

Report Workshop

Report Workshop © trichview.com

Table of Contents

Part I Report Workshop 9

Part II Overview 10

1..Definitions ... 10
2..Template Syntax ... 11
Fields ... 11
Data fields ... 12
Variables ... 15
Commands ... 16
Cross-tab header fields ... 19
Functions ... 20
Data field types ... 23
Format strings ... 24
3..Data queries ... 28
4..Extending Report Workshop ... 30
5..Limitations in old versions of Delphi ... 33
6..Version history ... 33

Part III Main Components 33

1..TCustomRVReportGenerator ... 34
Properties ... 34
TCustomRVReportGenerator.DataProvider ... 34
TCustomRVReportGenerator.ErrorColor ... 34
TCustomRVReportGenerator.Texts ... 35
TCustomRVReportGenerator.Variables ... 35
Methods ... 36
TCustomRVReportGenerator.EscapeDataQuery ... 36
TCustomRVReportGenerator.Execute ... 36
Events ... 36
TCustomRVReportGenerator.OnCreateQueryProcessor ... 36
TCustomRVReportGenerator.OnDataQueryProgress ... 37
TCustomRVReportGenerator.OnError ... 40
TCustomRVReportGenerator.OnProcessRecord ... 45
2..TRVReportGenerator ... 46

Part IV Data Provider Components 46

1..TRVReportAbsDataProvider ... 49
2..TRVReportADODataProvider ... 50
3..TRVReportBDEDataProvider ... 50
4..TRVReportCustomDBDataProvider ... 51
Properties ... 52
TRVReportCustomDBDataProvider.UseRecordCount ... 52
Events ... 52
TRVReportCustomDBDataProvider.OnCreateDataSet ... 52
TRVReportCustomDBDataProvider.OnDataSetCreated ... 52
TRVReportCustomDBDataProvider's query events ... 53

5..TRVReportDataProvider 53
Methods 54
TRVReportDataProvider.CreateQueryProcessor 54
6..TRVReportDBDataProvider 54
Properties 55
TRVReportDBDataProvider.DataSets 55
Classes of properties 55
TRVDataSetCollection 55
Properties 55
TRVDataSetCollection.Items 55
TRVDataSetItem 55
Properties 56
TRVDataSetItem.DataSet 56
TRVDataSetItem.Name 56
TRVDataSetItem.UseRecordCount 56
7..TRVReportDBISAMDataProvider 56
8..TRVReportDBXDataProvider 57
9..TRVReportEDBDataProvider 58
10..TRVReportFDDataProvider 58
11..TRVReportFDMongoDataProvider 59
12..TRVReportIBCDataProvider 60
13..TRVReportIBDataProvider 61
14..TRVReportIBODataProvider 62
15..TRVReportMyDataProvider 62
16..TRVReportMySQLDataProvider 63
17..TRVReportNxDataProvider 64
18..TRVReportPgDataProvider 65
19..TRVReportPSQLDataProvider 65
20..TRVReportUniDataProvider 66
21..TRVReportZEOSDSDataProvider 67

Part V Additional Components 68

1..TRVShape 68
Properties 68
TRVShape.BackgroundColor, BackgroundOpacity 68
TRVShape.Color, Opacity 69
TRVShape.EqualSides 69
TRVShape.GradientType 69
TRVShape.LineColor 69
TRVShape.LineUsesFillColor 70
TRVShape.LineWidth 70
TRVShape.Margin, Padding 70
TRVShape.ShapeProperties 70
TRVShape.StartColor 70
Methods 71
TCustomRVReportGenerator.EscapeDataQuery_2 71
TCustomRVReportGenerator.Execute_2 71
Events 71
TCustomRVReportGenerator.OnCreateQueryProcessor_2 71
TCustomRVReportGenerator.OnDataQueryProgress_2 72
TCustomRVReportGenerator.OnError_2 74

TCustomRVReportGenerator.OnProcessRecord_2 ... 79

Part VI Document Items 80

1..Report table - TRVReportTableItemInfo ... 80

Properties ... 80

TRVReportTableItemInfo.BackgroundColorChangers ... 80
TRVReportTableItemInfo.BackgroundVisualizers ... 81
TRVReportTableItemInfo.CrossTabulation ... 81
TRVReportTableItemInfo.Processed ... 81
TRVReportTableItemInfo.ReportCells ... 81
TRVReportTableItemInfo.Row GenerationRules ... 82
TRVReportTableItemInfo.SelectedRule ... 82
TRVReportTableItemInfo.Variables ... 82

Methods ... 83

TRVReportTableItemInfo.SetBackgroundColorChangers ... 83
TRVReportTableItemInfo.SetBackgroundVisualizers ... 83
TRVReportTableItemInfo.SetCellColorChanger ... 83
TRVReportTableItemInfo.SetCellDataQuery ... 83
TRVReportTableItemInfo.SetCellHighlightColor ... 83
TRVReportTableItemInfo.SetCellVisualizer ... 83
TRVReportTableItemInfo.SetCrossTabulation ... 84
TRVReportTableItemInfo.SetRow GenerationRules ... 84
TRVReportTableItemInfo.SetVariables ... 84

Classes of properties ... 84

TRVCrossTab ... 84
Properties ... 86
TRVCrossTab.Column, FirstRow ... 86
TRVCrossTab.ColumnGenerationType ... 88
TRVCrossTab.HighlightColor ... 91
TRVCrossTab.Levels ... 91
TRVCrossTab.MaxColCount ... 92
TRVCrossTab.TextForEmptyCells ... 92
TRVCrossTabLevel ... 92
Properties ... 93
TRVCrossTabLevel.CaptionFieldName ... 93
TRVCrossTabLevel.CaseSensitive ... 94
TRVCrossTabLevel.ClearEmptySummaryCols ... 95
TRVCrossTabLevel.DataQuery ... 95
TRVCrossTabLevel.DataQueryFieldName ... 97
TRVCrossTabLevel.FieldName ... 97
TRVCrossTabLevel.MaxColCount ... 98
TRVCrossTabLevel.MinValue, MaxValue, Step ... 98
TRVCrossTabLevel.SortType ... 99
TRVCrossTabLevels ... 99
Properties ... 100
TRVCrossTabLevels.Items ... 100
TRVReportColorChangers ... 100
Properties ... 100
TRVReportColorChangers.Items ... 100
TRVReportCustomValueVisualizerCollection ... 100
Properties ... 101
TRVReportCustomValueVisualizerCollection.ForbiddenIds ... 101
Methods ... 101
TRVReportCustomValueVisualizerCollection.AssignWithId ... 101
TRVReportCustomValueVisualizerCollection.FindById ... 101
TRVReportTableCellData ... 101
Properties ... 102

TRVReportTableCellData.ColorChangerId ... 102
TRVReportTableCellData.ColorValue ... 102
TRVReportTableCellData.HighlightColor ... 102
TRVReportTableCellData.DataQuery ... 103
TRVReportTableCellData.Value ... 103
TRVReportTableCellData.VisualizerId ... 104
TRVReportValueVisualizers ... 104
Properties ... 104
TRVReportValueVisualizers.Items ... 104
TRVRow GenerationCustomRule ... 105
Properties ... 105
TRVRow GenerationCustomRule.DataQuery ... 105
TRVRow GenerationCustomRule.FirstRow , Row Count ... 106
TRVRow GenerationCustomRule.HighlightColor ... 106
TRVRow GenerationCustomRule.Name ... 106
TRVRow GenerationCustomRule.SubRules ... 107
TRVRow GenerationNestedRule ... 109
Properties ... 109
TRVRow GenerationNestedRule.SkipLeftColCount, SkipRightColCount ... 109
TRVRow GenerationNestedRule.MergeWithPrevious ... 110
TRVRow GenerationNestedRules ... 111
Properties ... 111
TRVRow GenerationNestedRules.Items ... 111
TRVRow GenerationRule ... 111
Properties ... 112
TRVRow GenerationRule.Allow SummaryCols, Allow SummaryRow s ... 112
TRVRow GenerationRule.CaseSensitive ... 112
TRVRow GenerationRule.CopyFirstCol, CopyColCount, CopySpacingColCount, CopyMaxCount ... 113
TRVRow GenerationRule.KeyFieldNames ... 115
TRVRow GenerationRules ... 115
Properties ... 115
TRVRow GenerationRules.Items ... 115
.2..Shape - TRVShapeItemInfo ... 116
Properties ... 116
TRVShapeItemInfo.Alt ... 116
TRVShapeItemInfo.BackgroundOpacity ... 117
TRVShapeItemInfo.Color, Opacity ... 117
TRVShapeItemInfo.EqualSides ... 117
TRVShapeItemInfo.GradientType ... 117
TRVShapeItemInfo.LineColor ... 118
TRVShapeItemInfo.LineUsesFillColor ... 118
TRVShapeItemInfo.LineWidth ... 118
TRVShapeItemInfo.ShapeProperties ... 119
TRVShapeItemInfo.StartColor ... 119
TRVShapeItemInfo.Width, Height ... 119

Part VII Data Visualizers 119

.1..TRVReportAreaSizeVisualizer ... 121
Properties ... 122
TRVReportAreaSizeVisualizer.Gradient ... 122
TRVReportAreaSizeVisualizer.MinSize, MaxSize ... 122
TRVReportAreaSizeVisualizer.Shape ... 122
.2..TRVReportBarVisualizer ... 123
Properties ... 124
TRVReportBarVisualizer.AxisPosition, AxisColor ... 124
TRVReportBarVisualizer.BackgroundColor, BackgroundOpacity ... 125

TRVReportBarVisualizer.BarDirection 126
TRVReportBarVisualizer.FitWidth 126
TRVReportBarVisualizer.Gradient 127
TRVReportBarVisualizer.MaxWidth 127
TRVReportBarVisualizer.NegativeColor, NegativeLineColor 127
.3..TRVReportColorChangerItem 127
Properties 128
TRVReportColorChangerItem Colors 128
TRVReportColorChangerItem Opacities 128
TRVReportColorChangerItem.PercentMiddle 129
.4..TRVReportColoredShapeVisualizer 129
Properties 130
TRVReportColoredShapeVisualizer.AbsoluteValues 130
TRVReportColoredShapeVisualizer.ConditionalShapeProperties 131
TRVReportColoredShapeVisualizer.MaxSize 131
.5..TRVReportCustomShapeVisualizer 131
Properties 132
TRVReportCustomShapeVisualizer.Gradient 132
TRVReportCustomShapeVisualizer.LineUsesFillColor 132
TRVReportCustomShapeVisualizer.ShapeProperties 132
TRVReportCustomShapeVisualizer.ShapeScaleX 133
.6..TRVReportCustomValueVisualizer 133
Properties 133
TRVReportCustomValueVisualizer.Color, LineColor 133
TRVReportCustomValueVisualizer.HAlign, VAlign 134
TRVReportCustomValueVisualizer.Margin 134
.7..TRVReportCustomValueVisualizerBase 134
Properties 135
TRVReportCustomValueVisualizerBase.Id 135
TRVReportCustomValueVisualizerBase.Min-Max 135
TRVReportCustomValueVisualizerBase.ValueString 135
.8..TRVReportGaugeVisualizer 136
Properties 137
TRVReportGaugeVisualizer Area Colors 137
TRVReportGaugeVisualizer Area Percents 138
TRVReportGaugeVisualizer.Color, MiddleColor 138
TRVReportGaugeVisualizer.MaxSize 139
TRVReportGaugeVisualizer.Min-Max 139
TRVReportGaugeVisualizer.RedLow Values 139
TRVReportGaugeVisualizer.SingleColorMode 140
.9..TRVReportPieVisualizer 141
Properties 142
TRVReportPieVisualizer.Gap 142
TRVReportPieVisualizer.Gradient 142
TRVReportPieVisualizer.LineColor 142
TRVReportPieVisualizer.LineWidth 143
TRVReportPieVisualizer.MaxSize 143
TRVReportPieVisualizer.PartsCount 143
10..TRVReportShapeRepeaterVisualizer 144
Properties 145
TRVReportShapeRepeaterVisualizer.BackgroundColor 145
TRVReportShapeRepeaterVisualizer.ColCount, Row Count 145
TRVReportShapeRepeaterVisualizer.Color 146
TRVReportShapeRepeaterVisualizer.ColorEmpty, LineColorEmpty, OpacityEmpty 146
TRVReportShapeRepeaterVisualizer.Direction 146

TRVReportShapeRepeaterVisualizer.MaxShapeSize 147
TRVReportShapeRepeaterVisualizer.Padding 147
TRVReportShapeRepeaterVisualizer.Spacing 147
11..TRVReportSignalStrengthVisualizer 148
Properties 149
TRVReportSignalStrengthVisualizer.ColorEmpty, LineColorEmpty, OpacityEmpty 149
TRVReportSignalStrengthVisualizer.DisplayStyle 149
TRVReportSignalStrengthVisualizer.Gradient 150
TRVReportSignalStrengthVisualizer.LineUsesFillColor 150
TRVReportSignalStrengthVisualizer.MaxSize 150
TRVReportSignalStrengthVisualizer.PartsCount 151
TRVReportSignalStrengthVisualizer.Spacing 151

Part VIII Actions 151

.1..TrvrActionCellProperties 152
.2..TrvrActionConvertToReportTable 153
.3..TrvrActionCrossTab 153
.4..TrvrActionDocProperties 154
Properties 154
TrvrActionDocProperties.DataProvider 154
.5..TrvrActionInsertShape 154
Properties 155
TrvrActionInsertShape.BackgroundColor, BackgroundOpacity 155
TrvrActionInsertShape.BorderColor 155
TrvrActionInsertShape.BorderWidth 156
TrvrActionInsertShape.CallerControl 156
TrvrActionInsertShape.Color, Opacity 156
TrvrActionInsertShape.EqualSides 156
TrvrActionInsertShape.GradientType 156
TrvrActionInsertShape.LineColor 156
TrvrActionInsertShape.LineUsesFillColor 157
TrvrActionInsertShape.LineWidth 157
TrvrActionInsertShape.OuterHSpacing, OuterVSpacing 157
TrvrActionInsertShape.ShapeProperties 157
TrvrActionInsertShape.StartColor 157
TrvrActionInsertShape.UserInterface 158
TrvrActionInsertShape.Width, Height 158
Methods 158
TrvrActionInsertShape.ShowInsertDialog 158
.6..TrvrActionInsertTable 158
.7..TrvrActionRowGenerationRules 159
.8..TrvrCustomActionReportTable 160
Properties 160
TrvrCustomActionReportTableWithDataProvider.DataProvider 160

Part IX Types 160

.1..TRVGradientType 160
.2..TRVReportField 161
.3..TRVReportFieldType 161
.4..TRVReportRecordCountMode 161
.5..TRVReportShape 162
.6..TRVValueVisualizerId 165

Part X Constants 166

.1 rveipcXXX Constants ... 166

Part XI Procedures and Functions 167

.1 GenerateReport ... 167

Part XII Classes 167

.1 TRVConditionalShapePropertiesCollection ... 167

Properties ... 167

TRVConditionalShapePropertiesCollection.Items ... 167

.2 TRVConditionalShapePropertiesItem ... 168

Properties ... 168

TRVConditionalShapePropertiesItem.Color ... 168

TRVConditionalShapePropertiesItem.DeltaAngle ... 168

TRVConditionalShapePropertiesItem.Value ... 168

.3 TRVReportShapeProperties ... 169

Properties ... 169

TRVReportShapeProperties.MiddlePercent ... 169

TRVReportShapeProperties.PointCount ... 169

TRVReportShapeProperties.Shape ... 170

TRVReportShapeProperties.StartAngle ... 170

.4 TRVReportDocObject ... 170

Properties ... 171

TRVReportDocObject.DataQuery ... 171

TRVReportDocObject.HighlightRules ... 171

TRVReportDocObject.PageBreaksBetweenCopies ... 172

.5 TRVReportDBQueryProcessor ... 172

.6 TRVReportGenerationSession ... 172

Methods ... 173

TRVReportGenerationSession.GetVariableValue ... 173

TRVReportGenerationSession.GetQueryProcessor ... 173

.7 TRVReportQueryProcessor ... 174

Methods ... 174

TRVReportQueryProcessor.GetAs* ... 174

TRVReportQueryProcessor.GetField ... 174

TRVReportQueryProcessor.GetFieldType ... 174

TRVReportQueryProcessor.GetRecordCount ... 175

TRVReportQueryProcessor.IsFieldEmpty ... 175

Index 176

1 Report Workshop

Report Workshop is a set of VCL components producing reports from templates. **Report Workshop** is an add-on to **TRichView** components.

Features

What's new? 33

Features

- report templates are word processing documents, so users can use a customary user interface to create them;
- resulting reports are word processing documents as well, so users can print them, exported to HTML, DocX or RTF;
- resulting report may contain hyperlinks not only to external URLs, but to other fragments of the report;
- the report generator works by (1) applying data queries to rows of tables or to table cell content, replicating them for each record of returned data; (2) replacing field codes with values;
- although Report Workshop allows some commands to be inserted in text of templates, users do not need to know any special programming/scripting language to create report templates (except for a data query language, such as SQL);
- the main example of data query is SQL select statement, but Report Workshop allows using other, not necessary database-related, sources of data;
- Report Workshop implements data providers for many popular database components, other database components can be used as well.
- rich set of visualizers for numerical values
- any number of nested sub-reports
- very flexible cross-tab reports.

Overview

Definition of the main Report Workshop terms 10

Template syntax:

- Fields 11
- Data fields 12
- Variables 15
- Commands 16
- Cross-tab header fields 19
- Functions 20
- Data field types 23
- Format strings 24

Report tables 80

Components

Report generation


TRVReportGenerator 46: a component for making reports

Data providers

A list of data provider components 46

Additional components


TRVShape 68: a component that draws a shape, such as circle, polygon, star, flag, emoticon, etc.

Actions

Report Workshop includes a set of actions [151] providing user interface for editing report templates.

2 Overview

2.1 Definitions

Below you can find a list of definitions of key terms used by Report Workshop.

This list may be used as an introduction to Report Workshop: if you read this topic, you receive a general understanding how it works.

Definitions

Report template: a documents containing *fields* and *report tables*. This document can be saved in RVF format.

Final report: a document produced by a *report generator* from a *report template*.

Field [11]**:** a part of text (enclosed in curly brackets) that are processed by a *report generator* specially. There are three types of fields: *data field*, *variable*, *command*.

Report generator [46]**:** a component that processes a TRichView component containing a *report template* to produce a *final report*.

Data provider [53]**:** a component providing data for a *report generator*. These data are needed when calculating values of *data fields*. A data provider receives a *data query* and creates a *query processor* to process it.

Data query: a text string containing a query that are processed by a *query processor*. Data queries are contained in the report itself [171], in report table row generation rules [105], and in cells [103]. Example: SQL SELECT query.

Query processor [174]**:** a class that processes a data query. As a result, a query processor provides data in a tabular format. Columns correspond to *data fields* and are identified by names. Rows contain resulting data. Example: a query processor may be a wrapper for a TDataSet-based component.

Report table [80]**:** a document object (item) for insertion in *report templates*. It is based on a TRichView table item. A report table has a list of *row generation rules*, and a list of *variables*. Cells in report tables may have an associated *data query*.

Row generation rule [105]**:** a rule containing a *data query* that are applied to a range of rows of a *report table* specified in this rule. A *report generator* replicates these rows as many times as many records a *query processor* returns for this query. *Data fields* in these rows are replaced with values from the corresponding record. Row generation rules may have names. Optionally, a rule may define *source cells*. In this case, content of these cells are replicated from left to right within row(s), and only then new rows are added.

Report table cells [101]**:** a cell of a *report table*. It may have an associated *data query*. In this case, a report generator replicates a content of this cell as many times as many records a query processor returns for this query.

Data field [12]**:** one of types of *fields* in a *report template*. A *report generator* replaces a data field code with a value returned by a *query processor*. By default, the most recent *query processor* is used, but a data field may refer to a specific *row generation rule* (by its name); in this case, a *query processor* created for this rule's *data query* is used.

Variable [15]**:** one of types of *fields* in a *report template*. A *report generator* has a list of variables. Each *report tables* has its own list of local variables. A *report generator* replaces a variable code with a value of this variable.

Command [16]**:** one of types of *fields* in a *report template*; commands invoke some special operations.

Background visualizer: an object allowing to visualize a value associated with report table cells on these cells' background.

Background color changer: an object allowing to visualize a value associated with report table cells by changing these cells' background colors and opacity.

Definitions for cross-tab reports

A cross-tab report 84 allows displaying data as a grid, with rows representing one group of data (row fields), columns representing another group of data (column fields), and intersection of rows and columns containing the summarized information (value fields).

Column fields work like a filter: for each record, values from value fields are written to the column corresponding to the specific set of values of column fields. These columns are called *data columns*. Cross-tab columns may include data columns and *summary columns*.

Column fields are listed in an order, this order creates cross-tab levels 92 (the first column field corresponds to the highest level, the last column field corresponds to the lowest level). Levels allow grouping data columns: a group of data columns corresponds to the same values of column fields in higher cross-tab levels. A summary column (if exists for this cross-tab level) allows calculating aggregate functions 20 on values from this group of data. Functions can also be inserted in columns outside the cross-tab columns, in this case they calculate on data from all cross-tab columns.

The structure of columns of a cross-tab report is defined in a cross-tab header 86. A header defines which cross-tab levels have summary columns, how many summary columns they have, how many data columns in the lowest cross-tab level. Cells in a cross-tab header may have special fields 19 for displaying column captions.

Like for normal reports, data for cross-tab report are provided by row generation rules.

Rows below the cross-tab header, not belonging to any row generation rule, can be summary rows: functions inserted in cells of these rows calculate on data from their columns.

Note

There may be a confusion because the word "table" is used for two different types of objects:

- tables of a relational database, consisting of records and fields
- tables in TRichView documents, displaying text in rows and columns; in the Report Workshop, the most important type of these tables is report tables.

In this documentation, we refer the first type of tables as "database tables", the second type of tables are "report tables". Please do not confuse these terms.

2.2 Template Syntax

2.2.1 Fields

Fields are inserted in text of template documents. There is a limitation: the whole item text must be written using the same font and color (i.e., it must belong to a single TRichView text item).

Syntax definitions

::= means "defined as"

[x] means that x is optional

x | y means "either x or y"

x+ means x repeated 1 or more times.

<x> means that x is a term (that will be defined later)

x means 'x' or 'X' character (field text is case insensitive, unless explicitly specified otherwise)

Field syntax

<field> ::= **{**<data field> | <variable> | <command> | <cross-tab header> | <function> **}**

Data fields 12

<data field> ::= <full data field name>[<field type>][**"**<format>**"**]

Variables 15

<variable> ::= **%**<variable name>[<variable type>][**"**<format>**"**]

Commands 16

<command> ::= **$**<command name>[<command parameter>]

<command name> ::= <if> | <ifdef> | <ifndef> | **else** | **endif** | <listreset>

Cross-tab header fields 19

<cross-tab header> ::= **#**<cross-tab field>[<field type>][**"**<format>**"**]

Functions 20

<function> ::= <function name>**(**<param>**)**[<field type>][**"**<format>**"**]

Special characters in text

A template text may contain '{' characters that should not be treated as a field start.

If this character does not has '}' after it in the same text item (i.e. in a text fragment written using the same font and color, not containing non-text items, tab characters and paragraph breaks), it is not treated as a field start. Otherwise, use '{{'.

Example:

The text in a template:

```
The template may contain {{$IF} commands
```

produces in the final document:

```
The template may contain {$IF} commands
```

Fields in text strings

In addition to normal text, fields can be inserted in:

- data queries of table rules and cells (for example, to implement master-detail or grouped queries);
- hints (tool tips) of items;
- hints of table cells;
- tags of items (usually used as hyperlink targets);
- names and tags of checkpoints (used to mark the position in a document).

There are the following differences for fields in these places:

- commands are not supported;
- all values are inserted as strings (if a data field cannot be represented as a string, the field is erroneous).

2.2.2 Data fields

See the topic about fields in templates 11 for a general description of the syntax.

Syntax

<data field> ::= <full data field name>[<field type>][**"**<format>**"**]

<full data field name> ::= ['][<query processor>]<field name>[']

Full data field name may be enclosed in single quotes. These single quotes are necessary if <query processor> or <field name> contain space characters.

<query processor> ::= <rule>**:** | **:** | **^+**

<field name> is a name of a field in a table generated by the rule.

Field names are case insensitive.

<field type> – see "data field types" sections below.

<format> – see "data field format strings" sections below.

About data fields

Data fields allow retrieving values returned by query processors. A query processor may be created to process the following data queries:

- root data query 171
- table row generation rule's 105 data query 105
- table cell's 101 data query 103

A query processor is referred in <query processor>.

If it is omitted, the results of the current (most deeply nested) query processor are used.

If only ':' is included, the results of the root query processor are used (the root query processor is a query processor created for root data query 171; if this data query is not assigned, there is no root query processor).

If an identifier <rule> is included, a query processor for the table row generation rule 105 with name 106 = <rule> is used.

Special values:

- '^' – the parent query processor
- '^^' – the parent of the parent query processor

and so on

Table cell's query processors cannot be referred by a name. So they can be referred only as current or parent query processors.

Examples

{fieldname} – field 'fieldname' of the current query processor

{^:fieldname} – field 'fieldname' of the parent of the current query processor

{^^:fieldname} – field 'fieldname' of the grandparent of the current query processor

{:fieldname} – field 'fieldname' of the root query processor

{rowrule:fieldname} – field 'fieldname' of the query processor for table row generation rule 111 with name 106 = 'rowrule'

Data field types

A report generator supports the following types of fields:

- text
- integer value
- floating point value
- boolean
- date, time, date and time
- memo (a text document in ANSI or Unicode)
- RTF document (normal RTF and RTF converted to a Unicode string)
- RVF document
- image (bitmap, icon, metafile, jpeg, gif, png, tiff, if graphic classes for these formats are available).
- additional field types implemented by programmers

If not specified, a format returned by TRVReportQueryProcessor[174].GetFieldType[174] is used.

You can override the field type by specifying <field type>, read the topic about specifying data field types[23].

Data field format strings

Read the topic about format strings[24].

Use

A data field can be used:

- to insert its value in a template text;
- to insert a text representation of its value in a string (such as a data query or a hyperlink target)
- as a parameter of If, IfDef, IfNDef commands[16].

Examples

These examples use SQL-based data queries.

Example 1: basic example

Let a cell's data query = 'SELECT first_name, last_name FROM persons'.

Let a data processor returns two records: ('John', 'Smith') and ('Mary', 'Black').

The cell contains:

```
{first_name} {last_name}
```

It produces:

```
John Smith
Mary Black
```

Example 2: master-detail

Let we have two database tables: master_table and detail_table.

Let there is report table having a row generation rule with the data query = 'SELECT master_id from master_table'.

Let this table has a cell (in a row affected by this rule) having the data query = 'SELECT * from detail_table WHERE master_id={master_id}'

As a result, we have a master-detail report.

Example 3

Let we have three database tables: sellers, products, sales. The "sales" table contains fields: seller_id, product_id, quantity, date (so it establish "N to N" relation between sellers and products).

Let there is table having a row generation rule with the data query = 'SELECT seller_id from sales', name = 'sellers_rule'.

It has a nested table, having a row generation rule with the data query = 'SELECT product_id from products', name = 'products_rule'.

In its order, it has a nested table, having a row generation rule with the data query = 'SELECT quantity, date WHERE seller_id={sellers_rule:seller_id} AND product_id={products_rule:product_id}'.

As a result, we have a report of sales grouped first by products, then by sellers.

Note: since, for the third table, 'products_rule' is the most recent rule, we could omit 'products_rule:' in '{products_rule:product_id}'.

2.2.3 Variables

See the topic about fields in templates [11] for a general description of the syntax.

Syntax

<variable> ::= **%**['']<variable name>['][<variable type>][**"**<format>**"**]

A variable name may be enclosed in single quotes. These single quotes are necessary, if it contains space characters.

<variable type> ::= **object**

<format> – see "variable format strings" sections below.

Variable names are case insensitive.

About variables

Variables are defined in string lists of the whole report [35] and of report tables [82].

When evaluating a variable, the report generator tries to find it to the most nested table, then (if not found) in the table containing this table, and so on. Finally, it searches in the global report variables.

The report table's variables are used when:

- when processing content of this table's cells while applying row generation rules [82];
- when processing this table's cells' data query strings [103].

They are not used:

- when processing data query strings belonging to this table's row generation rules [82],
- when processing this table's hint (tool tip).
- when processing content of this table's cells that do no belong to any row generation rule and do not have a data query.

(in these cases, you can use only variables of parent tables and global report variables).

Variables are stored in a string list. Each string in this list must have the syntax:

<variable name>**=**<variable value>

Variable objects

It is possible that a string list item has an associated object. The variable string lists own their objects, so objects are freed when the lists are cleared or destroyed, so do not free these objects yourself.

If the field type "object" is specified, this object is inserted instead of <variable value>.

The current version of Report Workshop supports only one type of objects: an image (of a class inherited from TGraphic, such as TBitmap or TPngImage).

This feature can be used to generate pictures basing on data returned for a data query in OnProcessRecord [45] event.

Variable format strings

Read the topic about format strings [24].

Variables can use format strings for strings and for images.

Use

Variables can be used:

- to insert their value or object in text of templates;
- to insert their value in strings (such as data queries or hyperlink targets);
- as parameters of If, IfDef, IfNDef commands [16].

Examples

Example 1

Let the report has the following item in its variables:

phone2="Work phone"

The code

```
{%phone2}
```

inserts

```
Work phone
```

Example 2

Let the report has the following item in its variables:

barcode=

and this line has an associated TBitmap object

The code

```
{%barcode object}
```

inserts this bitmap in a document.

Example 3

Let the report has the following item in its variables:

fullreport=yes

And the following data fields are defined: name, address, phone

The code

```
Name: {name}
{$IFDEF %fullreport}
Address: {address}
Phone: {phone}
{$ENDIF}
```

generates the output like:

```
Name: John Smith
Address: 123 Way, Nethercity
Phone: 123456
```

2.2.4 Commands

See the topic about fields in templates [11] for a general description of the syntax.

All commands are case insensitive.

The current version of Report Workshop supports commands only inside a template text. They are not supported in strings (such as data queries, hyperlink targets, item hints and checkpoints).

"IF – ENDIF" or "IF – ELSE – ENDIF"

"If" command allows excluding some content from the report result. It may be optionally followed by "Else" command, and must be finished by "EndIf" command.

All these commands must be in the same sub-document (i.e. in the same table cell)

Syntax:

<if> ::= **if** <if condition>

<if condition> ::= <operand> [<comparison operation> <operand>]

<operand> ::= <full data field name>|**%**<variable name>|<integer constant>|<string constant>

In the first case, <if condition> is a single operand, which is evaluated as True or False.

<operand> may be either a data field or a variable. They must be defined, otherwise the command is erroneous.

For data fields having integer values, any non-zero value is evaluated as *True*, zero is evaluated as *False*.

Example:

```
{$IF count}Count of details: {count}{$ELSE}No details available{$ENDIF}
```

Variables and data fields of all other types are treated as string values.

Values evaluated as *True*	Values evaluated as *False*
t	f
y	n
true	false
yes	no
on	off
1	0
Texts 35.TrueText	Texts 35.FalseText

The second case, <if condition> consists of three parameters: <operand> <comparison operation> <operand>

Important: operands and operation must be delimited by space characters.

The command compares two operands using the specified comparison operation.

<comparison operation> ::= **=|<>|<|>|<=|>=|==|!=**

Comparison operation	*Meaning*
= ==	equal to
<> !=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

In addition to fields and variables, operand may be a string or an integer constant.

String constant is a text inside double quotes:

<string constant> ::= **"**|<text>|**"**

To include a double quote in text, duplicate it.

Integer constant consists of digits and an optional minus sign:

<integer constant> ::= [**-**]<digit>+

<digit> ::= **0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**

If the field name consists only of digits, to prevent interpreting it as an integer constant, include it in single quotes.

Examples

```
{$IF test = "Passed"}OK{$ENDIF}
{$IF weight > 200}Too heavy{$ENDIF}
```

Strings comparisons are case sensitive. When a string are compared to a numeric value, the component convert this string to a numeric value before the comparison (if it is not possible, "If" command is erroneous). The conversion to a floating value uses the system default numeric separator. String value can be compared only to another string or a number (it cannot be compared to a date, for example).

"IFDEF – ENDIF" or "IFDEF – ELSE – ENDIF"

"IfDef" command is similar to "If", it has a similar syntax and meaning, but a condition is evaluated differently.

Syntax:

<ifdef> ::= **ifdef** <condition>

<condition> ::= <full data field name>|**%**<variable name>

Example:

```
{$IFDEF phone2}Phone 2: {phone2}{$ENDIF}
```

<value> may be either a data field or a variable.

A data field in the condition must exist, otherwise the command is erroneous. NULL fields (or empty string fields) are evaluated as *False*, any other values are evaluated as *True*.

For variables, non-existent or empty variables are evaluated as *False*, variables having non-empty text are evaluated as *True*.

"IFNDEF – ENDIF" or "IFNDEF – ELSE – ENDIF"

"IfNDef" command is similar to "IfDef", but the rule of condition evaluating is exactly the opposite.

Syntax:

<ifndef> ::= **ifndef** <condition>

<condition> ::= <full data field name>|**%**<variable name>

Example:

```
{$IFnDEF comments}No comments{$ENDIF}
```

"LISTRESET"

"ListReset" specifies when the paragraph numbering must be reset to 1. This command must be inserted in a numbered paragraph, otherwise it is erroneous.

Syntax:

<listreset> ::= **listreset**[<depth>]

<depth> is an integer value, zero or positive. If omitted, zero is assumed.

By default, paragraph numbering is continuous through the whole document. This command allows to reset it to 1.

The depth parameter specifies the data query on which the list is reset. 0 means the most nested query, 1 means the query containing it, and so on.

Example:

Let we need to generate an organization structure. There are queries "bosses", "managers", "workers", having a master-detail relationship.

Without ListReset	{$LISTRESET 1}	{$LISTRESET} or {$LISTRESET 0}

boss A
  manager A
  1. worker A
  2. worker B
  manager B
  3. worker C
  4. worker D
boss B
  manager C
  5. worker E
  6. worker F
  manager D
  7. worker G
  8. worker H
	boss A
	  manager A
	  1. worker A
	  2. worker B
	  manager B
	  3. worker C
	  4. worker D
	boss B
	  manager C
	  1. worker E
	  2. worker F
	  manager D
	  3. worker G
	  4. worker H
		boss A
		  manager A
		  1. worker A
		  2. worker B
		  manager B
		  1. worker C
		  2. worker D
		boss B
		  manager C
		  1. worker E
		  2. worker F
		  manager D
		  1. worker G
		  2. worker H

2.2.5 Cross-tab header fields

See the topic about fields in templates [11] for a general description of the syntax.

See the topic about cross-tab reports [84].

Syntax

<cross-tab header> ::= **#**<cross-tab field>[<field type>][**"**<format>**"**]

<cross-tab field> is a value of one of fields specified in Table.CrossTabulation [81].Levels [91][].FieldName [97] or Table.CrossTabulation [81].Levels [91][].CaptionFieldName [93].

Field names are case insensitive.

About cross-tab header fields

References to cross-tab fields may occur:

- in cells of the cross-tab header [86];
- in Table.CrossTabulation [81].Levels [91][].DataQuery [95], if Table.CrossTabulation [81].ColumnGenerationType [88]=*rvcgtDataQueryCascade*

In the first case, you can refer to fields corresponding to the cross-tabulation level of this cell, and to fields corresponding to higher levels of the cross-tabulation.

In the second case, you can refer to fields corresponding to higher levels of the cross-tabulation.

<cross-tab field> can be empty. Such field is an equivalent to the lowest available cross-tab level.

Cross-tab header field types

Report Workshop supports the following types of cross-tab header fields (when referring to a value of FieldName [97] field):

- text
- integer value
- floating point value
- boolean
- date, time, date and time.

When referring to a value of CaptionFieldName [93], the field type is always text.

You can override the field type by specifying <field type>, read the topic about specifying data field types [23].

Cross-tab header field's format strings

Read the topic about format strings [24].

Example

Let we have:

- Table.CrossTabulation [81].Levels [91][0].CaptionFieldName [93] = 'Category'
- Table.CrossTabulation [81].Levels [91][1].CaptionFieldName [93] = 'Product'

A cross-tab report table may look like it is shown below (the pink color denotes cross-tab header cells corresponding to data columns, the light blue color denotes a header for a summary column):

	{#Category}	
	{#Product}	Total for {#Category}
{Year}	{Sales}	{sum(Sales)}

The same report table can be written in a shorter way:

	{#}	
	{#}	Total for {#Category}
{Year}	{Sales}	{sum(Sales)}

Additional examples can be found:

- in the topic about a cross-tab header [98] (replacing fields in header cells)
- in the topic about cross-tab data queries [95] (replacing fields in cross-tab data queries)

2.2.6 Functions

See the topic about fields in templates [11] for a general description of the syntax.

See the topic about cross-tab reports [84].

This type of field allows inserting the result of an aggregate functions in cross-tab tables.

Functions are not available for non-cross-tab tables. If you want to add summary row(s) or column(s) for non-cross-tab tables, you can:

- calculate them using SQL functions, if you use SQL data queries, or
- use OnProcessRecord [45] event to calculate the function in a variable [15].

Syntax

<function> ::= **=**<function name>**(**<param>**)**[<field type>][**"**<format>**"**]

<function name> ::= **min** | **max** | **sum** | **average** | **count** | **var.p** | **var.s** | **stddev.p** | **stddev.s** | <custom function name>

<param> ::= <field name>

<field type> – see "function results types" sections below.

<format> – see "format strings" sections below.

<custom function name> – see "additional functions" below.

About functions

This type of field allows inserting the result of an aggregate functions in cross-tab tables. It is not valid in all other places.

The argument of this function may be one of numeric value fields (see the topic about cross-tab reports [84]).

In other words, <param> is a name of one of data fields in the result of application of one of Table.RowGenerationRules [82][].DataQuery [105], providing that it:

- is not listed in Table.CrossTabulation [81].Levels [91][].FieldName [97]
- is not listed in Table.RowGenerationRules [82][].KeyFieldNames [115]
- have the type [161] either *rvrftInteger* or *rvrftFloat*.

A set of values of this parameter (used to apply the function) depends on the place of insertion of this field code:

- outside report tables, in a non-cross-tab report tables: not valid (no values)
- in rows of a cross-tab header [86] or above: not valid (no values)
- in the cells located in the intersection of rows corresponding to RowGenerationRules [82][] and:
  - ... data columns of the cross-tab headers: not valid (no values)
  - ... summary columns of the cross-tab header: values corresponding to this summary columns and this row
  - ... all other columns: all values corresponding to this row
- in the cells located in the intersection of all other rows and
  - ... data columns of the cross-tab headers: all values of this column
  - ... summary columns of the cross-tab header: all values corresponding to this summary columns and all rows
  - ... all other columns: all values

Example

In this example:

- the pink background color denotes headers of cross-tab data columns
- the light blue background color denotes a header of a cross-tab summary column
- the green background color denotes cells belonging to a row generation rule

Template

	{#Category}		Grand Tota
	{#Product}	Total for {#C	
{Year}	{Sales}	{sum(Sales)}	**{sum(Sales**
Grand Total	{sum(Sales)}	**{sum(Sales)**	**{sum(Sales**

Result Sample

	Fruits			Vegetables			Grand
	Apple	Orang	Total f	Tomat	Cucur	Total f	
2014	100	200	300	300	400	700	**1000**
2015	150	250	400	350	450	800	**1200**
Grand	250	450	**700**	650	850	**1500**	**2200**

As you can see, the same function code ('sum(Sales)') is inserted in several cells. But the results of this function are different, because a set of input values for the function depends on a cell location:

- normal blue font: this cell is in the summary column of the cross-tab, and in the row generation rule; it calculates a sum of values of this row at this level (300=100+200, 400=150+250, 700=300+400, 800=350+450)
- **bold blue font:** this cell is outside of cross-tab columns, and in the row generation rule; it calculates the sum of all values of this row (1000=100+200+300+400=300+700, 1200=150+250+350+450=400+800)
- normal green font: this cell is in the data cross-tab column, and outside the row generation rule; it calculates the sum of all values in this column (250=100+150, 450=200+250, 650=300+350, 850=400+450)

- **bold green font:** this cell is in the summary cross-tab column, and outside the row generation rule; it calculates the sum of all values corresponding to this summary column (700=100+200+150+250=300+400, 1500=300+400+350+450=700+800)
- **bold dark red font:** this cell is outside the cross-tab columns, and outside the row generation rule; it calculates the sum of all values (2200=100+200+300+400+150+250+350+450)

Functions list

<function name>	Meaning	Result type	Minimal necessary count of input values
min	returns the smallest value	type of the parameter	1
max	returns the largest value		1
sum	calculates the sum of values		0
count	returns the count of values	int	0
average	returns the average value (arithmetic mean) $\bar{x}$	float	1
var.p	calculates variance (based on the entire population) $\frac{\sum(x-\bar{x})^2}{n}$		1
var.s	estimates variance based on a sample $\frac{\sum(x-\bar{x})^2}{(n-1)}$		2
stddev.p	calculates standard deviation (based on the entire population) $\sqrt{\frac{\sum(x-\bar{x})^2}{n}}$		1
stddev.s	estimates standard deviation based on a sample $\sqrt{\frac{\sum(x-\bar{x})^2}{(n-1)}}$		2

If function cannot be calculates (because it is inserted in a wrong context, or because of not enough input values), a report generator inserts Texts [35].InvalidFunctionResult in place of the function field.

Additional functions

In addition to the functions listed above, programmers can implement their own functions.

As an example, ReportWorkshop includes TRVReportMedianCalculator unit implementing median.

<custom function name>	Meaning	Result type	Minimal necessary count of input values

median	calculates the median	float	1

See also: extending Report Workshop [30]

Functions results types

As you can see from the table above, a function may return either integer or floating point value.

You can override the result type by specifying <field type>, read the topic about specifying data field types [23].

Data field format strings

Read the topic about format strings [24].

Format string is not applied to Texts [35].InvalidFunctionResult.

2.2.7 Data field types

A type may be specified explicitly in:

- data fields [12]
- cross-tab header fields [19]
- functions [20]

Syntax

<field type> ::= **text** | **int** | **float** | **bool** | **datetime** | **date** | **time** | **blob** | **ansimemo** | **unicodememo** | **document** | **rtf** | **rvf** | **image** | **bitmap** | **gif** | **png** | **jpeg** | **tiff** | **icon** | **metafile** | <alternative rvf name> | <custom field type>

<alternative rvf name> is a text of RVReportGenerator [46].Texts [35].RVF.

Description

If specified, it is one of: text, int, float, bool, datetime, date, time, blob, ansimemo, unicodememo, document, rtf, rvf, image, bitmap, gif, png, jpeg, tiff, icon, metafile. As you can see, these types correspond to TRVReportFieldType [161].

Additionally, the value RVReportGenerator.Texts [35].RVF is treated as 'rvf'.

Additionally, programmers can register their own field types.

There are several reasons to specify a type:

- converting value to another format (for example, float to int, datetime to date)
- more efficient processing (for example, blob to rvf allows skipping a format detection step)
- allowing to use format string [24] from another field type (for example, int to float allows format integer values as floating point values)
- ignoring a field type returned by a query processor and using the specified type;
- visualizing values in special ways (especially when using custom field types).

Possible data field conversions

Not all combination of the original type and the specified type are allowed. If a combination of types is incorrect, the specified type is ignored.

All types can be converted to 'text'.

Additionally, the following conversions are possible.

Original field type	Can be converted to...

text	int, bool, float, datetime, date, time, datetime
int, float, bool	int, float, bool
time, date, datetime	time, date, datetime
blob, ansimemo, unicodememo, document, rtf, rvf, image, bitmap, gif, png, jpeg, tiff, icon, metafile	blob, ansimemo, unicodememo, document, rtf, rvf, image, bitmap, gif, png, jpeg, tiff, icon, metafile

Notes:

- when converting non-blob types, the value is received from a query processor basing on its original type; then this value is converted to the specified type
- when converting any blob type to each other, no value conversion occurs; instead, a report generator tries to load content in the specified format.
- when converting to bool, the same rules are used as for the condition evaluation in "If" command [16].
- when converting bool to int or float, *False* is converted to 0, *True* to 1.

Custom field types

Programmers can implement additional data types. Objects that process custom data types can get initial value and return its different representation; additionally, they can [pre]process format strings.

For example, RVReportSampleFieldObjects unit implements the following additional field types:

- *num2words* – spells a number in English (int to text); for example, '{value num2words}' for value 21 returns 'twenty one';
- *uppercase* – returns text in upper case (text to text); for example '{name uppercase}' for 'John' returns 'JOHN';
- *star* – returns an image of a star having as many points as defined in the input value (int to bitmap); this field type processes format strings itself; example: '{value start "size=100 color=red linecolor=blue gradient=1 middlepercent=60"}'
- *imageinfo* – returns text describing the input image (image to text); it returns text like '[Image 250×150]'

See also: extending Report Workshop [30]

2.2.8 Format strings

Format strings can be used in:

- data fields [12]
- variables [15]
- cross-tab header fields [19]
- functions [20]

They are optional.

Format strings are different for different types of data. The following types of data have their own formats of format strings:

- floating point values
- integer values
- boolean values
- date and time
- text strings
- documents (RVF and RTF)
- images

Note 1: a report generator treats '}' as a termination of the field code. To insert '}' in a format string, use '}}'.

Note 2: a format string is enclosed in double quotes; however, a format string can include double quote characters itself; you do not need to escape them.

Format strings for floating point values

Format strings for floating point values are the same as for FormatFloat() function from SysUtils unit, extended by optional colors and optional parameters section.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is if there is nothing between the semicolons that delimit the section, the section for positive values is used instead. For formatting purposes, sections contained only colors are treated as empty.

Each section may start from a color definition inside **[]**. See the "color definition" chapter below. This means that the format string in this section cannot start from **[** (you can include it in single or double quotes, though).

A format string may start from a parameters section. It has format **[fmt**<decimal separator>[<thousands separator>]**]**.

<decimal separator> and <thousands separator> are characters used as decimal and thousands separators. If a parameters section is omitted, locale default characters are used. You should specify decimal separator explicitly if this field is used in expressions; for example, SQL requires the dot character as a decimal separator.

Example:

- "0.00;[red]-0.00" – displays two digits after the decimal point, negative values are displayed red
- "select price * {discount "[fmt.]"} from mytable"

Format strings for integer values

Format strings for integer values consist of up to three sections. The first section represents positive values and define the number format. The second section represents negative, the third section represents zero values.

Format strings for integer values have the following format:

- <Int format string> ::= [**[**<color+>**]**][<number format>][**;[**<color->**]**[**;[**<color0>**]**]]

<number format> ::= **decimal**<min length> | **roman** | **ROMAN** | **alpha** | **ALPHA** | **hex**<min length> | **HEX**<min length>

<number format> is case sensitive.

number format	example
decimal	1, 2, 3, ..., 10, 11, 12
roman	i, ii, iii
ROMAN	I, II, III
alpha	a, b, c, ..., aa, ab, ac
ALPHA	A, B, C, ..., AA, AB, AC
hex	1, 2, 3, ..., a, b, c
HEX	1, 2, 3, ..., A, B, C

<min length> is a positive integer number, it specifies the minimal number of digits in the output string; if necessary, '0' are added to the resulting text.

<color+> (if specified) defines the color for positive values.

<color-> (if specified) defines the color for negative values (otherwise, <color+> is used)

<color0> (if specified) defines the color for zero (otherwise, <color+> is used)

Example:

- decimal[blue][red][black]
  sample output: -10 0 5
- hex4
  sample output: 0000, 001a, -00ff
- alpha
  sample output: a, 0, -8

Format strings for boolean values

Format strings for boolean values consist of two sections separated by a semicolon. The first section specifies how to represent *True*, the second section specifies how to represent *False*. Each section may contain a color and a text.

Format strings for boolean values have the following format:

<Bool format string> ::= [**[**<colorT>**]**]<textT>[**;[[**<colorF>**]]**<textF>]

<colorT> (if specified) defines the color for *True*, see the "color definition" below.

<colorF> (if specified) defines the color for *False*; if not specified, <colorT> is used for *False*.

<textT> (if specified) is a text for *True*; if empty, Texts [35].TrueText is used.

<textF> (if specified) is a text for *False*; if empty, Texts [35].FalseText is used.

Examples:

- "yes;no"
  sample output: no yes
- "[green]on;[red]off"
  sample output: off on

Format strings for dates and times

Format strings for date and time values are the same as for FormatDateTime() function from SysUtils unit.

Example:

- "d/m/y"

Format strings for strings

Format strings for string values are the same as for FormatMaskText() function from MaskUtils unit.

However, FormatMaskText() always uses space characters for missing characters and ignores a blank character specified in the format string.

Report Workshop uses the blank characters specified in the mask, or DefaultBlank global variable (from MaskUtils unit) if omitted. By default, DefaultBlank = '_'.

Example:

- "(000)000-0000;0; " – displays a phone number
  sample output: (123)456-7890

Format strings for documents

This type of format strings is used when displaying RVF and RTF fields.

The format string is a combination of the following values, separated by spaces: fontname, fontsize, color, backcolor.

The corresponding properties of text of the inserted document are ignored, i.e. properties of the font of the field code are used instead.

Item in the format string	Related property of TFontInfo	Comments
fontname	FontName	This option is not applied to text items having Charset=SYMBOL_CHARSET
fontsize	SizeDouble	This option is also applied to Font.Size of list markers
color	Color	This option is not applied to hyperlinks
backcolor	BackColor	

Example:

- "fontsize fontname" – ignores font size and name in inserted documents, uses these properties from a report template instead.

Format strings for images

This type of format strings is used when displaying images.

The format string is a combination of properties, separated by spaces. Each property has the form:

<property name>=<property value>

<property name> is one of: padding, borderwidth, vspace, hspace, bordercolor, align, width, height, minwidth, minheight, maxwidth, maxheight.

Property name	Related item property (TRVExtraItemProperty)	Meaning	Value type
padding	*rvepSpacing*	Spacing around the picture, inside its border. If a background color is defined, this area is colored	integer (in pixels)
vspace	*rvepOuterVSpacing*	Spacing above and below the border	integer (in pixels)
hspace	*rvepOuterHSpacing*	Spacing to the left and to the right of the border	integer (in pixels)
bordercolor	*rvepBorderColor*	Border color	color (see the "color definition" chapter)
borderwidth	*rvepBorderWidth*	Border width	integer (in pixels)
width	*rvepImageWidth*	The image is stretched to the specified width	integer (in pixels)
height	*rvepImageHeight*	The image is stretched to the specified height	integer (in pixels)

Note: a background color is not available as a property. It is taken from the background color of text of the field code.

The following properties allow stretching the image: width, height, minwidth, minheight, maxwidth, maxheight. If the properties contradict to each other, width and height have the highest priority, minwidth and minheight have a middle priority, maxwidth and maxheight have the lowest priority.

If both width and height are defined, min* and max* properties are ignored.

If one of width/height is defined, the other side is stretched proportionally, constrained to the corresponding min* and max* properties.

If none of width/height is defined, the image is stretched proportionally, according to min* and max* properties.

Example:

- "borderwidth=2 bordercolor=orange maxwidth=500"

Color definition

In certain places of format strings, users can specify colors.

Color codes are similar to HTML: you can define either a color name, or #RRGGBB code.

Color name is one of:

aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blanchedalmond, blue, blueviolet, brown, burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue, cornsilk, crimson, cyan, darkblue, darkcyan, darkgoldenrod, darkgray, darkgreen, darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, darkred, darksalmon, darkseagreen, darkslateblue, darkslategray, darkturquoise, darkviolet, deeppink, deepskyblue, dimgray, dodgerblue, firebrick, floralwhite, forestgreen, fuchsia, gainsboro, ghostwhite, gold, goldenrod, gray, green, greenyellow, honeydew, hotpink, indianred, indigo, ivory, khaki, lavender, lavenderblush, lawngreen, lemonchiffon, lightblue, lightcoral, lightcyan, lightgoldenrodyellow, lightgreen, lightgrey, lightpink, lightsalmon, lightseagreen, lightskyblue, lightslategray, lightsteelblue, lightyellow, lime, limegreen, linen, magenta, maroon, mediumaquamarine, mediumblue, mediumorchid, mediumpurple, mediumseagreen, mediumpurple, mediumslateblue, mediumspringgreen, mediumturquoise, mediumvioletred, midnightblue, mintcream, mistyrose, moccasin, navajowhite, navy, oldlace, olive, olivedrab, orange, orangered, orchid, palegoldenrod, palegreen, paleturquoise, palevioletred, papayawhip, peachpuff, peru, pink, plum, powderblue, purple, red, rosybrown, royalblue, saddlebrown, salmon, sandybrown, seagreen, seashell, sienna, silver, skyblue, slateblue, slategray, snow, springgreen, steelblue, tan, teal, thistle, tomato, turquoise, violet, wheat, white, whitesmoke, yellow, yellowgreen.

Color codes are case-insensitive.

Examples:

- floating point value format string: "0.00;[red]-0.00"
- image format string: "borderwidth=2 bordercolor=#ff0000"

2.3 Data queries

Data queries in reports

Data query is a text string containing a query.

A report generation mainly consists of execution of data queries and applying them to the corresponding part of the report template document.

Data queries may be associated with:

- the whole report (TRVReportDocObject[170].DataQuery[171])
- a group of table rows (TRVRowGenerationCustomRule[105].DataQuery[105])
- a table cell (TRVReportTableCellData[101].DataQuery[103])
- (there are also data queries for building columns of cross-tab reports, but we skip them here for a simplicity)

A content of a report template is nested in each other: any table is inserted in the root document, cells are inserted in a table, another table may be inserted in a cell. Thus the root document's query (if defined) is a parent query for table row queries (if they exist), they are, in their order, parents for cell queries (if they are defined), cell queries are parents for queries of rows of tables inserted in these cells, and so on.

A report generator starts to execute a root query; the corresponding content is replicated as many times as many records are returned when executing this query. In each copy of this content, child queries are executed, and so on.

The complete hierarchy of queries is the following:

1. the root document's query
2. queries of row generation rules of report tables
3. queries of row generation sub-rules of report tables, then sub-rules of these sub-rules, and so on
4. queries of cells of report tables

The levels 2-4 are repeated for each nested table.

In all levels of this hierarchy, the child objects are inserted in parent objects. The level 2 is an exception: rules and sub-rules are applied to the same table, parent rules are applied to larger table fragment than their child rules.

Master-detail reports

As it was said, data queries can be associated with nested objects. A master-detail relationships may be established between parent and child objects.

Example 1: SQL

Strings of data queries may contain fields referring to results of execution of parent queries. Thus you can implement master-detail or banded reports.

For example, the root data query may be:

```
select id, master_name from mastertable
```

This report may include a table with a row generation rule having the data query:

```
select * from detailtable where master_id={id}
```

In this example, {id} will be replaced to the value of "id" field from mastertable.

This is an universal method that can be used not only in SQL, but in all types of queries.

Example 2: nested datasets

If the master table has a field representing a nested dataset (TDataSetField), you can refer to this field using a data query like:

```
field:details
```

where 'field' is a reserved prefix, 'details' is a dataset field name.

The syntax of such data queries is explained below.

Example 3: master-detail relationship between existing TDataSet components

Let we have two components inherited from TDataSet, DataSet1 and DataSet2, and a master-detail relationship is established between them.

Let we use TRVReportDBDataProvider 54 component, and it has two items in DataSets 55 collection: ('mastertable', DataSet1) and ('detailtable', DataSet2).

Now the report template can use data queries

```
mastertable
```

and

```
detailtable
```

to use data from these datasets.

Types of data queries

Data queries are processed by query processors. A syntax of data queries depends on the query processors which you plan to use in your application.

The main way to create query processors for data queries is assigning a data provider component to the report generator's DataProvider[34] property. Report Workshop includes several data providers that allow processing data queries by retrieving records from database tables. So the most typical example of a data query is SQL SELECT statement.

Another way is registering "standard" query processors for execution of queries that start from some prefix. This prefix may contain English characters, '-', '_', and must be finished with ':'.

As an example, Report Workshop includes a query processor for executing queries like 'calendar:days of month 3 of 2016'. See the topic about extending Report Workshop[30].

One prefix is reserved: 'field'. It allows using a nested dataset as a query processor.

The syntax of this query is:

field:<full data field name>

where <full data field name> is defined in the same way as in data fields[12].

Fields in data queries

All '{' characters having the following '}' are treated as field codes[11]. To prevent it, write '{{' instead of '{'.

This type of escaping is not convenient if you use JSON queries, because they may contain many '{' and '}' characters.

If your data query does not contain fields, you can use TRVReportGenerator.EscapeDataQuery[36] to prepare queries.

2.4 Extending Report Workshop

Report Workshop can be customized by implementing plug-ins.

The following plug-in types can be implemented:

- additional types for data fields
- standard query processors
- additional data provider components
- additional value visualizers
- additional functions for cross-tab reports

We are interested in third-party plug-ins for Report Workshop. If you decide to make one, please contact us. We will offer a technical help, and rewards for authors of interesting plug-ins (free licenses or discounts on our products, and others).

Additional types for data fields

Data fields have the following syntax[12]: <full data field name>[<field type>]["<format>"]

Report Workshop implements standard field types[23], such as 'int', 'image', 'time', etc.

Programmers can implement additional data types. Objects that process custom data types can get initial value and return its different representation; additionally, they can [pre]process format strings.

▼ **Example**

For example, RVReportSampleFieldObjects unit implements the following additional field types:

- *num2words* – spells a number in English (int to text); for example, '{value num2words}' for value 21 returns 'twenty one';
- *uppercase* – returns text in upper case (text to text); for example '{name uppercase}' for 'John' returns 'JOHN';

- *star* – returns an image of a star having as many points as defined in the input value (int to bitmap); this field type processes format strings itself; example: '{value start "size=100 color=red linecolor=blue gradient=1 middlepercent=60"}'
- *imageinfo* – returns text describing the input image (image to text); it returns text like '[Image 250×150]'

Implementation

How to implement

Create a class inherited from TRVReportCustomFieldObject. Override the methods: GetValue, GetFieldType.

How to register

RVReportPluginManager.RegisterFieldObject(<prefix>, <object of your class>);

How to unregister

RVReportPluginManager.UnRegisterFieldObject(<prefix>, <object of your class>);

Standard query processors

The main way for handing data queries in TRVReportGenerator 46 is assigning a data provider component to its DataProvider 34 property. This data provider is responsible for creating query processors to execute data queries.

Additionally, you can create query processors in OnCreateQueryProcessor 36 event.

There is one more way: to implement a "standard" query processor class and register it for some prefix. After that, prefixed queries will be handled by this class of a query processor.

Standard query processor can use any prefix (consisting of English characters, digits and underscores), except for the reserved 'field' prefix.

Example

For example, RVReportDateTimeQueryProcessor unit implements data queries for 'calendar' prefix.

It processes the following types of queries:

1) 'calendar:months' returns 12 months of the year. The result has the following fields:

- *month* – number of month from 1 (int)
- *name* – localized name of the month (text)
- *short_name* – localized shortened name of the month (text)

2) 'calendar:week' returns 7 days of the week (starting from Monday). The result has the following fields:

- *day_of_week* – number of the day of the week from 1 (int)
- *name* – localized name of the day of the week (text)
- *short_name* – localized shortened name of the day of the week (text)

3) 'calendar:days of month *M* of *Y*' returns all days of the month *M* of the year *Y* (where *M* is from 1 to 12). The result has the following fields:

- *day* – number of the day in the month, from 1 (int)
- *day_of_week* – number of this day in the week from 1 (int)
- *week* – number of this day's week in the month, from 1 (int)

Implementation

How to implement

Create a class inherited from TRVReportStandardQueryProcessor. Override the methods: Execute, GetRecordCount, MoveToFirstRecord, MoveToNextRecord: Boolean, MoveBy, IsValidQuery, GetField (two versions), GetFieldType, GetFieldName, GetFieldCount, and some of GetAs*** functions.

How to register

RegisterClass(<your class>);

RVReportPluginManager.RegisterQueryProcessor(<prefix>, <name of your class>);

How to unregister

RVReportPluginManager.UnRegisterQueryProcessor(<prefix>, <name of your class>);

UnRegisterClass(<your class>);

Additional data provider components

The main way for handing data queries in TRVReportGenerator[46] is assigning a data provider component to its DataProvider[34] property. This data provider is responsible for creating query processors to execute data queries.

Report Workshop includes TRVReportDBDataProvider[54] component (a universal DB provider that uses TDataSet-based queries), and a set of providers for specific database components sets.

Programmers can implement additional DB-related data provider components (inherited from TRVReportDBDataProvider[54]), or another data provider components (inherited from TRVReportDataProvider[53]).

Additional data visualizers

Data visualizers[119] display diagrams that visualize values at backgrounds of report table cells[101].

Programmers can implement their own visualizers.

▼ **Implementation**

How to implement

Create a class inherited from TRVReportCustomValueVisualizer[133]. Override the methods: DisplayValue, GetContentSize.

Additional functions for cross-tab reports

Report Workshop provides a set of aggregate functions for cross-tab reports[20], including 'sum', 'min', 'max', 'average', etc.

Programmers can implement additional functions.

▼ **Example**

For example, RVReportMedianCalculator unit implements 'median' function. Input data: int or float. Result: float.

▼ **Implementation**

How to implement

Create a class inherited from TRVReportCustomFunctionCalculator. Override the methods: CalculateFunction, GetFunctionFieldType.

How to register

RVReportPluginManager.RegisterFunctionCalculator(<prefix>, <object of your class>);

How to unregister

RVReportPluginManager.UnRegisterFunctionCalculator(<prefix>, <object of your class>);

2.5 Limitations in old versions of Delphi

Delphi 5

In Delphi 5, format strings [24] for string fields are ignored.

Calendar data provider is not available for Delphi 5.

Delphi 5-2009

In Delphi versions prior to 2010, LineUsesFillColor properties of value visualizers [119] do not make line colors darker than fill colors.

Delphi 5-XE

In Delphi versions prior to XE2, GDI is used for value visualizers [119] instead of GDI+. No gradients, no anti-aliasing, no semitransparent filling of shapes.

2.6 Version history

Changes after version 1

Compatibility issues:

- TRVReportShapeProperties [169] is moved from RVReportValueVisualizer to RVReportShapes unit

New features

User interface is translated to multiple languages.

Functions for ScaleRichView [167] are added.

Data providers

 TRVReportZEOSDSDataProvider [67] – new data provider component for ZEOS library.

Commands

$If command [16] is extended: conditions may be comparison operators.

Shapes

New shape types [162] (the initial release included 7 shapes).

 TRVShape [68] – a new component that draws a shape.

TRVShapeItemInfo [116] – a new document object that draws a shape.

 TrvActionInsertShape [154] – a new action for inserting shape objects in editors. TrvActionInsertProperties can edit properties of shape items.

New property ShapeScaleX [133] for visualizers displaying shapes; it allows inscribing shapes in rectangles instead of squares.

3 Main Components

The main component included in ReportWorkshop:

 TRVReportGenerator [46].

This component takes TRichView control containing a report template, and generates a report.

3.1 TCustomRVReportGenerator

A base class for TRVReportGenerator[46] component.

Unit RVReportGenerator;

Syntax

```
TCustomRVReportGenerator = class (TComponent)
```

Hierarchy

TObject
TPersistent
TComponent

Description

This component is not used directly. Use TRVReportGenerator[46] instead.

Call Execute[36] to process a TRichView component containing a report template, in order to produce a final report.

While processing a report template, the report generator processes data queries by creating query processors. They are created either by DataProvider[34], or in OnCreateQueryProcessor[36] event.
While generating reports, the following events occur:

- OnDataQueryProgress[37] allows to show a progress indicator or to abort the generation;
- OnError[40] allows to process errors;
- OnProcessRecord[45] allows to execute your code before processing each record in results of a query processor.

The report generator has a list of global report variables[35].

See also

- Definitions of Report Workshop terms[10]

3.1.1 Properties

3.1.1.1 TCustomRVReportGenerator.DataProvider

Links the report generator with a data provider.

```
property DataProvider: TRVReportDataProvider[53];
```

A data provider receives data queries (for example, SQL SELECT statements) and create query processors[174] for these queries.

See also:

- Definitions of Report Workshop terms[10]

3.1.1.2 TCustomRVReportGenerator.ErrorColor

Color for error messages.

```
property ErrorColor: TColor;
```

This color is used for inserting error messages in place of erroneous fields.

If **ErrorColor** = *clNone*, error messages have the same color as the original text.

Default value:
clRed

3.1.1.3 TCustomRVReportGenerator.Texts

An object containing text strings used by the report generator.

```
type
  TRVReportGeneratorTexts = class (TPersistent);
property Texts: TRVReportGeneratorTexts;
```

This property can be used to localize a report generator.

Texts property has sub-properties listed in the table below.

Property	Meaning	Default value
ErrorUnknownVariable	this text inserted in a final report in place of a non-existent variable in variable fields [15].	'Unknown variable "%s"'
ErrorInvalidVariableValue	this text is inserted in place of a variable object field [15], if an object associated with this variable has an unsupported type	'Invalid value in variable "%s"'
ErrorUnknownTableGenerationRule	this text is inserted in place of a data field [12], if this data field refers to a non-existent table generation rule [82].	'Unknown rule "%s"'
ErrorUnknownField	this text is inserted in place of a data field [12], if this data field refers to a non-existent field in the query processor results	'Unknown field "%s"'
ErrorInvalidFieldValue	this text is inserted in place of a data field [12], if this data field contains data in an unsupported format	'Invalid or unsupported value in field "%s"'
ErrorUnsupportedFieldType	this text is inserted in place of a data field [12], if this data field has a type unsupported by the report generator	'Type of field "%s" is not supported'
RVF	an alternative name of rvf data field type [23]	'rvf'
TrueText, FalseText	default display values for boolean data fields [12]	'true', 'false'
InvalidFunctionResult	a value to insert as a function result, if this function cannot be calculates (for example, not enough input values, or the function field [20] is inserted in a wrong context)	'–' (n-dash)

3.1.1.4 TCustomRVReportGenerator.Variables

A string list containing global variables [15] for the report generator.

```
property Variables: TStringList;
```

This string list must have items in the form:

<variable name>=<variable value>

Items may have associated objects. They are owned by this string list: it frees these object when clearing or destroying.

While processing a report template, a report generator uses both these global variables, and local variables [82] in report tables.

See also:

- variables in templates [15]
- TRVReportTableItemInfo [80].Variables [82]
- TRVReportGenerationSession [172]

3.1.2 Methods

3.1.2.1 TCustomRVReportGenerator.EscapeDataQuery

Replaces all occurrences of '{' in **S** to '{{'.

```
class function EscapeDataQuery(const S: TRVUnicodeString): TRVUnicodeString;
```

This function takes the **S** parameter, replaces all '{' characters to '{{' in it, and returns the processed string.

After this processing, '{' characters are not treated as starting field code characters.

Use this function to prepare data queries that do not contain field codes. It is especial useful for JSON data queries, for example for MongoDB data provider [59].

See also

- data queries [28]

3.1.2.2 TCustomRVReportGenerator.Execute

Processes a report template contained in **RV** (or **RVData**), and produces a final report.

```
function Execute(RV: TCustomRichView): Boolean;
function Execute(RVData: TCustomRVData): Boolean;
```

The resulting document is not formatted, call **RV**.Format after calling this method.

The method finishes when the report generation is finished. It may take some time for large reports. You can provide feedback in OnDataQueryProgress [36] event.

If the generation is successful, the method returns *True*. There are may be some non-critical errors, though. You can control error handing using OnError [40] event.

3.1.3 Events

3.1.3.1 TCustomRVReportGenerator.OnCreateQueryProcessor

Allows providing a custom query processor for the specific **DataQuery**.

```
type
  TRVCreateQueryProcessorEvent = procedure (Sender: TCustomRVReportGenerator [34];
    const DataQuery: TRVUnicodeString;
    var QueryProcessor: TRVReportQueryProcessor [174]) of object;

property OnCreateQueryProcessor: TRVCreateQueryProcessorEvent;
```

This is an optional event. Normally, query processors are created by a linked [34] data provider component [53]. However, you can use this event:

- if you want to use a non-standard query processor;
- if you want to use different query processors for different data queries.

Input parameters

DataQuery – a data query string (such as SQL SELECT statement). This string is already processed: data fields and variables in it are replaced to their values.

Output parameters

QueryProcessor – a query processor to process **DataQuery**. If you return *nil*, a default processing will be created (a query processor will be requested from a linked data provider component). **QueryProcessor** will be owned by the report generator and will be freed when it finishes processing **DataQuery**, so do not free **QueryProcessor** yourself.

To create **QueryProcessor**, you can use CreateQueryProcessor 54 method of a data provider component.

See also:

- Definitions of Report Workshop terms 10

3.1.3.2 TCustomRVReportGenerator.OnDataQueryProgress

Occurs while processing a data query.

```
type
  TRVReportProgressStep = (rvrpsExecutingQuery, rvrpsApplying, rvrpsFinished);

  TRVDataQueryProgressEvent = procedure (Sender: TCustomRVReportGenerator 34;
    Rule: TRVRowGenerationCustomRule 105; Cell: TRVReportTableCellData 101;
    Step: TRVReportProgressStep; Level, Percent: Integer;
    var Proceed: Boolean) of object;

property OnDataQueryProgress: TRVDataQueryProgressEvent;
```

This event can be used to display a progress of processing a data query and to abort a report generation 36.

Parameters

Rule – a row generation rule of a report table. This parameter is assigned if the component currently processes a data query belonging to this rule, otherwise it is *nil*.

Cell – a cell of a report table.This parameter is assigned if the component currently processes a data query belonging to this cell, otherwise it is *nil*.

Level – a level of nesting of this data query. 1 for first level queries, 2 for queries nested in them, and so on. If you want to display a progress bar to show a progress, it makes sense to show it only for queries having Level = 1.

Step and **Percent** define a stage of progress.

You can assign *False* to **Proceed** to abort the report generation.

Stages

For each query:

1. at first, the event occurs before processing a data query (for example, before processing SQL SELECT statement) with **Step** = *rvrpsExecutingQuery*.
2. next, the event occurs multiple times while applying query results to table rows/a cell, with **Step** = *rvrpsApplying*, and **Percent** growing from 0 to 100. If the report generator cannot calculate percent (it happens when the total number of records is unknown), it calls the event with **Percent** = -1.
3. finally, the event occurs with **Step** = *rvrpsFinished*.

Aborting the report generation

A report generation is performed in the context of the main process. It means that an application is frozen until a report generation is finished.

You can unfreeze it by calling Application.ProcessMessages in this event, but be careful: this procedure is dangerous when used incorrectly!

You must:

- hide the TRichView control processed by this report generator; there must not be attempts to redraw it while the report is not finished and this TRichView is formatted;
- prevent the TRichView, TRVReportGenerator components (and all other components necessary for report generation) from destroying in this event; usually, it can be done by disallowing closing the form containing these components (by processing its OnCloseQuery event)
- prevent the report generation to start again while the current report generation is finished; usually, it can be done by disabling all controls on the form, may be except for an "Abort" button.

You can implement an "Abort" button. When the user clicks it, assign some form's boolean variable. In **OnDataQueryProgress**, check this variable and assign **Proceed** = *False* if necessary.

Example

Showing progress in ProgressBar1 and Label1.

```
procedure TForm1.RVReportGenerator1DataQueryProgress(
  Sender: TCustomRVReportGenerator[34];
  Rule: TRVRowGenerationRule[111]; Cell: TRVReportTableCellData[101];
  Step: TRVReportProgressStep; Level, Percent: Integer;
  var Proceed: Boolean);

  function GetRuleName: String;
  begin
    Result := '';
    if Rule<>nil then
      Result := Rule.Name;
    if Result='' then
      Result := '<untitled>';
  end;

begin
  if Level<>1 then
    exit;
  case Step of
    rvrpsExecutingQuery:
      begin
        ProgressBar1.State := pbsPaused;
        ProgressBar1.Visible := True;
```

```
        Label1.Visible := True;
        Label1.Caption := 'Executing query '+GetRuleName;
        Label1.Repaint;
        ProgressBar1.Repaint;
      end;
    rvrpsApplying:
      begin
        if ProgressBar1.State <> pbsNormal then
        begin
          Label1.Caption := 'Applying query '+GetRuleName;
          ProgressBar1.State := pbsNormal;
          Label1.Repaint;
          ProgressBar1.Repaint;
        end;
        ProgressBar1.Position := Percent;
      end;
    rvrpsFinished:
      begin
        ProgressBar1.Visible := False;
        Label1.Visible := False;
      end;
  end;
end;
```

3.1.3.3 TCustomRVReportGenerator.OnError

Occurs if an error happens while generating a report.

```
type
  TRVReportGeneratorError = (
    // rules
    rvrgeRuleInvalidRange, rvrgeRuleOverlaps,
    // queries
    rvrgeNoQueryProcessor, rvrgeQueryExecution, rvrgeRecordCountIsRequired,
    // variables
    rvrgeVariableUnknown, rvrgeVariableInvalidValue, rvrgeVariableInvalidName,
    rvrgeVariableUnsupportedObject,
    // data fields
    rvrgeFieldUnknown, rvrgeFieldInvalidValue, rvrgeFieldUnsupportedType,
    rvrgeFieldUnknownType, rvrgeFieldUnknownRule,
    // fields
    rvrgeInvalidTypeCast, rvrgeInvalidFormatString, rvrgeCustomFieldTypeError,
    // commands
    rvrgeCommandUnknown, rvrgeCommandNeedsParam,
    rvrgeCommandDoesNotNeedParam, rvrgeCommandWrongParamValue,
    rvrgeCommandSyntaxError, rvrgeCommandUnmatched,
    rvrgeCommandMustBeInQuery, rvrgeCommandWrongContext,
    // cross-tab
    rvrgeCrossTabInvalid, rvrgeCrossTabInvalidPosition,
    rvrgeCrossTabNoRules,
    rvrgeCrossTabUnknownField,
    rvrgeCrossTabWrongContext, rvrgeCrossTabRecordDoesNotMatch,
    rvrgeCrossTabDuplicateRecords,
    rvrgeCrossTabKeyFieldInCrossTabColumn,
    rvrgeCrossTabNoCrossTabColumns, rvrgeCrossTabTooManyColumns,
    rvrgeCrossTabTooManyTotalColumns,
    // functions
    rvrgeFuncParenthesisExpected, rvrgeFuncUnknown, rvrgeFuncBadParam,
    rvrgeFuncMustBeInCrossTabSummary,
    // others
    rvrgeInternalError
    );

  TRVReportGeneratorErrorEvent = procedure (Sender: TCustomRVReportGenerator[34];
    ErrorCode: TRVReportGeneratorError; const RelatedText: TRVUnicodeString;
    RelatedObject: TObject; var Proceed: Boolean) of object;

property OnError: TRVReportGeneratorErrorEvent;
```

This event can be processed:

- to display an error message,
- to abort a report generation on some (or all) errors.

Parameters

ErrorCode – a code identifying the error.

RelatedText, RelatedObject provide additional information about this error.

You can assign *False* to **Proceed** to abort the report generation.

Errors while checking table row generation rules for correctness

For this kind of errors, **RelatedObject** is TRVRowGenerationCustomRule[105].

Error	Meaning	RelatedText
rvrgeRuleInvalidRange	The rule has an incorrect range of rows (too small or too large row indexes, or these rows overlap other rows because their cells are merged vertically).	Empty
rvrgeRuleOverlaps	The rule has a range of rows intersecting with one of the previous rules in the same table	Empty

Errors while executing a data query (such as SQL SELECT statement)

For this kind of errors, **RelatedObject** is either TRVRowGenerationCustomRule[105] or TRVReportTableCellData[101].

Error	Meaning	RelatedText
rvrgeNoQueryProcessor	The report generator was not able to create a query processor for this data query.	Data query string
rvrgeQueryExecution	An error occurred while executing a data query	Data query string
rvrgeRecordCountIsRequired	This type of report requires a known record count before retrieving data, but the query processor cannot provide it. In the current version, a record count is needed: for column copying in row generation rules, for cross-tab reports	Empty

Errors while processing variables[15]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeVariableUnknown	A field refers to a non-existent variable	Variable name
rvrgeVariableInvalidValue	An improper value of a variable. For example, a variable in "If" command[16] cannot be evaluated as *True* or *False*.	Field code
rvrgeVariableInvalidName	An incorrect (for example, empty) variable name	Variable name
rvrgeVariableUnsupportedObject	A field needs insertion of an object associated with a variable, and this object has a class unsupported by the report generator	Variable name

Errors while processing data fields[12]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeFieldUnknown	A field refers to a non-existent data field	Data field name
rvrgeFieldInvalidValue	An improper value of a a data field. For example, a data field in "If" command[16] cannot be	Field code or data field name

	evaluated as *True* or *False*, or a format of content of a Blob field cannot be detected	
rvrgeFieldUnsupportedType	A data field has an unsupported type	Data field name
rvrgeFieldUnknownRule	A field refers to a non-existent row generation rule	Rule name

Common errors while processing data fields[12], cross-tab header fields,[19] functions[20]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeFieldUnknownType	Unknown value in <field type>[23]	Field code
rvrgeInvalidTypeCast	The value cannot be converted to <field type>[23]	Field code
rvrgeInvalidFormatString	Error in <format string>[24]	Field code
rvrgeCustomFieldTypeError	An error was reported by a custom field type[30] handler. The reason may be because the field data are inappropriate, or a format string is erroneous.	Field code

Errors while processing commands[16]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeCommandUnknown	Unknown command	Field code
rvrgeCommandNeedsParam	This command requires parameter(s)	Field code
rvrgeCommandDoesNotNeedParam	This command requires no parameters	Field code
rvrgeCommandWrongParamValue	A parameter value for this command is incorrect	Field code
rvrgeCommandSyntaxError	A syntax error in the command code	Field code
rvrgeCommandUnmatched	Unmatched commands (for example, "If" without "EndIf")	Field code
rvrgeCommandMustBeInQuery	This command must be inserted in a text affected by a data query	Field code
rvrgeCommandWrongContext	This command is not valid in this place of document (for example, "ListReset" command must be in a numbered paragraph)	Field code

Errors while making a cross-tab report[84]

Error	Meaning	RelatedText	RelatedObject

rvrgeCrossTabInvalid	Table.CrossTabulation [81] is incorrect. Possible reasons: • empty Table.CrossTabulation [81] .Levels [91] [].FieldName [97]; • empty Table.CrossTabulation.Levels[].DataQuery [95], if they are required for the given ColumnGenerationType [88]; • invalid values (or a combination of values) for MinValue, MaxValue, Step [98], if ColumnGenerationType [88] = *rvcgtRange*	Empty	TRVReportTableItemInfo [80]
rvrgeCrossTabInvalidPosition	Position or structure of a cross-tab header [86] is incorrect.	Empty	TRVReportTableItemInfo [80]
rvrgeCrossTabNoRules	A cross tabulation is defined for a table, but this table has no row generation rules [82], or the first rule is invalid	Empty	TRVReportTableItemInfo [80]
rvrgeCrossTabUnknownField	Table.CrossTabulation [81].Levels [91] [].FieldName [97] or CaptionFieldName [93] does not exists in the results of the respective data query (if ColumnGenerationType [88] <> *rvcgtRange*)	Field name	TRVCrossTabLevel [92]
	A cross-tab header field [19] refers to a non-existent data field	Field name	TRVReportGenerationSession [172]
rvrgeCrossTabWrongContext	A cross-tab header field [19] is found outside a cross-tab header cells	Field code	TRVReportGenerationSession [172]
rvrgeCrossTabRecordDoesNotMatch	A record in the result of a row generation rule's DataQuery [105] contains a combination of values of column fields [97] not corresponding to any cross-tab data column	Empty	TRVReportGenerationSession [172]
rvrgeCrossTabDuplicateRecords	A record in the result of a row generation rule's DataQuery [105] contains the same combination of values of key fields [115] and column fields [97] as one of previous records.	Empty	TRVReportGenerationSession [172]
rvrgeCrossTabKeyFieldInCrossTabColumn	One ore more key fields [115] is included in column fields [97]	Field name	TRVRowGenerationRule [111]
rvrgeCrossTabNoCrossTabColumns	A cross tabulation contains 0 data columns	Empty	TRVReportTableItemInfo [80] or TRVCrossTabLevel [92], depending on ColumnGenerationType [88]

rvrgeCrossTabTooManyColumns	A group of columns in a cross-tab level exceeds level.MaxColCount[98]. In this case, only the first MaxColCount[98] columns are used.	Empty	TRVCrossTabLevel[92]
rvrgeCrossTabTooManyTotalColumns	A total number of column repetitions exceeds MaxColCount[92]	Empty	TRVReportTableItemInfo[80]

Errors while processing functions[20]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeFuncParenthesisExpected	An opening or closing bracket around the function parameter is missing	Field code
rvrgeFuncUnknown	Unknown function name	Field code
rvrgeFuncBadParam	A function parameter is not a valid value field name	Field code
rvrgeFuncMustBeInCrossTabSummary	A function field is inserted not in a proper place	Field code

Other errors

Error	Meaning	RelatedText	RelatedObject
rvrgeInternalError	An exception occurred while generating a report	Exception message	Exception

Example

```
const ErrorMsg: array[TRVReportGeneratorError] of TRVUnicodeString =
  ('Rule error: invalid row range', 'Rule error: overlapping row range',
   'Query processing error: cann''t create query processor for %s',
   'Query execution error for %s', 'Variable error: unknown variable %s',
   'Variable error: invalid value for %s',
   'Variable error: invalid name %s',
   'Variable error: unsupported object in %s',
   'Field error: unknown field %s', 'Field error: invalid value for %s',
   'Field error: unsupported field type for %s', 'Field error: unknown rule %s',
   'Command error: unknown command %s', 'Command error: parameters are required in %s',
   'Command error: this command must not have parameters: %s',
   'Command error: wrong parameter value: %s',
   'Command error: syntax error %s', 'Command error: unmatched command %s',
   'Command error: a command must be in content affected by a query: %s',
   'Command error: the command is not valid in its context: %s');

procedure TForm1.RVReportGenerator1Error(Sender: TCustomRVReportGenerator[34];
  ErrorCode: TRVReportGeneratorError; const RelatedText: TRVUnicodeString;
  RelatedObject: TObject; var Proceed: Boolean);
var S: TRVUnicodeString;
begin
  S := Format(ErrorMsg[ErrorCode], [RelatedText]);
  Memo1.Lines.Add(S);
end;
```

3.1.3.4 TCustomRVReportGenerator.OnProcessRecord

Occurs before processing a record of a query processor.

```
type
  TRVProcessRecordEvent = procedure (Sender: TCustomRVReportGenerator[34];
    const DataQuery: TRVUnicodeString; Rule: TRVRowGenerationRule[111];
    Cell: TRVReportTableCellData[101];
    Session: TRVReportGenerationSession[172]; RecNo: Integer;
    QueryProcessor: TRVReportQueryProcessor[174]) of object;

property OnProcessRecord: TRVProcessRecordEvent;
```

This event may be used to assign values to variables based on the data from this record.

Parameters

DataQuery – a data query string (such as SQL SELECT statement). This string is already processed: data fields and variables in it are replaced to their values.

Rule – a row generation rule of a report table. This parameter is assigned if the **DataQuery** belongs to this rule, otherwise, it is *nil*.

Cell – a cell of a report table.This parameter is assigned if the **DataQuery** belongs to this cell, otherwise, it is *nil*.

Session – an object representing a report generation session. It can be used to get values of variables[15] and data fields[12].

RecNo – index of the record in the results of **QueryProcessor**, in the range 0..**QueryProcessor**.GetRecordCount-1.

QueryProcessor – a query processor created to process **DataQuery**.

See also

- Definitions of Report Workshop terms 10

3.2 TRVReportGenerator

A report generator component.

Unit RVReportGenerator;

Syntax

```
TRVReportGenerator = class (TCustomRVReportGenerator 34)
```

Hierarchy

TObject
TPersistent
TComponent
TCustomRVReportGenerator 34

Description

Call Execute 36 to processes a TRichView component containing a report template, in order to produce a final report.

While processing a report template, the report generator processes data queries by creating query processors. They are created either by DataProvider 34, or in OnCreateQueryProcessor 36 event.
While generating reports, the following events occur:

- OnDataQueryProgress 37 allows to show a progress indicator or to abort the generation;
- OnError 40 allows to process errors;
- OnProcessRecord 45 allows to execute your code before processing each record in results of a query processor.

The report generator has a list of global report variables 35.

See also

- Definitions of Report Workshop terms 10

4 Data Provider Components

Report Workshop includes the following data provider components working with TRVReportGenerator 46.

TRVReportDBDataProvider 54

Access to: any data stores available via TDataSet-based components

Supported types of data queries: names of TDataSet-based components

TRVReportAbsDataProvider 49

Access to: Absolute database

Based on components: Absolute database components by ***ComponentAce*** (TABSQuery and TABSTable)

Supported types of data queries: SQL or table name

TRVReportADODataProvider 50

Access to: multiple data stores accessed via ADO (ActiveX Data Objects)

Based on components: dbGo components by Borland/CodeGear/Embarcadero (TADOQuery and TADOTable)

Supported types of data queries: SQL or table name



TRVReportBDEDataProvider 50

Access to: multiple databases accessed via BDE (Borland Database Engine), including Paradox, dBASE, FoxPro, Access, databases via ODBC, and others

Based on components: BDE components by Borland/CodeGear/Embarcadero (TQuery and TTable), available in Delphi and C++Builder prior to XE8.

Supported types of data queries: SQL or table name



TRVReportDBISAMDataProvider 56

Access to: DBISAM databases

Based on components: DBISAM components by ***Elevate Software, Inc.*** (TDBISAMQuery and TDBISAMTable)

Supported types of data queries: SQL or table name



TRVReportDBXDataProvider 57

Access to: multiple databases via dbExpress drivers, including Oracle, Firebird, InterBase, DB2, Informix, SQL Server, MySQL and ODBC.

Based on components: dbExpress components by Borland/CodeGear/Embarcadero (TSQLQuery and TSQLTable)

Supported types of data queries: SQL or table name



TRVReportEDBDataProvider 58

Access to: ElevateDB databases

Based on components: ElevateDB components by ***Elevate Software, Inc.*** (TEDBQuery and TEDBTable)

Supported types of data queries: SQL or table name



TRVReportFDDataProvider 58

Access to: multiple databases via FireDAC

Based on components: FireDAC components by Embarcadero (TFDQuery and TFDTable), available since RAD Studio XE3

Supported types of data queries: SQL or table name



TRVReportFDMongoDataProvider 59

Access to: ***MogoDB*** databases

Based on components: FireDAC components by Embarcadero (TFDMongoQuery), avaliable since RAD Studio 10 Seattle.

Supported types of data queries: JSON



TRVReportIBCDataProvider 60

Access to: ***InterBase*** and ***Firebird*** databases

Based on components: IBDAC components by ***Devart*** (TIBCQuery and TIBCTable)

Supported types of data queries: SQL or table name

TRVReportIBDataProvider 61

Access to: *InterBase* and *Firebird* databases

Based on components: InterBase Express components by Borland/CodeGear/Embarcadero (TIBQuery and TIBTable)

Supported types of data queries: SQL or table name

TRVReportIBODataProvider 62

Access to: *InterBase* and *Firebird* databases

Based on components: IB Objects components by *Jason Wharton* (TIBOQuery and TIBOTable)

Supported types of data queries: SQL or table name

TRVReportMyDataProvider 62

Access to: *MySQL* databases

Based on components: MyDAC components by *Devart* (TMyQuery and TMyTable)

Supported types of data queries: SQL or table name

TRVReportMySQLDataProvider 63

Access to: *MySQL* databases

Based on components: DAC for MySQL components by *MicroOLAP Technologies LTD* (TMySQLQuery and TMySQLTable)

Supported types of data queries: SQL or table name

TRVReportNxDataProvider 64

Access to: NexusDB databases

Based on components: NexusDB components by *NexusQA Pty Ltd.* (TNxQuery and TNxTable)

Supported types of data queries: SQL or table name

TRVReportPgDataProvider 65

Access to: *PostgreSQL* databases

Based on components: PgDAC components by *Devart* (TPgQuery and TPgTable)

Supported types of data queries: SQL or table name

TRVReportPSQLDataProvider 65

Access to: *PostgreSQL* databases

Based on components: PostgresDAC components by *MicroOLAP Technologies LTD* (TPSQLQuery and TPSQLTable)

Supported types of data queries: SQL or table name

TRVReportUniDataProvider 66

Access to: multiple databases via UniDAC, including Oracle, Microsoft SQL Server, MySQL, InterBase, Firebird, PostgreSQL, SQLite, DB2, Microsoft Access, Sybase Advantage Database Server, Sybase Adaptive Server Enterprise, and other databases (using ODBC provider).

Based on components: UniDAC components by ***Devart*** (TUniQuery and TUniTable)

Supported types of data queries: SQL or table name

TRVReportZEOSDSDataProvider 67

Access to: multiple databases via ZeosLib, including MySQL, PostgreSQL, Interbase, Firebird, MS SQL, Sybase, Oracle and SQLite.

Based on components: ZeosLib components (***Download***, ***Forum***)

Supported types of data queries: SQL or table name

4.1 TRVReportAbsDataProvider

A data provider receiving data from ***Absolute databases***.

Unit RVReportAbsDataProvider;

Syntax

```
TRVReportAbsDataProvider = class (TRVReportCustomDBDataProvider 51)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 53
TRVReportCustomDBDataProvider 51

Description

This component provides data for a report generator component from ***Absolute databases***.

This component requires Absolute Database components by ***ComponentAce***.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider 34 property of TRVReportGenerator 46.

TRVReportAbsDataProvider introduces properties:

```
property DatabaseName: String;
property SessionName: String;
```

Also: InMemory property.

Internally, the component uses temporal TABSQuery and TABSTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TABSQuery or TABSTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

4.2 TRVReportADODataProvider

A data provider receiving data from ADO (ActiveX Data Objects) data stores.

Unit RVReportAdoDataProvider;

Syntax

```
TRVReportADODataProvider = class (TRVReportCustomDBDataProvider 51)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 53
TRVReportCustomDBDataProvider 51

Description

This component provides data for a report generator component from ADO data stores.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider 34 property of TRVReportGenerator 46.

TRVReportADODataProvider introduces properties:

```
property Connection: TADOConnection;
property ConnectionString: WideString;
```

Also: CacheSize, CursorLocation, CursorType, LockType, MaxRecords properties.

Internally, the component uses temporal dbGo (TADOQuery and TADOTable) components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TADOQuery or TADOTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

4.3 TRVReportBDEDataProvider

A data provider receiving data from BDE databases.

Unit RVReportBDEDataProvider;

Syntax

```
TRVReportBDEDataProvider = class (TRVReportCustomDBDataProvider 51)
```

Hierarchy

TObject
TPersistent

TComponent
TRVReportDataProvider 53
TRVReportCustomDBDataProvider 51

Description

This component provides data for a report generator component from databases accessed via BDE (Borland Database Engine), including Paradox, dBASE, FoxPro, Access, databases via ODBC, and others. BDE components are included in Delphi and C++Builder till version XE6 (inclusive).

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider 34 property of TRVReportGenerator 46.

TRVReportBDEDataProvider introduces properties:

```
property DatabaseName: String;
property SessionName: String;
```

Also: ObjectView property.

Internally, the component uses temporal TQuery and TTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TQuery or TTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

4.4 TRVReportCustomDBDataProvider

A base class of a database-related data provider for report generators.

Unit RVReportDBDataProvider;

Syntax

```
TRVReportDBDataProvider = class (TRVReportDataProvider 53)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 53

Description

This is a parent class for data providers components which uses TDataSet to process data queries. A typical data query is SQL SELECT statement.

A data provider receives data query strings from TRVReportGenerator 46 components, and creates TRVReportDBQueryProcessor 172 objects to handle them. Normally, this process is invisible for you, TRVReportDBQueryProcessor 172 is used internally.

See also

- Definitions of Report Workshop terms 10

4.4.1 Properties

4.4.1.1 TRVReportCustomDBDataProvider.UseRecordCount

Specifies whether the report generator can use RecordCount property of dataset.

```
property UseRecordCount: TRVReportRecordCountMode[161];
```

This property specifies whether the report generator can use RecordCount properties of datasets.

This property is used for datasets created by data providers automatically.

This property is not used for TRVReportDBDataProvider[54].DataSets[55], it has UseRecordCount[56] property in each item.

This property is used as an initial value of UseRecordCount parameter in OnCreateDataSet[52] event.

4.4.2 Events

4.4.2.1 TRVReportCustomDBDataProvider.OnCreateDataSet

Allows providing a dataset for the specified **DataQuery**.

```
type
  TRVCreateDataSetEvent = procedure (Sender: TRVReportCustomDBDataProvider[51];
    const DataQuery: TRVUnicodeString; var DataSet: TDataSet;
      var DestroyAfterUse: Boolean;
      var AUseRecordCount: TRVReportRecordCountMode[161]) of object;

property OnCreateDataSet: TRVCreateDataSetEvent;
```

This event is optional, because they data providers can create datasets themselves.

Input parameters

DataQuery – a data query string (such as SQL SELECT statement). This string is already processed: data fields and variables in it are replaced to their values.

Output parameters

DataSet – a dataset object to process **DataQuery**.

DestroyAfterUse defines whether the returned **DataSet** will be owned by the report generator component. If **DestroyAfterUse** = *True*, the report generator will free **DataSet** when it completes processing this **DataQuery**. Initival value of this parameter is *True*.

AUseRecordCount defines whether **DataSet** can return the correct total number of records, and supports First, Next and Last commands. Initial value of this parameter is equal to UseRecordCount[52].

See also:

- Definitions of Report Workshop terms[10]

4.4.2.2 TRVReportCustomDBDataProvider.OnDataSetCreated

Occurs after **DataSet** is created and initialized by a data provider.

```
type
  TRVDataSetCreatedEvent = procedure (Sender: TRVReportDBDataProvider[54];
    DataSet: TDataSet) of object;

property OnCreateDataSet: TRVCreateDataSetEvent;
```

This event allows assigning values to properties of **DataSet**.

Parameters

DataSet – a dataset object.

See also

- query events 53

4.4.2.3 TRVReportCustomDBDataProvider's query events

These events are assigned to created data sets.

```
property BeforeQueryOpen: TDataSetNotifyEvent;
property AfterQueryOpen: TDataSetNotifyEvent;
property BeforeQueryClose: TDataSetNotifyEvent;
property AfterQueryClose: TDataSetNotifyEvent;
property BeforeQueryScroll: TDataSetNotifyEvent;
property AfterQueryScroll: TDataSetNotifyEvent;
```

These values are assigned to events of created datasets: BeforeOpen, AfterOpen, BeforeClose, AfterClose, BeforeScroll, AfterScroll.

They are assigned only to datasets which are owned by the data provider. For example:

- they are not assigned to datasets returned from TRVReportDBDataProvider 54.DataSets 55
- they are assigned to datasets returned in OnCreateDataSet 52 event, only if DestroyAfterUse = *True*
- they are assigned to datasets created by data providers automatically

4.5 TRVReportDataProvider

A base class of a data provider for report generators.

Unit RVReportDataProvider;

Syntax

```
TRVReportDataProvider = class (TComponent)
```

Hierarchy

TObject
TPersistent
TComponent

Description

This component is not used directly. Use the components inherited from it: TRVReportDBDataProvider 54 and others.

Data providers are linked 34 to TRVReportGenerator 46 components. They create query processors for data queries.

Data provider receives data query strings from TRVReportGenerator 46 components, and creates TRVReportQueryProcessor 174 objects to handle them.

Inherited classes:

- TRVReportCustomDBDataProvider 51

See also

- Definitions of Report Workshop terms 10

4.5.1 Methods

4.5.1.1 TRVReportDataProvider.CreateQueryProcessor

The method creates a query processor for processing **DataQuery**.

```
function CreateQueryProcessor(const DataQuery: TRVUnicodeString):
  TRVReportQueryProcessor[174]; virtual; abstract;
```

Normally, you do not need to call this method. Just assign this data provider component to TRVReportGenerator[46]'s DataProvider[34] property, and it will create query processors automatically.

You can use this method in TRVReportGenerator[46].OnCreateQueryProcessor[36] event, if you want to implement processing data queries using multiple data provider components.

This is an abstract method in TRVReportDataProvider. It is implemented in inherited classes.

If **QueryProcessor** cannot be created, the method returns *nil*.

See also:

- Definitions of Report Workshop terms[10]

4.6 TRVReportDBDataProvider

An universal database-related data provider for report generators.

Unit RVReportDBDataProvider;

Syntax

```
TRVReportDBDataProvider = class (TRVReportDataProvider[53])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

Use this component to provide data from TDataSet-based components, if Report Workshop does not have special data providers for them.

This component offers two ways to provide datasets:

1. DataSets[55] collection allows using existing TDataSet components. A typical data query: a single word identifying a dataset.

2. OnCreateDataSet[52] event, where you can either create a new TDataSet component or return a link to existing TDataSet component. A typical data query: SQL SELECT statement, or a table name.

A data provider receives data query strings from TRVReportGenerator[46] components, and creates TRVReportDBQueryProcessor[172] objects to handle them. Normally, this process is invisible for you, TRVReportDBQueryProcessor[172] is used internally. You only need to provide a dataset object.

See also

- Definitions of Report Workshop terms[10]

4.6.1 Properties

4.6.1.1 TRVReportDBDataProvider.DataSets

Allows using existing datasets as sources of data for reports.

```
property DataSets: TRVDataSetCollection[55];
```

This is a collection of pairs (Name, DataSet). If data query[28] is equal to Name, the data provider creates a query processor using the corresponding DataSet.

Unlike DataSets created dynamically, these datasets are not freed when report generation is finished.

See also:

- Definitions of Report Workshop terms[10]

4.6.2 Classes of properties

4.6.2.1 TRVDataSetCollection

TRVDataSetCollection is a collection of datasets for use in report generator.

Unit RVReportTable;

Syntax

```
TRVDataSetCollection = class (TCollection);
```

Hierarchy

TObject
TPersistent
TCollection

Description

TRVDataSetCollection is a class of TRVReportDBDataProvider[54].DataSets[55] property. It is a collection of TRVDataSetItem[55] items.

Each item in this collection is a pair of (Name, DataSet).

4.6.2.1.1 Properties

4.6.2.1.1.1 TRVDataSetCollection.Items

Lists the items in the collection.

```
property Items[Index: Integer]: TRVDataSetItem[55]; default;
```

Use **Items** to access individual items in the collection.

4.6.2.2 TRVDataSetItem

TRVDataSetItem is an item in TRVDataSetCollection[55] collection.

Unit RVReportTable;

Syntax

```
TRVDataSetItem = class (TCollectionItem)
```

Hierarchy

TObject

TPersistent
TCollectionItem

Description

An object of this class allows using existing dataset DataSet[56] to provide data for the report generator. It is referred by Name[56].

4.6.2.2.1 Properties

4.6.2.2.1.1 TRVDataSetItem.DataSet

Specifies a dataset reference for TRVReportDBDataProvider[54].

```
property DataSet: TDataSet;
```

The dataset may be any TDataSet descendant. It can be referred in report templates by Name[56].

4.6.2.2.1.2 TRVDataSetItem.Name

Specifies a dataset name.

```
property Name: TRVUnicodeString;
```

If a report template contains a data query[28] equal to **Name**, data from DataSet[56] are used.

Each item in TRVDataSetCollection[55] must have an unique **Name**.

4.6.2.2.1.3 TRVDataSetItem.UseRecordCount

Specifies whether the report generator can use RecordCount property of dataset.

```
property UseRecordCount: TRVReportRecordCountMode[161];
```

This property specifies whether the report generator can use DataSet[56].RecordCount to get the total number of records.

Value of this property is used for datasets that are created automatically by data providers.

If a dataset is created in OnCreateDataSet[52] event,

4.7 TRVReportDBISAMDataProvider

A data provider receiving data from DBISAM databases.

Unit RVReportDBISAMDataProvider;

Syntax

```
TRVReportDBISAMDataProvider = class (TRVReportCustomDBDataProvider[51])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

This component provides data for a report generator component from ***DBISAM*** databases.

This component requires DBISAM components by ***Elevate Software, Inc.***

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider[34] property of TRVReportGenerator[46].

TRVReportDBISAMDataProvider introduces properties:

```
property DatabaseName: String;
property SessionName: String;
```

Also: MaxRowCount property.

Internally, the component uses temporal TDBISAMQuery and TDBISAMTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated[52] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[52] event. This DataSet is not necessary needed to be TDBISAMQuery or TDBISAMTable.

See also

- Definitions of Report Workshop terms[10]
- Data queries[28]

4.8 TRVReportDBXDataProvider

A data provider receiving data from databases via dbExpress drivers, including Oracle, Firebird, InterBase, DB2, Informix, SQL Server, MySQL and ODBC.

Unit RVReportDBXDataProvider;

Syntax

```
TRVReportDBXDataProvider = class (TRVReportCustomDBDataProvider[51])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

This component provides data for a report generator component from databases via dbExpress drivers, including Oracle, Firebird, InterBase, DB2, Informix, SQL Server, MySQL and ODBC.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider[34] property of TRVReportGenerator[46].

TRVReportDBXDataProvider introduces properties:

```
property SQLConnection: TSQLConnection;
```

Also: SchemaName, ObjectView.

Internally, the component uses temporal TSQLQuery components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated[52] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[52] event. This DataSet is not necessary needed to be TSQLQuery.

See also

- Definitions of Report Workshop terms[10]
- Data queries[28]

4.9 TRVReportEDBDataProvider

A data provider receiving data from ElevateDB databases.

Unit RVReportEDBDataProvider;

Syntax

```
TRVReportEDBDataProvider = class (TRVReportCustomDBDataProvider[51])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

This component provides data for a report generator component from ***ElevateDB*** databases.

This component requires ElevateDB components by ***Elevate Software, Inc.***

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider[34] property of TRVReportGenerator[46].

TRVReportEDBDataProvider introduces properties:

```
property DatabaseName: String;
property SessionName: String;
```

Internally, the component uses temporal TEDBQuery and TEDBTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated[52] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[52] event. This DataSet is not necessary needed to be TEDBQuery or TEDBTable.

See also

- Definitions of Report Workshop terms[10]
- Data queries[28]

4.10 TRVReportFDDataProvider

A data provider receiving data from various databases via FireDAC.

Unit RVReportFDDataProvider;

Syntax

```
TRVReportFDDataProvider = class (TRVReportCustomFDDataProvider)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 53
TRVReportCustomDBDataProvider 51
TRVReportCustomFDDataProvider

Description

This component provides data for a report generator component from various databases via FireDAC. FireDAC components are included in RAD Studio since XE3 version.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider 34 property of TRVReportGenerator 46.

TRVReportFDDataProvider introduces properties:

```
property Connection: TFDCustomConnection;
property ConnectionName: String;
```

Also: FieldOptions, LocalSQL properties.

Internally, the component uses temporal TFDQuery and TFDTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TFDQuery or TFDTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

4.11 TRVReportFDMongoDataProvider

A data provider receiving data from MongoDB databases via FireDAC.

Unit RVReportFDMongoDataProvider;

Syntax

```
TRVReportFDMongoDataProvider = class (TRVReportCustomFDMongoDataProvider)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 53
TRVReportCustomDBDataProvider 51
TRVReportCustomFDMongoDataProvider

Description

This component provides data for a report generator component from ***MogoDB*** databases via FireDAC. FireDAC for MongoDB components are included in RAD Studio since 10 Seattle version.

This data provider handles the following data queries:

- JSON queries containing $match, $sort, $limit, $project, $skip, $database, $collection items
- JSON queries containing only $match part

Examples of data queries:

```
// complete query syntax
RVReportGenerator1.EscapeDataQuery(
  '[{"$match": {"cuisine": "Italian", "address.zipcode": "10075"}},{"$limit":"10"}]')
// $match query syntax
RVReportGenerator1.EscapeDataQuery(
  '{"cuisine": "Italian", "address.zipcode": "10075"}');
// returning the whole collection:
RVReportGenerator1.EscapeDataQuery('{}')
```

To use this component, assign it to DataProvider property of TRVReportGenerator.

TRVReportFDMongoDataProvider introduces properties:

```
property Connection: TFDCustomConnection;
property DatabaseName: String;
property CollectionName: String;
```

Also: FieldOptions, ObjectView, LocalSQL property.

Database and Collection property specify the default values. They can be overridden by $database and $collection items in a data query.

Internally, the component uses temporal TFDMongoQuery components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet event. This DataSet is not necessary needed to be TFDMongoQuery.

See also

- Definitions of Report Workshop terms
- Data queries

4.12 TRVReportIBCDataProvider

A data provider receiving data from InterBase and Firebird databases via IBDAC.

Unit RVReportIBCDataProvider;

Syntax

```
TRVReportIBCDataProvider = class (TRVReportCustomDBDataProvider)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider
TRVReportCustomDBDataProvider

Description

This component provides data for a report generator component from ***InterBase*** and ***Firebird*** databases via IBDAC components by ***Devart***.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider property of TRVReportGenerator.

TRVReportIBCDataProvider introduces properties:

```
property Connection: TIBCConnection;
```

Also: Transaction, Options, FetchRows, FetchAll, DataTypeMap, Encryption, SmartFetch properties, AfterQueryExecute, AfterQueryFetch events.

Internally, the component uses temporal TIBCQuery and TIBCTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TIBCQuery or TIBCTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

4.13 TRVReportIBDataProvider

A data provider receiving data from InterBase and Firebird databases via InterBase Express components.

Unit RVReportIBDataProvider;

Syntax

```
TRVReportIBDataProvider = class (TRVReportCustomDBDataProvider 51)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 53
TRVReportCustomDBDataProvider 51

Description

This component provides data for a report generator component from ***InterBase*** and ***Firebird*** databases via Interbase Express components (included in Delphi and C++Builder).

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider 34 property of TRVReportGenerator 46.

TRVReportIBDataProvider introduces properties:

```
property Database: TIBDatabase;
```

Also: Transaction, FieldOptions, ObjectView.

Internally, the component uses temporal TIBQuery and TIBTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TIBQuery or TIBTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

4.14 TRVReportIBODataProvider

A data provider receiving data from InterBase and Firebird databases via IB Objects components.

Unit RVReportIBODataProvider;

Syntax

```
TRVReportIBODataProvider = class (TRVReportCustomDBDataProvider[51])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

This component provides data for a report generator component from ***InterBase*** and ***Firebird*** databases via IB Objects components by ***Jason Wharton***.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider[34] property of TRVReportGenerator[46].

TRVReportIBODataProvider introduces properties:

```
property IB_Connection: TIB_Connection;
property DatabaseName: String;
property SessionName: String;
```

Also: MaxRows, AutoFetchAll, FetchWholeRows, ColumnAttributes, FieldOptions properties.

Internally, the component uses temporal TIBOQuery and TIBOTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated[52] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[52] event. This DataSet is not necessary needed to be TIBOQuery or TIBOTable.

See also

- Definitions of Report Workshop terms[10]
- Data queries[28]

4.15 TRVReportMyDataProvider

A data provider receiving data from MySQL databases via MyDAC.

Unit RVReportMyDataProvider;

Syntax

```
TRVReportMyDataProvider = class (TRVReportCustomDBDataProvider[51])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

This component provides data for a report generator component from ***MySQL*** databases via MyDAC components by ***Devart***.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider[34] property of TRVReportGenerator[46].

TRVReportMyDataProvider introduces properties:

```
property Connection: TCustomMyConnection;
```

Also: Options, FetchRows, FetchAll, DataTypeMap, Encryption, SmartFetch properties, AfterQueryExecute, AfterQueryFetch events.

Internally, the component uses temporal TMyQuery and TMyTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated[52] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[52] event. This DataSet is not necessary needed to be TMyQuery or TMyTable.

See also

- Definitions of Report Workshop terms[10]
- Data queries[28]

4.16 TRVReportMySQLDataProvider

A data provider receiving data from MySQL databases via DAC for MySQL components.

Unit RVReportMySQLDataProvider;

Syntax

```
TRVReportMySQLDataProvider = class (TRVReportCustomDBDataProvider[51])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

This component provides data for a report generator component from ***MySQL*** databases via DAC for MySQL components by ***MicroOLAP Technologies LTD***.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider[34] property of TRVReportGenerator[46].

TRVReportMySQLDataProvider introduces properties:

```
property Database: TmySQLDatabase;
```

Also: SparseArrays, Options, ObjectView properties.

Internally, the component uses temporal TMySQLQuery and TMySQLTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TMySQLQuery or TMySQLTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

4.17 TRVReportNxDataProvider

A data provider receiving data from NexusDB databases.

Unit RVReportNxDataProvider;

Syntax

```
TRVReportNxDataProvider = class (TRVReportCustomDBDataProvider 51)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 53
TRVReportCustomDBDataProvider 51

Description

This component provides data for a report generator component from NexusDB components by ***NexusQA Pty Ltd.***

This data provider handles the following data queries:

- SQL select statement (must start from "select")
- table name (without space characters)
- table or view name with space characters (will be automatically transformed to *select * from "table name"* query)

To use this component, assign it to DataProvider 34 property of TRVReportGenerator 46.

TRVReportNxDataProvider introduces properties:

```
property Database: TnxDatabase;
property Session: TnxSession;
```

Also: Timeout property.

Internally, the component uses temporal TNxQuery and TNxTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TNxQuery and TNxTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

4.18 TRVReportPgDataProvider

A data provider receiving data from PostgreSQL databases via PgDAC.

Unit RVReportPgDataProvider;

Syntax

```
TRVReportPgDataProvider = class (TRVReportCustomDBDataProvider[51])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

This component provides data for a report generator component from ***PostgreSQL*** databases via PgDAC components by ***Devart***.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider[34] property of TRVReportGenerator[46].

TRVReportPgDataProvider introduces properties:

```
property Connection: TPgConnection;
```

Also: Options, FetchRows, FetchAll, DataTypeMap, Encryption, SmartFetch properties, AfterQueryExecute, AfterQueryFetch events.

Internally, the component uses temporal TPgQuery and TPgTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated[52] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[52] event. This DataSet is not necessary needed to be TPgQuery or TPgTable.

See also

- Definitions of Report Workshop terms[10]
- Data queries[28]

4.19 TRVReportPSQLDataProvider

A data provider receiving data from PostgreSQL databases via PostgreSQL components.

Unit RVReportPSQLDataProvider;

Syntax

```
TRVReportPSQLDataProvider = class (TRVReportCustomDBDataProvider[51])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

This component provides data for a report generator component from ***PostgreSQL*** databases via PostgresDAC components by ***MicroOLAP Technologies LTD***.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider[34] property of TRVReportGenerator[46].

TRVReportPSQLDataProvider introduces properties:

```
property Database: TPSQLDatabase;
```

Also: ObjectView property.

Internally, the component uses temporal TPSQLQuery and TPSQLTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated[52] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[52] event. This DataSet is not necessary needed to be TPSQLQuery and TPSQLTable.

See also

- Definitions of Report Workshop terms[10]
- Data queries[28]

4.20 TRVReportUniDataProvider

A data provider receiving data from multiple databases via UniDAC, including Oracle, Microsoft SQL Server, MySQL, InterBase, Firebird, PostgreSQL, SQLite, DB2, Microsoft Access, Sybase Advantage Database Server, Sybase Adaptive Server Enterprise, and other databases (using ODBC provider).

Unit RVReportUniDataProvider;

Syntax

```
TRVReportUniDataProvider = class (TRVReportCustomDBDataProvider[51])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[53]
TRVReportCustomDBDataProvider[51]

Description

This component provides data for a report generator component from multiple databases, including Oracle, Microsoft SQL Server, MySQL, InterBase, Firebird, PostgreSQL, SQLite, DB2, Microsoft Access, Sybase Advantage Database Server, Sybase Adaptive Server Enterprise, and other databases (using ODBC provider). It uses UniDAC components by ***Devart***.

This data provider handles the following data queries:

- SQL select statement
- table name

To use this component, assign it to DataProvider[34] property of TRVReportGenerator[46].

TRVReportUniDataProvider introduces properties:

```
property Connection: TUniConnection;
```

Also: Options, FetchRows, FetchAll, DataTypeMap, Encryption, SmartFetch properties, AfterQueryExecute, AfterQueryFetch events.

Internally, the component uses temporal TUniQuery and TUniTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TUniQuery or TUniTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

4.21 TRVReportZEOSDSDataProvider

A data provider receiving data from various databases using ***Zeos Lib***.

Unit RVReportZEOSDSDataProvider;

Syntax

```
TRVReportZEOSDSDataProvider = class (TRVReportCustomDBDataProvider 51)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 53
TRVReportCustomDBDataProvider 51

Description

This component provides data for a report generator component from various databases using Zeos Library (***Download***, ***Forum***).

This data provider handles the following data queries:

- SQL select statement (must start from "select")
- table name (without space characters)
- table or view name with space characters (will be automatically transformed to *select * from "table name"* query)

To use this component, assign it to DataProvider 34 property of TRVReportGenerator 46.

TRVReportZEOSDSDataProvider introduces the property:

```
property Connection: TZConnection;
```

Internally, the component uses temporal TZQuery and TZTable components to handle data queries. The data provider properties and events are assigned to these components. If you want to assign additional properties to them, use OnDataSetCreated 52 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 52 event. This DataSet is not necessary needed to be TZQuery or TZTable.

See also

- Definitions of Report Workshop terms 10
- Data queries 28

5 Additional Components

The following components are not directly related to reporting, but included in Report Workshop:

 TRVShape[68]. This component draws a shape, such as circle, polygon, star, flag, emoticon, etc.

5.1 TRVShape

TShape represents a geometric shape that can be drawn on a form.

This component is not related to reporting, it simply reuses shape drawing functions implemented for value visualizers[119].

Unit RVReportShapeComponent;

Syntax

```
TRVShape = class (TGraphicControl)
```

Hierarchy

TObject
TPersistent
TComponent
TComponent
TGraphicControl

Description

This component can draw shapes such as polygons, stars, emoticons, flags.

The shape is defined ShapeProperties[70].

It is colored using Color[69], StartColor[70], GradientType[69], Opacity[69] properties. It is outlined using LineColor[69], LineWidth[70], LineUsesFillColor[70] properties.

The component background is filled using BackgroundColor and BackgroundOpacity[68] properties.

Additional properties affecting the layout are: Margin, Padding[70], EqualSides[69].

If you redraw this component frequently, it's recommended to assign DoubleBuffered = *True* for its parent component.

See also

- TRVShapeItemInfo[116] – shape object for TRichView documents

5.1.1 Properties

5.1.1.1 TRVShape.BackgroundColor, BackgroundOpacity

Fill color and opacity for the component.

```
property BackgroundColor: TColor;
property BackgroundOpacity: TRVOpacity;
```

These properties define the component background fill. For shape fill, see Color and Opacity[69] properties.

BackgroundOpacity must be in range from 0 (transparent) to 100000 (opaque).

Opacity is used only if shapes are drawn using GDI+ (in Delphi XE2+).

Default value

- BackgroundColor: *clNone*
- BackgroundOpacity: 100000 (i.e. 100%)

5.1.1.2 TRVShape.Color, Opacity

Fill color and opacity for the shape.

```
property Color: TColor;
property Opacity: TRVOpacity;
```

If GradientType[69] = *rvgtNone*, the shape is filled with this color. Otherwise, this color is used as an ending gradient color (the starting color is StartColor[70]).

Opacity must be in range from 0 (transparent) to 100000 (opaque).

Gradient and opacity are used only if shapes are drawn using GDI+ (in Delphi XE2+).

Default value

- Color: $C68E63
- Opacity: 100000 (i.e. 100%)

5.1.1.3 TRVShape.EqualSides

Specifies whether the shape has the same width and height.

```
property EqualSides: Boolean;
```

If *False*, the shape is drawn in the rectangle Width × Height.

If *True*, the shape is drawn in the square having the side min(Width, Height).

Default value

True

5.1.1.4 TRVShape.GradientType

Type of gradient fill for shapes.

```
property GradientType: TRVGradientType[160];
```

Gradient is drawn from StartColor[70] to Color[69].

If **GradientType** = *rvgtNone*, the shape is filled with Color[69].

Gradient is used only if shapes are drawn using GDI+ (in Delphi XE2+).

Default value

rvgtNone

5.1.1.5 TRVShape.LineColor

Line color

```
property LineColor: TColor
```

Line color can be auto-calculated basing on Color[69], see LineUsesFillColor[70].

Default value

clBlack

See also

- LineWidth[70]

5.1.1.6 TRVShape.LineUsesFillColor

Specifies how shapes are outlined.

```
property LineUsesFillColor: Boolean;
```

If **LineUsesFillColor** = *False*:

Shapes are with LineColor[69].

If **LineUsesFillColor** = *True*:

If Color[69] = *clNone*, LineColor[69] is used. Otherwise, if GradientType[69] <> *rvgtNone*, shapes are outlined with Color[69]; for a flat fill, they are outlined with twice darker color.

Default value

False

5.1.1.7 TRVShape.LineWidth

Line width, in pixels

```
property LineWidth: Integer;
```

Assign 0 to hide lines.

Default value

1

See also

- LineColor[69]

5.1.1.8 TRVShape.Margin, Padding

Margin and padding, in pixels.

```
property Margin: Integer;
property Padding: Integer;
```

The properties define distance from the component edges to the shape.

Margin area is transparent. Padding area can be colored[68].

Default values

0

5.1.1.9 TRVShape.ShapeProperties

Specifies main properties of a shape.

```
property ShapeProperties: TRVReportShapeProperties[169];
```

This property contains sub-properties defining the shape type, rotation angle and other properties.

5.1.1.10 TRVShape.StartColor

Start color for gradient fill.

```
property StartColor: TColor;
```

Gradient uses **StartColor** and Color[69].

Gradient is used only if shapes are drawn using GDI+ (in Delphi XE2+).

Default value

clWhite

5.1.2 Methods

5.1.2.1 TCustomRVReportGenerator.EscapeDataQuery_2

Replaces all occurrences of '{' in **S** to '{{'.

```
class function EscapeDataQuery(const S: TRVUnicodeString): TRVUnicodeString;
```

This function takes the **S** parameter, replaces all '{' characters to '{{' in it, and returns the processed string.

After this processing, '{' characters are not treated as starting field code characters.

Use this function to prepare data queries that do not contain field codes. It is especial useful for JSON data queries, for example for MongoDB data provider [59].

See also

- data queries [28]

5.1.2.2 TCustomRVReportGenerator.Execute_2

Processes a report template contained in **RV** (or **RVData**), and produces a final report.

```
function Execute(RV: TCustomRichView): Boolean;
function Execute(RVData: TCustomRVData): Boolean;
```

The resulting document is not formatted, call **RV**.Format after calling this method.

The method finishes when the report generation is finished. It may take some time for large reports. You can provide feedback in OnDataQueryProgress [36] event.

If the generation is successful, the method returns *True*. There are may be some non-critical errors, though. You can control error handing using OnError [40] event.

5.1.3 Events

5.1.3.1 TCustomRVReportGenerator.OnCreateQueryProcessor_2

Allows providing a custom query processor for the specific **DataQuery**.

```
type
  TRVCreateQueryProcessorEvent = procedure (Sender: TCustomRVReportGenerator [34];
    const DataQuery: TRVUnicodeString;
    var QueryProcessor: TRVReportQueryProcessor [174]) of object;

property OnCreateQueryProcessor: TRVCreateQueryProcessorEvent;
```

This is an optional event. Normally, query processors are created by a linked [34] data provider component [53]. However, you can use this event:

- if you want to use a non-standard query processor;
- if you want to use different query processors for different data queries.

Input parameters

DataQuery – a data query string (such as SQL SELECT statement). This string is already processed: data fields and variables in it are replaced to their values.

Output parameters

QueryProcessor – a query processor to process **DataQuery**. If you return *nil*, a default processing will be created (a query processor will be requested from a linked data provider component). **QueryProcessor** will be owned by the report generator and will be freed when it finishes processing **DataQuery**, so do not free **QueryProcessor** yourself.

To create **QueryProcessor**, you can use CreateQueryProcessor[54] method of a data provider component.

See also:

- Definitions of Report Workshop terms[10]

5.1.3.2 TCustomRVReportGenerator.OnDataQueryProgress_2

Occurs while processing a data query.

```
type
  TRVReportProgressStep = (rvrpsExecutingQuery, rvrpsApplying, rvrpsFinished);

  TRVDataQueryProgressEvent = procedure (Sender: TCustomRVReportGenerator[34];
    Rule: TRVRowGenerationCustomRule[105]; Cell: TRVReportTableCellData[101];
    Step: TRVReportProgressStep; Level, Percent: Integer;
    var Proceed: Boolean) of object;

property OnDataQueryProgress: TRVDataQueryProgressEvent;
```

This event can be used to display a progress of processing a data query and to abort a report generation[36].

Parameters

Rule – a row generation rule of a report table. This parameter is assigned if the component currently processes a data query belonging to this rule, otherwise it is *nil*.

Cell – a cell of a report table.This parameter is assigned if the component currently processes a data query belonging to this cell, otherwise it is *nil*.

Level – a level of nesting of this data query. 1 for first level queries, 2 for queries nested in them, and so on. If you want to display a progress bar to show a progress, it makes sense to show it only for queries having Level = 1.

Step and **Percent** define a stage of progress.

You can assign *False* to **Proceed** to abort the report generation.

Stages

For each query:

1. at first, the event occurs before processing a data query (for example, before processing SQL SELECT statement) with **Step** = *rvrpsExecutingQuery*.
2. next, the event occurs multiple times while applying query results to table rows/a cell, with **Step** = *rvrpsApplying*, and **Percent** growing from 0 to 100. If the report generator cannot calculate percent (it happens when the total number of records is unknown), it calls the event with **Percent** = -1.
3. finally, the event occurs with **Step** = *rvrpsFinished*.

Aborting the report generation

A report generation is performed in the context of the main process. It means that an application is frozen until a report generation is finished.

You can unfreeze it by calling Application.ProcessMessages in this event, but be careful: this procedure is dangerous when used incorrectly!

You must:

- hide the TRichView control processed by this report generator; there must not be attempts to redraw it while the report is not finished and this TRichView is formatted;
- prevent the TRichView, TRVReportGenerator components (and all other components necessary for report generation) from destroying in this event; usually, it can be done by disallowing closing the form containing these components (by processing its OnCloseQuery event)

- prevent the report generation to start again while the current report generation is finished; usually, it can be done by disabling all controls on the form, may be except for an "Abort" button.

You can implement an "Abort" button. When the user clicks it, assign some form's boolean variable. In **OnDataQueryProgress**, check this variable and assign **Proceed** = *False* if necessary.

Example

Showing progress in ProgressBar1 and Label1.

```
procedure TForm1.RVReportGenerator1DataQueryProgress(
  Sender: TCustomRVReportGenerator[34];
  Rule: TRVRowGenerationRule[111]; Cell: TRVReportTableCellData[101];
  Step: TRVReportProgressStep; Level, Percent: Integer;
  var Proceed: Boolean);

  function GetRuleName: String;
  begin
    Result := '';
    if Rule<>nil then
      Result := Rule.Name;
    if Result='' then
      Result := '<untitled>';
  end;

begin
  if Level<>1 then
    exit;
  case Step of
    rvrpsExecutingQuery:
      begin
        ProgressBar1.State := pbsPaused;
        ProgressBar1.Visible := True;
        Label1.Visible := True;
        Label1.Caption := 'Executing query '+GetRuleName;
        Label1.Repaint;
        ProgressBar1.Repaint;
      end;
    rvrpsApplying:
      begin
        if ProgressBar1.State <> pbsNormal then
        begin
          Label1.Caption := 'Applying query '+GetRuleName;
          ProgressBar1.State := pbsNormal;
          Label1.Repaint;
          ProgressBar1.Repaint;
        end;
        ProgressBar1.Position := Percent;
      end;
    rvrpsFinished:
      begin
        ProgressBar1.Visible := False;
        Label1.Visible := False;
      end;
  end;
end;
```

5.1.3.3 TCustomRVReportGenerator.OnError_2

Occurs if an error happens while generating a report.

```
type
  TRVReportGeneratorError = (
    // rules
    rvrgeRuleInvalidRange, rvrgeRuleOverlaps,
    // queries
    rvrgeNoQueryProcessor, rvrgeQueryExecution, rvrgeRecordCountIsRequired,
    // variables
    rvrgeVariableUnknown, rvrgeVariableInvalidValue, rvrgeVariableInvalidName,
    rvrgeVariableUnsupportedObject,
    // data fields
    rvrgeFieldUnknown, rvrgeFieldInvalidValue, rvrgeFieldUnsupportedType,
    rvrgeFieldUnknownType, rvrgeFieldUnknownRule,
    // fields
    rvrgeInvalidTypeCast, rvrgeInvalidFormatString, rvrgeCustomFieldTypeError,
    // commands
    rvrgeCommandUnknown, rvrgeCommandNeedsParam,
    rvrgeCommandDoesNotNeedParam, rvrgeCommandWrongParamValue,
    rvrgeCommandSyntaxError, rvrgeCommandUnmatched,
    rvrgeCommandMustBeInQuery, rvrgeCommandWrongContext,
    // cross-tab
    rvrgeCrossTabInvalid, rvrgeCrossTabInvalidPosition,
    rvrgeCrossTabNoRules,
    rvrgeCrossTabUnknownField,
    rvrgeCrossTabWrongContext, rvrgeCrossTabRecordDoesNotMatch,
    rvrgeCrossTabDuplicateRecords,
    rvrgeCrossTabKeyFieldInCrossTabColumn,
    rvrgeCrossTabNoCrossTabColumns, rvrgeCrossTabTooManyColumns,
    rvrgeCrossTabTooManyTotalColumns,
    // functions
    rvrgeFuncParenthesisExpected, rvrgeFuncUnknown, rvrgeFuncBadParam,
    rvrgeFuncMustBeInCrossTabSummary,
    // others
    rvrgeInternalError
    );

  TRVReportGeneratorErrorEvent = procedure (Sender: TCustomRVReportGenerator[34];
    ErrorCode: TRVReportGeneratorError; const RelatedText: TRVUnicodeString;
    RelatedObject: TObject; var Proceed: Boolean) of object;

property OnError: TRVReportGeneratorErrorEvent;
```

This event can be processed:

- to display an error message,
- to abort a report generation on some (or all) errors.

Parameters

ErrorCode – a code identifying the error.

RelatedText, RelatedObject provide additional information about this error.

You can assign *False* to **Proceed** to abort the report generation.

Errors while checking table row generation rules for correctness

For this kind of errors, **RelatedObject** is TRVRowGenerationCustomRule[105].

Error	Meaning	RelatedText
rvrgeRuleInvalidRange	The rule has an incorrect range of rows (too small or too large row indexes, or these rows overlap other rows because their cells are merged vertically).	Empty
rvrgeRuleOverlaps	The rule has a range of rows intersecting with one of the previous rules in the same table	Empty

Errors while executing a data query (such as SQL SELECT statement)

For this kind of errors, **RelatedObject** is either TRVRowGenerationCustomRule[105] or TRVReportTableCellData[101].

Error	Meaning	RelatedText
rvrgeNoQueryProcessor	The report generator was not able to create a query processor for this data query.	Data query string
rvrgeQueryExecution	An error occurred while executing a data query	Data query string
rvrgeRecordCountIsRequired	This type of report requires a known record count before retrieving data, but the query processor cannot provide it. In the current version, a record count is needed: for column copying in row generation rules, for cross-tab reports	Empty

Errors while processing variables[15]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeVariableUnknown	A field refers to a non-existent variable	Variable name
rvrgeVariableInvalidValue	An improper value of a variable. For example, a variable in "If" command[16] cannot be evaluated as *True* or *False*.	Field code
rvrgeVariableInvalidName	An incorrect (for example, empty) variable name	Variable name
rvrgeVariableUnsupportedObject	A field needs insertion of an object associated with a variable, and this object has a class unsupported by the report generator	Variable name

Errors while processing data fields[12]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeFieldUnknown	A field refers to a non-existent data field	Data field name
rvrgeFieldInvalidValue	An improper value of a a data field. For example, a data field in "If" command[16] cannot be	Field code or data field name

	evaluated as *True* or *False*, or a format of content of a Blob field cannot be detected	
rvrgeFieldUnsupportedType	A data field has an unsupported type	Data field name
rvrgeFieldUnknownRule	A field refers to a non-existent row generation rule	Rule name

Common errors while processing data fields[12], cross-tab header fields,[19] functions[20]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeFieldUnknownType	Unknown value in <field type>[23]	Field code
rvrgeInvalidTypeCast	The value cannot be converted to <field type>[23]	Field code
rvrgeInvalidFormatString	Error in <format string>[24]	Field code
rvrgeCustomFieldTypeError	An error was reported by a custom field type[30] handler. The reason may be because the field data are inappropriate, or a format string is erroneous.	Field code

Errors while processing commands[16]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeCommandUnknown	Unknown command	Field code
rvrgeCommandNeedsParam	This command requires parameter(s)	Field code
rvrgeCommandDoesNotNeedParam	This command requires no parameters	Field code
rvrgeCommandWrongParamValue	A parameter value for this command is incorrect	Field code
rvrgeCommandSyntaxError	A syntax error in the command code	Field code
rvrgeCommandUnmatched	Unmatched commands (for example, "If" without "EndIf")	Field code
rvrgeCommandMustBeInQuery	This command must be inserted in a text affected by a data query	Field code
rvrgeCommandWrongContext	This command is not valid in this place of document (for example, "ListReset" command must be in a numbered paragraph)	Field code

Errors while making a cross-tab report[84]

Error	Meaning	RelatedText	RelatedObject

rvrgeCrossTabInvalid	Table.CrossTabulation [81] is incorrect. Possible reasons: • empty Table.CrossTabulation [81] .Levels [91] [].FieldName [97]; • empty Table.CrossTabulation.Levels[].DataQuery [95], if they are required for the given ColumnGenerationType [88]; • invalid values (or a combination of values) for MinValue, MaxValue, Step [98], if ColumnGenerationType [88] = *rvcgtRange*	Empty	TRVReportTableItemInfo [80]
rvrgeCrossTabInvalidPosition	Position or structure of a cross-tab header [86] is incorrect.	Empty	TRVReportTableItemInfo [80]
rvrgeCrossTabNoRules	A cross tabulation is defined for a table, but this table has no row generation rules [82], or the first rule is invalid	Empty	TRVReportTableItemInfo [80]
rvrgeCrossTabUnknownField	Table.CrossTabulation [81].Levels [91] [].FieldName [97] or CaptionFieldName [93] does not exists in the results of the respective data query (if ColumnGenerationType [88] <> *rvcgtRange*)	Field name	TRVCrossTabLevel [92]
	A cross-tab header field [19] refers to a non-existent data field	Field name	TRVReportGenerationSession [172]
rvrgeCrossTabWrongContext	A cross-tab header field [19] is found outside a cross-tab header cells	Field code	TRVReportGenerationSession [172]
rvrgeCrossTabRecordDoesNotMatch	A record in the result of a row generation rule's DataQuery [105] contains a combination of values of column fields [97] not corresponding to any cross-tab data column	Empty	TRVReportGenerationSession [172]
rvrgeCrossTabDuplicateRecords	A record in the result of a row generation rule's DataQuery [105] contains the same combination of values of key fields [115] and column fields [97] as one of previous records.	Empty	TRVReportGenerationSession [172]
rvrgeCrossTabKeyFieldInCrossTabColumn	One ore more key fields [115] is included in column fields [97]	Field name	TRVRowGenerationRule [111]
rvrgeCrossTabNoCrossTabColumns	A cross tabulation contains 0 data columns	Empty	TRVReportTableItemInfo [80] or TRVCrossTabLevel [92], depending on ColumnGenerationType [88]

rvrgeCrossTabTooManyColumns	A group of columns in a cross-tab level exceeds level.MaxColCount[98]. In this case, only the first MaxColCount[98] columns are used.	Empty	TRVCrossTabLevel[92]
rvrgeCrossTabTooManyTotalColumns	A total number of column repetitions exceeds MaxColCount[92]	Empty	TRVReportTableItemInfo[80]

Errors while processing functions[20]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[172].

Error	Meaning	RelatedText
rvrgeFuncParenthesisExpected	An opening or closing bracket around the function parameter is missing	Field code
rvrgeFuncUnknown	Unknown function name	Field code
rvrgeFuncBadParam	A function parameter is not a valid value field name	Field code
rvrgeFuncMustBeInCrossTabSummary	A function field is inserted not in a proper place	Field code

Other errors

Error	Meaning	RelatedText	RelatedObject
rvrgeInternalError	An exception occurred while generating a report	Exception message	Exception

Example

```
const ErrorMsg: array[TRVReportGeneratorError] of TRVUnicodeString =
  ('Rule error: invalid row range', 'Rule error: overlapping row range',
   'Query processing error: cann''t create query processor for %s',
   'Query execution error for %s', 'Variable error: unknown variable %s',
   'Variable error: invalid value for %s',
   'Variable error: invalid name %s',
   'Variable error: unsupported object in %s',
   'Field error: unknown field %s', 'Field error: invalid value for %s',
   'Field error: unsupported field type for %s', 'Field error: unknown rule %s',
   'Command error: unknown command %s', 'Command error: parameters are required in %s',
   'Command error: this command must not have parameters: %s',
   'Command error: wrong parameter value: %s',
   'Command error: syntax error %s', 'Command error: unmatched command %s',
   'Command error: a command must be in content affected by a query: %s',
   'Command error: the command is not valid in its context: %s');

procedure TForm1.RVReportGenerator1Error(Sender: TCustomRVReportGenerator[34];
  ErrorCode: TRVReportGeneratorError; const RelatedText: TRVUnicodeString;
  RelatedObject: TObject; var Proceed: Boolean);
var S: TRVUnicodeString;
begin
  S := Format(ErrorMsg[ErrorCode], [RelatedText]);
  Memo1.Lines.Add(S);
end;
```

5.1.3.4 TCustomRVReportGenerator.OnProcessRecord_2

Occurs before processing a record of a query processor.

```
type
  TRVProcessRecordEvent = procedure (Sender: TCustomRVReportGenerator[34];
    const DataQuery: TRVUnicodeString; Rule: TRVRowGenerationRule[111];
    Cell: TRVReportTableCellData[101];
    Session: TRVReportGenerationSession[172]; RecNo: Integer;
    QueryProcessor: TRVReportQueryProcessor[174]) of object;

property OnProcessRecord: TRVProcessRecordEvent;
```

This event may be used to assign values to variables based on the data from this record.

Parameters

DataQuery – a data query string (such as SQL SELECT statement). This string is already processed: data fields and variables in it are replaced to their values.

Rule – a row generation rule of a report table. This parameter is assigned if the **DataQuery** belongs to this rule, otherwise, it is *nil*.

Cell – a cell of a report table.This parameter is assigned if the **DataQuery** belongs to this cell, otherwise, it is *nil*.

Session – an object representing a report generation session. It can be used to get values of variables[15] and data fields[12].

RecNo – index of the record in the results of **QueryProcessor**, in the range 0..**QueryProcessor**.GetRecordCount-1.

QueryProcessor – a query processor created to process **DataQuery**.

See also

- Definitions of Report Workshop terms [10]

6 Document Items

6.1 Report table - TRVReportTableItemInfo

A class of objects in TRichView documents: a table having associated data queries (in row generation rules and cells)

Unit RVReportTable;

Syntax

```
TRVReportTableItemInfo = class (TRVTableItemInfo)
```

Hierarchy

TObject
TPersistent
TCustomRVItemInfo
TRVNonTextItemInfo
TRVFullLineItemInfo
TRVTableItemInfo

Description

This item looks like a normal table in TRichView document, and has all its properties.

Additionally, it has the properties:

- RowGenerationRules [82] – a collection of a table row generation rules, allowing to apply data queries to certain ranges of table rows.
- Variables [82] – a list of local variables [15] for this table
- BackgroundColorChangers [80] – a collections of color changers allowing to change colors and opacities of cells according to values associated with these cells.
- BackgroundVisualizers [81] – a collections of visualizers allowing to display values associated with cells on these cells' backgrounds.

A class of cells for report tables is TRVReportTableCellData [101], it allows associating data queries with cells.

The StyleNo of this item is rvsReportTable (-61).

The class of Cell[] property is still TRVTableCellData, so you can use ReportCells [81][] property:

```
Table.ReportCells [81][Row, Col].DataQuery [103] := 'SELECT * FROM MyTable';
```

See also

- Definitions of Report Workshop terms [10]

6.1.1 Properties

6.1.1.1 TRVReportTableItemInfo.BackgroundColorChangers

A collection containing background color changers for table cells.

```
property BackgroundColorChangers: TRVReportColorChangers [100];
```

If a report cell [101] is linked [102] to a color changer, this color changer changes the cell color and opacity according to a value [102] associated with this cell.

Cells are shaded with gradations of two or three colors that correspond to minimum, midpoint, and maximum thresholds.

A direct assignment to this property cannot be undone by the user. Use SetBackgroundColorChangers [83] to assign a new value to this property as an editing operation.

See also

- Definitions of Report Workshop terms [10]

6.1.1.2 TRVReportTableItemInfo.BackgroundVisualizers

A collection containing value visualizers [119] for table cells.

```
property BackgroundVisualizers: TRVReportValueVisualizers[104];
```

If a report cell [101] is linked [104] to a visualizer, this visualizer displays a value [103] associated with this cell on this cell's background.

There are many different value visualizers available, so classes of items in this collection are different. They are inherited from TRVReportCustomValueVisualizer [133].

This collection is a public, not a published property. However, it is stored together with its report table in RVF, like other collection properties introduced in TRVReportTableItemInfo [80] (we had to implement a special saving because items of this collection have different classes).

A direct assignment to this property cannot be undone by the user. Use SetBackgroundVisualizers [83] to assign a new value to this property as an editing operation.

See also

- Definitions of Report Workshop terms [10]

6.1.1.3 TRVReportTableItemInfo.CrossTabulation

A property defining parameters for cross-tab reports.

```
property CrossTabulation: TRVCrossTab[84];
```

A direct assignment to this property cannot be undone by the user. Use SetCrossTabulation [84] to assign a new value to this property as an editing operation.

6.1.1.4 TRVReportTableItemInfo.Processed

Specifies if this table was processed by a report generator [46].

```
property Processed: Boolean;
```

Initially, **Processed** = *False*. When a report generator's Execute [36] is called for TRichView containing this table, **Processed** becomes *True*.

Cross tabulation, row generation rules, and cells with assigned DataQuery can be highlighted only if **Processed** = False.

Default value

False

See also

- TRVReportDocObject [170].HighlightRules [171]

6.1.1.5 TRVReportTableItemInfo.ReportCells

Lists cells in the table.

```
property ReportCells[Row, Col: Integer]: TRVReportTableCellData[101];
```

This property returns the same values as Cells[Row, Col], but the result is typecasted TRVReportTableCellData [101] (i.e. to the actual class of cells).

Default value

False

6.1.1.6 TRVReportTableItemInfo.RowGenerationRules

A collection of row generation rules for this report table.

```
property RowGenerationRules: TRVRowGenerationRules[115];
```

Each item in this collections allows applying a data query (for example, SQL SELECT) to the specified range of table rows. These rows will be replicated as many times as many records a query processor returns for this data query, with each replication corresponds to one record.

A direct assignment to this property cannot be undone by the user. Use SetRowGenerationRules[84] to assign a new value to this property as an editing operation.

See also

- Definitions of Report Workshop terms[10]
- item in this collection: TRVRowGenerationRule[111]

6.1.1.7 TRVReportTableItemInfo.SelectedRule

A rule (or a cross tabulation header) to highlight as a selected one.

```
property SelectedRule: TRVRowGenerationCustomRule[105];
property CrossTabSelected: Boolean;
```

Assign one of RowGenerationRules[82] or their SubRules[107] to **SelectedRule** to draw a border around it.

Assign *True* to **CrossTabSelected** to draw a border around the cross tabulation[81] header.

These properties are used only when highlighting[171] is enabled.

A border is drawn around the cells of the selected rule / cross-tab header.

Special cases:

- an additional thin border is drawn around source cells, if the rule defines column copying[113];
- when drawing a border for a nested rule, it includes the leftmost and rightmost cells[109], while background filling does not.

See also

- TRVRowGenerationCustomRule[105].HighlightColor[106]
- TRVCrossTab[84].HighlightColor[91]

6.1.1.8 TRVReportTableItemInfo.Variables

A string list containing local variables[15] for this report table.

```
property Variables: TStringList;
```

This string list must have items in the form:

<variable name>=<variable value>

Items may have associated objects. They are owned by this string list: it frees these object when clearing or destroying.

A direct assignment to this property cannot be undone by the user. Use SetVariables[84] to assign a new value to this property as an editing operation.

See also:

- variables in templates[15]
- TCustomRVReportGenerator[34].Variables[35]
- TRVReportGenerationSession[172]

6.1.2 Methods

6.1.2.1 TRVReportTableItemInfo.SetBackgroundColorChangers

Assigns **ColorChangers** to BackgroundColorChangers [80] as an editing operation.

```
procedure SetBackgroundColorChangers(ColorChangers: TRVReportColorChangers [100]);
```

If the table is inserted in an editor, this method assigns the property as an editing (undoable) operation. Otherwise, it works (almost) like a direct assignment to the property. Unlike a direct assignment, this method copies Id [135] properties of items.

6.1.2.2 TRVReportTableItemInfo.SetBackgroundVisualizers

Assigns **Visualizers** to BackgroundVisualizers [81] as an editing operation.

```
procedure SetBackgroundVisualizers(Visualizers: TRVReportValueVisualizers [104]);
```

If the table is inserted in an editor, this method assigns the property as an editing (undoable) operation. Otherwise, it works (almost) like a direct assignment to the property. Unlike a direct assignment, this method copies Id [135] properties of items.

6.1.2.3 TRVReportTableItemInfo.SetCellColorChanger

Assigns **ColorChangerId** to ReportCells [81][**Row**, **Col**].ColorChangerId [102] as an editing operation.

```
procedure SetCellColorChanger(ColorChangerId: TRVValueVisualizerId [165]; Row, Col: Integer);
```

If the table is inserted in an editor, this method assigns the property as an editing (undoable) operation. Otherwise, it works like a direct assignment to the property.

6.1.2.4 TRVReportTableItemInfo.SetCellDataQuery

Assigns **Value** to ReportCells [81][**Row**, **Col**].DataQuery [103] as an editing operation.

```
procedure SetCellDataQuery(const Value: TRVUnicodeString; Row, Col: Integer);
```

If the table is inserted in an editor, this method assigns the property as an editing (undoable) operation. Otherwise, it works like a direct assignment to the property.

6.1.2.5 TRVReportTableItemInfo.SetCellHighlightColor

Assigns **Value** to ReportCells [81][**Row**, **Col**].HighlightColor [102] as an editing operation.

```
procedure SetCellHighlightColor(Value: TColor; Row, Col: Integer);
```

If the table is inserted in an editor, this method assigns the property as an editing (undoable) operation. Otherwise, it works like a direct assignment to the property.

6.1.2.6 TRVReportTableItemInfo.SetCellVisualizer

Assigns **VisualizerId** to ReportCells [81][**Row**, **Col**].VisualizerId [104] as an editing operation.

```
procedure SetCellVisualizer(VisualizerId: TRVValueVisualizerId [165]; Row, Col: Integer);
```

If the table is inserted in an editor, this method assigns the property as an editing (undoable) operation. Otherwise, it works like a direct assignment to the property.

6.1.2.7 TRVReportTableItemInfo.SetCrossTabulation

Assigns **CrossTab** to CrossTabulation [81] as an editing operation.

```
procedure SetCrossTabulation(CrossTab: TRVCrossTab [84]);
```

If the table is inserted in an editor, this method assigns the property as an editing (undoable) operation. Otherwise, it works like a direct assignment to the property.

6.1.2.8 TRVReportTableItemInfo.SetRowGenerationRules

Assigns **Rules** to RowGenerationRules [82] as an editing operation.

```
procedure SetRowGenerationRules(Rules: TRVRowGenerationRules [115]);
```

If the table is inserted in an editor, this method assigns the property as an editing (undoable) operation. Otherwise, it works like a direct assignment to the property.

6.1.2.9 TRVReportTableItemInfo.SetVariables

Assigns **AVariables** to Variables [82] as an editing operation.

```
procedure SetVariables(AVariables: TStringList);
```

If the table is inserted in an editor, this method assigns the property as an editing (undoable) operation. Otherwise, it works like a direct assignment to the property.

Since Variables [82].Objects[] are owned by Variables [82], this method assigns Variables [82].Objects[] and clears all **AVariables**.Objects[].

6.1.3 Classes of properties

6.1.3.1 TRVCrossTab

TRVCrossTab contains properties defining properties of cross tabulation reports.

Unit RVReportTable;

Syntax

```
TRVCrossTab = class (TPersistent);
```

Hierarchy

TObject
TPersistent

Description

Definition

TRVCrossTab is a class TRVReportTableItemInfo [80].CrossTabulation [81] property.

It defines the position of the cross-tab header in the table (Column and FirstRow [86] properties) and its levels (Levels [91] property).

If Levels [91] collection has at least one item, this table represents a cross-tab report.

A cross-tab report allows to display data as a grid, with rows representing one group of data ("row fields"), columns representing another group of data ("column fields"), and intersection of rows and columns containing a summarized information ("value fields").

TRVCrossTab represents group of data related to columns of a cross-tab report. *Column fields* are specified in Levels [91][].FieldName [97]. Each item in this collection corresponds to one level of a cross-tab report.

A report table may contain several cross-tab reports: as many reports as the count of items in Table.RowGenerationRules[82]. *Row fields* are listed in Table.RowGenerationRules[82][].KeyFieldNames[115], *column fields* are the same for all row generation rules. All other fields can be considered as *value fields*.

Cross-tab header

A structure of a cross-tab report is determined by the structure of cells in the cross-tab header. This header is located at the position specified in FirstRow and Column[86] properties. A cross-tab header defines:

- how many summary columns exist on each level;
- how many columns correspond to each combination of values of value fields

Examples and the detailed information about cross-tab headers can be found in the topic about FirstRow and Column[86] properties.

In the text below, the term "cross-tab columns" means columns below the cross-tab header.

Functions

Any value field of integer or floating point type can be used as a parameters of functions[20] fields. A range of values used to calculate a function depends on the cell position.

Building a cross-tab report

For each row generation rule, report generator creates a cross-tab report in the following way.

Stage 0: generating cross-tab columns

Stage 1: creating report rows and filling cells of cross-tab columns

For each record of the results of DataQuery[105], the report generator compares values of KeyFieldNames[115] with the previous report rows. If this record contains a unique set of values of these fields, a new report row is added*; otherwise, *value fields* from this record are written to the existing row having the same set of values in KeyFieldNames[115] fields.

Value fields are written in the intersection of this row and a cross-tab column** corresponding to the set of values of Items[91][].FieldName[97], equal to this record. Thus, *column fields* work like a filter of data.

At this stage, two errors are possible:

- a record contains values of Levels[91][].FieldName[97] fields, not corresponding to any *column fields* (it is not possible if ColumnGenerationType[88] = *rvcgtAutoSeparate* and *rvcgtAutoGroup*, but possible with other column generation types); OnError[40] event occurs with *rvrgeCrossTabRecordDoesNotMatch* parameter, and this record is ignored;
- a record contains values of Levels[91][].FieldName[97] and KeyFieldNames[115] fields, equal to values of another record (i.e., two or more records corresponds to the same cell*** of a cross-tab report); OnError[40] event occurs with *rvrgeCrossTabDuplicateRecords* parameter, and this record is ignored (only the first record containing this set of values is used).

The results of this stage:

- all table rows corresponding to this row generation rule are added;
- all cells of cross-tabs columns of these rows are filled, except for cells of summary columns;
- the report generator accumulates data necessary for calculating aggregate functions in summary columns and rows.

Stage 2: filling the rest of cells of report rows

The report generator navigates through records of results of DataQuery[105] again. This time it enumerates only first records corresponding to each report row. At this stage, the rest of cells of the report rows are filled. It was not possible at the first stage, because we did not have data for calculating aggregate functions.

Notes

* strictly speaking, it may be not a single row, but a group of RowCount[106] rows

** strictly speaking, it may be not a single column, but several columns corresponding to each set of values in *column fields*; this count of columns is defined in the bottom row of the cross-tab header, see the topic about FirstRow and Column 86 properties

*** strictly speaking, it may be a group of cells having the number of rows and columns described in "*" and "**" notes

6.1.3.1.1 Properties

6.1.3.1.1.1 TRVCrossTab.Column, FirstRow

These properties define the location of a cross-tab header in the report table

```
property FirstRow: Integer;
property Column: Integer;
```

A cross-tab header consists of cells located in:

- rows from **FirstRow** to **FirstRow** + Count - 1
- columns from **Column** to Table.Cells[**FirstRow**, **Column**].**ColSpan** - 1

Structure of a cross-tab header

The cross-tab header has Levels 91.Count rows.

- the **FirstRow**-th row corresponds to Items[0];
- the (**FirstRow** +1)-th row corresponds to Items[1];
- and so on.

The **Column**-th column is the leftmost column of the cross-tab header. It contains cells corresponding to data columns. Other cross-tab header columns, if they exist, correspond to summary columns.

For every row *R* in the cross-tab header, except for the first row, Table.Cells[R, **Column**].ColSpan <= Table.Cells[*R* - 1, **Column**].ColSpan. If this is a strict inequality, the right column(s) define headers for summary columns at this level. For the first row, all columns outside the cross-tab columns can be considered as summary columns. Summary header cells can span across columns, but not beyond the cross-tab header's right side.

If the cell in the leftmost column of the last cross-tab level has ColSpan > 1, this means that a group of ColSpan columns is replicated for each cross-tab data.

For every row *R* in the cross-tab header, Table.Cells[R, **Column**].RowSpan = 1. Other (summary header) cells may span across rows, but not beyond the cross-tab header's bottom side.

Below you can find some examples. The pink color shows cross-tab header cells corresponding to data columns, the light blue color shows cross-tab header cells corresponding to summary columns.

Example 1

The most simple example, only one item in Levels 91.

Template

	{#Person}
{Year}	${Salary}

Result Sample

	Alice	Bob	Carol
2014	$1000	$2000	$3000
2015	$4000	$5000	$6000

Example 2

One item in Levels 91, two data columns

Template

Result Sample

	{#Person}	
	Salary	Days
{Year}	${Salary}	{WorkDays}

	Alice		Bob		Carol	
	Salary	Days	Salary	Days	Salary	Days
2014	$1000	100	$2000	200	$3000	300
2015	$4000	300	$5000	300	$6000	300

Example 2

Two items in Levels [91]

Template

	{#Question}
	{#Option}
{Age}	{Percent}%

Result Sample

	Do you like porridge?		Have you ever seen	
	Yes	No	Yes	Never
Below 20	10%	90%	10%	80%
20 and older	50%	50%	10%	0%

Example 3

Two items in Levels [91], a summary column. Additionally, the rightmost table column is used as a grand summary.

Template

	{#Category}		Grand Total
	{#Product}	Total for {#Category}	
{Year}	{Sales}	{sum(Sales)}	**{sum(Sales)**

Result Sample

	Fruits				
	Apples	Oranges	Total for 2014	Tomatoes	C
2014	100	200	300	300	
2015	150	250	400	350	

Example 3

Three items in Levels [91], two summary columns in each level.

Template

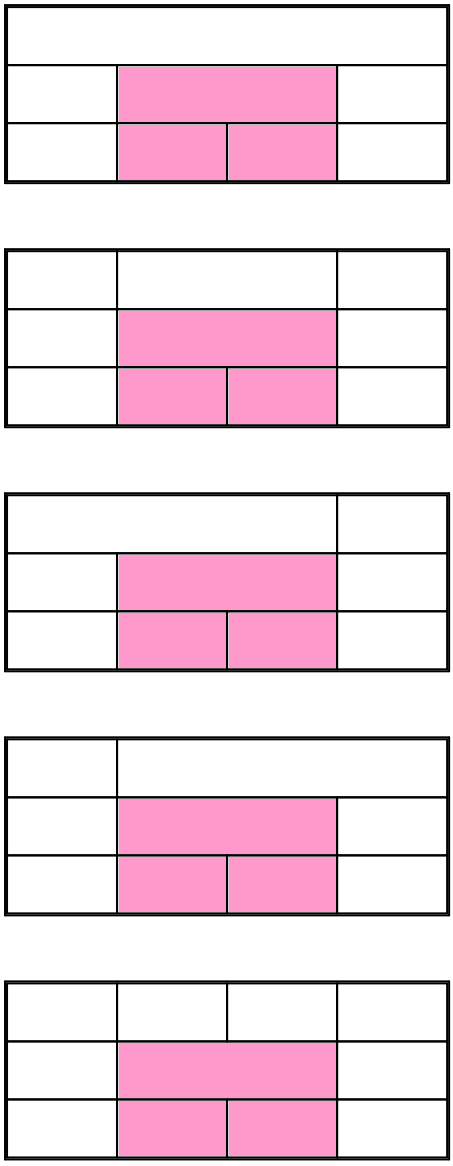
	{#Category}					Total	Average
	{#Product}			Total for {#	Average fo		
	{#Year}	Total fo	Averag				
{Region}	{Sales}	{sum(S	{averag	{sum(Sales	{average(S	**{sum(Sales)}**	**{average(Sales)}**

Position of a cross-tab header in a table

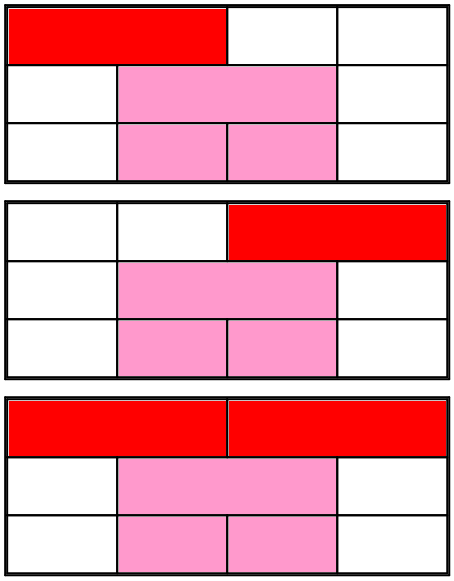
The whole cross-tab header must be inside a rectangular area, i.e. no cells may span across its sides.

Cells above the header may span across the cross-tab header columns, but such cells must not end or start between the header columns. Below you can find the examples for this rule, the pink color shows a cross-tab header.

Good:



Bad:



The cells below the cross-tab header may not span across cross-tab columns.

Default values:

0

See also

- Definitions of Report Workshop terms [10]

6.1.3.1.1.2 TRVCrossTab.ColumnGenerationType

Specifies how cross-tabs columns are generated.

```
type
  TRVColGenerationType = (rvcgtAutoSeparate, rvcgtAutoGroup,
    rvcgtDataQuerySeparate, rvcgtDataQueryCascade,
```

```
    rvcgtRange);

property ColumnGenerationType: TRVColGenerationType;
```

This property affects all levels of cross-tab.

- *rvcgtAutoSeparate* and *rvcgtAutoGroup*, columns are generated basing on data of the report itself.
- *rvcgtDataQuerySeparate* and *rvcgtDataQueryCascade*: columns are generated by applying special data queries for each level of cross-tab.
- *rvcgtRange*: columns are generated by enumerating values in the specified range

Comparison table

<table>
<tr><th></th><th>rvcgtAutoSeparate</th><th>rvcgtAutoGroup</th><th>rvcgtDataQuerySeparate</th><th>rvcgtDataQueryCascade</th><th>rvcgtRange</th></tr>
<tr><td>Columns are generated basing on...</td><td colspan="2">report data
(results of Table.RowGenerationRules[82][0].DataQuery[105])</td><td colspan="2">special data query
(results on Levels[91][].DataQuery[95])</td><td>range of values</td></tr>
<tr><td>Each group of columns are generated from an unique set of values, depending on the value of the parent cross-tab column
(if not, all groups of columns on each level are generated from the same sets of values)</td><td>–</td><td>Yes</td><td>–</td><td>Yes</td><td>–</td></tr>
<tr><td>Can use special data field for captions[93]</td><td colspan="4">Yes</td><td>–</td></tr>
<tr><td>Can use integer, boolean, floating point, date-time data fields for values</td><td colspan="5">Yes</td></tr>
<tr><td>Can use text data fields for values</td><td colspan="4">Yes</td><td>–</td></tr>
<tr><td>Sort order</td><td colspan="2">defined in Levels[91][].SortType[99]</td><td colspan="2">as returned by a data query</td><td>ascending or descending, depending on Levels[91][].Step[98]</td></tr>
</table>

rvcgtAutoSeparate and rvcgtAutoGroup

For *rvcgtAutoSeparate* and *rvcgtAutoGroup*, columns are generated basing on data of the report itself.

ColumnGenerationType	Comments

rvcgtAutoSeparate	Columns are generated basing on data field values returned by the application of Table.RowGenerationRules[82][0].DataQuery[105]. Values for each cross-tab level are collected independently.
rvcgtAutoGroup	The same, but data for cross-tab levels are collected together. This option generates the minimal necessary set of columns.

Example

Let we have a data table containing the following data:

Student	Year	Subject	Mark
Alice	2014	Math	7
Alice	2015	Law	6
Bob	2014	Math	3
Bob	2015	Math	9

We want to build a cross-tab report on fields "Year" and "Subject", with "Student" used as a key field. The result is shown below.

rvcgtAutoSeparate

Student	2014		2015	
	Math	Law	Math	Law
Alice	7			6
Bob	3		9	

rvcgtAutoGroup

Student	2014	2015	
	Math	Math	Law
Alice	7		6
Bob	3	9	

As you can see, the right table does not have (2014, 'Law') column, because this combination of values does not exist in the data table.

For each cross-tab level (i.e., for each item in Levels[91]):

- values for column generation are taken from FieldName[97] field;
- additionally, captions may be taken from CaptionFieldNameField[93];
- columns are sorted according to SortType[99].

rvcgtDataQuerySeparate and rvcgtDataQueryCascade

For *rvcgtDataQuerySeparate* and *rvcgtDataQueryCascade*, columns are generated from data queries specified in cross-tab levels, i.e in Levels[91][].DataQuery[95].

ColumnGenerationType	Comments
rvcgtDataQuerySeparate	All Table.Levels[91][].DataQuery[95] are independent from each other. Each such a data query is executed only once. If each level of cross-tab repeats the same group of data, use this generation type, because it is more efficient.
rvcgtDataQueryCascade	Levels[91][*N*].DataQuery[95] may refer[19] to data queries of higher levels, i.e. the results of execution of Levels[91][*M*].DataQuery[95]-s, where *N* is from 1 to Count-1, *M* can be from 0 to N-1.

	I.e. the *N*-th data query may use the results of 0..*N*-1 data queries. See Levels[91][].DataQuery[95] for an example.

For each cross-tab level (i.e., for each item in Levels[91]):

- values for column generation are taken either from DataQueryFieldName[97] or from FieldName[97] field;
- additionally, captions may be taken from CaptionFieldNameField[93];

rvcgtRange

For *rvcgtRange*, columns are generated from enumerations of values in the specified range.

For each cross-tab level (i.e., for each item in Levels[91]), this enumeration is defined by MinValue, MaxValue, and Step[98] properties.

Default value

rvcgtAutoGroup

6.1.3.1.1.3 TRVCrossTab.HighlightColor

Specifies the color to highlight the cross-tab header in the report template.

```
property HighlightColor: TColor;
```

The cross-tab header is highlighted if:

- the first TRVReportDocObject[170] item in TRichView.DocObjects has HighlightRules[171] = *True*;
- the owner table's Processed[81] = *False*
- FirstRow[86] and Column[86] define the position of the cross-tab header inside the table
- the owner table's Cells[FirstRow[86], Column[86]] is not *nil*.

The header is shaded with this color.

Default value:

$FF0099

See also

- Definitions of Report Workshop terms[10]
- TRVReportTableCellData[102].HighlightColor[102]
- TRVRowGenerationCustomRule[105].HighlightColor[106]
- TRVReportTableItemInfo[80].CrossTabSelected[82]

6.1.3.1.1.4 TRVCrossTab.Levels

Collection of levels of cross-tab columns.

```
property Levels: TRVCrossTabLevels[99]; default;
```

This is a collection of TRVCrossTabLevel[92] items.

A cross tabulation is defined, if this collection has at least one item.

Levels[0] corresponds to ReportTable.Cells[FirstRow, Column[86]].
Levels[1] corresponds to ReportTable.Cells[FirstRow + 1, Column].
Levels[2] corresponds to ReportTable.Cells[FirstRow + 2, Column].
and so on.

6.1.3.1.1.5 TRVCrossTab.MaxColCount

A maximal possible count of columns generated for the lowest cross-tab level (in total).

```
property MaxColCount: Integer;
```

If the report generator generates more than **MaxColCount** columns, *rvrgeCrossTabTooManyTotalColumns* error occurs [40], and the cross-tabulation for this table is not applied.

Strictly speaking, this value limits not a number of columns at the lowest level, but a number of column repetitions on the lowest level (there are may be more actual columns, if any level contains more than one data columns). Only repetitions of data columns are counted, summary columns are ignored.

Default value:

1000

See also:

- TRVCrossTabLevel [92].MaxColCount [98]

6.1.3.1.1.6 TRVCrossTab.TextForEmptyCells

Specifies a text to insert in empty cells instead of their content.

```
property TextForEmptyCells: TRVUnicodeString;
```

If a cell in a cross-tab column does not correspond to any record of data, this cell is cleared, and **TextForEmptyCells** is inserted in this cell.

The same processing may be made for summary columns, if Items[].ClearEmptySummaryCols [95] = True.

Default value:

" (empty string)

See also

- Definitions of Report Workshop terms [10]

6.1.3.2 TRVCrossTabLevel

TRVCrossTabLevel is an item in TRVCrossTab [84] collection.

Unit RVReportTable;

Syntax

```
TRVCrossTabLevel = class (TCollectionItem);
```

Hierarchy

TObject
TPersistent
TCollectionItem

Description

TRVCrossTabLevel is a class of items in the collection TRVCrossTabLevels [99], available as TRVCrossTab [84].Levels [91] property.

TRVCrossTabLevel defines a level of cross-tab report, i.e. how columns are generated at this level.

The column generation method is common for all items in this collection, it is defined in TRVReportTableItemInfo [80].CrossTabulation [81].ColumnGenerationType [88].

Different properties of **TRVCrossTabLevel** are used for different values of ColumnGenerationType.

ColumnGenerationType	Properties of TRVCrossTabLevel

rvcgtAutoSeparate, *rvcgtAutoGroup*	• FieldName 97 • CaptionFieldName 93 • SortType 99 • CaseSensitive 94
rvcgtDataQuerySeparate, *rvcgtDataQueryCascade*	• DataQuery 95 • FieldName 97 • DataQueryFieldName 97 • CaptionFieldName 93 • CaseSensitive 94
rvcgtRange	• MinValue, MaxValue, Step 98

ClearEmptySummaryCols 95 affects summary columns started on the next cross-tab level.

MaxColCount 98 allows restricting the count of columns at this level.

6.1.3.2.1 Properties

6.1.3.2.1.1 TRVCrossTabLevel.CaptionFieldName

A name of data field that can be used to show captions of cross-tab columns.

```
property CaptionFieldName: TRVUnicodeString;
```

If this property is not empty, it defines an alternative caption for columns of this level of the cross-tab table. This caption can be displayed (see cross-tab heading field 19 syntax).

This property is used if TRVReportTableItemInfo 80.CrossTabulation 81.ColumnGenerationType 88 is one of:

- *rvcgtDataQuerySeparate*
- *rvcgtDataQueryCascade.*
- *rvcgtAutoSeparate*
- *rvcgtAutoGroup*

This property is useful if FieldName field contains some identifier, while a text description is stored in another field.

If SortType 99 = *rvrstCaptionsAscending* or *rvrstCaptionsDescending*, and TRVReportTableItemInfo 80.CrossTabulation 81.ColumnGenerationType 88 = *rvcgtAutoSeparate* or *rvcgtAutoGroup*, captions are used when sorting columns at this level of cross-tab (see also CaseSensitive 94 property).

Example

A data table for a cross-tab report is named "OrdersTable", it has fields: OrderId, ItemId, Quantity.

There is another table "ItemsTable", it has fields: ItemId, ItemName.

We want to build a cross-tab report that uses ItemId for cross-tab columns, but we want to display ItemName in the table header.

The simplest example of a template table for this report is below:

Order	**{#ItemName}**
N {OrderID}	{Quantity}

```
Table.CrossTabulation[81].ColumnGenerationType[88] := rvcgtDataQuerySeparate;
Table.CrossTabulation.Column[86] := 1;
Table.CrossTabulation.FirstRow[86] := 0;
with Table.CrossTabulation.Levels[91].Add do
begin
  DataQuery := 'SELECT ItemId, ItemName FROM ItemsTable ORDER BY ItemName';
  FieldName[97] := 'ItemId';
  CaptionFieldName := 'ItemName';
end;
with Table.RowGenerationRules[82].Add do
begin
  FirstRow[106] := 1;
  RowCount[106] := 1;
  DataQuery[105] := 'SELECT * FROM OrdersTable';
  KeyFieldNames[115] := 'OrderId';
end;
```

The resulting report table looks like:

Order	Item A	Item B	Item C
N 10001	1	2	3
N 10002	3	3	3
N 10003	4	1	1
N 10004	2	3	3

Default value:

" (empty string)

See also

- Definitions of Report Workshop terms[10]

6.1.3.2.1.2 TRVCrossTabLevel.CaseSensitive

Specifies how text strings are compared.

```
property CaseSensitive: Boolean;
```

1. This property is used when generating columns for this cross-tab level, if the cross-tab field contains text values:

- if TRVReportTableItemInfo[80].CrossTabulation[81].ColumnGenerationType[88] = *rvcgtDataQuerySeparate* or *rvcgtDataQueryCascade*, this property is used when building a list of values for cross-tab columns and removing duplicates from this list (data are generated by DataQuery[95], values are taken from DataQueryFieldName[97] or FieldName[97] field)
- if TRVReportTableItemInfo[80].CrossTabulation.ColumnGenerationType = *rvcgtAutoSeparate* or *rvcgtAutoGroup*, this property is used when building a list of values for cross-tab columns, sorting and removing duplicates from this list (data are generated by TRVReportTableItemInfo[80].RowGenerationRules[82][0].DataQuery[95], values are taken from FieldName[97] field)

2. if SortType[99] = *rvrstCaptionsAscending* or *rvrstCaptionsDescending*, and TRVReportTableItemInfo[80].CrossTabulation.ColumnGenerationType = *rvcgtAutoSeparate* or *rvcgtAutoGroup,* this property is used when sorting columns (captions are taken from CaptionFieldName[93] field)

Default value

False

6.1.3.2.1.3 TRVCrossTabLevel.ClearEmptySummaryCols

Allows or disallows clearing summary columns containing no values.

```
property ClearEmptySummaryCols: Boolean;
```

This value affects summary columns started on the next cross-tab level.

For example, in this template (the colored cells belong to the cross-tab header, the light blue color denotes the header for summary columns), the summary column is affected by TRVReportTableItemInfo[80].CrossTabulation[81].Levels[91][0].**ClearEmptySummaryCols**.

	{#Category}	
	{#Product}	Total for {#Category}
{Year}	{Sales}	{sum(Sales)}

If the whole group of columns to the left of the summary column(s) contain no values, they are cleared and replaced to Table.CrossTabulation.TextForEmptyCells[92].

Note: as you can see, this property affects only summary columns inside the cross-tab columns. It does not affect to summary columns for the whole rows, and to summary rows, they cannot be cleared.

Default value

False

6.1.3.2.1.4 TRVCrossTabLevel.DataQuery

A data query for cross-tab columns generation.

```
property DataQuery: TRVUnicodeString;
```

This property is used only if TRVReportTableItemInfo[80].CrossTabulation[81].ColumnGenerationType[88] = *rvcgtDataQuerySeparate* or *rvcgtDataQueryCascade*.

If there is only one cross-tab level (i.e. TRVReportTableItemInfo[80].CrossTabulation.Levels[91].Count = 1), these generation types are identical. The difference exists only if there are two or more cross-tab levels.

ColumnGenerationType	Comments
rvcgtDataQuerySeparate	All Table.CrossTabulation.Levels[91][].**DataQuery** are independent from each other. Each such a data query is executed only once. If each level of cross-tab repeats the same group of data, use this generation type, because it is more efficient.
rvcgtDataQueryCascade	Table.CrossTabulation.Levels[91][].**DataQuery** may refer[19] to data queries of higher levels. I.e. the *N*-th data query may use the results of 0..*N*-1 data queries.

Values for cross-tab columns generation are taken from DataQueryFieldName[97] field of the result of application of **DataQuery**. If DataQueryFieldName[97] is empty, FieldName[97] field is used.

This field must be of one of the following types[161]:

- *rvrftText,*
- *rvrftInteger,*
- *rvrftFloat,*
- *rvrftBoolean,*
- *rvrftDateTime, rvrftDate, rvrftTime*

The values used for column generation are ordered as they appear in records returned by an application of **DataQuery**. Only unique values are used, without repetitions. When comparing text values (to remove duplicates), CaseSensitive property is taken into account.

Example:

Let we have the following tables:

'QuestionsTable' containing identifiers and text of poll questions.

'QuestionOptionsTable' containing identifiers and text of possible answers to questions.

```
Table.CrossTabulation[81].ColumnGenerationType[88] := rvcgtDataQueryCascade;
Table.CrossTabulation.Column[86] := 1;
Table.CrossTabulation.FirstRow[86] := 0;
with Table.CrossTabulation.Levels[91].Add do
begin
  DataQuery := 'SELECT QuestionText, QuestionId FROM QuestionsTable';
  FieldName[97] := 'QuestionId';
  CaptionFieldName[93] := 'QuestionText';
end;
with Table.CrossTabulation.Levels[91].Add do
begin
  DataQuery := 'SELECT OptionText, OptionId FROM QuestionsOptionsTable WHERE QuestionId={#Q
  FieldName[97] := 'QuestionId';
  CaptionFieldName[93] := 'QuestionText';
end;
```

The report template table may look like:

Age	{#QuestionText}
	{#OptionText}
{Age}	{Percent}%

The resulting table may look like:

Age	Do you ... ?		Are you ... ?		
	yes	no	yes, completely	yes, partially	no
15 and under	10%	90%	25%	50%	25%
16-25	21%	79%	50%	25%	25%
26-40	1%	99%	15%	60%	25%
over 40	90%	10%	0%	100%	0%

Default value:

" (empty string)

See also

- Definitions of Report Workshop terms [10]
- Information about data queries [28]
- TRVReportDocObject [170].DataQuery [171]
- TRVRowGenerationRule [111].DataQuery [105]
- TRVReportTableCellData [101].DataQuery [103]

6.1.3.2.1.5 TRVCrossTabLevel.DataQueryFieldName

A name of data field that can be used to generate cross-tab columns, if they are generated by special data queries.

```
property DataQueryFieldName: TRVUnicodeString;
```

This property is used only if TRVReportTableItemInfo[80].CrossTabulation[81].ColumnGenerationType[88] = *rvcgtDataQuerySeparate* or *rvcgtDataQueryCascade*.

This field must be of one of the following types[161]:

- *rvrftText,*
- *rvrftInteger,*
- *rvrftFloat,*
- *rvrftBoolean,*
- *rvrftDateTime, rvrftDate, rvrftTime*

If this property is empty, FieldName[97] is used instead.

Default value:

" (empty string)

See also

- Definitions of Report Workshop terms[10]

6.1.3.2.1.6 TRVCrossTabLevel.FieldName

A name of data field corresponding to this level of the cross-tab table.

```
property FieldName: TRVUnicodeString;
```

Primarily, this property is used when processing records produced by executing a row generation rule's DataQuery[105]. A record is matched to columns of this level of the cross-tab table, if its **FieldName** field contains a value equal to the value corresponding to these columns.

Additionally, this field can be used:

- when displaying a value corresponding to columns of this level of the cross-tab table (see cross-tab heading field[19] syntax); see also CaptionFieldName[93];
- when generating columns of this level of the cross-tab table in the following cases:
  - if TRVReportTableItemInfo[80].CrossTabulation[81].ColumnGenerationType[88] = *rvcgtDataQuerySeparate* or *rvcgtDataQueryCascade*, and DataQueryFieldName[97] is empty;
  - if TRVReportTableItemInfo.CrossTabulation.ColumnGenerationType = *rvcgtAutoSeparate* or *rvcgtAutoGroup*.

This field must be of one of the following types[161]:

- *rvrftText,*
- *rvrftInteger,*
- *rvrftFloat,*
- *rvrftBoolean,*
- *rvrftDateTime, rvrftDate, rvrftTime*

Default value:

" (empty string)

See also

- Definitions of Report Workshop terms[10]

6.1.3.2.1.7 TRVCrossTabLevel.MaxColCount

A maximal possible count of columns generated for this cross-tab level (in each group).

```
property MaxColCount: Integer;
```

If the report generator generates more than **MaxColCount** columns in a group for this level, *rvrgeCrossTabTooManyColumns* error occurs [40], and the first **MaxColCount** columns are used.

Note 1: summary columns are ignored, only data columns are counted;

Note 2: strictly speaking, not columns, but a number of column repetitions is counted (the actual number of generated columns = number of repetitions * count of data columns in the table template for this level)

Note 3 a top level has a single group of columns; a lower level has as many groups of columns as many column repetitions are on its parent level

Default value:

100

See also:

- TRVCrossTab [84].MaxColCount [92]

6.1.3.2.1.8 TRVCrossTabLevel.MinValue, MaxValue, Step

The properties define a range of field values to generate columns of this cross-tab report level.

```
property MinValue: Variant;
property MaxValue: Variant;
property Step: Variant;
```

These properties are used only if TRVReportTableItemInfo [80].CrossTabulation [81].ColumnGenerationType [88] = *rvcgtRange*.

For this column generation type, columns are generated from values in the specified range, without using data queries.

The first value for the columns is **MinValue**, the next value is **MinValue** + **Step**, the next value is **MinValue** + **Step***2, and so on, until the value exceeds **MaxValue**. **MinValue** is always included, **MaxValue** is included if incrementing by **Step** allows it (or for boolean values, where **Step** is ignored).

Limitations:

- allowed variant types: floating point values, integer values, date-time, boolean;
- **MaxValue** and **MinValue** must be of the same type (for example, floating point **MaxValue** and integer **MinValue** are not allowed);
- **Step** must have the same type as well*, with the exception: for date-time **MinValue** and **MaxValue**, **Step** must be a floating point value;
- **Step** must not be zero*;
- if **MinValue** < **MaxValue**, **Step** must be positive, if **MinValue** > **MaxValue**, **Step** must be negative*.

Comment:

* **Step** is ignored for boolean values, and if **MinValue** = **MaxValue**

Default values:

Undefined

See also

- Definitions of Report Workshop terms [10]

6.1.3.2.1.9 TRVCrossTabLevel.SortType

Specifies how columns are ordered at this level of a cross-tab report.

```
type
  TRVReportSortType = (rvrstNoSort, rvrstAscending, rvrstDescending,
    rvrstCaptionsAscending, rvrstCaptionsDescending);
property SortType: TRVReportSortType;
```

This property is used if TRVReportTableItemInfo[80].CrossTabulation[81].ColumnGenerationType[88] is one of:

- *rvcgtDataQuerySeparate*
- *rvcgtDataQueryCascade.*

In these column generation modes, values (and, optionally, captions) are taken from the results of TRVReportTableItemInfo[80].RowGenerationRules[82][0].DataQuery[95].

Values are in FieldName[97] field, captions are in CaptionFieldName[93] field.

Value	Meaning
rvrstNoSort	No sorting, values are sorted in the order of their appearance in the results of a date query
rvrstAscending	Values are ordered from low to high
rvrstDescending	Values are ordered from high to low
rvrstCaptionsAscending	Values are sorted so that captions are ordered low to high; if captions are not defined, this mode works like *rvrstAscending*
rvrstCaptionsDescending	Values are sorted so that captions are ordered high to low; if captions are not defined, this mode works like *rvrstDescending*

Text values and captions are compared using CaseSensitive[94] property.

Default value:

rvrstAscending

See also

- Definitions of Report Workshop terms[10]

6.1.3.3 TRVCrossTabLevels

TRVCrossTabLevels is a collection of cross-tab levels for a report table[80].

Unit RVReportTable;

Syntax

```
TRVCrossTabLevels = class (TOwnedCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection

Description

TRVCrossTabLevels is a class of TRVCrossTab[84].Levels[91] property.

In other words, it is a class of TRVReportTableItemInfo[80].CrossTabulation[81].Levels[91] property.

It is a collection of TRVCrossTabLevel[92] items.

6.1.3.3.1 Properties

6.1.3.3.1.1 TRVCrossTabLevels.Items

Lists the items in the collection.

```
property Items[Index: Integer]: TRVCrossTabLevel[92]; default;
```

Use **Items** to access individual items in the collection.

6.1.3.4 TRVReportColorChangers

TRVReportColorChangers is a collection of color changers for a report table[80].

Unit RVReportColorChanger;

Syntax

```
TRVReportColorChangers = class (TCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection
TRVReportCustomValueVisualizerCollection[100]

Description

TRVReportColorChangers is a class of TRVReportTableItemInfo[80].BackgroundColorChangers[80] property. It is a collection of TRVReportColorChangerItem[127] items.

See also

- TRVReportTableCellData[101].ColorChangerId[102]

6.1.3.4.1 Properties

6.1.3.4.1.1 TRVReportColorChangers.Items

Lists the items in the collection.

```
property Items[Index: Integer]: TRVReportColorChangerItem[127]; default;
```

Use **Items** to access individual items in the collection.

6.1.3.5 TRVReportCustomValueVisualizerCollection

TRVReportCustomValueVisualizerCollection is a parent class for TRVReportValueVisualizers[104] and TRVReportColorChangers[100].

Unit RVReportValueVisualizer;

Syntax

```
TRVReportValueVisualizers = class (TCollection);
```

Hierarchy

TObject

TPersistent
TCollection
TOwnedCollection

Description

This class is not used directly.

Use TRVReportValueVisualizers[104] and TRVReportColorChangers[100].

6.1.3.5.1 Properties

6.1.3.5.1.1 TRVReportCustomValueVisualizerCollection.ForbiddenIds

A list of forbidden values for Id[135] properties of items.

```
property ForbiddenIds: TRVIntegerList;
```

This property lists values that cannot be assigned to new items of the collection.

It is useful when editing the collection: when the user deletes an item in the collection, add its Id[135] to this list. It prevents reusing this value for new items.

6.1.3.5.2 Methods

6.1.3.5.2.1 TRVReportCustomValueVisualizerCollection.AssignWithId

Assigns Source to the collection.

```
procedure AssignWithId(Source: TRVReportCustomValueVisualizerCollection[100]);
```

Unlike Assign method, AssignWithId copies all properties of items, including Id[135]

6.1.3.5.2.2 TRVReportCustomValueVisualizerCollection.FindById

Searches for the item with the specified value of Id[135] property.

```
function FindById(Id: TRVValueVisualizerId[165]): Integer;
function FindItemById(Id: TRVValueVisualizerId[165]):
  TRVReportCustomValueVisualizerBase[134];
```

FindById returns the index of the item, or -1 if not found.

FindItemById returns the item, or *nil* if not found.

6.1.3.6 TRVReportTableCellData

TRVReportTableCellData is a class of cell of a report table (TRVReportTableItemInfo[80]).

Unit RVReportTable;

Syntax

```
TRVReportTableCellData = class (TRVTableCellData)
```

Hierarchy

TObject
TPersistent
TCustomRVData
TCustomRVFormattedData
TRVItemFormattedData
TRVTableCellData

Description

In addition to the standard TRichView table cell properties, this class introduces DataQuery[103]. If DataQuery is specified, the content of this cell is repeated for each record returned by the execution of this query.

Also, this cell may be linked to a color changer (ColorChangerId[102]) or value visualizer (VisualizerId[104]). They allow changing background colors and displaying a diagram proportionally to the value calculated for this cell.

Cells are accessible using TRVReportTableItemInfo[80].ReportCells[81] property.

6.1.3.6.1 Properties

6.1.3.6.1.1 TRVReportTableCellData.ColorChangerId

This property allows linking this cell with a color changer.

```
property ColorChangerId: TRVValueVisualizerId[165];
```

This property allows referring to the item in the report table's BackgroundColorChangers[80] collection by its Id[135] property. (or -1, if this cell is not linked to a color changer).

When TRVReportGenerator[46] generates a report, it calculates a value of this item's ValueString[135] for this cell and assigns it to ColorValue[102].

After applying table's RowGenerationRules[82] and CrossTabulation[81], multiple cells may be produced from this cell. All these cells will be linked to the specified color changer.

In a report template (before a report generation) several cells may be linked to the same color changer. In this case, for all cells produced from these cells, background color and opacity will be changed using the same minimal and maximal values.

A direct assignment to this property cannot be undone by the user. Use Table.SetCellColorChanger[83] to assign a new value to this property as an editing operation.

Default value:

-1

See also

- Definitions of Report Workshop terms[10]

6.1.3.6.1.2 TRVReportTableCellData.ColorValue

This property contains a value used for the cell's background color changer.

```
property ColorValue: Variant;
```

When TRVReportGenerator[46] generates a report, it calculates values of table.BackgroundColorChangers[80].FindItemById[101](ColorChangerId[102]).ValueString[135] for all cells produced from this cell, and assigns results to their **ColorValue** properties.

After that, these cells' background colors and/or opacities are changed according to these values.

Unlike Value, **ColorValue** is not stored in RVF (it's not necessary, because color and opacity of cells are stored). So this property is used to store a value temporarily.

See also

- Definitions of Report Workshop terms[10]

6.1.3.6.1.3 TRVReportTableCellData.HighlightColor

Specifies the color to highlight this cell in the report template.

```
property HighlightColor: TColor;
```

The cell is highlighted if:

- its DataQuery[103] is not empty
- the first TRVReportDocObject[170] item in TRichView.DocObjects has HighlightRules[171] = *True*;
- the owner table's Processed[81] = *False*

The cell is shaded with this color.

A direct assignment to this property cannot be undone by the user. Use Table.SetCellHighlightColor[83] to assign a new value to this property as an editing operation.

Default value:

$41C589

See also

- Definitions of Report Workshop terms[10]
- TRVRowGenerationCustomRule[105].HighlightColor[106]
- TRVCrossTab[84].HighlightColor[91]

6.1.3.6.1.4 TRVReportTableCellData.DataQuery

A string containing a data query.

```
property DataQuery: TRVUnicodeString;
```

An example of a DataQuery is a SQL query, like 'SELECT * FROM MyTable'.

TRVReportGenerator[46] creates a query processor[174] for this data query. This query processor returns data for this query, and the report generator replicates a content of this cell as many times as many records in the returned data. Then the report generator processes these replicated contents, replacing data fields in them accordingly.

A direct assignment to this property cannot be undone by the user. Use Table.SetCellDataQuery[83] to assign a new value to this property as an editing operation.

Default value:

" (empty string)

See also

- Definitions of Report Workshop terms[10]
- Information about data queries[28]
- TRVReportDocObject[170].DataQuery[171]
- TRVRowGenerationCustomRule[105].DataQuery[105]
- TRVCrossTabLevel[92].DataQuery[95]

6.1.3.6.1.5 TRVReportTableCellData.Value

This property contains a value used for the cell's value visualizer.

```
property Value: Variant;
```

When TRVReportGenerator[46] generates a report, it calculates values of table.BackgroundVisualizers[81].FindItemById[101](VizualizerId[104]).ValueString[135] for all cells produced from this cell, and assigns results to their **Value** properties.

After that, this value is visualized at these cells' backgrounds.

See also

- Definitions of Report Workshop terms[10]

6.1.3.6.1.6 TRVReportTableCellData.VisualizerId

This property allows linking this cell with a value visualizer.

```
property VisualizerId: TRVValueVisualizerId;
```

This property allows referring to the item in the report table's BackgroundVisualizers collection (or -1, if this cell is not linked to a background visualizer).

When TRVReportGenerator generates a report, it calculates a value of this item's ValueString for this cell and assigns it to Value.

After applying table's RowGenerationRules and CrossTabulation, multiple cells may be produced from this cell. All these cells will be linked to the specified visualizer.

In a report template (before a report generation) several cells may be linked to the same visualizer. In this case, for all cells produced from these cells, data will be visualized using the same minimal and maximal values, and the same visualizer size.

A direct assignment to this property cannot be undone by the user. Use Table.SetCellVisualizer to assign a new value to this property as an editing operation.

Default value:

-1

See also

- Definitions of Report Workshop terms

6.1.3.7 TRVReportValueVisualizers

TRVReportValueVisualizers is a collection of value visualizers for a report table.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportValueVisualizers = class (TCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection
TRVReportCustomValueVisualizerCollection

Description

TRVReportValueVisualizers is a class of TRVReportTableItemInfo.BackgroundVisualizers property. This collection may include items of different classes, inherited from TRVReportCustomValueVisualizer.

See also

- TRVReportTableCellData.VisualizerId

6.1.3.7.1 Properties

6.1.3.7.1.1 TRVReportValueVisualizers.Items

Lists the items in the collection.

```
property Items[Index: Integer]: TRVReportCustomValueVisualizer; default;
```

Use **Items** to access individual items in the collection.

TRVReportCustomValueVisualizer is a parent class for items of this collection. To assign properties specific for the given visualizer, typecast Items[] to its class.

6.1.3.8 TRVRowGenerationCustomRule

TRVRowGenerationCustomRule is a base class for table row generation rules.

Unit RVReportTable;

Syntax

```
TRVRowGenerationRule = class (TCollectionItem)
```

Hierarchy

TObject
TPersistent
TCollectionItem

Description

This class is not used directly. The following classes are inherited from it:

- TRVReportGenerationRule 172 – a class of items in the collection TRVReportTableItemInfo 80 .RowGenerationRules 82.
- TRVReportGenerationCustomRule 172 – a class of items in the collection SubRules 107.

Main properties:

- DataQuery 105 – a data query to apply to the rows
- FirstRow, RowCount 106 – a range of table rows that are replicated when applying the rule
- Name 106 – a name of the rule, allowing to refer to it from data fields 12
- SubRules 107 – rules applied to subsets of cells belonging to the rows defined with FirstRow, RowCount 106

6.1.3.8.1 Properties

6.1.3.8.1.1 TRVRow GenerationCustomRule.DataQuery

A string containing a data query.

```
property DataQuery: TRVUnicodeString;
```

An example of a DataQuery is a SQL query, like 'SELECT * FROM MyTable'.

TRVReportGenerator 46 creates a query processor 174 for this data query. This query processor returns data for this query, and the report generator replicates rows specified in FirstRow and RowCount 106 properties as many times as many records exist in the returned data. Then the report generator processes these replicated rows, replacing data fields in them accordingly.

Normally, the report generator travels through the results of this query only once. But for cross-tab reports, the report generator may travel 3 up times:

1. when generating columns, if Table.CrossTabulation 81.ColumnGenerationType 88 = *rvcgtAutoSeparate* or *rvcgtAutoGroup* (only for the first rule in the table)
2. when filling cells of cross-tab columns;
3. when filling the rest of cells.

Default value:

" (empty string)

See also

- Definitions of Report Workshop terms 10
- Information about data queries 28
- TRVReportDocObject 170.DataQuery 171
- TRVReportTableCellData 101.DataQuery 103
- TRVCrossTabLevel 92.DataQuery 95

6.1.3.8.1.2 TRVRow GenerationCustomRule.FirstRow , Row Count

The properties defining a range of rows of the report table [80] to apply this rule.

```
property FirstRow: Integer;
property RowCount: Integer;
```

The rule is applied to the row range **FirstRow**..**FirstRow**+**RowCount**-1. Indexes of rows are counted from 0.

If this is a nested rule [109], **FirstRow** is added to the position of its parent rule.

The rule is applied only if the range of rows is correct, i.e.:

- for a root rule [111], this range is inside 0..Table.RowCount-1;
- for a nested rule [109], this range is inside 0..**RowCount**-1 of the parent rule;
- table rows belonging to this range do not overlap with rows below and above (because of a vertical cell merging)
- this range does not intersect with ranges of rules having lower indexes in the collection.

The row indexes are counted in the original table in a report template (not in a table produced by an application of previous rules).

Default values:

- FirstRow = 0
- RowCount = 1

6.1.3.8.1.3 TRVRow GenerationCustomRule.HighlightColor

Specifies the color to highlight row(s) belonging to this rule in the report template.

```
property HighlightColor: TColor;
```

The row(s) are highlighted if:

- the first TRVReportDocObject [170] item in TRichView.DocObjects has HighlightRules [171] = *True*;
- the owner table's Processed [81] = *False*
- FirstRow [106] and RowCount [106] define the position inside the table.

The rows are shaded with this color.

One of rules in the table can be drawn specially, it is specified in TRVReportTableItemInfo [80].SelectedRule [82].

Default value:

$B18419

See also

- Definitions of Report Workshop terms [10]
- TRVReportTableCellData [102].HighlightColor [102]
- TRVCrossTab [84].HighlightColor [91]
- TRVReportTableItemInfo [80].SelectedRule [82]

6.1.3.8.1.4 TRVRow GenerationCustomRule.Name

A name of this row generation rule.

```
property Name: TRVUnicodeString;
```

A name allows data fields [12] to refer to data produced by an application of this rule. Also, it makes it easier to identify a rule in the report generator events, such as OnDataQueryProgress [37].

A name is optional: if it is omitted in a data field, results of the most recent query processor (for a table rule or a cell) is used.

Example:

Let we have database tables:

- MasterTable having fields: master_id, master_name

- DetailTable having fields: master_id, detail_name and others

We need to produce a master-detail report.

We add a report table with a rule:

- Name='Master',
- DataQuery='SELECT * FROM MasterTable',
- FirstRow=0, RowCount=1.

In the top left cell of this table, we insert:

- text: 'Master name: {Master:master_name}
- another table with a rule:
  - Name='Detail',
  - DataQuery='SELECT * FROM DetailTable WHERE master_id={Master:master_id}'
    text in a cell: 'Detail name: {Detail:detail_name}'

In all cases in this example, we can omit references to the rules (i.e. 'Master:' and 'Detail:' inside {}), because in this example all data fields refer to the most recent rule in their context.

Default value:

" (empty string)

See also

- Definitions of Report Workshop terms [10]

6.1.3.8.1.5 TRVRowGenerationCustomRule.SubRules

A collection of nested rules for this rule.

```
property SubRules: TRVRowGenerationNestedRules[111];
```

Nested rules are applied only if:

- if cross-tabulation is not defined (i.e. the table's CrossTabulation[84].Levels[91].Count = 0)
- copying is not defined in the rule (i.e. CopyColCount[113] = 0).

Nested rules allow applying data queries to [parts of] rows belonging to this rule.

Rows of nested rules must be included in rows of this rule. Rows of nested rules must not overlap with each other.

Example 1

Let we have the table "Product" having the fields "Category" and "Product".

The table template is:

Products
{Category}
{Product}

```
with Table.RowGenerationRules[82].Add do
  FirstRow[106] := 1;
  RowCount[106] := 2;
  DataQuery[105] := 'SELECT Category FROM Products';
  with SubRules[107].Add do
  begin
    FirstRow[106] := 1; // the actual first row is 1 + 1 = 2
    RowCount[106] := 1;
    DataQuery[105] := 'SELECT Product FROM Products WHERE Category=''{Category}''';
 end;
```

As you can see, all two green rows belong to the parent rule, and the light green row also belongs to the nested rule.

The result would be like this:

Products
Fruits
apples
oranges
Vegetables
tomatoes
cucumbers

Example 2

The same data as in the example 1, but a different report layout.

The table template is:

Products	
{Category}	{Product}

```
with Table.RowGenerationRules[82].Add do
  FirstRow[106] := 1;
  RowCount[106] := 1;
  DataQuery[105] := 'SELECT Category FROM Products';
  with SubRules[107].Add do
  begin
    FirstRow[106] := 0; // the actual first row is 1 + 0 = 1
    RowCount[106] := 1;
    SkipLeftColCount[109] := 1;
    DataQuery[105] := 'SELECT Product FROM Products WHERE Category=''{Category}''';
 end;
```

As you can see, all two green cells belong to the parent rule, and the light green cell also belongs to the nested rule.

The result would be like this:

Products	
Fruits	apples
	oranges
Vegetables	tomatoes
	cucumbers

Order processing

The report generator processes rules and also cells of report tables.

The order of processing is the following

1. the first sub-rule (rows are replicated)
2. cells belonging to the first sub-rule's replicated fragment, i.e. all cells between its SkipLeftColCount and SkipRightColCount[109] columns;
3. other cells of the first sub-rule, i.e. all cells of the outermost SkipLeftColCount and SkipRightColCount columns;
4. the same for the second sub-rule, and so on
5. cells of this rule, not belonging to sub-rules

6.1.3.9 TRVRowGenerationNestedRule

TRVRowGenerationNestedRule is an item in TRVRowGenerationRules[111] collection.

Unit RVReportTable;

Syntax

```
TRVRowGenerationNestedRule = class (TRVRowGenerationCustomRule[105])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVRowGenerationCustomRule[105]

Description

TRVRowGenerationNestedRule is a class of items in the collection TRVRowGenerationCustomRule[105].SubRules[107].

The report table[80] has RowGenerationRules[82] property, a collection of TRVRowGenerationRule[111] items. Each item has SubRules[107] property, a collection of **TRVRowGenerationNestedRule**. Each item in this collection has its on SubRules[107], and so on. Thus, rules are organized in a tree-like hierarchy.

Like for the root rules, the position of this rule's rows are defined in FirstRow and RowCount[106] properties. However, rows are counted from the parent rule's first row, not from the table begin.

This class introduces new properties allowing to define outermost columns: SkipLeftColCount and SkipRightColCount[109]. Cells of these columns are not replicated when applying this rule. Instead, they are merged vertically.

6.1.3.9.1 Properties

6.1.3.9.1.1 TRVRow GenerationNestedRule.SkipLeftColCount, SkipRightColCount

The properties define outermost columns that are not replicated when this sub-rule is applied.

```
property SkipLeftColCount: Integer;
property SkipRightColCount: Integer;
```

When this sub-rule is applied, its rows are replicated. However, cells in outermost columns may be not replicated but merged across all replicated rows.

SkipLeftColCount defines the number of such columns from the left side (withing columns of the parent rule).

SkipRightColCount defines the number of such columns from the right side (withing columns of the parent rule).

So, FirstRow and RowCount[106] define the range of rows which will be replicated, **SkipLeftColCount** and **SkipRightColCount** define replicated and not-replicated columns.

However, there is a difference. All rows of the parent rule outside the range defined by this sub-rule's FirstRow and RowCount[106] do not belong to this sub-rule (they may belong to another sub-rule, or not). All columns of the

parent rule outside the range defined by **SkipLeftColCount** and **SkipRightColCount** are still considered as belonging to this rule, and they are processed immediately after processing the replicated part of this sub-rule.

When this sub-rule is applied, all cells belonging to these outermost columns are merged vertically (see the "Example 2" in the topic about SubRules[107]). If MergeWithPrevious[110] = *True*, all cells belonging to these outermost columns are merged with cells of the previous sub-rule, if the following conditions are satisfied:

- this is not the first sub-rule,
- rows of this sub-rule are located immediately below the previous sub-rule,
- the previous sub-rule has equal values of **SkipLeftColCount** (cells of the leftmost columns are merged) and/or **SkipRightColCount** (cells of the rightmost columns are merged)

Default values:

0

6.1.3.9.1.2 TRVRow GenerationNestedRule.MergeWithPrevious

Specifies whether cells of the outermost columns of the sub-rule will be merged with cells of the previous rule.

```
property MergeWithPrevious: Boolean;
```

The counts of outermost columns are specified in SkipLeftColCount and SkipRightColCount[109].

Example

Let we have the table "Product" having the fields "Category" and "Product".

The table template is:

Products	
{Category}	{Product}
	{Count}

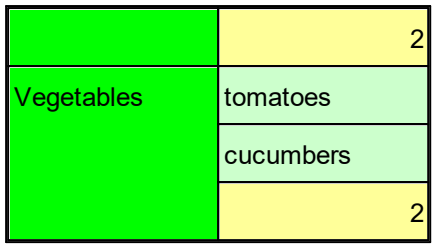
All colored cells belong to the parent rule, light green cell belongs to the first sub-rule, yellow cell belongs to the second sub-rule.

The results with **MergeWithPrevious** = *False*:

Products	
Fruits	apples
	oranges
	2
Vegetables	tomatoes
	cucumbers
	2

The results with **MergeWithPrevious** = *True*:

Products	
Fruits	apples
	oranges



Default value:

False

6.1.3.10 TRVRowGenerationNestedRules

TRVRowGenerationNestedRules is a collection of row generation sub-rules for a report table [80].

Unit RVReportTable;

Syntax

```
TRVRowGenerationNestedRules = class (TOwnedCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection

Description

TRVRowGenerationRules [115] is a class of TRVReportTableItemInfo [80].RowGenerationRules [82] property. It is a collection of TRVRowGenerationRule [111] items.

Each item in this collection has SubRules [107] property of **TRVRowGenerationNestedRules** class. It is a collection of TRVRowGenerationNestedRule [109] items. Each item in its order has its own SubRules [107] property.

6.1.3.10.1 Properties

6.1.3.10.1.1 TRVRow GenerationNestedRules.Items

Lists the items in the collection.

```
property Items[Index: Integer]: TRVRowGenerationRule [111]; default;
```

Use **Items** to access individual items in the collection.

6.1.3.11 TRVRowGenerationRule

TRVRowGenerationRule is an item in TRVRowGenerationRules [115] collection.

Unit RVReportTable;

Syntax

```
TRVRowGenerationRule = class (TRVRowGenerationCustomRule [105])
```

Hierarchy

TObject
TPersistent
TCollectionItem

TRVRowGenerationCustomRule 105

Description

TRVRowGenerationRule is a class of items in the collection TRVReportTableItemInfo 80 .RowGenerationRules 82.

In addition to properties inherited from TRVRowGenerationCustomRule 105, this class introduce the properties listed below.

Properties used when building a cross-tab report:

- KeyFieldNames 115 – a list of fields allowing to distinguish records when grouping them in table rows
- CaseSensitive 112 specifies how values of text key fields are compared
- AllowSummaryCols, AllowSummaryRows 112 allow generating reports that do not have summary rows and columns more efficiently.

Properties used for copying columns:

- CopyFirstCol, CopyColCount, CopySpacingColCount, CopyMaxCount 113

Special report features

There are three special report features that can be used in rules:

- cross-tabulation (mainly defined in the table's CrossTabulation 84 property);
- column copying (content of selected cells are copied in multiple columns, and only then a new row is added);
- nested rules (defined in SubRules 107, nested rules applied to rows belonging to rows of their parent rule).

Only one of these features can be used at the same time:

- if cross-tabulation is defined (i.e. the table's CrossTabulation 84.Levels 91.Count > 0), neither column copying nor nested rules are applied;
- if column copying is defined in the rule (i.e. CopyColCount 113 > 0), nested rules are not applied; however, different rules in the same table may have column copying and nested rules.

6.1.3.11.1 Properties

6.1.3.11.1.1 TRVRow GenerationRule.Allow SummaryCols, Allow SummaryRow s

The properties allow generating cross-tab reports 84 that do not have summary columns and rows more efficiently.

```
property AllowSummaryCols: Boolean;
property AllowSummaryRows: Boolean;
```

These properties are used only when building a cross-tab report. Otherwise, they are ignored.

You can assign **AllowSummaryCols** = *False* if the report table 80 does not have functions in summary columns for this rule. You can assign **AllowSummaryRows** = *False* if the report table does not have functions in summary rows for this rule. These assignments allow building a report faster and using less memory.

Default value:

True

6.1.3.11.1.2 TRVRow GenerationRule.CaseSensitive

Defines how values of text fields in KeyFieldNames 115 are compared when building a cross-tab 84 report.

```
property CaseSensitive: Boolean;
```

Default value:

False

6.1.3.11.1.3 TRVRow GenerationRule.CopyFirstCol, CopyColCount, CopySpacingColCount, CopyMaxCount

The properties defining table columns copying.

```
property CopyFirstCol: Integer;
property CopyColCount: Integer;
property CopySpacingColCount: Integer;
property CopyMaxCount: Integer;
```

These properties are used only if the table is not used for building a cross-tab report, i.e. if its CrossTabulation[81].Levels[91].Count = 0.

Columns copying is activated only if **CopyColCount** > 0. If it is activated, data are propagated from left to right (by copying content), then from top to bottom (by adding rows).

There is an important difference between a columns copying and an application of the rule to rows: when applying the rule to table rows, new rows are added; when copying columns, contents of the source cells are copied to existing cells, without adding new columns.

The source cells are defined as cells in the intersection of rows from FirstRow to FirstRow+RowCount-1[106], and columns from **CopyFirstCol** to **CopyFirstCol**+**CopyColCount**-1. These cells are copied to columns to the right, leaving **CopySpacingColCount** columns between copies. The number of copies is calculated automatically. If **CopyMaxCount** > 0, it defines the maximal possible count of copies (per a group of rows).

The following cases are not considered as errors (the rule is applied ignoring these properties):

- CrossTabulation[81].Levels[91].Count > 0
- incorrect values of these properties, if **CopyColCount** <= 0.

The following cases are considered as errors (the rule is not applied):

- incorrect values of properties (specifying indexes of columns outside the table)
- only one instance of repeated cells (i.e. only source cells, so no copying can occur; because of it, you cannot assign **CopyMaxCount** = 1, the minimal valid value is 2).
- incorrect structure of the source cells (if some cells intersect the range of the main cells' rows and columns because of cell merging)
- incorrect structure of destination cells (every destination cell must have the same values of ColSpan and RowSpan properties as the corresponding source cell)

If **CopyColCount** > 1, SubRules[107] are not applied.

Examples

Example 1: the simplest case, labels for a cookbook.

Column 1	Column 2
{Label}	

Let this table has a rule with the properties: FirstRow[106] = 1, RowCount[106] = 1, **CopyColCount** = 1, all other Copy* properties are zeros.

Let the data query for this rule returns records having the following values in the 'Label' field: 'Soups', 'Salads', 'Main Dishes', 'Cakes', 'Beverages'.

The result is:

Column 1	Column 2
Soups	Salads
Main Dishes	Cakes
Beverages	

Example 2:

Staff						
{Name}		*empty*		*empty*		
{Title}	Phone: {Phone}					

Let this table has a rule with the properties: FirstRow[106] = 1, RowCount[106] = 2, **CopyColCount** = 1, **CopyFirstCol** = 1, **CopySpacingColCount** = 1

Let the data query for this rule returns records about 5 persons: Alice, Bob, Carol, Dave, Eve.

The result is:

Staff						
Alice		**Bob**		**Carol**		
Director General	Phone: 123456	Director (Development)	Phone: 234567	Director (Planning)	Phone: 345678	
Dave		**Eve**		*empty*		
Director (Finance)	Phone: 456789	Internal Auditor	Phone: 567890			

Data fields in other cells

The rest of cells belonging to the rule (cells to the left of the source cells, cells to the right of the last copy, cells between copies) may also contain data fields. The report generator fills them in the following way:

- the leftmost cells are filled using data from the record corresponding to the leftmost copy;
- the rightmost cells are filled using data from the record corresponding to the rightmost copy;
- cells between copies are filled using data from the record corresponding to the copy to the left of them.

The table below shows record indexes used to fill the table from the previous example:

this cell does not belong to the rule									
0	0		0	1		1	2		2
	0	0		1	1		2	2	
3	3		3	4		4	4		4
	3	3		4	4		4	4	

Default values:

0

See also

- Definitions of Report Workshop terms[10]

6.1.3.11.1.4 TRVRow GenerationRule.KeyFieldNames

A semicolon-delimited list of fields used to distinguish records in cross-tab reports.

```
property KeyFieldNames: TRVUnicodeString;
```

This property is used only when building a cross-tab report. Otherwise, it is ignored.

All records (returned by en execution of DataQuery[105]) having equal values of these fields are mapped to the same table row.

If **KeyFieldNames** is empty, all records are mapped to a single row.

If a field listed in **KeyFieldNames** does not exists in results of DataQuery[105] execution, this field is ignored, and OnError[40] event occurs with the parameter *rvrgeCrossTabUnknownField*.

Fields listed in the property may have one of the following types[161]: *rvrftInteger, rvrftBoolean, rvrftDateTime, rvrftDate, rvrftTime, rvrftText, rvrftFloat*. Otherwise, this field is ignored, and OnError[40] event occurs with the parameter *rvrgeFieldUnsupportedType*.

If a field listed in this property has *rvrftText* type, its values are compared using CaseSensitive[112] property.

Default value:

" (empty string)

Example:

'id1;id2'

See also:

- information about building a cross-tab report in the topic on TRVCrossTab[84]

6.1.3.12 TRVRowGenerationRules

TRVRowGenerationRules is a collection of row generation rules for a report table[80].

Unit RVReportTable;

Syntax

```
TRVRowGenerationRules = class (TOwnedCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection

Description

TRVRowGenerationRules is a class of TRVReportTableItemInfo[80].RowGenerationRules[82] property. It is a collection of TRVRowGenerationRule[111] items.

Each item in this collection has SubRules[107] property of TRVRowGenerationNestedRules[111] class. It is a collection of TRVRowGenerationNestedRule[109] items. Each item in its order has its own SubRules[107] property.

6.1.3.12.1 Properties

6.1.3.12.1.1 TRVRow GenerationRules.Items

Lists the items in the collection.

```
property Items[Index: Integer]: TRVRowGenerationRule[111]; default;
```

Use **Items** to access individual items in the collection.

6.2 Shape - TRVShapeItemInfo

A class of objects in TRichView documents: a geometric shape.

This object is not related to reporting, it simply reuses shape drawing functions implemented for value visualizers[119].

Unit RVReportShapeItem;

Syntax

```
TRVShapeItemInfo = class (TRVRectItemInfo)
```

Hierarchy

TObject
TPersistent
TCustomRVItemInfo
TRVNonTextItemInfo
TRVSimpleRectItemInfo
TRVRectItemInfo

Description

This object can draw shapes such as polygons, stars, emoticons, flags.

The shape is defined ShapeProperties[119]. Its size is specified in Width and Height[119] and EqualSides[117] properties.

The shape is colored using Color[117], StartColor[119], GradientType[117], Opacity[117] properties. It is outlined using LineColor[118], LineWidth[118], LineUsesFillColor[118] properties.

The object background is filled using BackgroundColor and BackgroundOpacity[117] properties.

The StyleNo of this item is rvsShape (-211).

See also

- TRVShape[68] component

6.2.1 Properties

6.2.1.1 TRVShapeItemInfo.Alt

Text representation of this item.

```
Alt: String;
```

This property is used when exporting this item to HTML.

Use TRichViewEdit.SetCurrentItemExtraStrProperty to change value of these properties as an editing operation (*rvespAlt*).

Default value

" (empty string)

6.2.1.2 TRVShapeItemInfo.BackgroundOpacity

Opacity of item background.

```
BackgroundOpacity: TRVOpacity;
```

The property must be in range from 0 (transparent) to 100000 (opaque).

Background color is specified in BackgroundColor property (inherited from TRVRectItemInfo).

Opacity is used only if shapes are drawn using GDI+ (in Delphi XE2+).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcBackgroundOpacity*[166]).

Default value

- 100000 (i.e. 100%)

6.2.1.3 TRVShapeItemInfo.Color, Opacity

Fill color and opacity for the shape.

```
Color: TColor;
Opacity: TRVOpacity;
```

If GradientType[117] = *rvgtNone*, the shape is filled with this color. Otherwise, this color is used as an ending gradient color (the starting color is StartColor[119]).

Opacity must be in range from 0 (transparent) to 100000 (opaque).

Gradient and opacity are used only if shapes are drawn using GDI+ (in Delphi XE2+).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of these properties as an editing operation (*rveipcShapeColor, rveipcShapeOpacity*[166]).

Default values

- Color: $C68E63
- Opacity: 100000 (i.e. 100%)

6.2.1.4 TRVShapeItemInfo.EqualSides

Specifies whether the shape has the same width and height.

```
EqualSides: Boolean;
```

If *False*, the shape is drawn in the rectangle Width[119] × Height[119].

If *True*, the shape is drawn in the square having the side min(Width[119], Height[119]).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcEqualSides*[166]).

Default value

True

6.2.1.5 TRVShapeItemInfo.GradientType

Type of gradient fill for shapes.

```
GradientType: TRVGradientType[160];
```

Gradient is drawn from StartColor[119] to Color[117].

If **GradientType** = *rvgtNone*, the shape is filled with Color[117].

Gradient is used only if shapes are drawn using GDI+ (in Delphi XE2+).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcGradientType*[166]).

Default value

rvgtNone

6.2.1.6 TRVShapeItemInfo.LineColor

Line color

```
LineColor: TColor
```

Line color can be auto-calculated basing on Color[117], see LineUsesFillColor[118].

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcLineColor*[166]).

Default value

clBlack

See also

- LineWidth[118]

6.2.1.7 TRVShapeItemInfo.LineUsesFillColor

Specifies how shapes are outlined.

```
property LineUsesFillColor: Boolean;
```

If **LineUsesFillColor** = *False*:

Shapes are with LineColor[118].

If **LineUsesFillColor** = *True*:

If Color[117] = *clNone*, LineColor[118] is used. Otherwise, if GradientType[117] <> *rvgtNone*, shapes are outlined with Color[117]; for a flat fill, they are outlined with twice darker color.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcLineWidth*[166]).

Default value

False

6.2.1.8 TRVShapeItemInfo.LineWidth

Line width, in pixels or twips (depending on RVStyle.Units)

```
LineWidth: TRVStyleLength;
```

Assign 0 to hide lines.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcLineWidth*[166]).

Default value

1

See also

- LineColor[118]

6.2.1.9 TRVShapeItemInfo.ShapeProperties

Specifies main properties of a shape.

```
property ShapeProperties: TRVReportShapeProperties[169];
```

This property contains sub-properties defining the shape type, rotation angle and other properties.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of sub-properties as an editing operation (*rveipcShapeType, rveipcPointCount, rveipcMiddlePercent, rveipcStartAngle*[166]).

6.2.1.10 TRVShapeItemInfo.StartColor

Start color for gradient fill.

```
property StartColor: TColor;
```

Gradient uses **StartColor** and Color[117].

Gradient is used only if shapes are drawn using GDI+ (in Delphi XE2+).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcStartShapeColor*[166]).

Default value

clWhite

6.2.1.11 TRVShapeItemInfo.Width, Height

Shape size, in pixels or twips (depending on RVStyle.Units)

```
Width, Height: TRVStyleLength;
```

The full item width is **Width** + (Spacing + BorderWidth + OuterHSpacing) * 2.

The full item height is **Height** + (Spacing + BorderWidth + OuterVSpacing) * 2.

Use TRichViewEdit.SetCurrentItemExtraIntProperty to change value of these properties as an editing operation (*rvepImageWidth, rvepImageHeight*).

Default value

48

See also

- EqualSides[117]

7 Data Visualizers

Data visualizers may be associated with cells[101] of report tables[80].

There are two group of visualizers:

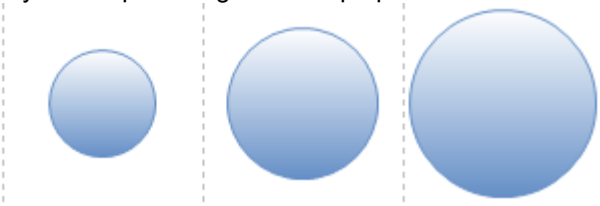
- for changing background colors depending on values
- for displaying diagrams that visualize values (a diagram is drawn on top on the standard cell background, but below the cell content)

The same cell may have both a color changer and a diagram.

This feature is similar to conditional formatting that can be found in Microsoft Excel, however, there are important differences:

- in Excel, you can visualize the same value as entered in the cell; in Report Workshop, this value is specified separately. So you can, for example, display a price, but show the icon identifying the price change; or show a movie title, but visualize its rating
- Report Workshop does not use raster images to visualize values, so diagrams may be of any size and printed without losing quality

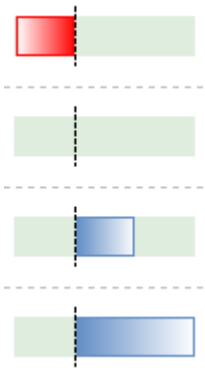
All visualizers are inherited from TRVReportCustomValueVisualizerBase[134]. It defines the string that must be evaluated to get values for each cell, and specify how minimal and maximal values are calculated.
The first group of visualizers (color changers) are represented by a single class: TRVReportColorChangerItem[127]
.
The second group of visualizers (value visualizers displaying diagrams) include:
TRVReportAreaSizeVisualizer[121]: displays a shape having the area proportional to the value



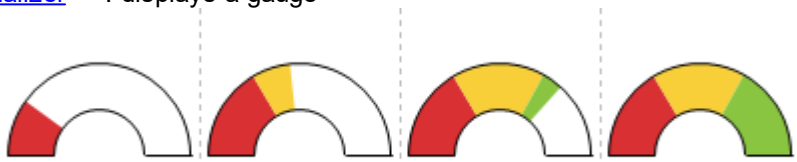
TRVReportColoredShapeVisualizer[129]: displays a shape having colors and rotation depending on the value



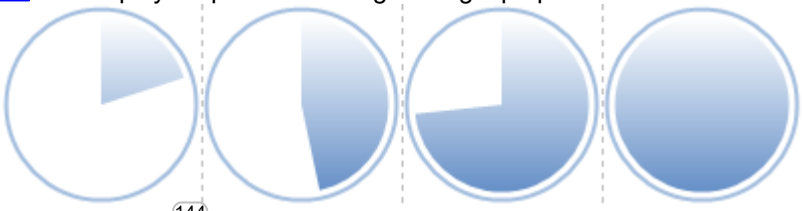
TRVReportBarVisualizer[123]: displays a horizontal or vertical bar having length proportional to the value



TRVReportGaugeVisualizer[136]: displays a gauge



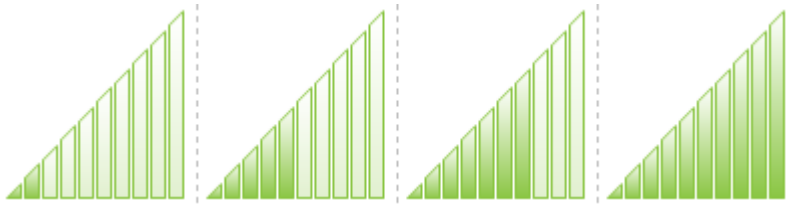
TRVReportPieVisualizer[141]: displays a pie slice having an angle proportional to the value



TRVReportShapeRepeaterVisualizer[144]: displays the count of shapes proportional to the value



TRVReportSignalStrengthVisualizer[148]: displays a diagram that is usually used to show a signal strength or a volume



7.1 TRVReportAreaSizeVisualizer

TRVReportAreaSizeVisualizer visualizes positive numeric values by displaying shapes having areas proportional to these values.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportAreaSizeVisualizer = class (TRVReportCustomValueVisualizer[133])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[134]
TRVReportCustomValueVisualizer[133]

Description

This visualizer displays shapes (circles or squares) having areas proportional to displayed values.

This visualizer displays only positive values (nothing is displayed for negative values).

The shape is specified in the Shape[122] property.

Shapes are drawn using Color[133], LineColor[133], and Gradient[122] properties.

Sizes for the minimal and the maximal values can be specified in MinSize and MaxSize[122] properties.

Examples

These examples display the following values:

0.0	33.3	66.7	100.0

MinValue[135] = 0, MaxValue[135] = 100.

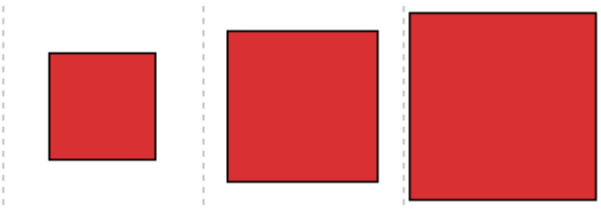
Example1:

Shape[122] = *rvrsshCircle*, Gradient[122] = *True*, Color[133] = LineColor[133] = $C68E63



Example 2

Shape[122] = *rvrsshSquare*, Gradient[122] = *False*, Color[133] = $3330D9, LineColor[133] = *clBlack*.



See also

- TRVReportTableItemInfo[80].BackgroundVisualizers[81]

7.1.1 Properties

7.1.1.1 TRVReportAreaSizeVisualizer.Gradient

Specifies whether shapes must be painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, shapes are filled with Color[133].

If *True*, shapes are filled with a gradient from Color[133] (at bottom) to a twice lighter color (at top).

Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+).

Default value

False

7.1.1.2 TRVReportAreaSizeVisualizer.MinSize, MaxSize

These properties define the minimal and maximal sizes for displayed shapes.

```
property MinSize: TRVStyleLength;
property MaxSize: TRVStyleLength;
```

These values are measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

MinSize specifies the size of a shape corresponding to MinValue[135] (or to 0, if MinValue[135] is negative).

The maximal shape size (corresponding to MaxValue[135]) is calculated from cell sizes, but it cannot be greater than **MaxSize** (if **MaxSize** is positive).

MaxValue cannot be less than **MinValue** (otherwise, it is ignored).

Shape	The size defines
rvrsshCircle	diameter
rvrsshSquare	side length

Default values

0 (meaning no shape for MinValue[135], and maximal possible shape size for MaxValue[135])

7.1.1.3 TRVReportAreaSizeVisualizer.Shape

A shape to display.

```
type
  TRVReportSimpleShape = (rvrsshCircle, rvrsshSquare);
property Shape: TRVReportSimpleShape;
```

Shape	Meaning

rvrsshCircle	circle
rvrsshSquare	square

Default value

rvrsshCircle

7.2 TRVReportBarVisualizer

TRVReportBarVisualizer visualizes numeric values by displaying bars having lengths proportional to these values.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportBarVisualizer = class (TRVReportCustomValueVisualizer[133])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[134]
TRVReportCustomValueVisualizer[133]

Description

This visualizer shows bars having lengths proportional to visualized values.

Bar lengths are calculated for absolute values.

AxisPosition[124] defines how negative values and axis are displayed.

Bars can be directed from left to right, from right to left, from top to bottom, from bottom to top, depending on BarDirection[126].

Bars for positive values are drawn with Color and LineColor[133], bars for negative values are drawn using NegativeColor and NegativeLineColor[127]. Background can be solid or gradient[127].

Optionally, a background is drawn using BackgroundColor and BackgroundOpacity[125] properties.

An axis, if visible, is drawn using AxisColor[124].

Widths of bars are defined in FitWidth[126] and MaxWidth[127] properties.

Examples

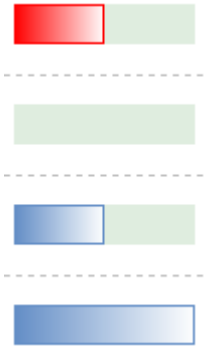
These examples display the following values:

-50	0	50	100

MinValue[135] = -50, MaxValue[135] = 100.

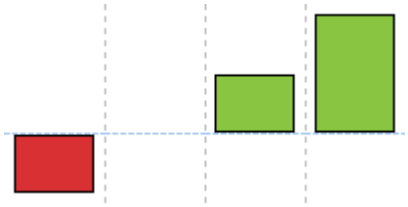
Example 1:

AxisPosition[124] = *rvrbapSide*, Gradient[127] = *True*, BarDirection[126] = *rvrbdRight* (default), Color[133] = $C68E63 (default), NegativeColor[127] = *clRed* (default), LineColor = Color[133], NegativeLineColor = NegativeColor[127], Margin[134] = 5, BackgroundColor[125] = *clMoneyGreen*



Example 2

AxisPosition[124] = *rvrbapAuto* (default), Gradient[127] = *False* (default), BarDirection[126] = *rvrbdUp*, Color[133] = $41C589, NegativeColor[127] = $3330D9, LineColor[133] = *clBlack* (default), NegativeLineColor[127] = *clNone* (default), Margin[134] = 5, , BackgroundColor[125] = *clNone* (default), AxisColor[124] = *clSkyBlue*



See also

- TRVReportTableItemInfo[80].BackgroundVisualizers[81]

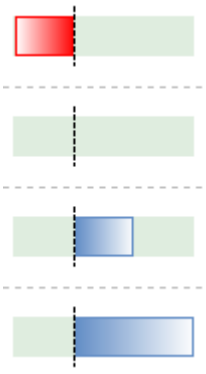
7.2.1 Properties

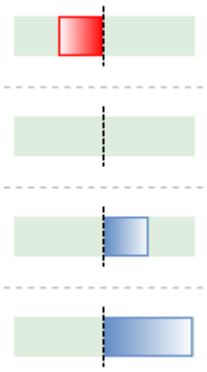
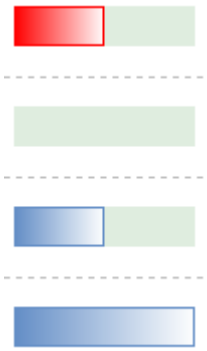
7.2.1.1 TRVReportBarVisualizer.AxisPosition, AxisColor

These properties specify the axis position and color.

```
type
  TRVReportBarAxisPosition = (rvrbapAuto, rvrbapMiddle, rvrbapSide);
property AxisPosition: TRVReportBarAxisPosition;
property AxisColor: TColor;
```

AxisPosition is taken into account only if MinValue[135] < 0. Otherwise, all bars are started from a side (for example, if BarDirection[126] = *rvrbdRight*, bars are started from the left side), and an axis is not drawn.

AxisPosition	Meaning	Example
rvrbapAuto	Axis position is calculated automatically from MinValue[135] and MaxValue[135] Bars for positive and negative values are shown in the opposite directions.	

rvrbapMiddle	Axis is always at the middle. Bars for positive and negative values are shown in the opposite directions.	
rvrbapSide	Axis is at a side (and is not shown) Bars for positive and negative values are shown in the same direction	

In the table above, examples use the following data:

-50	0	50	100

AxisColor specifies the color for drawing an axis line. If **AxisColor** = *clNone*, an axis is not drawn. Axis is drawn on margins[134] as well.

Default values:

- AxisPosition: *rvrbapAuto*
- AxisColor: *clBlack*

7.2.1.2 TRVReportBarVisualizer.BackgroundColor, BackgroundOpacity

A background color for the visualizer.

```
property BackgroundColor: TColor;
property BackgroundOpacity: TRVOpacity;
```

Background color is used to fill space "below" the bar.

Opacity is defined in 1/1000 of percent, it must be in the range 0..100000. Opacity is used only if visualizers are drawn using GDI+ (in Delphi XE2+).

Example (BackgroundColor = *clMoneyGreen*)**:**



You can see that the background area has the same width as the bar. Margins[134] are not filled with background.

Default value

- BackgroundColor: *clNone* (no background)
- BackgroundOpacity: 50000 (i.e. 50%), ignored for BackgroundColor = *clNone*

7.2.1.3 TRVReportBarVisualizer.BarDirection

A direction for bars

```
type
  TRVReportBarDirection = (rvrbdRight, rvrbdLeft, rvrbdUp, rvrbdDown);
property BarDirection: TRVReportBarDirection;
```

Value	Meaning
rvrbdRight	from left to right
rvrbdLeft	from right to left
rvrbdUp	from bottom to top
rvrbdDown	from top to bottom

This direction is used to display bars for positive values. Bars for negative values may be directed in the opposite side, see AxisPosition[124].

Default value

rvrbdRight

7.2.1.4 TRVReportBarVisualizer.FitWidth

Specifies whether all bars have the same width.

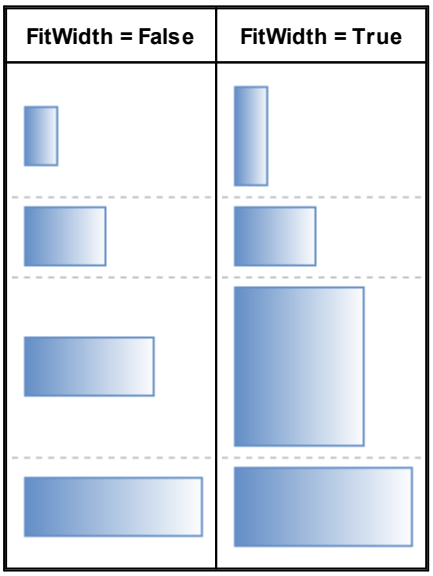
```
property FitWidth: Boolean;
```

If **FitWidth** = *False*, all bars have the same width, to fit the narrowest cell.

If **FitWidth** = *True*, width of each bar is calculated individually, to fit the container cell.

Bar widths may be limited by MaxWidth[127].

Example:



In this example, Margin[134] = 5

Default value

True

7.2.1.5 TRVReportBarVisualizer.Gradient

Specifies whether bars must be painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, bars are filled with Color[133].

If *True*, bars are filled with a gradient from Color[133] to a twice lighter color, in the direction specified in BarDirection[126].

For negative values, NegativeColor[127] is used instead, and the direction may be opposite, see AxisPosition[124].

Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+).

Default value

False

7.2.1.6 TRVReportBarVisualizer.MaxWidth

Specifies the maximal bar width.

```
property MaxWidth: TRVStyleLength;
```

If **MaxWidth** > 0, it defines the maximal bar width.

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

Default value

0

7.2.1.7 TRVReportBarVisualizer.NegativeColor, NegativeLineColor

These properties specify colors for bars displayed for negative values.

```
property NegativeColor: TColor;
property NegativeLineColor: TColor;
```

NegativeColor is used to fill bars displayed for negative values.

NegativeLineColor is used to draw a rectangle around these bars. If **NegativeLineColor** = *clNone*, LineColor[133] is used.

Note: for bars displayed for positive values, Color and LineColor[133] are used

Default value

- NegativeColor: *clRed*
- NegativeLineColor: *clNone*

7.3 TRVReportColorChangerItem

TRVReportColorChangerItem is a class for color changers.

Unit RVReportColorChanger;

Syntax

```
TRVReportColorChangerItem = class (TRVReportCustomValueVisualizerBase[134])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[134]

Description

TRVReportColorChangerItem is a class of items in TRVReportTableItemInfo[80].BackgroundColorChangers[80].

If a report cell[101] is linked[102] to a color changer, this color changer changes the cell color and opacity according to a value[102] associated with this cell.

Cells are shaded with gradations of two or three colors that correspond to minimum, midpoint, and maximum thresholds.

This class introduces new properties:

- OpacityBegin, OpacityMiddle, OpacityEnd[128] – colors for defining a color scale
- ColorBegin, ColorMiddle, ColorEnd[128] – the same for opacity
- PercentMiddle[129] defines the midpoint.

The following properties are inherited:

- ValueString[135] – expression to evaluate for cells.
- MinValue, MaxValue, AutoMinValue, AutoMaxValue[135] – properties for calibration of the color scale

See also

- Definitions of Report Workshop terms[10]

7.3.1 Properties

7.3.1.1 TRVReportColorChangerItem Colors

These properties specify colors.

```
property ColorBegin: TColor
property ColorMiddle: TColor
property ColorEnd: TColor;
```

Cells are shaded with gradations of two or three colors that correspond to minimum, midpoint, and maximum thresholds.

ColorBegin is used for values equal to MinValue[135] (all lesser values are treated as MinValue)

ColorEnd is used for values equal to MaxValue[135] (all greater values are treated as MaxValue).

For other values:

- If **ColorMiddle** = *clNone*, the cell is shaded with gradations of **ColorBegin** and **ColorEnd**, proportionally to (value - MinValue) / (MaxValue - MinValue).
- If **ColorMiddle** <> *clNone*, the cell is shaded with gradations of **ColorBegin** and **ColorMiddle**, if it comes to the lower interval defined by PercentMiddle; or with the gradation of **ColorMiddle** and **ColorEnd** if it comes to the upper interval.

Default values

- ColorBegin: $7BBE63
- ColorMiddle: $84EBFF
- ColorEnd: $6B69F8

See also

- OpacityBegin, OpacityMiddle, OpacityEnd[128]
- PercentMiddle[129]

7.3.1.2 TRVReportColorChangerItem Opacities

These properties specify the scales of opacity.

```
property OpacityBegin: TRVOpacity;
property OpacityMiddle: TRVOpacity;
property OpacityEnd: TRVOpacity;
```

Cells are shaded with gradations of two or three opacities that correspond to minimum, midpoint, and maximum thresholds.

Opacities are defined in 1/1000 of percent, they must be in the range 0..100000.

OpacityMiddle is used only if ColorMiddle[128] <> clNone.

An opacity corresponding the given value is calculated similarly to colors[128].

Default value

100000 (100%)

See also

- ColorBegin, ColorMiddle, ColorEnd[128]
- PercentMiddle[129]

7.3.1.3 TRVReportColorChangerItem.PercentMiddle

Specify the location of the midpoint color and opacity.

```
property PercentMiddle: Single;
```

A value of this property must be greater than 0 and less than 100.

This property is used only if ColorMiddle[128] <> *clNone*.

The midpoint value is calculated as MinValue[135] + (MaxValue[135] - MinValue[135]) * **PercentMiddle** / 100. This value corresponds to ColorMiddle[128] and OpacityMiddle[128].

Values between MinValue and the midpoint are shaded between ColorBegin and ColorMiddle, OpacityBegin and OpacityMiddle.

Values between the midpoint and MaxValue are shaded between ColorMiddle and ColorEnd, OpacityMiddle and OpacityEnd.

Default value

50

See also

- OpacityBegin, OpacityMiddle, OpacityEnd[128]
- ColorBegin, ColorMiddle, ColorEnd[128]

7.4 TRVReportColoredShapeVisualizer

TRVReportColoredShapeVisualizer visualizes numeric values by applying color and rotation to a shape.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportColoredShapeVisualizer = class (TRVReportCustomShapeVisualizer[131])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[134]
TRVReportCustomValueVisualizer[133]
TRVReportCustomShapeVisualizer[131]

Description

The shape is defined in ShapeProperties[132].

A color and a rotation angle of this shape depend on the visualized value. The visualizer searches in ConditionalShapeProperties[131] for the item corresponding to the visualized value. If none of the items corresponds to the visualized value (it may happen for low values), the default color and rotation angle is used.

Shape size may be limited by MaxSize[131].

By default, the shape is inscribed in a square. You can stretch it by changing ShapeScaleX[133].

Examples

Example 1: ShapeProperties[132].Shape[170] = *rvrshArrow1*, ShapeProperties[132].StartAngle[170] = 180, angles in ConditionalShapeProperties[131] are from -60 to -180 (so arrows are rotated by 180°, 120°, 60°, 0°), Gradient[132] = *False*



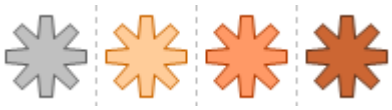
Example 2: The same settings, Gradient[132] = *True*



Example 3: ShapeProperties[132].Shape[170] = *rvrshCircle*, LineUsesFillColor[132] = *False*, LineColor[133] = *clSilver*



Example 4: ShapeProperties[132].Shape[170] = *rvrshBluntPointStar*, ShapeProperties[132].PointCount[169] = 8, Gradient[132] = *False*



Note: in all these examples, a visualized value is increased from left to right. The leftmost shape is drawn using Color[133] and ShapeProperties[132].StartAngle[170]. Colors and rotations of other shapes are defined in ConditionalShapeProperties[131]

See also

- TRVReportTableItemInfo[80].BackgroundVisualizers[81]

7.4.1 Properties

7.4.1.1 TRVReportColoredShapeVisualizer.AbsoluteValues

Specifies whether values in ConditionalShapeProperties[131] are defined as absolute values or as percent.

```
property AbsoluteValues: Boolean;
```

If *False*, all ConditionalShapeProperties[131][].Value[168]s are specified in percent.

If *True*, all ConditionalShapeProperties[131][].Value[168]s are specified in absolute values

Default value

False

7.4.1.2 TRVReportColoredShapeVisualizer.ConditionalShapeProperties

Specifies a set of shape properties that are applied depending on the visualized value.

```
property ConditionalShapeProperties: TRVConditionalShapePropertiesCollection[167];
```

The programmer must ensure that items in this collection are sorted in an ascending order by **ConditionalShapeProperties**[].Value[168].

The visualizer searches the item to apply using the following algorithm. Items are enumerated from the last one to the first one. If the comparison between the current item's Value[168] and the visualized value returns *True*, this item is used. If all comparisons return *False*, Color[133] and ShapeProperties[132].StartAngle[170] are applied.

If the *Index*-th item of this collection is used, a shape is displayed using **ConditionalShapeProperties**[*Index*].Color[168] and ShapeProperties[132].StartAngle[170] + **ConditionalShapeProperties**[*Index*].DeltaAngle[168].

Comparisons with the visualized value depend on AbsoluteValues[130] property.

- If AbsoluteValues[130] = *True*, values are compared as: *Visualized_Value* >= **ConditionalShapeProperties**[].Value[168]
- If AbsoluteValues[130] = *False*, values are compared as: (*Visualized_Value* - MinValue[135]) *100 / (MaxValue[135] - MinValue[135]) >= **ConditionalShapeProperties**[].Value[168])

Values less than MinValue[135] are treated as MinValue[135], values greater than MaxValue[135] are treated as MaxValue[135].

Example

Let this collection has two items:

- **ConditionalShapeProperties**[0].Value[168] = 3
- **ConditionalShapeProperties**[1].Value[168] = 7

AbsoluteValues[130] = *True*

In this case:

- for all values >=7, **ConditionalShapeProperties**[1] is applied
- for all values >=3 and <7, **ConditionalShapeProperties**[0] is applied
- for all values < 3, default properties are applied.

7.4.1.3 TRVReportColoredShapeVisualizer.MaxSize

Maximal possible size of shapes displayed by the visualizer.

```
property MaxSize: TRVStyleLength;
```

If **MaxSize** > 0, it specifies the the maximal possible diameter of an imaginary circle around shapes.

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

Default value

0

7.5 TRVReportCustomShapeVisualizer

TRVReportCustomShapeVisualizer is a base class for visualizers that draw shapes.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportCustomShapeVisualizer = class (TRVReportCustomValueVisualizer[133])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase 134
TRVReportCustomValueVisualizer 133

Description

This class is not used directly. The following visualizers are inherited from this class:

- TRVReportColoredShapeVisualizer 129
- TRVReportShapeRepeaterVisualizer 144

TRVReportCustomShapeVisualizer introduces properties:

- ShapeProperties 132 – shape type and other properties,
- LineUsesFillColor 132, Gradient 132 specify how shapes are outlined and filled.

See also

- TRVReportTableItemInfo 80.BackgroundVisualizers 81

7.5.1 Properties

7.5.1.1 TRVReportCustomShapeVisualizer.Gradient

Specifies whether shapes are painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, shapes are filled with Color 133.

If *True*, shapes are filled with a gradient from Color 133 (at bottom) to a twice lighter color (at top).

Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+).

Default value

True

7.5.1.2 TRVReportCustomShapeVisualizer.LineUsesFillColor

Specifies how shapes are outlined.

```
property LineUsesFillColor: Boolean;
```

If **LineUsesFillColor** = *False*:

Shapes are with LineColor 133.

If **LineUsesFillColor** = *True*:

If the corresponding fill color is equal to *clNone*, LineColor 133 is used. Otherwise, if Gradient 132 = *True*, shapes are outlined with Color 133; if Gradient 132 = *False*, they are outlined with twice darker colors.

Note: for Delphi versions prior to 2010, this option simply uses Color as LineColor.

Default value

True

7.5.1.3 TRVReportCustomShapeVisualizer.ShapeProperties

Specifies main properties of a shape

```
property ShapeProperties: TRVReportShapeProperties 169;
```

This property contains sub-properties defining the shape type, rotation angle and other properties.

7.5.1.4 TRVReportCustomShapeVisualizer.ShapeScaleX

Horizontal scale ratio for the shape, percent

```
property ShapeScaleX: Integer;
```

"Shape width" is calculated as "shape height" * ShapeScaleX / 100.

7.6 TRVReportCustomValueVisualizer

TRVReportColorChangerItem is a base class for value visualizers.

Unit RVReportColorChanger;

Syntax

```
TRVReportCustomValueVisualizer = class (TRVReportCustomValueVisualizerBase[134])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[134]

Description

This class is not used directly. All value visualizers are inherited from this class.

TRVReportCustomValueVisualizer is a base class for items in TRVReportTableItemInfo[80].BackgroundVisualizers[81].

This class introduces new properties, common for all value visualizers:

- HAlign, VAlign[134] – alignment of images displayed by the visualizers relative to respective cells.
- Margin[134] – space around the image.
- Color, LineColor[133] – colors of shapes drawn by the visualizer.

The following properties are inherited:

- ValueString[135] – expression to evaluate for cells.
- MinValue, MaxValue, AutoMinValue, AutoMaxValue[135] – properties for calibration of the visualizer.

See also

- Definitions of Report Workshop terms[10]

7.6.1 Properties

7.6.1.1 TRVReportCustomValueVisualizer.Color, LineColor

These properties specify colors.

```
property Color: TColor
property LineColor: TColor
```

The use of these properties depends on the specific visualizer. But any visualizer draws some shape, and it uses **Color** to fill its interior, and **LineColor** to draw its outline.

Some visualizers have Gradient property. If Gradient = *True*, shapes are painted with a gradient fill (color 1: **Color**; color2: a color twice lighter than **Color**)

Some visualizers have LineUsesFillColor property. If LineUsesFillColor = *True* and **Color** <> clNone, an outline color for shapes is calculated basing on **Color**:

- for gradient fills, the outline color is **Color**
- for plain color fills, the online color is twice darker than **Color**

Default values

- Color: $C68E63
- LineColor: clBlack

7.6.1.2 TRVReportCustomValueVisualizer.HAlign, VAlign

These properties define positions of images displayed by the visualizer, relative to the respective cells.

```
property HAligb: TRVCellHAlign; // rvcLeft, rvcCenter or rvcRight
property VAlign: TRVCellVAlign2; // rvcTop, rvcMiddle or rvcBottom
```

Image positioning ignores cell padding (the minimal distance to the cell border is defined in Margin[134])

Default values

rvcCenter, rvcMiddle

7.6.1.3 TRVReportCustomValueVisualizer.Margin

Sets the space around images displayed by the visualizer.

```
property Margin: TRVStyleLength;
```

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

Default value

0

7.7 TRVReportCustomValueVisualizerBase

TRVReportCustomValueVisualizerBase is a base class for color changers and value visualizers.

Unit RVReportColorChanger;

Syntax

```
TRVReportCustomValueVisualizerBase = class (TCollectionItem)
```

Hierarchy

TObject
TPersistent
TCollectionItem

Description

This class is not used directly.

Two groups of objects are inherited from this class:

- color changers,
- value visualizers.

Color changers are represented by a single class TRVReportColorChangerItem[127]. This is a class of items in TRVReportTableItemInfo[80].BackgroundColorChangers[80]. Color changers allow modifying background color and opacity for cells depending on the value of expression[135] calculated for these cells.

There are many *value visualizers* available, all of them are inherited from TRVReportCustomValueVisualizer[133]. They are classes of items in TRVReportTableItemInfo[80].BackgroundVisualizers[81]. Value visualizers allow showing values at the background of cells in different ways: as shapes of different size or color, histograms, gauges, pies, etc.

TRVReportCustomValueVisualizerBase contains properties necessary for all color changers and visualizers:

- ValueString[135] – expression to evaluate for cells
- MinValue, MaxValue, AutoMinValue, AutoMaxValue[135] – properties for calibration of visualizers

See also

- Definitions of Report Workshop terms 10

7.7.1 Properties

7.7.1.1 TRVReportCustomValueVisualizerBase.Id

An unique identifier of this item in the collection.

```
property Id: TRVValueVisualizerId 165;
```

This value is unique for all items in the collection. A valid identifier is a zero or positive value.

This property is read-only, however, it is stored with the item when the table is saved in RVF.

When you call Assign method to copy one item to another, **Id** is not copied. Use AssignIdFrom method to copy **Id** from one item to another (do not do it for items in the same collection, otherwise this property will not be unique).

7.7.1.2 TRVReportCustomValueVisualizerBase.Min-Max

Values for the visualizer calibration.

```
property MinValue: Variant;
property MaxValue: Variant;
property AutoMinValue: Boolean;
property AutoMaxValue: Boolean;
```

To show values, value visualizers and color changers need calibration, i.e. they must know minimal and maximal values.

For example, the pie visualizer displays an empty circle for the minimal value, and a full circle for the maximal value.

By default, these properties have the following initial values: **MinValue**=**MaxValue**=*Null*, **AutoMinValue**=**AutoMaxValue**=*True*. This means that **MinValue** and **MaxValue** are calculated automatically on the set of values to visualize.

You can assign some initial numeric value to **MinValue** (or **MaxValue**). In this case, they still will be calculated automatically, but the result cannot be greater than the initial value of **MinValue** (or less than the initial value of **MaxValue**).

If you assign **AutoMinValue** (or **AutoMaxValue**) = *False*, the report generator will not change them. In this case, you must assign a numeric value to **MinValue** (or **MaxValue**).

If you assign **MinValue** and **MaxValue**, you must ensure that **MinValue** < **MaxValue**.

All values less than **MinValue** are treated as **MinValue**, all values greater than **MaxValue** are treated as **MaxValue**.

Default values

Null, Null, True, True (but these values may be different in some visualizers)

7.7.1.3 TRVReportCustomValueVisualizerBase.ValueString

An expression to calculate.

```
property ValueString: TRVUnicodeString;
```

When generating reports, TRVReportGenerator 46 evaluates an expression stored in this string and store in the corresponding report table cell. Later, the result of this evaluation will be visualized on this cell background.

The syntax is similar to data fields 12, but:

- curly brackets are not used;

- cross-tab header fields are not supported
- format and type are not supported

Syntax:

<value string> ::= <full data field name> | % <variable name> | <function name> (<param>)

Examples:

- 'Myquery:Myfield' – value of 'Myfield' field from the result of 'Myquery' row generation rule
- '%count' – value of 'count' variable
- 'sum(salary)' – sum of 'salary' field values for cross-tab reports (a set of values to summarize depends on the cell location, it may be in a summary column or in a summary row)

Default value

" (empty string)

7.8 TRVReportGaugeVisualizer

TRVReportGaugeVisualizer visualizes numeric values by displaying them on a gauge.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportGaugeVisualizer = class (TRVReportCustomValueVisualizer)
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase
TRVReportCustomValueVisualizer

Description

This visualizer shows a gauge and fills it from the left side to the position corresponding to the value.

This visualizer is useful for displaying values that can be separated into "good", "neutral" and "bad". They are displayed as "green", "yellow" and "red" areas of a gauge.

By default, large values are "bad" values, but you can reverse it using RedLowValues property.

You can define relative sizes and colors of these areas.

This visualizer overrides initial values of MinValue, MaxValue, AutoMinValue, AutoMaxValue properties: MinValue = 0, MaxValue = 100, and they are not modified while generating reports.

All values less than MinValue are displayed as MinValue, all values greater than MaxValue are displayed as MaxValue.

Gauges can be filled using all area colors, or by a single color, depending on SingleColorMode.

Gauge size may be limited by MaxSize.

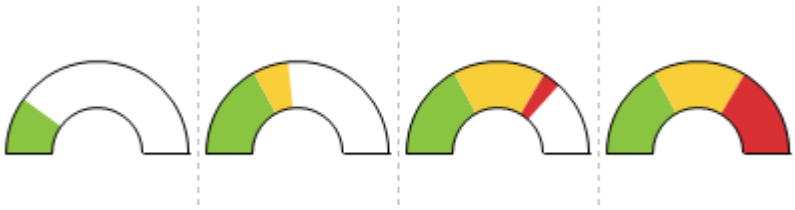
Examples

The examples use the following data:

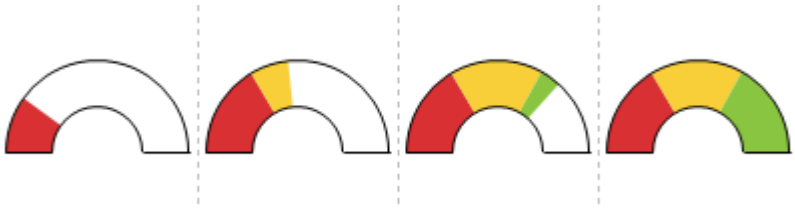
20	46.7	73.3	100

MinValue = 0, MaxValue = 100.

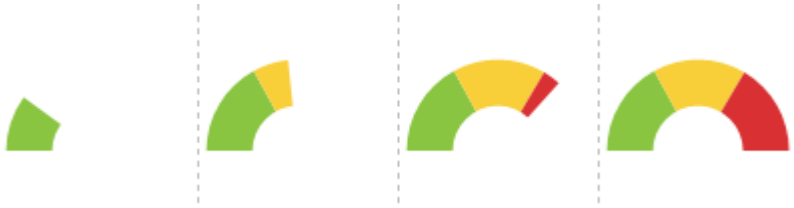
Example 1: Default property values



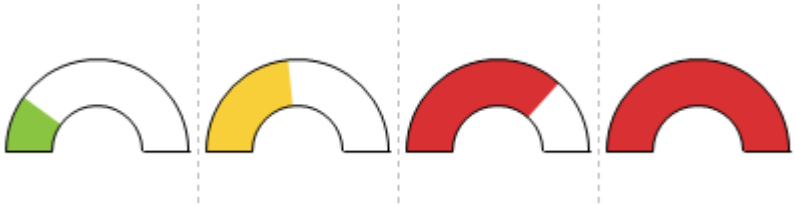
Example 2: RedLowValues[139] = *True*



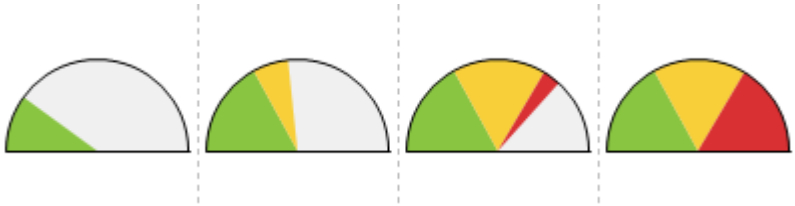
Example 3: LineColor[133] = *clNone*



Example 4: SingleColorMode[140] = *True*



Example 5: MiddleColor[138] = *clNone*, Color[138] = $F0F0F0



See also

- TRVReportTableItemInfo[80].BackgroundVisualizers[81]

7.8.1 Properties

7.8.1.1 TRVReportGaugeVisualizer Area Colors

The properties define colors of the gauge areas.

```
property GreenAreaColor: TColor;
property YellowAreaColor: TColor;
property RedAreaColor: TColor;
```

The range of values from MinValue to MaxValue[139] are separated into three areas: "green", "yellow", "red". Their colors are defined in these properties.

It's supposed that "green" area contains good values, "red" value contains bad values, "yellow" area contains neutral values.

Default values:

- GreenAreaColor: $41C589
- YellowAreaColor: $39CFF8
- RedAreaColor: $3330D9

See also

- RedLowValues 139
- SingleColorMode 140
- RedAreaPercent, YellowAreaPercent 138

7.8.1.2 TRVReportGaugeVisualizer Area Percents

The properties define relative sizes of the gauge values.

```
property RedAreaPercent: Integer;
property YellowAreaPercent: Integer;
```

The range of values from MinValue to MaxValue 139 are separated into three areas: "green", "yellow", "red". Their relative size is defined in these properties.

It's supposed that "green" area contains good values, "red" value contains bad values, "yellow" area contains neutral values.

The relative size of the "green" area is calculated as 100 - **YellowAreaPercent** - **RedAreaPercent**.

The areas can be ordered as green-yellow-red or red-yellow-green, depending on RedLowValues 139.

Default values:

33, 33 (so the green area is 34%)

7.8.1.3 TRVReportGaugeVisualizer.Color, MiddleColor

Background colors for gauges.

```
property Color: TColor;
property MiddleColor: TColor;
```

Color is inherited from TRVReportCustomValueVisualizer 133. See the description of the original property 133. TRVReportGaugeVisualizer overrides the initial value of this property.

Color is used to fill the rest of a gauge, not filled by colors corresponding to a value.

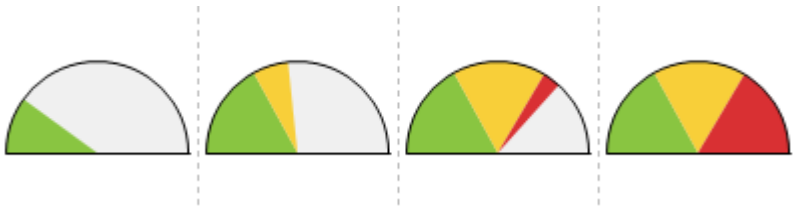
MiddleColor is used to fill the center of a gauge. If **MiddleColor** = *clNone*, gauges look like a semicircles, otherwise they look like arcs of semicircles.

Example:

MiddleColor = *clWhite*, **Color** = $F0F0F0



MiddleColor = *clNone*, **Color** = $F0F0F0



Default values

- Color: *clWhite*
- MiddleColor: *clWhite*

7.8.1.4 TRVReportGaugeVisualizer.MaxSize

Maximal diameter of gauge circles.

```
property MaxSize: TRVStyleLength;
```

All gauges displayed by the visualizer have the same diameter. It is calculated to fit the smallest cell.

If **MaxSize** > 0, it specifies the maximal possible diameter.

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

Default value

0

7.8.1.5 TRVReportGaugeVisualizer.Min-Max

Values for the visualizer calibration.

```
property MinValue: Variant;
property MaxValue: Variant;
property AutoMinValue: Boolean;
property AutoMaxValue: Boolean;
```

These properties are inherited from TRVReportCustomValueVisualizerBase[134]. See the description of the original properties[135].

TRVReportGaugeVisualizer overrides the initial values of these properties.

Default values

0, 100, False, False

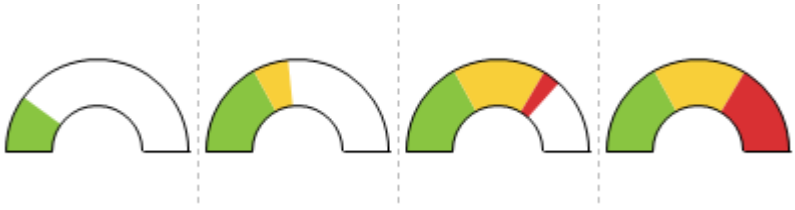
7.8.1.6 TRVReportGaugeVisualizer.RedLowValues

This property defines the order of areas of a gauge.

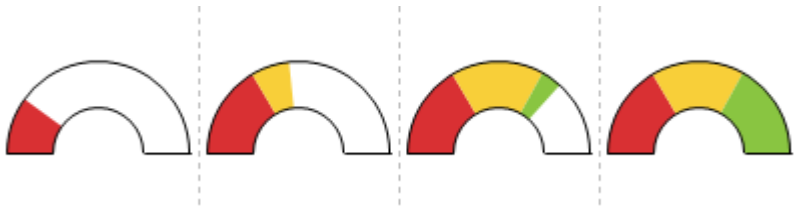
```
property RedLowValues: Boolean;
```

The range of values from MinValue to MaxValue[139] are separated into three areas: "green", "yellow", "red".

If **RedLowValues** = *False*, areas are ordered: "green", "yellow", "red".



If **RedLowValues** = *True*, areas are ordered: "red", "yellow", "green".



The examples use the following data:

20	46.7	73.3	100

MinValue[139] = 0, MaxValue[139] = 100.

It's supposed that "green" area contains good values, "red" value contains bad values, "yellow" area contains neutral values.

Default value

False

See also

- RedAreaPercent, YellowAreaPercent[138]
- RedAreaColor, YellowAreaColor, GreenAreaColor[137]
- SingleColorMode[140]

7.8.1.7 TRVReportGaugeVisualizer.SingleColorMode

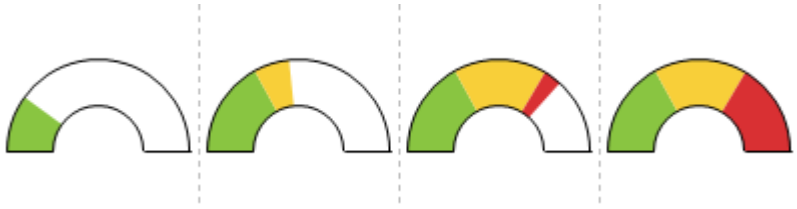
This property specifies whether values are displayed using all colors or a single color.

```
property SingleColorMode: Boolean;
```

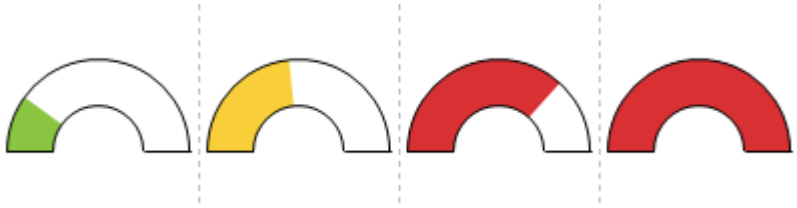
The range of values from MinValue to MaxValue[139] are separated into three areas: "green", "yellow", "red".

A gauge is filled from the left side to the position corresponding to a value.

If **SingleColorMode** = *False*, this fill uses all area colors.



If **RedLowValues** = *True*, this fill uses a single color corresponding to the value's area.



The examples use the following data:

20	46.7	73.3	100

MinValue[139] = 0, MaxValue[139] = 100.

It's supposed that "green" area contains good values, "red" value contains bad values, "yellow" area contains neutral values.

Default value

False

See also

- RedAreaPercent, YellowAreaPercent 138
- RedAreaColor, YellowAreaColor, GreenAreaColor 137
- RedLowValues 139

7.9 TRVReportPieVisualizer

TRVReportPieVisualizer visualizes numeric values by displaying pie slices (i.e. sectors of circle) having lengths of arcs proportional to these values.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportPieVisualizer = class (TRVReportCustomValueVisualizer 133)
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase 134
TRVReportCustomValueVisualizer 133

Description

This visualizer displays a pie diagram for each value. Each such diagram contains one slice having size of an arc proportional to the corresponding value.

Empty circle corresponds to MinValue 135, full circle corresponds to MaxValue 135. Values less than MinValue are displayed as MinValue, values greater than MaxValue are displayed as MaxValue.

Pie slices are filled with Color 133 and Gradient 142. Circles around slices are drawn using LineColor 142 and LineWidth 143. Gap 142 is a difference between radii of these circles and of pies.

Sizes of pie slices can be changed either smoothly or by steps, depending on PartsCount 143 property.

Size of these diagrams may be limited by MaxSize 143.

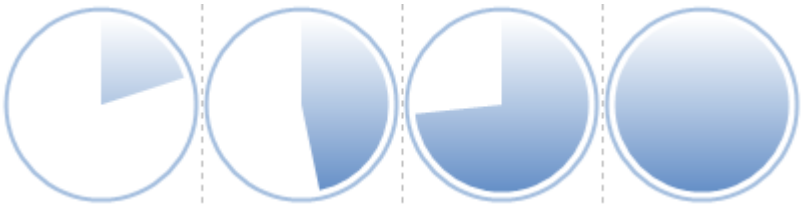
Example

The example uses the following data:

20	46.7	73.3	100

MinValue 139 = 0, MaxValue 139 = 100.

Gradient 142 = True, LineColor 142 is lighter than Color 133 by 1.3, LineWidth 143 = 2, Gap 142 = 4.



See also

- TRVReportTableItemInfo 80.BackgroundVisualizers 81

7.9.1 Properties

7.9.1.1 TRVReportPieVisualizer.Gap

A difference between the radii of the pie itself and of the surrounding circle.

```
property MaxSize: TRVStyleLength;
```

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

Examples:

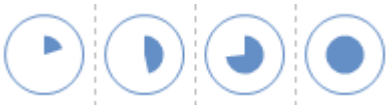
Gap = 0



Gap = 2



Gap = 10



Default value

2

7.9.1.2 TRVReportPieVisualizer.Gradient

Specifies whether pie slices are painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, pie slices are filled with Color[133].

If *True*, pie slices are filled with a gradient from Color[133] (at bottom) to a twice lighter color (at top).

Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+).

Default value

False

7.9.1.3 TRVReportPieVisualizer.LineColor

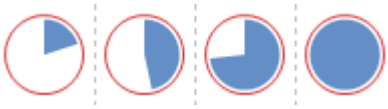
A color of a circles surrounding pies.

```
property LineColor: TColor
```

LineColor is inherited from TRVReportCustomValueVisualizer[133]. See the description of the original property[133]. TRVReportPieVisualizer overrides the initial value of this property: by default, it is the same as Color[133].

This circle is drawn if **LineColor** <> *clNone* and LineWidth[143] > 0.

Example: red line color



Default value

$C68E63

7.9.1.4 TRVReportPieVisualizer.LineWidth

A line width of a circles what surround pies.

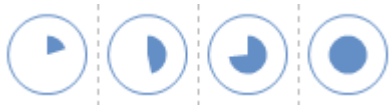
```
property LineWidth: TRVStyleLength;
```

This circle is drawn if LineColor[142] <> *clNone* and **LineWidth** > 0.

Examples:

In these examples, Gap[142] = 10.

LineWidth = 1



LineWidth = 5



Default value

1

7.9.1.5 TRVReportPieVisualizer.MaxSize

Maximal diameter of pies displayed by the visualizer.

```
property MaxSize: TRVStyleLength;
```

All pies displayed by the visualizer have the same diameter. It is calculated to fit the smallest cell.

If **MaxSize** > 0, it specifies the maximal possible diameter (more exactly, it defines the maximal diameter of circles surrounding pies)

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

Default value

0

7.9.1.6 TRVReportPieVisualizer.PartsCount

Defines whether angles of pie slices are changed smoothly or by steps.

```
property PartsCount: Integer;
```

If **PartsCount** = 0, pie slices may have any angle.

If **PartsCount** is a positive value, pie slices have angles which are multiples of 360° / **PartsCount**.

Examples

The examples use the following data:

20	46.7	73.3	100

MinValue[135] = 0, MaxValue[135] = 100.

PartsCount = 0



PartsCount = 4



PartsCount = 3



Default value

0

7.10 TRVReportShapeRepeaterVisualizer

TRVReportShapeRepeaterVisualizer visualizes numeric values by displaying the count of shapes proportional to these values.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportShapeRepeaterVisualizer = class (TRVReportCustomShapeVisualizer[131])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[134]
TRVReportCustomValueVisualizer[133]
TRVReportCustomShapeVisualizer[131]

Description

Values are visualized by filling shapes. The count of full shapes is proportional to the visualized value. Displaying all empty shapes corresponds to MinValue[135], displaying all full shapes corresponds to MaxValue[135]. Values less than MinValue are displayed as MinValue, values greater than MaxValue are displayed as MaxValue.

Shapes are shown in RowCount[145] rows and ColCount[145] columns, the filling direction is specified in Direction[146].

The shape is defined in ShapeProperties[132].

Full shapes are drawn using Color[146] and LineColor[133], empty shapes are drawn using ColorEmpty and LineColorEmpty[146]. Empty shapes can be semitransparent[146]

The following properties affect shape drawing: Gradient[132] and LineUsesFillColor[132].

Optionally, background can be filled using BackgroundColor[145].

Distance between shapes is specified in Spacing[147], space around the whole diagram is specified in Padding[147] and Margin[134].

Shape size may be limited by MaxShapeSize[147].

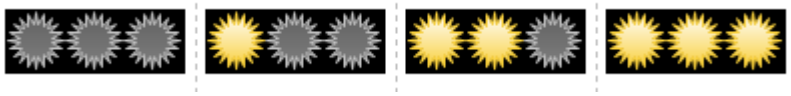
By default, shapes are inscribed in squares. You can stretch them by changing ShapeScaleX[133].

Examples

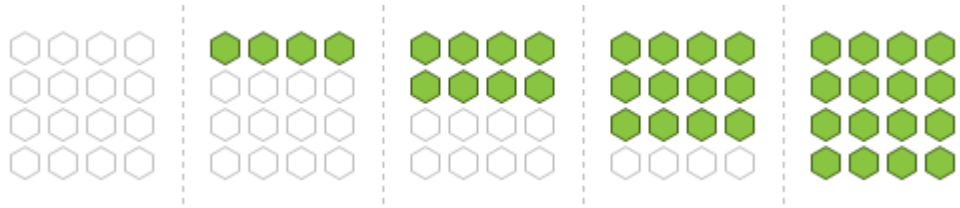
Example 1: ColCount 145 = 4; ShapeProperties 132: Shape 170 = *rvrshStar*, PointCount 169 = 5, MiddlePercent 169 40; Gradient 132 = *False*



Example 2: ColCount 145 = 3; ShapeProperties 132: Shape 170 = *rvrshStar*, PointCount 169 = 20, MiddlePercent 169 60; BackgroundColor 145 = *clBlack*, Padding 147 = 2



Example 3: ColCount 145 = RowCount 145 = 4; ShapeProperties 132: Shape 170 = *rvrshPolygon*, PointCount 169 = 6, Spacing 147 = 2, Gradient 132 = *False*, ColorEmpty 146 = *clNone*, LineColorEmpty 146 = *clSilver*, Color 146 = $41C589:



See also

- TRVReportTableItemInfo 80.BackgroundVisualizers 81

7.10.1 Properties

7.10.1.1 TRVReportShapeRepeaterVisualizer.BackgroundColor

Background color

```
property BackgroundColor: TColor;
```

This property defines the color of a rectangle shown below shapes.

Example: BackgroundColor = *clGray*



Default value

clNone

See also

- Padding 147

7.10.1.2 TRVReportShapeRepeaterVisualizer.ColCount, RowCount

The properties specify how many shapes are displayed by the visualizer.

```
property RowCount: Integer;
property ColCount: Integer;
```

Shapes are displayed in rows and columns. These properties specify the counts of rows and columns.

Default values

- RowCount: 1

- ColCount: 5

7.10.1.3 TRVReportShapeRepeaterVisualizer.Color

Color for full shapes.

```
property Color: TColor;
```

Color is inherited from TRVReportCustomValueVisualizer[133]. See the description of the original property[133]. TRVReportShapeRepeaterVisualizer overrides the initial value of this property.

Default value

$39CFF8

See also

- ColorEmpty[146]

7.10.1.4 TRVReportShapeRepeaterVisualizer.ColorEmpty, LineColorEmpty, OpacityEmpty

These properties define colors and opacity for empty shapes.

```
property LineColorEmpty: TColor;
property ColorEmpty: TColor;
property OpacityEmpty: TRVOpacity;
```

Empty shapes are filled with **ColorEmpty** and **OpacityEmpty**, and outlined with **LineColorEmpty** (unless LineUsesFillColor[132] = *True*)

Full shapes are filled with Color[146] (fully opaque), and outlined with LineColor[133] (unless LineUsesFillColor[132] = *True*).

Default values

- ColorEmpty: *clSilver*
- LineColorEmpty: *clBlack*
- OpacityEmpty: 50000 (i.e. 50%)

7.10.1.5 TRVReportShapeRepeaterVisualizer.Direction

A direction in which shapes are displayed

```
type
  TRVReportBarDirection = (rvrbdRight, rvrbdLeft, rvrbdUp, rvrbdDown);
property Direction: TRVReportBarDirection;
```

Shapes are displayed in rows and columns. Full shapes are added according to **Direction**.

Value	Meaning	Example
rvrbdRight	From left to right, then from top to bottom	
rvrbdLeft	From right to left, then from bottom to top	
rvrbdUp	From bottom to top, then from right to left	
rvrbdDown	From top to bottom, then from left to right	

Default values

rvrbdRight

7.10.1.6 TRVReportShapeRepeaterVisualizer.MaxShapeSize

Maximal possible size of shapes displayed by the visualizer.

```
property MaxShapeSize: TRVStyleLength;
```

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

If **MaxShapeSize** > 0, it specifies the the maximal possible diameter of an imaginary circle around each shape.

Unlike the most of other visualizers' "max size" properties, this property limits a size not of the whole diagram, but of its fragment (shapes).

If **MaxShapeSize** > 0, the maximal possible diagram size is:

- width: **MaxShapeSize** * ColCount[145] + Spacing[147] * (ColCount[145] - 1) + Padding[147] * 2
- height: **MaxShapeSize** * RowCount[145] + Spacing[147] * (RowCount[145] - 1) + Padding[147] * 2

(additionally, there is space around the diagram specified in Margin[134])

7.10.1.7 TRVReportShapeRepeaterVisualizer.Padding

Sets a colored space around images displayed by the visualizer.

```
property Padding: TRVStyleLength;
```

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

If BackgroundColor[145] = *clNone*, this value is simply added to Margin[134]. Otherwise, this space is colored, while margins are transparent.

Examples

Padding = 2



Padding = 10



Default value

1

See also

- Spacing[147]

7.10.1.8 TRVReportShapeRepeaterVisualizer.Spacing

A distance between shapes

```
property Spacing: TRVStyleLength;
```

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

Default value

1

See also

- Padding[147]

7.11 TRVReportSignalStrengthVisualizer

TRVReportSignalStrengthVisualizer visualizes numeric values by a drawing a "signal strength" diagrams.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportSignalStrengthVisualizer = class (TRVReportCustomValueVisualizer[133])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[134]
TRVReportCustomValueVisualizer[133]

Description

This visualizer displays a "signal strength" diagram for each value. This diagram consists of shapes of increased heights (so this diagram is inscribed in an imaginary right triangle).

This visualizer supports two types of shapes: bars and wedges, specified in DisplayStyle[149] property. For simplicity, we will call them "bars".

The count of bars is specified in PartsCount[151], a distance between them is specified in Spacing[151].

Values are visualized by filling bars. The count of full bars is proportional to the visualized value. Displaying all empty bars corresponds to MinValue[135], displaying all full bars corresponds to MaxValue[135]. Values less than MinValue are displayed as MinValue, values greater than MaxValue are displayed as MaxValue.

Full bars are drawn using Color and LineColor[133], empty bars are drawn using ColorEmpty and LineColorEmpty[149]. Empty bars can be semitransparent[149].

The following properties affect bar drawing: Gradient[150] and LineUsesFillColor[150].

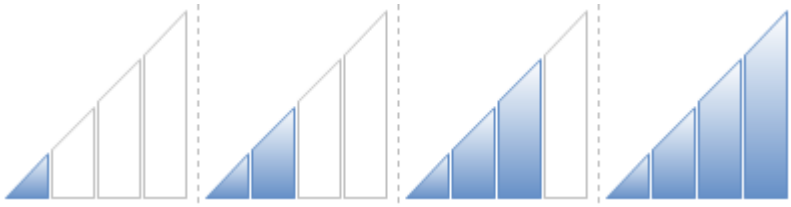
Examples

The examples use the following data:

20	46.7	73.3	100

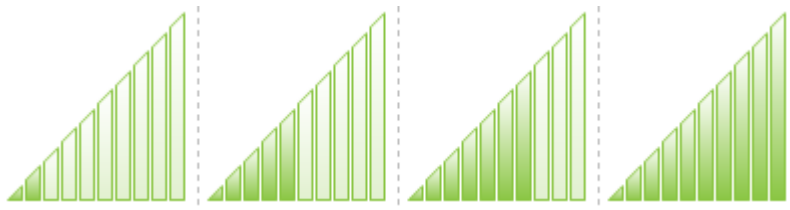
MinValue[139] = 0, MaxValue[139] = 100.

Example 1: default values



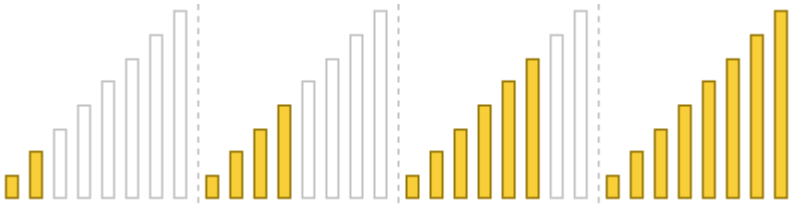
Example 2:

Color[133] = ColorEmpty[149] = $41C589, OpacityEmpty[149] = 25000, PartsCount[151] = 10



Example 3:

Color[133] = $39CFF8, PartsCount[151] = 8, Spacing[151] = 5, Gradient[150] = *False*, DisplayStyle[149] = *rvrsssBars*



See also

- TRVReportTableItemInfo[80].BackgroundVisualizers[81]

7.11.1 Properties

7.11.1.1 TRVReportSignalStrengthVisualizer.ColorEmpty, LineColorEmpty, OpacityEmpty

These properties define colors and opacity for empty bars or wedges.

```
property ColorEmpty: TColor;
property LineColorEmpty: TColor;
property OpacityEmpty: TRVOpacity;
```

Empty bars are filled with **ColorEmpty** and **OpacityEmpty**, and outlined with **LineColorEmpty** (unless LineUsesFillColor[150] = *True*)

Full bars are filled with Color[133] (fully opaque), and outlined with LineColor[133] (unless LineUsesFillColor[150] = *True*).

Default values

- ColorEmpty: *clNone*
- LineColorEmpty: *clSilver*
- OpacityEmpty: 50000 (i.e. 50%)

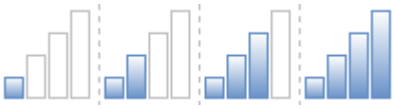
See also:

- Gradient[150]

7.11.1.2 TRVReportSignalStrengthVisualizer.DisplayStyle

Specifies shapes used in diagrams.

```
type
  TRVReportSignalStrengthStyle = (rvrsssBars, rvrsssWedges);
property DisplayStyle: TRVReportSignalStrengthStyle;
```

Value	Example
rvrsssBars	

rvrsssWedges	

Default value

rvrsssWedges

7.11.1.3 TRVReportSignalStrengthVisualizer.Gradient

Specifies whether bars (or wedges) are painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, bars are filled with Color[133] / ColorEmpty[149].

If *True*, bars are filled with a gradient from Color[133] / ColorEmpty[149] (at bottom) to a twice lighter color (at top).

Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+).

Default value

True

7.11.1.4 TRVReportSignalStrengthVisualizer.LineUsesFillColor

Specifies how bars (or wedges) are outlined.

```
property LineUsesFillColor: Boolean;
```

If **LineUsesFillColor** = *False*:

Full bars are outlined with LineColor[133], empty bars are outlined with LineColorEmpty[149].

If **LineUsesFillColor** = *True*:

If the corresponding fill color is equal to *clNone*, LineColor[133]/LineColorEmpty[149] is used. Otherwise, if Gradient[150] = *True*, bars are outlined with Color[133]/ColorEmpty[149]; if Gradient[150] = *False*, they are outlined with twice darker colors.

Note: for Delphi versions prior to 2010, this option simply uses Color as LineColor.

Default value

True

7.11.1.5 TRVReportSignalStrengthVisualizer.MaxSize

Maximal size of diagrams displayed by this visualizer.

```
property MaxSize: TRVStyleLength;
```

All images displayed by the visualizer have the same size. It is calculated to fit the smallest cell.

If **MaxSize** > 0, it specifies the maximal possible size.

This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

Default value

0

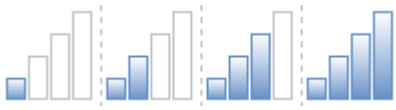
7.11.1.6 TRVReportSignalStrengthVisualizer.PartsCount

A count of bars (or wedges) in each diagram displayed by the visualizer.

```
property PartsCount: Integer;
```

Examples

PartsCount = 4



PartsCount = 10



Default value

4

7.11.1.7 TRVReportSignalStrengthVisualizer.Spacing

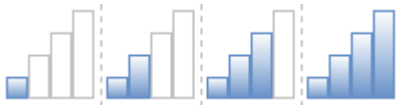
A distance between bars (or wedges)

```
property Spacing: TRVStyleLength;
```

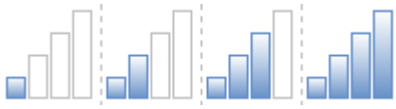
This value is measured in screen pixels or in twips (1/20 of point), depending on Units property of TRVStyle component.

Examples

Spacing = 0



Spacing = 1



Spacing = 8



Default value

1

8 Actions

Report Workshop includes a set of actions for editing report templates. They can be used in addition to other RichViewActions.

Report properties

TrvActionDocProperties 154 allows editing report-related document properties

Properties of report tables

 TrvrActionRowGenerationRules[159] allows editing row generation rules[82].

 TrvrActionCrossTab[153] allows editing a cross tabulation[81].

 TrvrActionCellProperties[152] allows editing report table cell[101] properties, as well as BackgroundVisualizers[81] and BackgroundColorChangers[80] collections

The actions above may be applied to normal tables: they converts them to report tables (after a confirmation)

Additional reporting actions

 TrvrActionConvertToReportTable[153] converts the selected normal table to a report table.

 TrvrActionInsertTable[158] creates a new a report table.

These actions are redundant, because the actions for editing report table properties may convert normal tables to report tables.

Other actions

 TrvrActionInsertShape[154] inserts a geometric shape into the editor.

8.1 TrvrActionCellProperties

TrvrActionCellProperties is an action for editing report table cell[101] properties.

Unit RVReportActions;

Syntax

```
TrvrActionCellProperties = class (TrvrCustomActionReportTableWithDataProvider[160]);
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionTableCell
TrvrCustomActionReportTable[160]
TrvrCustomActionReportTableWithDataProvider[160]

Description

This action displays a dialog window for editing properties of report table cell[101] selected in the target editor.

This action can also be applied to normal tables (it may convert them to report tables).

The action edits the following cell properties:

- DataQuery[103]
- VisualizerId[104]
- ColorChangerId[102]

Additionally, it edits BackgroundVisualizers[81] and BackgroundColorChangers[80] properties of the table.

If DataProvider[160] property of this action is assigned, the dialog displays lists of tables and fields to help editing data queries associated with the cells.

8.2 TrvrActionConvertToReportTable

TrvrActionConvertToReportTable converts the selected normal table to a report table[80].

Unit RVReportActions;

Syntax

```
TrvrActionConvertToReportTable = class (TrvActionTableCell)
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionTableCell

Description

TrvrActionConvertToReportTable converts the selected normal table to a report table[80].

This action is redundant: the actions for editing report table properties may convert normal tables to report tables themselves.

8.3 TrvrActionCrossTab

TrvrActionCrossTab is an action for editing a cross tabulation[81] for the selected report table[80].

Unit RVReportActions;

Syntax

```
TrvrActionCrossTab = class (TrvrCustomActionReportTableWithDataProvider[160]);
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionTableCell
TrvrCustomActionReportTable[160]
TrvrCustomActionReportTableWithDataProvider[160]

Description

This action displays a dialog window for editing cross tabulation[81] for the report table[80] selected in the target editor.

This action can also be applied to normal tables (it may convert them to report tables).

This action allows defining the cross-tab header position and column properties. Properties of rows are defined by TrvrActionRowGenerationRules[159].

If DataProvider[160] property of this action is assigned, the dialog displays lists of tables and fields to help editing data queries for column generation.

8.4 TrvrActionDocProperties

TrvrActionDocProperties is an action for editing report-related document properties

Unit RVReportActions;

Syntax

```
TrvrActionDocProperties = class (TrvAction);
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction

Description

This action changes properties of TRVReportDocObject[170] item in DocObjects property of the target editor:

- DataQuery[171]
- PageBreaksBetweenCopies[172]

8.4.1 Properties

8.4.1.1 TrvrActionDocProperties.DataProvider

Allows linking the action with a data provider.

```
property DataProvider: TRVReportDataProvider[53];
```

This data provider is used to get a list of tables and fields, to help users writing data queries.

8.5 TrvrActionInsertShape

TrvrActionInsertShape is an action for inserting a geometric shape[116] in the caret position of the editor.

This action is not related to reporting, it simply reuses shape drawing functions implemented for value visualizers[119].

Unit RVReportShapeAction;

Syntax

```
TrvrActionInsertShape = class (TrvAction);
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction

Description

The action has the same properties as the shape item, defining its shape, colors and size. These properties are assigned to inserted items.

If UserInterface[158] = *False*, the specified shape is simply inserted in the editor.

If UserInterface[158] = *True*, the action shows a shape selection window and inserts the chosen shape.

TrvActionItemProperties action allows editing properties of shape items. It displays a dialog which has "Default" check box. If checked, values entered in this dialog are applied to all **TrvrActionInsertShape** action on the same form as TrvActionItemProperties, changing their properties.

8.5.1 Properties

8.5.1.1 TrvrActionInsertShape.BackgroundColor, BackgroundOpacity

Specify the item background color and opacity

```
property BackgroundColor: TColor;
property BackgroundOpacity: TRVOpacity;
```

These properties are assigned to BackgroundColor and BackgroundOpacity[117] of the inserted item.

Default value

- BackgroundColor: *clNone*
- BackgroundOpacity: 100000 (i.e. 100%)

8.5.1.2 TrvrActionInsertShape.BorderColor

Color of border around the item.

```
property BorderColor: TColor;
```

This property is assigned to BorderColor of the inserted item.

Default value

clBlack

8.5.1.3 TrvrActionInsertShape.BorderWidth

Width of border around the item, in pixels or twips (depending on ControlPanel.UnitsProgram)

```
property BorderWidth: TRVStyleLength;
```

This property is assigned to BorderWidth of the inserted item. A conversion between twips and pixels is performed, if necessary.

Default value

0

8.5.1.4 TrvrActionInsertShape.CallerControl

Specifies the control (for example a toolbar button) which executed this action.

```
property CallerControl: TControl;
```

Assign this property if you use Delphi 4 or 5. For the newer versions of Delphi, the caller control is determined automatically (using ActionComponent property).

A shape selection window is positioned relative to this control. If this control is undefined, a color-picker window is displayed at the mouse pointer position.

8.5.1.5 TrvrActionInsertShape.Color, Opacity

Fill color and opacity for the shape.

```
property Color: TColor;
property Opacity: TRVOpacity;
```

These properties are assigned to Color and Opacity[117] of the inserted item.

Default values

- Color: $C68E63
- Opacity: 100000 (i.e. 100%)

8.5.1.6 TrvrActionInsertShape.EqualSides

Specifies whether the shape has the same width and height.

```
property EqualSides: Boolean;
```

This property is assigned to EqualSides[117] of the inserted item.

Default value

True

8.5.1.7 TrvrActionInsertShape.GradientType

Type of gradient fill for shapes.

```
property GradientType: TRVGradientType[160];
```

This property is assigned to GradientType[117] of the inserted item.

Default value

rvgtNone

8.5.1.8 TrvrActionInsertShape.LineColor

Line color

```
property LineColor: TColor;
```

This property is assigned to LineColor[118] of the inserted item.

Default value

clBlack

8.5.1.9 TrvrActionInsertShape.LineUsesFillColor

Specifies how shapes are outlined.

```
property LineUsesFillColor: Boolean;
```

This property is assigned to LineUsesFillColor[118] of the inserted item.

Default value

False

8.5.1.10 TrvrActionInsertShape.LineWidth

Line width, in pixels or twips (depending on ControlPanel.UnitsProgram)

```
property LineWidth: TRVStyleLength;
```

This property is assigned to LineWidth[118] of the inserted item. A conversion between twips and pixels is performed, if necessary.

Default value

1

8.5.1.11 TrvrActionInsertShape.OuterHSpacing, OuterVSpacing

Horizontal and vertical space around the item (around the border, if it exists), in pixels or twips (depending on ControlPanel.UnitsProgram)

```
property OuterHSpacing: TRVStyleLength;
property OuterVSpacing: TRVStyleLength;
```

This property is assigned to OuterHSpacing and OuterVSpacing of the inserted item. A conversion between twips and pixels is performed, if necessary.

Default value

0

8.5.1.12 TrvrActionInsertShape.ShapeProperties

Specifies main properties of a shape.

```
property ShapeProperties: TRVReportShapeProperties[169];
```

This property is assigned to ShapeProperties[119] of the inserted item.

8.5.1.13 TrvrActionInsertShape.StartColor

Start color for gradient fill.

```
property StartColor: TColor;
```

This property is assigned to StartColor[119] of the inserted item.

Default value

clWhite

8.5.1.14 TrvrActionInsertShape.UserInterface

Specify whether the action shows a shape selection window.

```
property UserInterface: Boolean;
```

If *False*: the action simple inserts a shape and assigns its properties.

If *True*: the action displays a shape selection window, then inserts the chosen shape. Some shape properties are taken from the shape selection window, others are taken from the action properties.

The shape selection window is the same as the window displayed by ShowInsertDialog[158] method. If action was initiated by a button, the window is displayed below this button (for old version of Delphi, use CallerControl[156]), otherwise it is displayed at the position of the mouse pointer.

Default value

True

8.5.1.15 TrvrActionInsertShape.Width, Height

Shape size, in pixels or twips (depending on ControlPanel.UnitsProgram)

```
property Width: TRVStyleLength;
property Height: TRVStyleLength;
```

These properties are assigned to Width and Height[119] of the inserted item. A conversion between twips and pixels is performed, if necessary.

Default value

48

8.5.2 Methods

8.5.2.1 TrvrActionInsertShape.ShowInsertDialog

The methods show a window for choosing a shape to insert, then insert the chosen shape.

```
procedure ShowInsertDialog(Target: TCustomRichViewEdit;
  Button: TControl); overload;
procedure ShowInsertDialog(Target: TCustomRichViewEdit;
  const ButtonRect: TRect); overload;
```

The first method shows this window below the specified **Button** control. If **Button** = *nil*, it shows this window in the position of the mouse pointer.

The second method shows this window below the specified **ButtonRect** (it contains screen coordinates). This method is useful for displaying a window below the button that is not inherited from TControl.

The inserted shape has properties defined in this action, with some properties overridden by the selection in this window.

When the action is assigned to a button, a shape selection window is displayed below this button automatically, when the action is executed. However, you may wish to use a combo-style toolbar button (which shows a triangle a the right side). To do it, you can assign a empty TPopupMenu component to toolbar button's DropdownMenu, and call **ShowInsertDialog** in this menu's OnPopup event.

8.6 TrvrActionInsertTable

TrvrActionInsertTable is an action for inserting a report table[80] in the caret position

Unit RVReportActions;

Syntax

```
TrvrActionInsertTable = class (TrvActionInsertTable)
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionInsertTable

Description

This action works exactly like RichViewActions' TrvActionInsertTable, but inserts TRVReportTableItemInfo[80] instead of TRVTableItemInfo.

This action is redundant: you can use TrvActionInsertTable instead, and the actions for editing report table properties may convert normal tables to report tables.

8.7 TrvrActionRowGenerationRules

TrvrActionRowGenerationRules is an action for editing row generation rules[82] for the selected report table[80].

Unit RVReportActions;

Syntax

```
TrvrActionRowGenerationRules = class (TrvrCustomActionReportTableWithDataProvider[160]);
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionTableCell
TrvrCustomActionReportTable[160]
TrvrCustomActionReportTableWithDataProvider[160]

Description

This action displays a dialog window for editing row generation rules[82] for the report table[80] selected in the target editor.

This action can also be applied to normal tables (it may convert them to report tables).

If DataProvider[160] property of this action is assigned, the dialog displays lists of tables and fields to help editing data queries for row generation.

8.8 TrvrCustomActionReportTable

Basic reporting actions for editing properties of report tables [80].

Unit RVReportActions;

Syntax

```
TrvrCustomActionReportTable = class (TrvActionTableCell);
TrvrCustomActionReportTableWithDataProvider = class (TrvrCustomActionReportTable);
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionTableCell

Description

These actions are not used directly. The following actions are inherited from TrvrCustomActionReportTableWithDataProvider:

- TrvrActionRowGenerationRules [159]
- TrvrActionCrossTab [153]
- TrvrActionCellProperties [152]

TrvrCustomActionReportTableWithDataProvider introduces DataProvider [160] property.

8.8.1 Properties

8.8.1.1 TrvrCustomActionReportTableWithDataProvider.DataProvider

Allows linking the action with a data provider.

```
property DataProvider: TRVReportDataProvider [53];
```

This data provider is used to get a list of tables and fields, to help users writing data queries.

9 Types

9.1 TRVGradientType

Type of gradient fill.

Unit RVReportTypes;

Syntax

```
type
  TRVGradientType = (rvgtNone, rvgtVertical, rvgtHorizontal);
```

Value	Meaning

rvgtNone	No gradient, flat color
rvgtVertical	Gradient from top to bottom
rvgtHorizontal	Gradient from left to right

9.2 TRVReportField

Identifies a data field in TRVReportQueryProcessor[174].

Unit RVReportDataProvider;

Syntax

```
type
  TRVReportField = Pointer;
```

9.3 TRVReportFieldType

Identifies a type of a data field of TRVReportQueryProcessor[174].

Unit RVReportDataProvider;

Syntax

```
type
 TRVReportFieldType = (rvrftUnknown,
    rvrftText, rvrftInteger, rvrftFloat, rvrftBoolean,
    rvrftDateTime,
      rvrftDate,
      rvrftTime,
    rvrftBlob,
      rvrftAnsiMemo,
      rvrftUnicodeMemo,
      rvrftDocument,
        rvrftRTF,
        rvrftRVF,
      rvrftGraphic,
        rvrftBitmap,
        rvrftGif,
        rvrftPng,
        rvrftJpeg,
        rvrftTiff,
        rvrftIcon,
        rvrftMetafile);
```

As you can see, these values allow to specify the field type with different degree of precision.
For example, *rvrftBlob* may be clarified as *rvrftGraphic*, *rvrftGraphic* may be clarified as *rvrftBitmap*.

See also:

- Data field types[23]

9.4 TRVReportRecordCountMode

Unit RVReportDBDataProvider;

Syntax

```
type
  TRVReportRecordCountMode =
```

```
  (rvrrcmDoNotUse, rvrrcmUseIfAvailable, rvrrcmUseAlways);
```

Value	Meaning
rvrrcmDoNotUse	Report Workshop avoids using DataSet.RecordCount property. If possible, unidirectional datasets are created in this mode. If Report Workshop needs a second pass through the results of DataSet, it closes and re-opens the dataset before moving to the first record. Pros: • queries may be executed faster; Contras: • Report Workshop still requires record count for some operations (if RowGenerationRule[82].CopyColCount[113] > 1, or for cross-tab reports[81]); in this case, one more query execution is added to calculate the record count; • some operations are performed less efficiently; for example, when applying queries to table rows, rows are added one by one instead of adding all rows at once; • OnDataQueryProgress[37] is called without calculating Percent parameter
rvrrcmUseIfAvailable	Report Workshop uses DataSet.RecordCount property. It assumes that the dataset correctly processes First, Next, and Last commands. This value cannot be used for unidirectional datasets that cannot move back to the first record.
rvrrcmUseAlways	When opening a dataset for the first time, one more query execution is performed to calculate record count. After that, the dataset is closed and re-opened.

9.5 TRVReportShape

Identifies a shape type.

Unit RVReportShapes;

Syntax

```
type
  TRVReportShape = (rvrshCircle, rvrshPolygon, rvrshStar, rvrshRoughGear,
    rvrshBluntPointStar, rvrshArrow1, rvrshArrow2, rvrshRing,
    rvrshPolygonInCircle, rvrshStarInCircle, rvrshStarInPolygon,
    rvrshSquare,
    rvrshStopRect, rvrshNo, rvrshCheck,
    rvrshEmoticonSmile, rvrshEmoticonLaugh, rvrshEmoticonStraight, rvrshEmoticonSad,
    rvrshEmoticonShout, rvrshEmoticonWink,
    rvrshFlag,
    rvrshMan1, rvrshWoman1, rvrshMan2, rvrshWoman2
    );
```

A shape of each type is defined by the following parameters:

- diameter of an imaginary circle surrounding the shape
- diameter of an imaginary internal circle
- count of points

In the examples below, the same shape is displayed with different parameters. Horizontally, the count of points is increased from 3 to 7. Vertically, shapes are displayed with internal radii 30%, 50%, 70% of the external radius.

Note: the shapes may have unequal width and height. In this case, circles become ellipses, squares become rectangles, etc.

All shapes are shown unrotated.

Value	Meaning	Rotatable	Examples
rvrshCircle	Circle The internal circle and the count of points are ignored.		
rvrshPolygon	Regular polygon. Count of sides = count of points. The internal circle is ignored.	yes	
rvrshStar	Star polygon Vertexes are placed on the external and the internal circles alternately. Count of vertexes on each circle = count of points.	yes	
rvrshRoughGear	Gear-like polygon Vertexes are placed on the external and the internal circles; two vertexes on the external circle, then two vertexes on the internal circle, and so on. Count of vertexes on each circle = count of points * 2.	yes	
rvrshBluntPointStar	Blunt-pointed star polygon Vertexes are placed on the external and the internal circles; two vertexes on the external circle, then one vertex on the internal circle, and so on. Count of vertexes on the internal circle = count of points	yes	
rvrshArrow1	Arrow, version 1 Width of the arrow shaft = diameter of the internal circle. 3 vertexes of the arrow head are located on the external circle, the arrow head angle = 90° The count of points is ignored.	yes	
rvrshArrow2	Arrow, version 2 Width of the arrow shaft = diameter of the internal circle. The arrow head touches the internal circle, the arrow head angle = 90° The count of points is ignored.	yes	
rvrshRing	Ring / Doughnut Two circles, a colored area is in the middle. The count of points is ignored.		
rvrshPolygonInCircle	Polygon inside circle Count of polygon sides = count of points. Polygon vertexes are on the internal circle.	yes	

rvrshStarInCircle	Star inside circle The star parameters are the same as for *rvrshStar*	yes	
rvrshStarInPolygon	Star inside polygon Star parameters are the same as for *rvrshStar.* Polygon parameters are the same as for *rvrshPolygon.*	yes	
rvrshSquare	Square Unlike *rvrshPolygon* rotated by 45° (a square inscribed in the outer circle), this shape occupies maximum size (the outer circle is inscribed in this square). The count of points is ignored.		
rvrshStopRect	"No entry" traffic sign, rectangle inside circle The rectangle parameters are the same as for *rvrshNo* The count of points is ignored.	yes	
rvrshNo	No sign, strike-through ring The middle line has the same width as the ring. To make a slashed/backslashed circle, rotate by 45°/-45°. The count of points is ignored.	yes	
rvrshCheck	Check mark Internal circle defines the width of line. The count of points is ignored.		
rvrshEmoticonSmile	Smiling face Internal circle affects face features. The count of points is ignored.		
rvrshEmoticonLaugh	Laughing face Internal circle affects face features. The count of points is ignored.		
rvrshEmoticonStraight	Neutral face Internal circle affects face features. The count of points is ignored.		

rvrshEmoticonSad	Sad face Internal circle affects face features. The count of points is ignored.		
rvrshEmoticonShout,	Shouting/surprised face Internal circle affects face features. The count of points is ignored.		
rvrshEmoticonWink	Winking face Internal circle affects face features. The count of points is ignored.		
rvrshFlag	Flag Internal circle defines the banner height. The count of points (3, 4, or 5) defines the banner shape. Less then 3 is counted as 3, more than 5 is counted as 5.		
rvrshMan1	Male icon, version 1 Internal circle defines the figure width. The count of points is ignored.		
rvrshWoman1	Female icon, version 1 Internal circle defines the figure width. The count of points is ignored.		
rvrshMan2	Male icon, version 2 Internal circle defines the figure width. The count of points is ignored.		
rvrshWoman2	Female icon, version 2 Internal circle defines the figure width. The count of points is ignored.		

9.6 TRVValueVisualizerId

This type is used for unique identifiers of data visualizers and color changers.

Unit RVReportColorChanger;

Syntax

```
type
  TRVValueVisualizerId = type Integer;
```

This type is used in the properties:

- Id[135] property of value visualizers[133] and color changers[127]; this property is an unique identifier of items in their collections;
- VisualizerId[104] property of report table cell[101]; this property links the cell with the item of the report table[80]'s BackgroundVisualizers[81] collection;
- ColorChangerId[102] property of report table cell[101]; this property links the cell with the item of the report table[80]'s BackgroundColorChangers[80] collection.

10 Constants

10.1 rveipcXXX Constants

These constants identify integer properties of TRichView items.

They are used in the following methods:

- TRichView.GetExtraItemIntPropertyEx
- TRichView.SetExtraItemIntPropertyEx
- TRichViewEdit.SetItemExtraIntPropertyExEd
- TRichViewEdit.GetCurrentItemExtraIntPropertyEx
- TRichViewEdit.SetCurrentItemExtraIntPropertyEx

Properties of shapes[116]

These properties are defined in RVReportShapeItem unit.

Constant	Value	Property
rveipcShapeColor	200	Color[117]
rveipcStartShapeColor	201	StartColor[119]
rveipcLineColor	202	LineColor[118]
rveipcLineWidth	203	LineWidth[118]
rveipcLineUsesFillColor	204	LineUsesFillColor[118]
rveipcEqualSides	205	EqualSides[117]*
rveipcGradientType	206	GradientType[117]*
rveipcBackgroundOpacity	207	BackgroundOpacity[117]
rveipcShapeOpacity	208	Opacity[117]
rveipcShapeType	209	ShapeProperties.Shape[170]*
rveipcPointCount	210	ShapeProperties.PointCount[169]
rveipcMiddlePercent	211	ShapeProperties.MiddlePercent[169]
rveipcStartAngle	212	ShapeProperties.StartAngle[170]**

* the property is type-casted to Integer

** the property is multiplied by 100 and rounded

11 Procedures and Functions

11.1 GenerateReport

Generates error for ScaleRichView editor **Editor**.

Unit RVReportSRVGenerator;

Syntax

```
function GenerateReport(Editor: TSRichViewEdit;
 Generator: TRVReportGenerator[46]; IncludeInvisibleHeaders: Boolean = False): Boolean;
```

This method calls **Generator**.Execute[36] for Editor.**RichViewEdit** (main document) and all **Editor**.SubDocuments[] (headers and footers).

If **IncludeInvisibleHeaders** = *False*, only visible **Editor**.SubDocuments[] are processed. Headers and footers may be hidden by the following properties:

- Editor.PageProperty.TitlePage
- Editor.PageProperty.FacingPages
- Editor.ViewProperty.HeaderVisible
- Editor.ViewProperty.FooterVisible

If **IncludeInvisibleHeaders** = *True*, all **Editor**.SubDocuments[] are processed.

12 Classes

12.1 TRVConditionalShapePropertiesCollection

TRVConditionalShapePropertiesCollection is a collection of shape properties applied on a condition.

Unit RVReportValueVisualizer;

Syntax

```
TRVConditionalShapePropertiesCollection = class (TCollection)
```

Hierarchy

TObject
TPersistent
TCollectionI

Description

This is collection of TRVConditionalShapePropertiesItem[168].

Value[168] properties of all items[167] must be assigned, and items must be sorted by Value[168] in ascending order.

12.1.1 Properties

12.1.1.1 TRVConditionalShapePropertiesCollection.Items

Lists the items in the collection.

```
property Items[Index: Integer]: TRVConditionalShapePropertiesItem[168]; default;
```

Use **Items** to access individual items in the collection.

12.2 TRVConditionalShapePropertiesItem

TRVConditionalShapePropertiesItem contains several shape properties.

Unit RVReportValueVisualizer;

Syntax

```
TRVConditionalShapePropertiesItem = class (TCollectionItem)
```

Hierarchy

TObject
TPersistent
TCollectionItem

Description

TRVConditionalShapePropertiesItem is a class of items in TRVConditionalShapePropertiesCollection[167].

This item contains the following shape properties:

- Color[168]
- DeltaAngle[168]

Additionally, it has Value[168] property. An owner of this item's collection compares a visualized value with Value[168] property of all items in the collection to find the proper item.

12.2.1 Properties

12.2.1.1 TRVConditionalShapePropertiesItem.Color

Shape fill color

```
property Color: TColor;
```

Default value

clWhite

12.2.1.2 TRVConditionalShapePropertiesItem.DeltaAngle

Additional rotation angle.

```
property DeltaAngle: Double;
```

This value is measured in degrees. Positive values rotate a shape clockwise, negative values rotate a shape counterclockwise.

When a shape is displayed, this value is added to the default rotation angle, and its shape is displayed rotated.

Default value

0

12.2.1.3 TRVConditionalShapePropertiesItem.Value

A numeric value associated with this item.

```
property Value: Variant;
```

This value defines when properties of this item are used.

Default value

(no value)

12.3 TRVReportShapeProperties

TRVReportShapeProperties contains shape properties.

Unit RVReportShapes;

Syntax

```
TRVReportShapeProperties = class (TPersistent)
```

Hierarchy

TObject
TPersistent

Description

This item contains the following shape properties:

- MiddlePercent 169 defines the radius of an internal circle for the shape
- PointCount 169 – count of points in the shape
- Shape 170 – shape type (circle, regular polygon, star, arrow, etc.)
- StartAngle 170 – rotation angle for the shape

This class is used for the following properties:

- TRVReportCustomShapeVisualizer.ShapeProperties 132
- TRVShape.ShapeProperties 70
- TRVShapeItemInfo.ShapeProperties 119

12.3.1 Properties

12.3.1.1 TRVReportShapeProperties.MiddlePercent

Allows to calculate the radius of an internal circle for the shape.

```
property MiddlePercent: Integer;
```

If the radius of an external circle is *R*, the radius of an internal circle is (*R* * **MiddlePercent**) / 100.

Value of this property must be in range 0..99.

The meaning of this property depends on the shape type (Shape 162)

Some shapes (regular polygons) ignore this property. Meaning of this property for other shapes is described in TRVReportShape 162.

Default value

50

12.3.1.2 TRVReportShapeProperties.PointCount

A count of points in the shape.

```
property PointCount: Integer;
```

The meaning of this property depends on the shape type (Shape 162)

Some shapes (circle, arrows) ignore this property. Meaning of this property for other shapes is described in TRVReportShape 162.

Default value

5

12.3.1.3 TRVReportShapeProperties.Shape

A shape type

```
property Shape: TRVReportShape;
```

Default value

rvrshCircle

12.3.1.4 TRVReportShapeProperties.StartAngle

An angle of the shape rotation.

```
property StartAngle: Double;
```

This value is measured in degrees.

In the topic on TRVReportShape you can find examples of unrotated shapes (for **StartAngle** = 0).

Positive values rotate the shape clockwise, negative values rotate it counterclockwise.

Default value

0

12.4 TRVReportDocObject

TRVReportDocObject contains report-related properties for storing in report templates.

Unit RVReportDocObject;

Syntax

```
TRVReportDocObject = class (TRVDocObject)
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVDocObject

Description

An object of this class can be added in TRichView.DocObjects collection. Objects in this collection can be stored and loaded in RVF (RichView Format) files.

It's highly recommended to add only a single object of this class in this collection.

The recommended way of working with properties of this class is using the following functions:

```
function GetRVReportDocObject(rv: TCustomRichView;
  AllowCreate: Boolean): TRVReportDocObject;
function RVReportGetRootDataQuery(rv: TCustomRichView): TRVUnicodeString;
procedure RVReportSetRootDataQuery(rv: TCustomRichView;
  const Value: TRVUnicodeString; EditMode: Boolean);
function RVReportGetHighlightRules(rv: TCustomRichView): Boolean;
procedure RVReportSetHighlightRules(rv: TCustomRichView; Value: Boolean);
```

GetRVReportDocObject returns the first TRVReportDocObject item in **rv**.DocObjects. If not found, it may return nil (if **AllowCreate** = *False*) or add a new object to the end of the collection and return it (if **AllowCreate** = *True*).

RVReportGetRootDataQuery returns a value of the DataQuery property of the first TRVReportDocObject item in **rv**.DocObjects, or empty string if not found.

RVReportSetRootDataQuery assigns a new value to the DataQuery[171] property of the first TRVReportDocObject item in **rv**.DocObjects. If such an object is not found, it creates it. If **EditMode** = True and (**rv** is TCustomRichViewEdit), this assignment is performed as an editing operation (undoable).

Similarly, **RVReportGetHighlightRules** and **RVReportSetHighlightRules** provide access to HighlightRules[171] property.

12.4.1 Properties

12.4.1.1 TRVReportDocObject.DataQuery

A string containing a data query.

```
property DataQuery: TRVUnicodeString;
```

An example of a DataQuery is a SQL query, like 'SELECT * FROM MyTable', or simply a table name, like 'MyTable'.

TRVReportGenerator[46] creates a query processor[174] for this data query. This query processor returns data for this query, and the report generator replicates the whole content of TRichView as many times as many records exist in the returned data. Then the report generator processes these replicated contents, replacing data fields in them accordingly.

TRVReportGenerator[46] starts processing this data query before any other data queries.

TRVReportGenerator[46] processes this property only for the first object of TRVReportDocObject class in TRichView.DocObjects collection.

Default value:

" (empty string)

See also

- RVReportGetRootDataQuery[170] function
- RVReportSetRootDataQuery[170] procedure

See also

- Definitions of Report Workshop terms[10]
- Information about data queries[28]
- PafeBreaksBetweenCopies[172]
- TRVRowGenerationCustomRule[105].DataQuery[105]
- TRVReportTableCellData[101].DataQuery[103]
- TRVCrossTabLevel[92].DataQuery[95]

12.4.1.2 TRVReportDocObject.HighlightRules

Allows/disallows highlighting in all report tables[80].

```
property HighlightRules: Boolean;
```

You can highlight:

- report cells using TRVReportTableCellData[102].HighlightColor[102] (if they have a data query associated)
- row generation rules using TRVRowGenerationCustomRule[105].HighlightColor[106]
- cross-tab header using TRVCrossTab[84].HighlightColor[91]

These objects can be highlighted in unprocessed report template, to allow better understanding its structure.

The property only for the first object of TRVReportDocObject class in TRichView.DocObjects collection is used.

Default value

False

See also

- RVReportGetHighlightRules[170] function
- RVReportSetHighlightRules[170] procedure

12.4.1.3 TRVReportDocObject.PageBreaksBetweenCopies

Specifies whether the report generator must add page breaks between copies of data generated when applying DataQuery[171].

```
property PageBreaksBetweenCopies: Boolean;
```

This property is used only if DataQuery[171] is not empty.

Default value:

False

12.5 TRVReportDBQueryProcessor

The class of an object processing a database-related data query.

unit RVReportDBDataProvider;

```
TRVReportDBQueryProcessor = class (TRVReportQueryProcessor[174]);
```

Hierarchy

TObject
TRVReportQueryProcessor[174]

Description

This class is a wrapper around TDataSet-based components processing database-related data queries, normally, SQL SELECT statements.

Normally, this class is used internally in TRVReportGenerator[46] component, and you need information about this class only if you want to implement your own non-standard query processor.

However, this class may be useful to get values of data fields, if you want to assign values for variables basing on data in OnProcessRecord[45] event.

See also

- Definitions of Report Workshop terms[10]
- TRVReportGenerationSession[172].GetQueryProcessor[173]

12.6 TRVReportGenerationSession

The class of an object containing information about a report generation session. It can be used to get values of variables[15] and data fields[12].

Unit RVReportGenerator;

Syntax

```
TRVReportGenerationSession = class (TRVList)
```

Hierarchy

TObject
TList
TRVList

Description

Methods:

- GetQueryProcessor[173] returns a query processor; it can be used to get values of data fields[12].
- GetVariableValue[173] returns a value of a variable[15].

See also

- Definitions of Report Workshop terms 10

12.6.1 Methods

12.6.1.1 TRVReportGenerationSession.GetVariableValue

Returns a value and an associated object for the given variable 15.

```
function GetVariableValue(const Name: TRVUnicodeString;
  var Value: TRVUnicodeString; var Obj: TObject): Boolean;
```

Input parameter

Name – variable name (without a starting '%'). The method looks for this variable in Variables 82 of the most deeply nested report table, then in the table containing it, and so on. Finally, it looks in the TRVReportGenerator's Variables 35.

Note: when processing variables in a report table properties (e.g. in row generation rule's data query, and in a hint), or cells outside any row generation rule, variables of this table are not used.

Output parameters

These parameters are valid only if the method returns *True*.

Value – variable value

Obj – an object associated with this variable

Return value

True if this variable exists

See also:

- Definitions of Report Workshop terms 10

12.6.1.2 TRVReportGenerationSession.GetQueryProcessor

Returns a query processor identified by **Name**.

```
function GetQueryProcessor(const Name: TRVUnicodeString):
  TRVReportQueryProcessor 174;
```

Input parameter

Name can be one of the following values:

- *table row generation rule name* 106. A data query 105 of this rule must be processed at the moment of this call. If several rules have the same name, the method returns a query processor for the more deeply nested rule.
- *empty string*. The method returns the current (i.e. the most deeply nested) query processor;
- '^'. The method returns the parent query processor of the current query processor, '^^' returns the grandparent, and so on.

Return value

A query processor object, or *nil*, if not found.

See also:

- Definitions of Report Workshop terms 10

12.7 TRVReportQueryProcessor

The class of an object processing a data query.

unit RVReportDataProvider;

```
TRVReportQueryProcessor = class;
```

Hierarchy

TObject

Description

Normally, this class is used internally in TRVReportGenerator[46] component, and you need information about this class only if you want to implement your own non-standard query processor.

However, this class may be useful to get values of data fields, if you want to assign values for variables basing on data in OnProcessRecord[45] event.

See also

- Definitions of Report Workshop terms[10]
- TRVReportGenerationSession[172].GetQueryProcessor[173]

12.7.1 Methods

12.7.1.1 TRVReportQueryProcessor.GetAs*

Returns a value for the specified field.

```
function GetAsString(Field: TRVReportField[161]): TRVUnicodeString;
function GetAsInteger(Field: TRVReportField[161]): Int64;
function GetAsFloat(Field: TRVReportField[161]): Extended;*
function GetAsDateTime(Field: TRVReportField[161]): TDateTime;
function GetAsStream(Field: TRVReportField[161]): TStream;
```

* for Delphi 6-2007, this function returns Double

To get a value for the **Field** parameter, use GetField[174].

12.7.1.2 TRVReportQueryProcessor.GetField

Returns an object identifying a field having the specified name.

```
function GetField(const FieldName: TRVUnicodeString): TRVReportField[161];
```

If a field is not found, the function returns *nil*.

12.7.1.3 TRVReportQueryProcessor.GetFieldType

Returns a type for the specified field.

```
function GetFieldType(Field: TRVReportField[161]): TRVReportFieldType[161];
```

To get a value for the **Field** parameter, use GetField[174].

12.7.1.4 TRVReportQueryProcessor.GetRecordCount

Returns the count of records returned for a data query.

```
function GetRecordCount: Integer;
```

A query processor is not necessary supports quick calculation of record counts. So it may return MaxInt, if it has a non-zero count of records. In this case, a report generator will enumerate records until MoveToNextRecord returns *False*.

You rarely need to use this method. Usually, query processors are used to get field values from the current record.

12.7.1.5 TRVReportQueryProcessor.IsFieldEmpty

Returns: is the field value empty/undefined?

```
function IsFieldEmpty(Field: TRVReportField): Boolean;
```

To get a value for the **Field** parameter, use GetField.

This method is used by a report generator to check a parameter in "IfDef" and "IfNDef" commands.

Index

- A -

AbsoluteValues 130
TRVReportColoredShapeVisualizer 130
Actions 151
AfterQueryClose 53
TRVReportCustomDBDataProvider 53
AfterQueryOpen 53
TRVReportCustomDBDataProvider 53
AfterQueryScroll 53
TRVReportCustomDBDataProvider 53
AllowSummaryCols 112
TRVRowGenerationRule 112
AllowSummaryRows 112
TRVRowGenerationRule 112
Alt 116
TRVShapeItemInfo 116
AssignWithId 101
TRVReportCustomValueVisualizerCollection 101
AutoMaxValue 135, 139
TRVReportCustomValueVisualizerBase 135
TRVReportGaugeVisualizer 139
AutoMinValue 135, 139
TRVReportCustomValueVisualizerBase 135
TRVReportGaugeVisualizer 139
AxisColor 124
TRVReportBarVisualizer 124
AxisPosition 124
TRVReportBarVisualizer 124

- B -

BackgroundColor 68, 125, 145, 155
TrvrActionInsertShape 155
TRVReportBarVisualizer 125
TRVReportShapeRepeaterVisualizer 145
TRVShape 68
BackgroundColorChangers 80
TRVReportTableItemInfo 80
BackgroundOpacity 68, 117, 125, 155
TrvrActionInsertShape 155
TRVReportBarVisualizer 125
TRVShape 68
TRVShapeItemInfo 117
BackgroundVisualizers 81
TRVReportTableItemInfo 81
BarDirection 126
TRVReportBarVisualizer 126
BeforeQueryClose 53
TRVReportCustomDBDataProvider 53
BeforeQueryOpen 53
TRVReportCustomDBDataProvider 53
BeforeQueryScroll 53
TRVReportCustomDBDataProvider 53
BorderColor 155
TrvrActionInsertShape 155
BorderWidth 156
TrvrActionInsertShape 156

- C -

CallerControl 156
TrvrActionInsertShape 156
CaptionFieldName 93
TRVCrossTabLevel 93
CaseSensitive 94, 112
TRVCrossTabLevel 94
TRVRowGenerationRule 112
ClearEmptySummaryCols 95
TRVCrossTabLevel 95
ColCount 145
TRVReportShapeRepeaterVisualizer 145
Color 69, 117, 133, 138, 146, 156, 168
TRVConditionalShapePropertiesItem 168
TrvrActionInsertShape 156
TRVReportCustomValueVisualizer 133
TRVReportGaugeVisualizer 138
TRVReportShapeRepeaterVisualizer 146
TRVShape 69
TRVShapeItemInfo 117
ColorBegin 128
TRVReportColorChangerItem 128
ColorChangerId 102
TRVReportTableCellData 102
ColorEmpty 146, 149
TRVReportShapeRepeaterVisualizer 146
TRVReportSignalStrengthVisualizer 149
ColorEnd 128
TRVReportColorChangerItem 128
ColorMiddle 128
TRVReportColorChangerItem 128
ColorValue 102
TRVReportTableCellData 102
Column 86

Column 86
TRVCrossTab 86
ColumnGenerationType 88
TRVCrossTab 88
Commands 16
ConditionalShapeProperties 131
TRVReportColoredShapeVisualizer 131
CopyColCount 113
TRVRowGenerationRule 113
CopyFirstCol 113
TRVRowGenerationRule 113
CopyMaxCount 113
TRVRowGenerationRule 113
CopySpacingColCount 113
TRVRowGenerationRule 113
CreateQueryProcessor 54
TRVReportDataProvider 54
CrossTabulation 81
TRVReportTableItemInfo 81

- D -

Data field types 19, 23
Data fields 12
Data queries 28
Data visualizers 119
DataProvider 34, 154, 160
TCustomRVReportGenerator 34
TrvrActionDocProperties 154
TrvrCustomActionReportTableWithDataProvider 160
DataQuery 95, 103, 105, 171
TRVCrossTabLevel 95
TRVReportDocObject 171
TRVReportTableCellData 103
TRVRowGenerationCustomRule 105
DataQueryFieldName 97
TRVCrossTabLevel 97
DataSet 56
TRVDataSetItem 56
DataSets 55
TRVReportDBDataProvider 55
Definitions 10
DeltaAngle 168
TRVConditionalShapePropertiesItem 168
Direction 146
TRVReportShapeRepeaterVisualizer 146
DisplayStyle 149
TRVReportSignalStrengthVisualizer 149

- E -

EqualSides 69, 117, 156
TrvrActionInsertShape 156
TRVShape 69
TRVShapeItemInfo 117
ErrorColor 34
TCustomRVReportGenerator 34
EscapeDataQuery 36, 71
TCustomRVReportGenerator 36, 71
Execute 36, 71
TCustomRVReportGenerator 36, 71
Extending Report Workshop 30

- F -

FieldName 97
TRVCrossTabLevel 97
Fields 11
FindById 101
TRVReportCustomValueVisualizerCollection 101
FindItemById 101
TRVReportCustomValueVisualizerCollection 101
FirstRow 86, 106
TRVCrossTab 86
TRVRowGenerationCustomRule 106
FitWidth 126
TRVReportBarVisualizer 126
ForbiddenIds 101
TRVReportCustomValueVisualizerCollection 101
Format strings 24
Functions 20

- G -

Gap 142
TRVReportPieVisualizer 142
GenerateReport 167
GetAsDateTime 174
TRVReportQueryProcessor 174
GetAsFloat 174
TRVReportQueryProcessor 174
GetAsInteger 174
TRVReportQueryProcessor 174
GetAsStream 174
TRVReportQueryProcessor 174

GetAsString 174
TRVReportQueryProcessor 174
GetField 174
TRVReportQueryProcessor 174
GetFieldType 174
TRVReportQueryProcessor 174
GetQueryProcessor 173
TRVReportGenerationSession 173
GetRecordCount 175
TRVReportQueryProcessor 175
GetRVReportDocObject 170
GetVariableValue 173
TRVReportGenerationSession 173
Gradient 122, 127, 132, 142, 150
TRVReportAreaSizeVisualizer 122
TRVReportBarVisualizer 127
TRVReportCustomShapeVisualizer 132
TRVReportPieVisualizer 142
TRVReportSignalStrengthVisualizer 150
GradientType 69, 117, 156
TrvrActionInsertShape 156
TRVShape 69
TRVShapeItemInfo 117
GreenAreaColor 137
TRVReportGaugeVisualizer 137

- H -

HAlign 134
TRVReportCustomValueVisualizer 134
Height 119, 158
TrvrActionInsertShape 158
TRVShapeItemInfo 119
HighlightColor 91, 102, 106
TRVCrossTab 91
TRVReportTableCellData 102
TRVRowGenerationCustomRule 106
HighlightRules 171
TRVReportDocObject 171

- I -

Id 135
TRVReportCustomValueVisualizerBase 135
IsFieldEmpty 175
TRVReportQueryProcessor 175
Items 55, 100, 104, 111, 115, 167
TRVConditionalShapePropertiesCollection 167
TRVCrossTabLevels 100
TRVDataSetCollection 55
TRVReportColorChangers 100
TRVReportValueVisualizers 104
TRVRowGenerationNestedRules 111
TRVRowGenerationRules 115

- K -

KeyFieldNames 115
TRVRowGenerationRule 115

- L -

Levels 91
TRVCrossTab 91
Limitations in old versions of Delphi 33
LineColor 69, 118, 133, 142, 156
TrvrActionInsertShape 156
TRVReportCustomValueVisualizer 133
TRVReportPieVisualizer 142
TRVShape 69
TRVShapeItemInfo 118
LineColorEmpty 146, 149
TRVReportShapeRepeaterVisualizer 146
TRVReportSignalStrengthVisualizer 149
LineUsesFillColor 70, 118, 132, 150, 157
TrvrActionInsertShape 157
TRVReportCustomShapeVisualizer 132
TRVReportSignalStrengthVisualizer 150
TRVShape 70
TRVShapeItemInfo 118
LineWidth 70, 118, 143, 157
TrvrActionInsertShape 157
TRVReportPieVisualizer 143
TRVShape 70
TRVShapeItemInfo 118

- M -

Margin 70, 134
TRVReportCustomValueVisualizer 134
TRVShape 70
MaxColCount 92, 98
TRVCrossTab 92
TRVCrossTabLevel 98
MaxShapeSize 147
TRVReportShapeRepeaterVisualizer 147
MaxSize 122, 131, 139, 143, 150
TRVReportAreaSizeVisualizer 122
TRVReportBarVisualizer 139
TRVReportColoredShapeVisualizer 131

MaxSize 122, 131, 139, 143, 150
TRVReportPieVisualizer 143
TRVReportSignalStrengthVisualizer 150
MaxValue 98, 135, 139
TRVCrossTabLevel 98
TRVReportBarVisualizer 139
TRVReportCustomValueVisualizerBase 135
MaxWidth 127
TRVReportBarVisualizer 127
MergeWithPrevious 110
TRVRowGenerationNestedRule 110
MiddleColor 138
TRVReportGaugeVisualizer 138
MiddlePercent 169
TRVReportShapeProperties 169
MinSize 122
TRVReportAreaSizeVisualizer 122
MinValue 98, 135, 139
TRVCrossTabLevel 98
TRVReportCustomValueVisualizerBase 135
TRVReportGaugeVisualizer 139

- N -

Name 56, 106
TRVDataSetItem 56
TRVRowGenerationCustomRule 106
NegativeColor 127
TRVReportBarVisualizer 127
NegativeLineColor 127
TRVReportBarVisualizer 127

- O -

OnCreateDataSet 52
TRVReportCustomDBDataProvider 52
OnCreateQueryProcessor 36, 71
TCustomRVReportGenerator 36, 71
OnDataQueryProgress 37, 72
TCustomRVReportGenerator 37, 72
OnDataSetCreated 52
TRVReportCustomDBDataProvider 52
OnError 40, 74
TCustomRVReportGenerator 40, 74
OnProcessRecord 45, 79
TCustomRVReportGenerator 45, 79
Opacity 69, 117, 156
TrvrActionInsertShape 156
TRVShape 69
TRVShapeItemInfo 117
OpacityBegin 128
TRVReportColorChangerItem 128
OpacityEmpty 146, 149
TRVReportShapeRepeaterVisualizer 146
TRVReportSignalStrengthVisualizer 149
OpacityEnd 128
TRVReportColorChangerItem 128
OpacityMiddle 128
TRVReportColorChangerItem 128
OuterHSpacing 157
TrvrActionInsertShape 157
OuterVSpacing 157
TrvrActionInsertShape 157

- P -

Padding 70, 147
TRVReportShapeRepeaterVisualizer 147
TRVShape 70
PageBreaksBetweenCopies 172
TRVReportDocObject 172
PartsCount 143, 151
TRVReportPieVisualizer 143
TRVReportSignalStrengthVisualizer 151
PercentMiddle 129
TRVReportColorChangerItem 129
PointCount 169
TRVReportShapeProperties 169
Processed 81
TRVReportTableItemInfo 81

- R -

RedAreaColor 137
TRVReportGaugeVisualizer 137
RedAreaPercent 138
TRVReportGaugeVisualizer 138
RedLowValues 139
TRVReportGaugeVisualizer 139
Report Workshop
Actions 151
Additional components 68
Data provider components 46
Data queries 28
Data visualizers 119
Definitions 10
Limitations in old versions of Delphi 33
Main components 33
Overview 9
ReportCells 81

ReportCells 81
TRVReportTableItemInfo 81
RowCount 106, 145
TRVReportShapeRepeaterVisualizer 145
TRVRowGenerationCustomRule 106
RowGenerationRules 82
TRVReportTableItemInfo 82
RVReportGetHighlightRules 170
RVReportGetRootDataQuery 170
RVReportSetHighlightRules 170
RVReportSetRootDataQuery 170

- S -

SelectedRule 82
TRVReportTableItemInfo 82
SetBackgroundColorChangers 83
TRVReportTableItemInfo 83
SetBackgroundVisualizers 83
TRVReportTableItemInfo 83
SetCellColorChanger 83
TRVReportTableItemInfo 83
SetCellDataQuery 83
TRVReportTableItemInfo 83
SetCellHighlightColor 83
TRVReportTableItemInfo 83
SetCellVisualizer 83
TRVReportTableItemInfo 83
SetCrossTabulation 84
TRVReportTableItemInfo 84
SetRowGenerationRules 84
TRVReportTableItemInfo 84
SetVariables 84
TRVReportTableItemInfo 84
Shape 122, 170
TRVReportAreaSizeVisualizer 122
TRVReportShapeProperties 170
ShapeProperties 70, 119, 132, 157, 169
TrvrActionInsertShape 157
TRVReportCustomShapeVisualizer 132
TRVShape 70
TRVShapeItemInfo 119
ShapeScaleX 133
TRVReportCustomShapeVisualizer 133
ShowInsertDialog 158
TrvrActionInsertShape 158
SingleColorMode 140
TRVReportGaugeVisualizer 140
SkipLeftColCount 109
TRVRowGenerationNestedRule 109
SkipRightColCount 109
TRVRowGenerationNestedRule 109
SortType 99
TRVCrossTabLevel 99
Spacing 144, 147, 151
TRVReportShapeRepeaterVisualizer 147
TRVReportSignalStrengthVisualizer 151
StartAngle 170
TRVReportShapeProperties 170
StartColor 70, 119, 157
TrvrActionInsertShape 157
TRVShape 70
TRVShapeItemInfo 119
Step 98
TRVCrossTabLevel 98
SubRules 107
TRVRowGenerationCustomRule 107

- T -

TCustomRVReportGenerator 34
TCustomRVReportGenerator.DataProvider 34
TCustomRVReportGenerator.ErrorColor 34
TCustomRVReportGenerator.EscapeDataQuery 36, 71
TCustomRVReportGenerator.Execute 36, 71
TCustomRVReportGenerator.OnCreateQueryProcessor 36, 71
TCustomRVReportGenerator.OnDataQueryProgress 37, 72
TCustomRVReportGenerator.OnError 40, 74
TCustomRVReportGenerator.OnProcessRecord 45, 79
TCustomRVReportGenerator.Texts 35
TCustomRVReportGenerator.Variables 35
Template syntax
Commands 16
Cross-tab heading 19
Data field types 23
Data fields 12
Fields 11
Format strings 24
Functions 20
Variables 15
TextForEmptyCells 92
TRVCrossTab 92
Texts 35
TCustomRVReportGenerator 35
TRVColGenerationType 88
TRVConditionalShapePropertiesCollection 167

TRVConditionalShapePropertiesCollection.Items 167
TRVConditionalShapePropertiesItem 168
TRVConditionalShapePropertiesItem.Color 168
TRVConditionalShapePropertiesItem.DeltaAngle 168
TRVConditionalShapePropertiesItem.Value 168
TRVCreateDataSetEvent 52
TRVCreateQueryProcessorEvent 36, 71
TRVCrossTab 84
TRVCrossTab.Column 86
TRVCrossTab.ColumnGenerationType 88
TRVCrossTab.FirstRow 86
TRVCrossTab.HighlightColor 91
TRVCrossTab.Levels 91
TRVCrossTab.MaxColCount 92
TRVCrossTab.TextForEmptyCells 92
TRVCrossTabLevel 92
TRVCrossTabLevel.CaptionFieldName 93
TRVCrossTabLevel.CaseSensitive 94
TRVCrossTabLevel.ClearEmptySummaryCols 95
TRVCrossTabLevel.DataQuery 95
TRVCrossTabLevel.DataQueryFieldName 97
TRVCrossTabLevel.FieldName 97
TRVCrossTabLevel.MaxColCount 98
TRVCrossTabLevel.MaxValue 98
TRVCrossTabLevel.MinValue 98
TRVCrossTabLevel.SortType 99
TRVCrossTabLevel.Step 98
TRVCrossTabLevels 99
TRVCrossTabLevels.Items 100
TRVDataSetCollection 55
TRVDataSetCollection.Items 55
TRVDataSetCreatedEvent 52
TRVDataSetItem 55
TRVDataSetItem.DataSet 56
TRVDataSetItem.Name 56
TRVDataSetItem.UseRecordCount 56
TRVGradientType 160, 166
TRVProcessRecordEvent 37, 45, 72, 79
TrvrActionCellProperties 152
TrvrActionConvertToReportTable 153
TrvrActionCrossTab 153
TrvrActionDocProperties 154
TrvrActionDocProperties.DataProvider 154
TrvrActionInsertShape 154
TrvrActionInsertShape.BackgroundColor 155
TrvrActionInsertShape.BackgroundOpacity 155
TrvrActionInsertShape.BorderColor 155
TrvrActionInsertShape.BorderWidth 156
TrvrActionInsertShape.CallerControl 156
TrvrActionInsertShape.Color 156
TrvrActionInsertShape.EqualSides 156
TrvrActionInsertShape.GradientType 156
TrvrActionInsertShape.Height 158
TrvrActionInsertShape.LineColor 156
TrvrActionInsertShape.LineUsesFillColor 157
TrvrActionInsertShape.LineWidth 157
TrvrActionInsertShape.Opacity 156
TrvrActionInsertShape.OuterHSpacing 157
TrvrActionInsertShape.OuterVSpacing 157
TrvrActionInsertShape.ShapeProperties 157
TrvrActionInsertShape.ShowInsertDialog 158
TrvrActionInsertShape.StartColor 157
TrvrActionInsertShape.UserInterface 158
TrvrActionInsertShape.Width 158
TrvrActionInsertTable 158
TrvrActionRowGenerationRules 159
TrvrCustomActionReportTable 160
TrvrCustomActionReportTableWithDataProvider 160
TrvrCustomActionReportTableWithDataProvider.DataProvider 160
TRVReportAbsDataProvider 49
TRVReportADODataProvider 50
TRVReportAreaSizeVisualizer 121
TRVReportAreaSizeVisualizer.Gradient 122
TRVReportAreaSizeVisualizer.MaxSize 122
TRVReportAreaSizeVisualizer.MinSize 122
TRVReportAreaSizeVisualizer.Shape 122
TRVReportBarAxisPosition 124
TRVReportBarDirection 126, 146
TRVReportBarVisualizer 123
TRVReportBarVisualizer.AxisColor 124
TRVReportBarVisualizer.AxisPosition 124
TRVReportBarVisualizer.BackgroundColor 125
TRVReportBarVisualizer.BackgroundOpacity 125
TRVReportBarVisualizer.BarDirection 126
TRVReportBarVisualizer.FitWidth 126
TRVReportBarVisualizer.Gradient 127
TRVReportBarVisualizer.MaxWidth 127
TRVReportBarVisualizer.NegativeColor 127
TRVReportBarVisualizer.NegativeLineColor 127
TRVReportBDEDataProvider 50
TRVReportColorChangerItem 127
TRVReportColorChangerItem.ColorBegin 128
TRVReportColorChangerItem.ColorEnd 128
TRVReportColorChangerItem.ColorMiddle 128
TRVReportColorChangerItem.OpacityBegin 128

TRVReportColorChangerItem.OpacityEnd 128
TRVReportColorChangerItem.OpacityMiddle 128
TRVReportColorChangerItem.PercentMiddle 129
TRVReportColorChangers 100
TRVReportColorChangers.Items 100
TRVReportColoredShapeVisualizer 129
TRVReportColoredShapeVisualizer.AbsoluteValues 130
TRVReportColoredShapeVisualizer.ConditionalShapeProperties 131
TRVReportColoredShapeVisualizer.MaxSize 131
TRVReportCustomDBDataProvider 51
TRVReportCustomDBDataProvider.AfterQueryClose 53
TRVReportCustomDBDataProvider.AfterQueryOpen 53
TRVReportCustomDBDataProvider.AfterQueryScroll 53
TRVReportCustomDBDataProvider.BeforeQueryClose 53
TRVReportCustomDBDataProvider.BeforeQueryOpen 53
TRVReportCustomDBDataProvider.BeforeQueryScroll 53
TRVReportCustomDBDataProvider.OnCreateDataSet 52
TRVReportCustomDBDataProvider.OnDataSetCreated 52
TRVReportCustomDBDataProvider.UseRecordCount 52
TRVReportCustomShapeVisualizer 131
TRVReportCustomShapeVisualizer.Gradient 132
TRVReportCustomShapeVisualizer.LineUsesFillColor 132
TRVReportCustomShapeVisualizer.ShapeProperties 132
TRVReportCustomShapeVisualizer.ShapeScaleX 133
TRVReportCustomValueVisualizer 133
TRVReportCustomValueVisualizer.Color 133
TRVReportCustomValueVisualizer.HAlign 134
TRVReportCustomValueVisualizer.LineColor 133
TRVReportCustomValueVisualizer.Margin 134
TRVReportCustomValueVisualizer.VAlign 134
TRVReportCustomValueVisualizerBase 134
TRVReportCustomValueVisualizerBase.AutoMaxValue 135
TRVReportCustomValueVisualizerBase.AutoMinValue 135
TRVReportCustomValueVisualizerBase.Id 135
TRVReportCustomValueVisualizerBase.MaxValue 135
TRVReportCustomValueVisualizerBase.MinValue 135
TRVReportCustomValueVisualizerBase.ValueString 135
TRVReportCustomValueVisualizerCollection 100
TRVReportCustomValueVisualizerCollection.AssignWithId 101
TRVReportCustomValueVisualizerCollection.FindById 101
TRVReportCustomValueVisualizerCollection.FindItemById 101
TRVReportCustomValueVisualizerCollection.ForbiddenIds 101
TRVReportDataProvider 53
TRVReportDataProvider.CreateQueryProcessor 54
TRVReportDBDataProvider 54
TRVReportDBDataProvider.DataSets 55
TRVReportDBISAMDataProvider 56
TRVReportDBQueryProcessor 172
TRVReportDBXDataProvider 57
TRVReportDocObject 170
TRVReportDocObject.DataQuery 171
TRVReportDocObject.HighlightRules 171
TRVReportDocObject.PageBreaksBetweenCopies 172
TRVReportEDBDataProvider 58
TRVReportFDDataProvider 58
TRVReportFDMongoDataProvider 59
TRVReportField 161
TRVReportFieldType 161
TRVReportGaugeVisualizer 136
TRVReportGaugeVisualizer.AutoMaxValue 139
TRVReportGaugeVisualizer.AutoMinValue 139
TRVReportGaugeVisualizer.Color 138
TRVReportGaugeVisualizer.GreenAreaColor 137
TRVReportGaugeVisualizer.MaxSize 139
TRVReportGaugeVisualizer.MaxValue 139
TRVReportGaugeVisualizer.MiddleColor 138
TRVReportGaugeVisualizer.MinValue 139
TRVReportGaugeVisualizer.RedAreaColor 137
TRVReportGaugeVisualizer.RedAreaPercent 138
TRVReportGaugeVisualizer.RedLowValues 139
TRVReportGaugeVisualizer.SingleColorMode 140
TRVReportGaugeVisualizer.YellowAreaColor 137
TRVReportGaugeVisualizer.YellowAreaPercent 138
TRVReportGenerationSession 172
TRVReportGenerationSession.GetQueryProcessor 173

TRVReportGenerationSession.GetVariableValue 173
TRVReportGenerator 46
TRVReportGeneratorError 40, 74
TRVReportGeneratorTexts 35
TRVReportIBCDataProvider 60
TRVReportIBDataProvider 61
TRVReportIBODataProvider 62
TRVReportMyDataProvider 62
TRVReportMySQLDataProvider 63
TRVReportNxDataProvider 64
TRVReportPgDataProvider 65
TRVReportPieVisualizer 141
TRVReportPieVisualizer.Gap 142
TRVReportPieVisualizer.Gradient 142
TRVReportPieVisualizer.LineColor 142
TRVReportPieVisualizer.LineWidth 143
TRVReportPieVisualizer.MaxSize 143
TRVReportPieVisualizer.PartsCount 143
TRVReportProgressStep 37, 72
TRVReportPSQLDataProvider 65
TRVReportQueryProcessor 174
TRVReportQueryProcessor.GetAsDateTime 174
TRVReportQueryProcessor.GetAsFloat 174
TRVReportQueryProcessor.GetAsInteger 174
TRVReportQueryProcessor.GetAsStream 174
TRVReportQueryProcessor.GetAsString 174
TRVReportQueryProcessor.GetField 174
TRVReportQueryProcessor.GetFieldType 174
TRVReportQueryProcessor.GetRecordCount 175
TRVReportQueryProcessor.IsFieldEmpty 175
TRVReportRecordCountMode 161
TRVReportShape 162
TRVReportShapeProperties 169
TRVReportShapeProperties.MiddlePercent 169
TRVReportShapeProperties.PointCount 169
TRVReportShapeProperties.Shape 170
TRVReportShapeProperties.StartAngle 170
TRVReportShapeRepeaterVisualizer 144
TRVReportShapeRepeaterVisualizer.BackgroundColor 145
TRVReportShapeRepeaterVisualizer.ColCount 145
TRVReportShapeRepeaterVisualizer.Color 146
TRVReportShapeRepeaterVisualizer.ColorEmpty 146
TRVReportShapeRepeaterVisualizer.Direction 146
TRVReportShapeRepeaterVisualizer.LineColorEmpty 146
TRVReportShapeRepeaterVisualizer.MaxShapeSize 147
TRVReportShapeRepeaterVisualizer.OpacityEmpty 146
TRVReportShapeRepeaterVisualizer.Padding 147
TRVReportShapeRepeaterVisualizer.RowCount 145
TRVReportShapeRepeaterVisualizer.Spacing 147
TRVReportSignalStrengthStyle 149
TRVReportSignalStrengthVisualizer 148
TRVReportSignalStrengthVisualizer.ColorEmpty 149
TRVReportSignalStrengthVisualizer.DisplayStyle 149
TRVReportSignalStrengthVisualizer.Gradient 150
TRVReportSignalStrengthVisualizer.LineColorEmpty 149
TRVReportSignalStrengthVisualizer.LineUsesFillColor 150
TRVReportSignalStrengthVisualizer.MaxSize 150
TRVReportSignalStrengthVisualizer.OpacityEmpty 149
TRVReportSignalStrengthVisualizer.PartsCount 151
TRVReportSignalStrengthVisualizer.Spacing 151
TRVReportSortType 99
TRVReportTableCellData 101
TRVReportTableCellData.ColorChangerId 102
TRVReportTableCellData.ColorValue 102
TRVReportTableCellData.DataQuery 103
TRVReportTableCellData.HighlightColor 102
TRVReportTableCellData.Value 103
TRVReportTableCellData.VisualizerId 104
TRVReportTableItemInfo 80
TRVReportTableItemInfo.BackgroundColorChangers 80
TRVReportTableItemInfo.BackgroundVisualizers 81
TRVReportTableItemInfo.CrossTabulation 81
TRVReportTableItemInfo.Processed 81
TRVReportTableItemInfo.ReportCells 81
TRVReportTableItemInfo.RowGenerationRules 82
TRVReportTableItemInfo.SelectedRule 82
TRVReportTableItemInfo.SetBackgroundColorChangers 83
TRVReportTableItemInfo.SetBackgroundVisualizers 83
TRVReportTableItemInfo.SetCellColorChanger 83
TRVReportTableItemInfo.SetCellDataQuery 83
TRVReportTableItemInfo.SetCellHighlightColor 83
TRVReportTableItemInfo.SetCellVisualizer 83
TRVReportTableItemInfo.SetCrossTabulation 84

TRVReportTableItemInfo.SetRowGenerationRules 84
TRVReportTableItemInfo.SetVariables 84
TRVReportTableItemInfo.Variables 82
TRVReportUniDataProvider 66
TRVReportValueVisualizers 104
TRVReportValueVisualizers.Items 104
TRVReportZEOSDSDataProvider 67
TRVRowGenerationCustomRule 105, 109
TRVRowGenerationCustomRule.DataQuery 105
TRVRowGenerationCustomRule.FirstRow 106
TRVRowGenerationCustomRule.HighlightColor 106
TRVRowGenerationCustomRule.Name 106
TRVRowGenerationCustomRule.RowCount 106
TRVRowGenerationCustomRule.SubRules 107
TRVRowGenerationNestedRule 109
TRVRowGenerationNestedRule.MergeWithPrevious 110
TRVRowGenerationNestedRule.SkipLeftColCount 109
TRVRowGenerationNestedRule.SkipRightColCount 109
TRVRowGenerationNestedRules 111
TRVRowGenerationNestedRules.Items 111
TRVRowGenerationRule 111
TRVRowGenerationRule.AllowSummaryCols 112
TRVRowGenerationRule.AllowSummaryRows 112
TRVRowGenerationRule.CaseSensitive 112
TRVRowGenerationRule.CopyColCount 113
TRVRowGenerationRule.CopyFirstCol 113
TRVRowGenerationRule.CopyMaxCount 113
TRVRowGenerationRule.CopySpacingColCount 113
TRVRowGenerationRule.KeyFieldNames 115
TRVRowGenerationRules 115
TRVRowGenerationRules.Items 115
TRVShape 68
TRVShape.BackgroundColor 68
TRVShape.BackgroundOpacity 68
TRVShape.Color 69
TRVShape.EqualSides 69
TRVShape.GradientType 69
TRVShape.LineColor 69
TRVShape.LineUsesFillColor 70
TRVShape.LineWidth 70
TRVShape.Margin 70
TRVShape.Opacity 69
TRVShape.Padding 70
TRVShape.ShapeProperties 70
TRVShape.StartColor 70
TRVShapeItemInfo 116
TRVShapeItemInfo.Alt 116
TRVShapeItemInfo.BackgroundOpacity 117
TRVShapeItemInfo.Color 117
TRVShapeItemInfo.EqualSides 117
TRVShapeItemInfo.GradientType 117
TRVShapeItemInfo.Height 119
TRVShapeItemInfo.LineColor 118
TRVShapeItemInfo.LineUsesFillColor 118
TRVShapeItemInfo.LineWidth 118
TRVShapeItemInfo.Opacity 117
TRVShapeItemInfo.ShapeProperties 119
TRVShapeItemInfo.StartColor 119
TRVShapeItemInfo.Width 119
TRVValueVisualizerId 165

- U -

UseRecordCount 52, 56
  TRVDataSetItem 56
  TRVReportCustomDBDataProvider 52
UserInterface 158
  TrvrActionInsertShape 158

- V -

VAlign 134
  TRVReportCustomValueVisualizer 134
Value 103, 168
  TRVConditionalShapePropertiesItem 168
  TRVReportTableCellData 103
Value visualizers 119
ValueString 135
  TRVReportCustomValueVisualizerBase 135
Variables 15, 35, 82
  TCustomRVReportGenerator 35
  TRVReportTableItemInfo 82
VisualizerId 104
  TRVReportTableCellData 104

- W -

Width 119, 158
  TrvrActionInsertShape 158
  TRVShapeItemInfo 119

- Y -

YellowAreaColor 137
  TRVReportGaugeVisualizer 137

YellowAreaPercent 138
TRVReportGaugeVisualizer 138