all records**)
'None':	ControlVarxx = datafield1
'Power':	ControlVarxx = datafield1^exponent (exponent **integer valued**)
'Product':	ControlVarxx = datafield1*datafield2
'Quotient':	ControlVarxx = datafield1/datafield2 (datafield2 **nonzero for all records**)
'Scaled':	ControlVarxx = datafield1/scalefactor (scalefactor **nonzero**)
'Sqrt':	ControlVarxx = sqrt(datafield1) (datafield1 **positive for all records**)

Before using one of the transformations above, the user should have sufficient reason to believe that none of the values contained in the records in the dataset violate the conditions of the transformation.

Example

```
begin
  with MLRTable2 do
    begin
      ResponseVarY.SetTransformation('Scaled');
      ResponseVarY.SetField('Lottery Ticket Sales');
      ResponseVarY.SetScaleFactor('1000');
      ControlVar01.SetTransformation('Scaled');
      ControlVar01.SetField1('Local Population');
      ControlVar01.SetScaleFactor('1000');
      ControlVar02.SetTransformation('Product');
      ControlVar02.SetField1('Total Local Income');
      ControlVar02.SetField2('Local Population')
    end
end;
```

NOTE:
The above example can be implemented without code, using the ControlVar and ResponseVar property editors.

Transformation method

Applies To
TDependentVar, TIndepVar

Declaration
```
function Transformation: string;
```

Description
The Transformation function returns the string-valued representation of the user-defined transformation. This function is valid for any ControlVarxx or ResponseVarY property. It is expressed in the context of the name of the datafield(s) attached by the ControlVar or ResponseVar property.

For the ControlVarxx property, eight (8) (string-valued) transformations are defined:

Transformation	**Meaning**
'Inactive':	ControlVarxx not in current model
'Inverse':	ControlVarxx = 1/datafield1
'Log':	ControlVarxx = ln(datafield1)
'None':	ControlVarxx = datafield1
'Power':	ControlVarxx = datafield1^exponent
'Product':	ControlVarxx = datafield1*datafield2
'Quotient':	ControlVarxx = datafield1/datafield2
'Scaled':	ControlVarxx = datafield1/scalefactor
'Sqrt':	ControlVarxx = sqrt(datafield1)

For the ResponseVarY property, four (4) (string-valued) transformations are defined:

Transformation	**Meaning**
'Inverse':	ResponseVarY = 1/datafield
'Log':	ResponseVarY = ln(datafield)
'None':	ResponseVarY = datafield
'Scaled':	ResponseVarY = datafield/scalefactor

TIndepVar Class

Methods

Unit

MIrtypes

Description

The TIndepVar class defines the capabilities and parameters for the control variable(s) property of the MLRQuery and MLRTable components. The declaration is:

```
TIndepVar = class(TStringList);
```

All properties are stored as elements of the string list. The elements are set, modified, and accessed through the following (public) methods:

Modeling Methods:

```
procedure SetTransformation(const parValue: string);
procedure SetField1(const parValue: string);
procedure SetField2(const parValue: string);
procedure SetUsage(const parValue: string);
procedure SetExponent(const parValue: string);
procedure SetScaleFactor(const parValue: string);

function Process(DataSet: TDatasetPtr): double;
```

Output Methods:

```
function Transformation: string;
function Field1: string;
function Field2: string;
function Exponent: string;
function ScaleFactor: string;
function OutputTerm: string;
```

Typically, the user will have need for the Output Methods for regression reporting. The modeling methods need to be used only when creating and modeling components programmatically. Otherwise, design-time properties are set in the property editors (highly recommended).

Methods

Key methods

- Exponent
- Field1
- Field2
- OutputTerm
- Process
- ScaleFactor
- SetExponent
- SetField1
- SetField2
- SetParameters
- SetScaleFactor
- SetTransformation
- SetUsage
- Transformation
- Usage

Exponent method

Applies To
TIndepVar

Declaration
```
function Exponent: string;
```

Description
The Exponent method returns the string representation of the value of the exponent under the *Power* transformation. Note that ControlVarxx.Exponent only returns a *valid* expression when ControlVarxx.Transformation = 'Power'.

Field1 method

Applies To
TIndepVar

Declaration
```
function Field1: string;
```

Description
The Field1 method accesses the *datafield1* value defined in a ControlVarxx transformation. This the actual name of the field in the underlying table. This value will always be valid for any defined ControlVar.

Field2 method

Applies To
TIndepVar

Declaration
```
function Field2: string;
```

Description
The Field2 method accesses the *datafield2* value defined in a ControlVarxx transformation. This the actual name of the field in the underlying table. This field name returned by Field2 is only valid when the transformation is either 'Product' or 'Quotient'.

SetExponent method

Applies To
TIndepVar

Declaration
```
procedure SetExponent(const parValue: string);
```

Description
The SetExponent procedure is used under the *Power* transformation to set the power to which to raise DataField1.

This procedure is used internally by the control variable property editor. When creating and setting components programmatically, the parValue parameter would be set to the string representation of the integer to raise DataField1. parValue must represent a positive integer. In addition, parValue can be set to '1/2' to allow a *square root* transformation. The string value '1/2' is the only value other than a positive integer that will be accepted.

Example
```
begin
  with MLRTable2 do
    begin
       ResponseVarY.SetTransformation('None');
       ResponseVarY.SetField('Wattage');
       ControlVar01.SetTransformation('Power');
       ControlVar01.SetField1('Current');
       ControlVar01.SetExponent('2')
     end
end;
```

NOTE:
The above example can be implemented without code, using the ControlVar and ResponseVar property editors.

SetField1 method

Applies To
TIndepVar

Declaration
```
procedure SetField1(const parValue: string);
```

Description
The procedure Setfield1 is used to set the ControlVarxx DataField1 to the datafield name in the MLRTable or MLRQuery which will be used for the dependent variable.

This procedure is used internally by the control variable property editor. When creating and setting components programmatically, the parValue parameter would be set to the name of the datafield in the underlying dataset. parValue must match **exactly**, the name of an existing numeric type datafield.

Example
```
begin
  MLRTable1.ControlVar01.SetField1('Average Education')
end;
```

NOTE:
DataField1 is a "private" property implemented as an element of the TStringList array, which is the base class for TIndepVar. DataField1 is used in ***all*** transformations for a ControlVarxx property.

SetField2 method

Applies To
TIndepVar

Declaration
```
procedure SetField2(const parValue: string);
```

Description
The procedure Setfield2 is used to set the optional ControlVarxx DataField2 to the datafield name in the MLRTable or MLRQuery which will be used for the dependent variable under certain transformations.

This procedure is used internally by the control variable property editor. When creating and setting components programmatically, the parValue parameter would be set to the name of the datafield in the underlying dataset. parValue must match **exactly**, the name of an existing numeric type datafield.

Example
```
begin
   with MLRTable1 do
     begin
        ControlVar01.SetField1('Volume');
        ControlVar01.SetField2('Concentration')
     end
end;
```

NOTE:
DataField2 is a "private" property implemented as an element of the TStringList array, which is the base class for TIndepVar. DataField2 is used in ***only the following*** transformations for a ControlVarxx property: *Product, Quotient*.

SetUsage method

Applies To
TIndepVar

Declaration
```
procedure SetUsage(const parValue: string);
```

Description
Used internally.

Usage method

Applies To
TIndepVar

Declaration
```
function Usage: string;
```

Description
This is an internally used function for the control variable property editor.

TDatasetPtr type

Unit
Mlrtypes

Declaration
```
TDatasetPtr = ^TDataset;
```

Description
This defines a pointer to a dataset (TQuery or TTable and descendants). Used internally by several procedures.

cMAXDATA constant

Unit
Maxdata

Declaration
```
cMAXDATA = 125;  { for 16-bit
cMAXDATA = 1500;   for 32-bit }
```

Description
The constant cMAXDATA allows the user to modify the maximum size of the regression buffer. This should be adjusted to accommodate the maximum amount of data based on available memory.

This constant usually requires no adjustment under 32-bit versions.

glindex type

Unit
Matrix

Declaration
```
glindex = array[1..cMAXVARS] of integer;
```

Description
Used internally by matrix routines.

glmbynp type

Unit
Matrix

Declaration
```
glmbynp = array[1..cMAXDATA,1..cMAXVARS] of float;
```

Description
Used internally by matrix routines.

glmvector type

Unit
Matrix

Declaration
```
glmvector = array[1..cMAXDATA] of float;
```

Description
Used internally by matrix routines.

glnpbym type

Unit
Matrix

Declaration
```
glnpbym = array[1..cMAXVARS,1..cMAXDATA] of float;
```

Description
Used internally by matrix routines.

glnpbynp type

Unit
Matrix

Declaration
```
glnpbynp = array[1..cMAXVARS,1..cMAXVARS] of float;
```

Description
Used internally by matrix routines.

glnpvarray type

Unit
Matrix

Declaration
```
glnpvarray = array[1..cMAXVARS] of glnpvector;
```

Description
Used internally by matrix routines.

NumericField routine

Unit
Mlrtypes

Declaration
```
function NumericField(field: TField): boolean;
```

Description
The NumericField function is used primarily by the MLRCtrl and MLRResp property editors to list only the numeric fields in a table to include in a regression model. The user can access this function to verify that a field to include in a regression is a valid numeric field when using the controls programmatically (rather than design-time).