Report Workshop

Table of Contents

1 Report Workshop

Report Workshop is a set of VCL, Lazarus (for Windows), and FireMonkey components producing rich text reports from templates. **Report Workshop** is an add-on to **TRichView** components.

VCL and Lazarus versions are complete.

FireMonkey version does not include actions[215] (user interface for editing report templates).

Features

What's new?[21]

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 and export 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 the whole document, 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 LiveBindings, 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[11]

Template syntax:

- Fields[26]
- Data fields[27]
- Variables[30]
- Commands[32]
- Cross-tab header fields[36]
- Functions[37]
- Data field types[53]
- Format strings[56]

Report tables[124]

Components

Report generation

TRVReportGenerator [79]: a component for making reports

Data providers

A list of data provider components [80]

Additional components

TRVShape [114]: a component that draws a shape, such as circle, polygon, star, flag, emoticon, etc.

Actions

Report Workshop includes a set of actions [215] providing user interface for editing report templates.

1.1 Overview

General Information

Definitions [11]

Additional Information

- Data queries [13]
- Extending Report Workshop [17]
- Limitations in old versions of Delphi [20]
- Version history [21]

1.1.1 Definitions

Below you can find a list of definitions of key terms used by Report Workshop [10].

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 [26]**:** a part of text (enclosed in curly brackets) that are processed by a *report generator* specially. There are six types of fields:

- *data field,*
- *variable,*
- *command,*

- *cross-tab header field,*
- *aggregate function,*
- *expression.*

Report generator[79]**:** a component that processes a TRichView component containing a *report template* to produce a *final report.*

Data provider[92]**:** 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[13]**:** a text string containing a query that are processed by a *query processor*. Data queries are contained in the report itself[249], in report table row generation rules[156], and in cells[152]. Example: SQL SELECT query.

Query processor[252]**:** 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[124]**:** 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[155]: 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[150]**:** 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.Report table cells may have names.

Data field[27]**:** 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* or *report table cell* (by its name); in this case, a *query processor* created for this rule's or cell's *data query*.

Variable[30]**:** 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[32]**:** one of types of *fields* in a *report template*; commands invoke some special operations.

Background visualizer[174]**:** an object allowing to visualize values associated with report table cells on these cells' background.

Background color changer[174]**:** an object allowing to visualize values associated with report table cells by changing these cells' background colors and opacity.

Definitions for cross-tab reports

A cross-tab report[130] 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[140] (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[37] 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[132]. 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[36] 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 (datasets)
- tables (grids) 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.

1.1.2 Data queries

Data queries in reports

Data query is a text string containing a query.

A report generation mainly consists of executions 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[247].DataQuery[249])
- a group of table rows (TRVRowGenerationCustomRule[155].DataQuery[156])
- a table cell (TRVReportTableCellData[150].DataQuery[152])
- Query()[49] function in expressions

- (there are also data queries for building columns of cross-tab reports, but we skip them here for 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.

Additionally, Query() function may be inserted in any text, and Query() functions may be nested.

In all levels of this hierarchy, 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[94] component, and it has two items in DataSets[97] 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[62] 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[17].

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[27].

Fields in data queries

All '{' characters having the following '}' are treated as field codes[26]. 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[66] to prepare queries.

1.1.3 Scripts

Scripts in reports

Script is a text string containing commands to execute.

Scripts are executed on report generation, when certain events occur.

The following scripts are available:

- Script_OnStart, Script_BeforeRecord, Script_AfterRecord, Script_OnEnd [250] associated with a whole report template
- Script_OnStart, Script_BeforeRecord, Script_AfterRecord, Script_OnEnd [168] associated with a table row generation rule [163].

The main purpose of scripts is calculation of values of variables [30].

Syntax

Each script line is one command.

Empty lines and lines that start from "//" are ignored.

The script supports the commands: $if, $else, $endif, $set. Each line must include exactly one command.

The syntax of commands is the same as the syntax of commands that can be inserted as a field in text [32].

In scripts, the initial "$" character can be omitted, so you can write both

```
set %myvar = 1
```

and

```
$set %myvar = 1
```

Example

Let we have the following variables:

- %count - the count of items;
- %sum_price - the sum of prices of all items.

We need to calculate an average price and assign it to %avg_price variable.

The script is:

```
if +%count=0
set %avg_price = 0
else
set %avg_price = +%sum_price/+%count
endif
```

Note 1: since all variables are text variables, we need to convert them to numbers before calculation. We do it using "+" operator. See details in the topic about expressions [41].

Note 2: we cannot use this code:

```
// wrong code
set %avg_price = If(+%count=0,0,+%sum_price/+%count)
```

It's because If() calculates all its parameters, and a division by zero would occur if %count = '0'.

1.1.4 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 expressions
- additional aggregate 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 [27]: <full data field name>[<field type>]["<format>"]

Report Workshop implements standard field types [53], 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]'

Note that a part of this functionality can be implemented using functions for expressions (see below). Moreover, we already added Upper() and SpellNumber() functions. When you need working with text, number, logical and date-time values, functions are preferred. But data types are useful to support additional types of binary data, or additional document formats.

Implementation

How to implement

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

How to register

RVReportPluginManager.RegisterFieldObject(<field type name>, <object of your class>);

How to unregister

RVReportPluginManager.UnRegisterFieldObject(<field type name>, <object of your class>);

Standard query processors

The main way for handing data queries in TRVReportGenerator[79] is assigning a data provider component to its DataProvider[62] property. This data provider is responsible for creating query processors to execute data queries.

Additionally, you can create query processors in OnCreateQueryProcessor[67] 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[79] is assigning a data provider component to its DataProvider[62] property. This data provider is responsible for creating query processors to execute data queries.

Report Workshop includes TRVReportDBDataProvider[94] 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[94]), or another data provider components (inherited from TRVReportDataProvider[92]).

Additional data visualizers

Data visualizers[174] display diagrams that visualize values at backgrounds of report table cells[150].

Programmers can implement their own visualizers.

▼ **Implementation**

How to implement

Create a class inherited from TRVReportCustomValueVisualizer[191]. Override the methods: DisplayValue, GetContentSize.

Additional functions for expressions

Report Workshop provides a set of functions that can be used in expressions[41].

Programmers can implement additional functions.

▼ **Example**

For example, RVReportCharCodeCalculator unit implements 'char' function. The parameter is a character code (UTF-32). The result is the character.

▼ **Implementation**

How to implement

Create a class inherited from TRVReportCustomExpressionFunctionCalculator. Override the methods: CalculateFunction, GetFunctionArgCount, GetFunctionInfo.

How to register

RVReportPluginManager.RegisterExpressionFunctionCalculator(<function name>, <object of your class>);

How to unregister

RVReportPluginManager.UnRegisterExpressionFunctionCalculator(<function name>, <object of your class>);

Additional aggregate functions for cross-tab reports

Report Workshop provides a set of aggregate functions for cross-tab reports [37], 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 TRVReportCustomAggregateFunctionCalculator. Override the methods: CalculateFunction, GetFunctionFieldType.

How to register

RVReportPluginManager.RegisterAggregateFunctionCalculator(<function name>, <object of your class>);

How to unregister

RVReportPluginManager.UnRegisterAggregateFunctionCalculator(<function name>, <object of your class>);

1.1.5 Limitations in Lazarus and old versions of Delphi

Delphi 5

In Delphi 5, format strings [56] for string fields are ignored.

Calendar data provider is not available for Delphi 5; however, all calendar functions are available in expressions [41].

Delphi 5-XE

In Delphi versions prior to XE2, GDI (standard Windows drawing) is used for value visualizers [174] instead of GDI+. No gradients, no anti-aliasing, no semitransparent filling of shapes.

Lazarus

In Lazarus, by default, GDI (standard Windows drawing) is used for value visualizers [174] instead of GDI+. No gradients, no anti-aliasing, no semitransparent filling of shapes.

However, GDI+ can be used optionally. To use it:

1. Download ***GDI+ Library for Delphi and Lazarus*** and compile lazgdiplus.lpk. Alternatively, you can install it using Lazarus Online package manager (menu "Package | Online Package Manager")
2. Compile <TRichView Dir>\TRichView\Source\rvgdipluslaz.lpk (Note: it is included in <TRichView Dir>\TRichViewLazarus_Optional.lpg project group)
3. Include a reference to rvgdipluslaz package in your project (Menu "Project | Project Inspector" to open Project Inspector window. Right click "Required Packages", select menu "Add...". Select RVGDIPlusLaz)
4. Include the unit RVGDIPlusGrInLaz in "uses" of the main form unit.

Note: there is an alternative GDI+ support package for Lazarus (rv**p**gdipluslaz.lpk, containing RV**P**GDIPlusGrInLaz unit). It uses GDI+ library by Progdigy. It is not available now at its original location, but ***can be downloaded from our website***. Still, I recommend using the package above.

1.1.6 Version history

Changes after 6

Compatibility issues:

Changes in packages for Lazarus: The separation of runtime and designtime packages for Lazarus has been removed: instead of the two packages *rvreportworkshoplaz.lpk* (runtime) and *rvreportworkshoplaz_dsgn.lpk* (designtime), there is now a single package, *rvreportworkshoplaz.lpk* (runtime + designtime). These changes have been applied to all Lazarus packages.

Changes in version 6

(including changes in version 6.1)

Delphi and C++Builder 12 are supported.

Support of "Windows 64-bit (Modern)" platform in C++Builder 12+.

FireMonkey

FireMonkey is supported for Delphi XE6 and newer, all platforms.

Ported to FireMonkey: everything except for actions [215]. The following data providers are available for FireMonkey: TRVReportBindSourceDataProvider [86], TRVReportDBDataProvider [94], TRVReportFDDataProvider [102], TRVReportFDMongoDataProvider [103], TRVReportDBXDataProvider [100], TRVReportIBODataProvider [106].

Lazarus

Lazarus for Windows is supported. The following data providers are available for Lazarus:

TRVReportDbfDataProvider [97] (data from DBF tables)

TRVReportSQLDataProvider [111] (data from various SQL databases)

Report generation in a background thread

TRVReportGenerator[61].Execute[66] has a new optional UseThread parameter. If *True*, a report is generated in a background thread.

The same parameter is added the the function that generates a report in ScaleRichView: GenerateReport[241].

New event: OnGenerated[78] occurs when report generation is complete.

New properties: Generating[63], SynchronizedEvents[63].

Barcode extension

Barcodes with Zint for Delphi[118].

Other changes

Script_OnStart and Script_OnEnd[250] (associated with a whole report template) are executed even if the global DataQuery[249] is not defined.

Changes in version 5

Compatibility issues:

Empty (NULL) integer values are not converted to 0 automatically anymore.

Details

It may be a problem if the field is used in a SQL statement like this:

```
select * from MyTable where MyField={ValueThatCanBeNULL}
```

For NULL fields, it now produces incorrect statement

```
select * from MyTable where MyField=
```

To fix this problem, this statement can be changed to:

```
select * from MyTable where
  MyField={=If(Defined(ValueThatCanBeNULL),ValueThatCanBeNULL,0)}
```

This expression[41] returns 0 for NULL fields.

Another option, to check for NULL fields in SQL statement:

```
select * from MyTable where
  {'=If(Defined(ValueThatCanBeNULL),"MyField="+ValueThatCanBeNULL,"M
```

LiveBingings

New data provider component for Delphi XE3 or newer: TRVReportBindSourceDataProvider[86]. It provides data for reports using LiveBindings.

Report template generation

TrvrActionInsertTable[224] can insert a report table generated basing on a data table (i.e., on a dataset).

New action: TrvrActionReportWizard[227] generates a new report template. It supports multilevel master-detail reports.

Empty fields

Change: since this version, empty (NULL) database fields are not displayed in data fields [27]; expressions process empty fields specially [52].

New data function for expressions [49]: Defined(Value).

HTML

A new field type [53] is supported: HTML.

Report cells

Report table cells now may have names [153], and can be referred from data fields [27] by names, like row generation rules.

TrvrActionCellProperties's [216] dialog allows defining this property.

UI

Improvement: TrvrActionRowGenerationRules's [231] dialog allows defining Essential [167] property of rules (checkbox "Delete the whole table if no results")

Commands

$IFDEF and $IFnDEF [32] commands support cross-tab heading fields [36] (in addition to data fields and variables).

Changes in v4

RAD Studio 11 Alexandria

Delphi and C++Builder 11 Alexandria are supported.

Expressions

New functions for expressions [41]: GetDayOfWeek, GetWeekOfMonth, MonthName, MonthShortName, DayOfWeekName, DayOfWeekShortName.

New commands:

{$HEADER} and {$FOOTER} [32]

Field types

A new field type [53] is supported: Markdown.

Field formats

A new format option [56] for integer values: lower Greek

Help files

ReportWorkshop supports help files for end users (see TRVAControlPanel.UseHelpFiles)

Changes in v3

Compatibility issues:

- A new parameter (RelatedObject) is added to RVReportGetErrorString[242] procedure.
- In fields, if a variable name is in single quotes, '%' character must be inside the quotes.
- Some classes and methods related to implementation of custom aggregate functions[17] are renamed: TRVReportCustomFunctionCalculator to TRVReportCustomAggregateFunctionCalculator, RegisterFunctionCalculator to RegisterAggregateFunctionCalculator, UnregisterFunctionCalculator to UnregisterAggregateFunctionCalculator. It was done to avoid confusion with functions used in expressions.

RAD Studio 10.4 Sydney

Delphi and C++Builder 10.4 Sydney are supported, including per-control VCL styles; in dialogs, previews use the style of the target editor.

Expressions (new!)

New field type: expressions[41].

Expressions are also used in $IF and $SET command[32] and value visualizers[195].

Query() function is worth a special notice, it is a new way of applying data queries[13].

Commands

Expressions are used as conditions of $IF command.

A new command is added: $SET. It allows assigning a value to a report variable.

Scripts (new!)

New feature: scripts[15] that can be executed while a report is generated.

Field types

A new field format[53] is supported: DocX.

Other changes

- In all places where a text string is converted to a number, both dot and comma characters are allowed as a decimal separator.
- Cross-tab header fields can be visualized[195].
- Ability to delete the whole report table if its row generation rule returned 0 records: Essential[167] property for row generation rules.

Changes in v2

Compatibility issues:

- TRVReportShapeProperties[254] is moved from RVReportValueVisualizer to RVReportShapes unit.
- Demo projects for Delphi are moved to Demos\Delphi\ folder. To prevent duplicates, uninstall the previous version of Report Workshop before installing v2.0

RAD Studio 10.3 Rio

Delphi and C++Builder 10.3 Rio are supported.

Hi-DPI

Report Workshop supports high-dpi display modes and zooming in TRichView (dialogs and visualizers were modified). "Per monitor v2" is supported in RAD Studio 10.3.

New features

User interface is translated to multiple languages.

Functions for ScaleRichView [241] are added.

Data providers

TRVReportZEOSDSDataProvider [113] – new data provider component for ZEOS library.

TRVReportDxMemDataProvider [101] – new data provider allowing to implement master/detail reports on TdxMemData datasets (by ***Developer Express Inc.***)

Commands

$If command [32] is extended: conditions may be comparison operators.

Data fields

New data field types [53]: minutes, seconds, mseconds. They allow to represent an integer (containing a time interval) as a time value.

Shapes

New shape types [236] (the initial release included 7 shapes).

TRVShape [114] – a new component that draws a shape.

TRVShapeItemInfo [169] – a new document object that draws a shape.

TrvrActionInsertShape [219] – a new action for inserting shape objects in editors. TrvActionInsertProperties can edit properties of shape items.

New property ShapeScaleX [191] for visualizers displaying shapes; it allows inscribing shapes in rectangles instead of squares.

Demo projects

All Demo projects are moved from Demos\ to Demos\Delphi\ folder. ReportEditor demos are restructured: now they all use the same main form unit. New versions of ReportEditor demos were added for Delphi 10.3+: they use virtual image lists (containing 16x16, 32x32 and selected 64x64 images) and support "per monitor v2" mode. A complete set of ReportEditor demos (for Delphi 5-2007 with 16 color toolbar images, for Delphi 2009+ with modern images, for Delphi 2009+ with moden images using ScaleRichView, for Delphi 10.3+ using virtual image lists, for Delphi 10.3+ using virtual image lists with ScaleRichView) is available for many database components, where it is possible (FireDAC, IBX, MicroOLAP DACs, NexusDB, ElevateDB, ZEOSLib, BDE).

Other

new event: TRVReportGenerator [79].OnGetField [78] (an informational event)

1.2 Template syntax

Syntax of Report Templates

- Fields 26
- Data fields 27
- Variables 30
- Commands 32
- Cross-tab header fields 36
- Aggregate functions 37
- Expressions 41
- Data field types 53
- Format strings 56

1.2.1 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> | <aggregate function> | <expression> **}**

Data fields 27

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

Variables 30

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

Commands 32

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

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

Cross-tab header fields 36

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

Aggregate functions 37

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

Expressions [41]

<expression> ::= **=**<expression text>[<result 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, to insert '{' in text, 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).

1.2.2 Data fields

See the topic about fields in templates [26] 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> ::= <name>**:** | **:** | **^+:**

<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[28] section below.

<format> – see data field format strings[29] section 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[249]
- table row generation rule's[155] data query[156]
- table cell's[150] data query[152]
- Query() function in expressions[41]

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[249]; if this data query is not assigned, there is no root query processor).

If an identifier <name> is included, it must be equal to name[157] of table row generation rule[155], or a name of name[153] of report table cell[150]; a query processor associated with the referred object is used.

Special values:

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

and so on

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[163] with name[157] = 'rowrule'

Data field types

A report generator supports the following types of fields:

- text
- integer value
- floating point value
- boolean (logical)
- date, time, date and time
- memo (a text document in ANSI or Unicode)
- RTF document (normal RTF and RTF converted to a Unicode string)
- HTML document

- DocX document
- Markdown document
- RVF document
- image (bitmap, icon, metafile, jpeg, gif, png, tiff, svg, if graphic classes for these formats are available).
- additional field types implemented by programmers

If not specified, a format returned by TRVReportQueryProcessor[252].GetFieldType[253] is used.

You can override the field type by specifying <field type>, read the topic about specifying data field types[53].

Data field format strings

Read the topic about format strings[56].

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[32].

Note: if the field value is empty (NULL), it is not inserted in template text or strings.

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}'.

1.2.3 Variables

See the topic about fields in templates [26] for a general description of the syntax.

Syntax

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

A variable name (including the starting '%' character) may be enclosed in single quotes. These single quotes are necessary, if it contains space characters.

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

<format> – see variable format strings [31] sections below.

Variable names are case insensitive.

About variables

Variables are defined in string lists of the whole report [65] and of report tables [127].

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 processing content of this table's cells while applying row generation rules [126];
- when processing this table's cells' data query strings [152].

They are not used:

- when processing data query strings belonging to this table's row generation rules [126],
- 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>

When variables are assigned

Before the report generation, you can create an initial set of variables for the whole report[65] and for report tables[127].

You can change values of variables and add new variables:

- using $SET command[32] inserted in a report template text
- using scripts[15]
- in your code, in events.

When a report generation is finished, all variables that were calculated during the generation remain valid. So make sure to reset global report variables[65] to default values before the next generation.

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[79] event.

Variable format strings

Read the topic about format strings[56].

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[32];
- in expressions[41].

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}
{$IF %fullreport="yes"}
Address: {address}
Phone: {phone}
{$ENDIF}
```

generates the output like:

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

1.2.4 Commands

See the topic about fields in templates [26] 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).

"HEADER" and "FOOTER"

These commands define a part of document to apply the document's data query [249].

This data query is applied to the part of document between {$HEADER} and {$FOOTER}

These commands are special:

- they are valid only if the document's data query is defined
- they must start a new paragraph
- the next item (if exists) must start a new paragraph as well
- all text on the same line as {$HEADER} and {$FOOTER} is discared
- these commands are processed before any other commands, so they cannot be placed inside "If" command.

Example:

```
Fruits:
{$HEADER}
- {FruitName}
{$FOOTER}
Generated {=CurDate()}
```

produces a result like this:

```
Fruits:
- Apples
- Oranges
- Pears
Generated 21.12.2021
```

Do not rely to the order of processing headers and footers (it may be important if you use "Set" commands to assign variable values).

Currently, the order is the following:

- if the document's data query returns 0 records, a header is processed before a footer;
- otherwise, the order of processing: a repeated part, a footer, a header.

But this order may be changed in future.

"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> ::= <expression text>

See the topic about expressions [41] on definition of <expression text>.

Example 1:

Two identical examples:

```
{$IF count}Count of details: {count}{$ELSE}No details available{$ENDIF}
{$IF count<>0}Count of details: {count}{$ELSE}No details available{$ENDIF}
```

```
{$IF test="Passed"}OK{$ENDIF}
```

```
{$IF weight>200}Too heavy{$ENDIF}
```

This command can also be used in scripts [15].

"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>|**#**<cross-tab field>

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>|**#**<cross-tab field>

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

"SET"

Assigns the variable value.

Syntax:

<set> := **set** [']**%**<variable name>['] **to** <expression text> | **set** [']**%**<variable name>['] **=** <expression text>

See the descriptions of variables[30] and expressions[41].

These two versions of SET command are almost identical, but there is a difference in handling undefined variables. The version with "TO" reports an error for unknown variables. The version with "=" adds a new variable in global Variables[65].

This command can also be used in scripts[15] (and it's recommended to use scripts for variables assignments).

Example:

```
{$set %hello = "Hello world"}{%hello}
```

produces

```
Hello world
```

Example:

Please note that values of variables are strings, so

```
{$set %v to 1}{$set %v to %v+1}{%v}
```

produces

```
11
```

because the variable value becomes "11".

To produce 2, convert the variable to a number:

```
{$set %v to 1}{$set %v to ToNumber(%v)+1}{%v}
```

or

```
{$set %v to 1}{$set %v to +%v+1}{%v}
```

1.2.5 Cross-tab header fields

See the topic about fields in templates[26] for a general description of the syntax.

See the topic about cross-tab reports[130].

Syntax

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

A cross-tab field name (including the starting '#' character) may be enclosed in single quotes. These single quotes are necessary, if it contains space characters.

<cross-tab field> is a value of one of fields specified in Table.CrossTabulation[126].Levels[139][].FieldName[145] or Table.CrossTabulation[126].Levels[139][].CaptionFieldName[141].

Field names are case insensitive.

About cross-tab header fields

References to cross-tab fields may occur:

- in cells of the cross-tab header[132];
- in Table.CrossTabulation[126].Levels[139][].DataQuery[143], if Table.CrossTabulation[126].ColumnGenerationType[135]=*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[145] field):

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

When referring to a value of CaptionFieldName[141], the field type is always text.

You can override the field type by specifying <field type>, read the topic about specifying data field types[53].

Cross-tab header field's format strings

Read the topic about format strings[56].

Example

Let we have:

- Table.CrossTabulation[126].Levels[139][0].CaptionFieldName[141] = 'Category'

- Table.CrossTabulation[126].Levels[139][1].CaptionFieldName[141] = '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[146] (replacing fields in header cells)
- in the topic about cross-tab data queries[143] (replacing fields in cross-tab data queries)

1.2.6 Aggregate functions

See the topic about fields in templates[26] for a general description of the syntax.

See the topic about cross-tab reports[130].

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

Aggregate 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[79] event to calculate the function in a variable[30].

Syntax

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

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

<param> ::= <field name>

<result type> – see function results types[40] sections below.

<format> – see format strings[40] sections below.

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

About aggregate functions

This type of field allows inserting the result of an aggregate function 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[130]).

In other words, <param> is a name of one of data fields in the result of application of one of Table.RowGenerationRules[126][].DataQuery[156], providing that it:

- is not listed in Table.CrossTabulation[126].Levels[139][].FieldName[145]
- is not listed in Table.RowGenerationRules[126][].KeyFieldNames[168]
- have the type[234] 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[132] or above: not valid (no values)
- in the cells located in the intersection of rows corresponding to RowGenerationRules[126][] 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 Tot**
	{#Product}	Total for {#	
{Year}	{Sales}	{sum(Sales	**{sum(Sales**
Grand Tot	{sum(Sales	**{sum(Sales**	**{sum(Sales**

Result Sample

	Fruits			Vegetables			**Gran**
	Appl	Oran	Total	Toma	Cucu	Total	
2014	100	200	300	300	400	700	**1000**
2015	150	250	400	350	450	800	**1200**
Gran	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)

Aggregate 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 the aggregate function cannot be calculated (because it is inserted in a wrong context, or because of not enough input values), a report generator inserts Texts[64].InvalidFunctionResult in place of the function field. If the aggregate function is used in an expression[41], and it cannot be calculated, the generator reports an error.

Additional aggregate 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[17]

Aggregate functions results types

As you can see from the table above, an aggregate 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[53].

Using aggregate functions in fields of other types

Aggregate functions may be used not only in a special field type; they can also be used in expressions (in fields of expression type[41], and in $IF commands[32]).

In expressions, parameters must be enclosed in double quotes, for example: {$IF Sum("Sales")>0}, or {=Max("Date")-Min("Date")}.

When used separately, double quotes are optional; you can write {Sum(Sales)} or {Sum("Sales")}.

Data field format strings

Read the topic about format strings[56].

Format string is not applied to Texts 64.InvalidFunctionResult.

1.2.7 Expressions

See the topic about fields in templates 26 for a general description of the syntax.

Syntax

<expression> ::= [']=<expression text>['][<result type>]["<format>"]

An expression text (including the starting '=' character) may be enclosed in single quotes. These single quotes are necessary, if it contains space characters. To insert a single quote in a quoted expression, use a double single quote.

<result type> – see "expression results types" section below.

<format> – see "expression result format strings" section below.

Examples:

- {=Value+1}
- {=(x+ln(x))/2}
- {'=if(Score>=10,"win","try again")'}

Syntax in $IF and $SET commands

{$IF <expression text>}

{$SET %variable TO <expression text>} or {$SET %variable = <expression text>}

When used in $IF and $SET, expression text does not need single quotes.

About expressions

An expression is a construction that returns a value.

Expressions allow calculating a numeric, text, date-time, or boolean (logical) values.

Expressions are used:

- in fields 26, to insert a result in a report
- in $if command 32 as a condition
- in $iset command to assign a result to a variable 30
- in $if and $set commands in scripts 15.

Operators

List of operators

Expressions can include binary operators listed in the next table.

Character	Operator

+	for numbers: addition
	for strings: concatenation

-	subtraction of numbers
*	multiplication of numbers
/	division of numbers
|	logical OR
&	logical AND
= or ==	equality
<> or !=	inequality
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Expressions can include unary operators listed in the next table.

Character	**Operator**
+	sign identity
-	sign negation
!	logical negation

Precedence

In complex expressions, rules of precedence determine the order in which operations are performed.

Precedence of operators:

- unary + - ! (highest)
- * /
- binary + -
- = == < > <= >= <> !=
- &
- | (lowest)

An operator with higher precedence is evaluated before an operator with lower precedence, while operators of equal precedence are evaluated from left to right (except for unary operators, which are evaluated from right to left).

You can use parentheses to override these precedence rules. An expression within parentheses is evaluated first, then treated as a single operand.

Example:

```
(A+B)*C
```

multiplies C times the sum of A and B.

Relational operators

Relational operators (such as = or <) are used to compare two operands.

Operands may be converted to another type before the comparison (see the section about type conversions[50] below):

- empty (NULL) value is less than any other value
- if at least one of operands is a Boolean value, the second operand is converted to a Boolean value as well (e.g. {="true"=True()} returns *True*)
- otherwise, if one operand is text and another operand is numeric, text operand is converted to a number (e.g. {="1.1"=1.1} returns *True*)
- otherwise, operands are compared if they have compatible types.

If operands cannot be converted, or operands have incomparable types, an error occurs.

Strings are compared in lexicographical order, letter by letter, from left to right. The comparison is case sensitive.

For Boolean (logical) operands, *True* > *False*.

Addition and concatenation (+)

If one operand is text, another operand is converted to text, and the operator performs concatenation of strings.

Operands

The following values can be used as operands:

- numeric constants
- string constants
- variables
- cross-tab header fields
- aggregate functions
- other functions

Numeric constants

Examples: 12, 12.34, 12.34e56,12.34e+56, 12.34e-56.

The dot character is used as a decimal separator. 'E' or 'e' can be used to separate a mantissa and an exponent.

Note: if a string value is converted to a number, both dot and comma characters can be used as a decimal separator; but in numbers in an expression, only dot can be used.

String constants

Example: "Hello world!", "so called ""doctors""".

A string constant is enclosed in double quotes; to use a double quote in a string, insert it twice. If the expression includes strings containing space characters, the whole expression text must be enclosed in single quotes, e.g: {'=lower("Hello world!")'} (this rule is only for expression field types; the expression does not need to be quoted when used in $IF command)

Variables 30

Examples: %myvariable, '%my variable', or [%my variable].

Names of variables may be enclosed in single quotes or square brackets; it is necessary if they include space characters. If the expression includes variable names containing space characters, the whole expression text must be enclosed in single quotes, e.g: {'=[%my variable]+1'} or {'=''%my variable''+1'} (single quotes inside single quotes must be doubled) (this rule is only for expression field types; the expression does not need to be quoted when used in $IF command)

Only text values of variables are supported in expressions (graphic values are not supported)

Data fields 27

Examples: Total, SalesTable:Total, :Total, ^:Total, 'Total sales', [SalesTable:Total sales].

Data fields may be enclosed in single quotes or square brackets; it is necessary if they include space characters in names. If the expression includes data fields containing space characters, the whole expression text must be enclosed in single quotes (single quotes inside single quotes must be doubled) (this rule is only for expression field types; the expression does not need to be quoted when used in $IF command).

Names of data fields in expressions must not be the same as function names. To make them distinct, you can use full names, or enclose them in single quotes or square brackets

Example (incorrect, there is len() function): {=len}

Correct: {=mytable:len+1} {=[len]+1} {'=''len''+1'} (a single quote before the opening '=' character is a quote around the whole expression text; so quotes around the field name are doubled)

Only text, numeric, boolean (logical), and date-time field values are supported.

Cross-tab header fields 36

Example: #Category

Aggregate functions 37

Example: Sum("Sales"), Min("Sales").

Arguments (field names) must be strings. Like other string constants, they must be enclosed in double quotes.

Other functions

Example: Lower("HELLO"), MakeTime(Hour, Minutes, Seconds), Sqrt(2).

Functions are discussed in the next section of this topic.

Functions

Function names are case insensitive.

Basic math functions

Function	Meaning
Div(X,Y)	Calculates integer division of *X* by *Y*

Mod(X,Y)	Returns the reminder of integer division of *X* by *Y*
Round(X,Digits)	Rounds *X* to *Digits* number of digits. Examples: • Round(1.234, 2) = 1.23 • Round(123.4, -2) = 100 • Round(2.5, 0) = 3 (integer value) The function works by rounding numbers 1-4 down, and rounding numbers 5-9 up.
Ceiling(X)	Return the smallest integer value that is greater than or equal to *X*.
Floor(X)	Returns the largest integer value that is smaller than or equal to *X*.
Trunc(X)	Rounds *X* toward zero.
Abs(X)	Returns the absolute value of *X*
Sign(X)	Returns: • 0, if X is zero. • 1, if X is greater than zero. • -1, if X is less than zero.

Trigonometric functions

Function	**Meaning**
Sin(X)	The sine of *X* (in radians)
Cos(X)	The cosine of *X* (in radians)
Tan(X)	The tangent of *X* (in radians)
Cotan(X)	The cotangent of *X* (in radians)
Radians(X)	Converts *X* (in degrees) to radians
Degrees(X)	Converts *X* (in radians) to degrees
Pi()	π (approx. 3.14159)

Logarithms and exponents

Function	**Meaning**
Log(X,Base)	Returns the logarithm base *Base* of *X*

	$\log_{Base}X$
Log10(X)	Returns the logarithm base 10 of *X* $\log_{10}X$
Ln(X)	Returns the natural logarithm of X $\ln X = \log_e X$
Exp(X)	Returns the exponential of *X* e^X
Power(Base,Exponent)	Raises *Base* to *Exponent* power $Base^{Exponent}$
Sqrt(X)	Returns the square root of X $\sqrt{x}$

Min and max

See the section about relational operators [43] above for information about comparisons.

Report Workshop uses "Min" and "Max" names for aggregate functions [37], so it uses "MinVal" and "MaxVal" names for functions that compare two operands.

Function	Meaning
MinVal(X,Y)	Returns the lesser of *X* and *Y*
MaxVall(X,Y)	Returns the greater of *X* and *Y*

Text functions

Function	Meaning
Upper(S)	Converts *S* to upper case
Lower(S)	Converts *S* to lower case
UpperFirst(S)	Converts the first letter of S to upper case (note: the count of characters may be changed, if the first letter is a ligature)
Trim(S)	Trims leading and trailing spaces and control characters from *S*
LTrim(S)	Trims leading spaces and control characters from *S*
RTrim(S)	Trims trailing spaces and control characters from *S*

Substring(S,Index,Count)	Returns a substring of *S* containing *Count* characters starting from the *Index*-th character. Index of the first character is 1.
Left(S,Count)	Returns a substring of *S* containing first *Count* characters
Right(S, Count)	Returns a substring of *S* containing last *Count* characters
Repeat(S,Count)	Repeats the string *S Count* times
Len(S)	Returns the count of characters in *S*

Date and time functions

Date-time values may contain a date, a time, or both date and time.

Function	Meaning
Now()	Returns the current date and time
CurDate()	Returns the current date
CurTime()	Returns the current time
MakeDate(Year,Days)	Returns a date based on *Year* and a number of days *Days*. The first day in a year is 1.
DateFromParts(Year,Month, Days)	Returns a date based on *Year, Month,* and *Days*. *Month* is from 1 to 12, *Days* is from 1 to 28 or 31, depending on the month.
MakeTime(Hour,Minutes, Seconds)	Returns a time based on the specified *Hour, Minutes*, and *Seconds*
GetDay(Date)	Returns the day of the month (from 1 to 31) of the specified *Date*
GetMonth(Date)	Returns the month (from 1 to 12) of the specified *Date*
GetYear(Date)	Returns the year of the specified *Date*
GetHour(Time)	Returns the hour of day (from 0 to 23) of *Time*
GetMinutes(Time)	Returns the minutes (from 0 to 59) of *Time*
GetSeconds(Time)	Returns the seconds (from 0 to 59) of *Time*

GetDayOfWeek(Date)	Returns the day of week (from 1 to 7, where 1 is Monday) of *Date*
GetWeekOfMonth(Date)	Returns a number (from 1 to 5) indicating which week of the month the date *Date* falls in
MonthName(Month)	Returns a name of the *Month* (where *Month* is a number from 1 to 12)*
MonthShortName(Month)	Returns a shortened name of the *Month* (where *Month* is a number from 1 to 12)*
DayOfWeekName(DayOfWeek)	Returns a name of the *DayOfWeek* (where *DayOfWeek* is a number from 1 to 7, 1 means Monday)*
DayOfWeekShortName(DayOfWeek)	Returns a shortened name of the *DayOfWeek* (where *DayOfWeek* is a number from 1 to 7, 1 means Monday)*

* names are returned in a system default language

For empty (NULL) date-time parameters, the functions above return the empty (NULL) value.

Logical functions

See the section about conversions below.

Function	**Meaning**
False()	Returns *False*
True()	Returns *True*
If(Condition,A,B)	If *Condition* is *True*, returns *A*. If *Condition* is *False*, returns *B*. This function is especially useful in tags of items, hints and checkpoint names (where $IF command is not available).

Note: both *A* and *B* parameters are calculated before If() calculation. This means that If(y=0, 0, x/y) produces "division by 0" error if y = 0.

Conversion functions

See the section about conversions below.

Function	**Meaning**

ToText(X)	Converts *X* to a text string
ToNumber(X)	Converts *X* to a number (the same as unary + operator)

Conversion functions, numbers to words

See the section about number to words conversion [50] below.

Function	Meaning
SpellNumber(Value, Language, Options)	Converts the integer *Value* to text in the specified *Language*
SpellCurrency(Value, Language, Currency, Options)	Converts the monetary *Value* to text in the specified *Language*

Data functions

Function	Meaning
Defined(Value)	Returns *True* if *Value* is not empty, returns *False* otherwise. Empty (NULL) values may come from empty database field
RecNo(Depth)	Returns a index of the currently processed record. Records are counted from 1. *Depth* = 0 means the most nested query, 1 means the query containing it, and so on.
Query(DataQuery,Format, Delimiter)	Executes the DataQuery [13] and returns the result. See the section about Query() function [52] below.

Example:

- If(Defined(Date),Date,"undefined date") returns the value of Date if it is not NULL, and "undefined date" text otherwise.

Custom functions

Programmers can add their own functions, see the topic explaining how to extend Report Workshop [17].

The following functions are implemented as an example.

Function	Meaning

Char(Code)	Returns the character for the given UTF-32 *Code*. (this function becomes available if you include RVReportCharCodeCalculator unit in your project)

Type conversion

Types of values may be converted explicitly (using ToText() and ToNumber() functions) or implicitly (when a function or an operator needs a value of another type).

Conversion to boolean values (*True* or *False*)

Text to boolean values:

Values evaluated as *True*	Values evaluated as *False*
"t"	"f"
"y"	"n"
"true"	"false"
"yes"	"no"
"on"	"off"
"1"	"0"
Texts 64 .TrueText	Texts 64 .FalseText

Number to boolean values: any non-zero value is converted to *True*, zero is converted to *False*.

Empty (NULL) value is converted to *False*.

Conversion to numbers

Boolean values to numbers: *True* is converted to 1, *False* is converted to 0.

Empty (NULL) value is converted to 0.

Strings to numbers: both dot and comma characters can be used as a decimal separator.

Example:

```
=1+3+"x"
```

returns "4x"

The first operation is 1+3 (sum of integer values), the second operation is 4+"x" (concatenation of strings, 4 is converted to "4").

Numbers to words conversion

SpellNumber() returns a text representation of an integer numeric value.

SpellCurrency() returns a text representation of a monetary value.

Language

The following values of *Language* parameter are supported:

Language	**Meaning**
• "ru" or • any string starting from "ru-"	Russian
• "pt" or • "pt-br"	Brazilian Portuguese
• "pt-pt"	European Portuguese
• any other string	English

The language string is case-insensitive.

SpellNumber() parameters

Options is a string that may contain the following characters:

***Options* character**	**Meaning**
"m" or "f" or "n"	Grammatical gender: male/female/neuter (male is assumed, if none is specified)
"$"	If included, monetary spelling rules are applied

The *Options* string is case-insensitive.

SpellCurrency() parameters

Options is a string that may contain the following characters:

***Options* character**	**Meaning**
"0"	If included, if fractional part of *Value* is zero (i.e. 0 cents), it is dropped.
"#"	If included, a fractional part of *Value* (i.e. cents) is written using digits instead of words.

The *Options* string is case-insensitive.

Options is a string that may contain the following characters:

Currency	**Meaning**
"usd" or "$"	United States dollar

"eur" or "euro" or "€"	Euro
"brl" or "r$"	Brazilian real
"rub" or "rur" or "sur"	Russian ruble
any other string	Default currency

The *Currency* string is case-insensitive.

The default currency depends on the *Language:*

Language	**Default currency**
English	United States dollar
Russian	Russian ruble
Brazilian Portuguese	Brazilian real
European Portuguese	Euro

Examples:

- SpellNumber(-10, "en", "") returns "minus ten"
- SpellCurrency(1.2, "en", "usd", "") returns "one dollar and twenty cents"

Query()

Query(DataQuery,Format,Delimiter) executes *DataQuery* [13] specified in the parameter. Each record of the result is applied to *Format* string, so processed *Format* is repeated record count times. *Delimiters* are inserted between.

For example, let we have a data query "select * from FruitTable" that returns 3 records with "apple", "orange", "peach" in "FruitName" field.

The call Query("select * from FruitTable", "{FruitName}", ", ") returns the string "apple, orange, peach".

Please note that when inserted in a field:

- "}" character must be doubled, otherwise it will be treated as a field end;
- since there are space characters, the whole field must be in single quotes.

```
We have: {'=Query("select * from FruitTable", "{FruitName}}", ", ")'}.
```

Empty (NULL) values in expressions

The following rules are applied to NULL values:

- any date-time function returns NULL for NULL parameter
- when used as a number, NULL is converted to 0
- when used as a Boolean value, NULL is converted to *False*
- when used as a string value, NULL is converted to empty string
- when comparing, NULL is less than any other value.

Expression result types

The expression may return values of the following types:

- text
- integer value
- floating point value
- boolean
- date and time
- empty value (NULL)

You can override the field type by specifying <result type>, read the topic about specifying data field types [53].

Expression result format strings

Read the topic about format strings [56].

1.2.8 Data field types

A type may be specified explicitly in:

- data fields [27]
- cross-tab header fields [36]
- functions [37]
- expressions [41]

Syntax

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

<alternative rvf name> is a text of RVReportGenerator [79].Texts [64].RVF.

Description

If specified, it is one of: text, int, float, bool, datetime, date, time, minutes, seconds, mseconds, blob, ansimemo, unicodememo, document, rtf, docx, html, markdown, rvf, image, bitmap, gif, png, jpeg, tiff, icon, metafile. As you can see, these types correspond to TRVReportFieldType [234].

Additionally, the value RVReportGenerator.Texts [64].RVF is treated as 'rvf'.

Additionally, programmers can register their own field types.

Field type	Meaning
text	text string, may be multiline
int	integer value (Int64)

float	floating point value (Extended for Delphi 2009 or newer, Double otherwise)
bool	boolean value
datetime	date and time (TDateTime)
date	date (TDateTime)
time	time (TDateTime)
minutes	converts integer value (number of minutes) to time (TDateTime)*
seconds	converts integer value (number of seconds) to time (TDateTime)*
mseconds	converts integer value (number of milliseconds) to time (TDateTime)*
blob	BLOB field containing either a document or a graphic; the actual content type is detected by format**,***,****
ansimemo	BLOB field containing ANSI text**, HTML, or RTF
unicodememo	BLOB field containing either Unicode (UTF-16) text or RTF (converted to UTF-16), or HTML (in UTF-16 encoding)
document	BLOB field containing text**,***, RTF****, DocX, HTML, or RVF
rtf	RTF****
docx	Microsoft Word Document (DocX)
html	HTML
markdown	Markdown (in UTF-8 encoding)
rvf, RVReportGenerator.Texts[64].RVF	RVF
image	graphic; format is detected by content
bitmap, gif, png, jpeg, tiff, icon, metafile	graphic of the specific format

* the result must be less than 24 hours

** if the field contains text, the report generator tries to detect if it contains Unicode (UTF-16) or ANSI

*** ANSI text encoding is defined in DefCodePage property of TRVStyle component linked to the editor containing the report template; by default, this is a default Windows code page

**** if the field contains RTF, the report generator tries to detect if RTF codes are stored as Unicode (UTF-16) or ANSI

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[56] from another field type (for example, int to float allows formatting integer values as floating point values)
- ignoring a field type returned by a query processor, and using the specified type;
- specifying a format that cannot be auto-detected (markdown)
- visualizing values in special ways (especially when using custom field types).

Field types in extension

Barcodes with Zint for Delphi[118] adds two new field types: barcode and qrcode.

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' (pictures are converted to empty 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, minutes, seconds, mseconds
time, date, datetime	time, date, datetime
blob, ansimemo, unicodememo, document, rtf, docx, html, markdown, rvf, image, bitmap, gif, png, jpeg, tiff, icon, metafile	blob, ansimemo, unicodememo, document, rtf, docx, rvf, html, markdown, 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 or to "text", 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 conversions in expressions[50].
- 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 [17]

1.2.9 Format strings

Format strings can be used in:

- data fields [27]
- variables [30]
- cross-tab header fields [36]
- functions [37]
- expressions [41]

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, RTF, DocX, HTML, Markdown)
- 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.

Note 3: empty (NULL) values are not inserted in text, regardless format strings

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" – the result uses dot as a decimal separator

Format strings for integer values

Format strings for integer values consist of up to three sections.

The first section represents positive values and defines the number format.

The second section represents negative values

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> | **greek**

<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
greek	α, β, γ, ..., αα, αβ, αγ

<min length> is a positive integer number, it specifies the minimal number of digits in the output string; if necessary, '0's 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, w, 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[64].TrueText is used.

<textF> (if specified) is a text for *False*; if empty, Texts[64].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, RTF, DocX, HTML, Markdown 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)
align	*rvepVAlign*	Vertical image alignment in a line, or alignment to the left/right side	one of: baseline, middle, abstop, absbottom, absmiddle, left, right

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"

1.3 Main Components

The main component included in ReportWorkshop:

TRVReportGenerator[79].

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

1.3.1 TCustomRVReportGenerator

A base class for TRVReportGenerator[79] component.

Unit RVReportGenerator;

Syntax

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

Hierarchy

TObject

TPersistent
TComponent

Description

This component is not used directly. Use TRVReportGenerator [79] instead.

Call Execute [66] to process a TRichView component containing a report template, in order to produce a final report. The report may be generated in the context of the main process, or in a background thread.

While processing a report template, the report generator processes data queries by creating query processors. They are created either by DataProvider [62], or in OnCreateQueryProcessor [67] event.
While generating reports, the following events occur:
- OnDataQueryProgress [68] allows to show a progress indicator or to abort the generation;
- OnError [70] allows to process errors;
- OnProcessRecord [79] allows to execute your code before processing each record in results of a query processor.

When report generation is finished, OnGenerated [78] event occurs.

The report generator has a list of global report variables [65].

See also

- Definitions of Report Workshop terms [11]

1.3.1.1 Properties

In TCustomRVReportGenerator

- 🟨 DataProvider [62]
- 🟩 ErrorColor [63]
- ▶ Generating [63]
- 🟩 Language [63]
- 🟩 SynchronizedEvents [63]
- 🟩 Texts [64]
- 🟩 Variables [65]

1.3.1.1.1 TCustomRVReportGenerator.DataProvider

Links the report generator with a data provider.

```
property DataProvider: TRVReportDataProvider [92];
```

A data provider receives data queries (for example, SQL SELECT statements) and create query processors [252] for these queries.

This is the main way to provide data for report generation.

Alternative (additional) ways:
- OnCreateQueryProcessor [67] event
- "standard" query processors [17] (processing data queries started from a registered prefixes)

See also:

- Definitions of Report Workshop terms [11]

1.3.1.1.2 TCustomRVReportGenerator.ErrorColor

Color for error messages.

```
property ErrorColor: TRVColor;
```

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

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

Default value:

rvclRed

1.3.1.1.3 TCustomRVReportGenerator.Generating

Returns *True* is a report is being generated.

```
property Generating: Boolean;
```

1.3.1.1.4 TCustomRVReportGenerator.Language

Specifies the current language of user interface.

```
type
  TRVALanguageName = type String;
property Language: TRVALanguageName;
```

This property is only available in FireMonkey version of ReportWorkshop (as a temporary solution).

VCL and Lazarus versions of ReportWorkshop use Language property of TRVAControlPanel component.

See also

- RWA_LocalizeErrorMessages [242] procedure
- RWA_LocalizeReportGenerator [243] procedure

1.3.1.1.5 TCustomRVReportGenerator.SynchronizedEvents

A set of events that are always called in the context of the main process.

```
type
  TRVReportGeneratorEventId =
    (rvrgeOnDataQueryProgress, rvrgeOnError);
  TRVReportGeneratorEventIds =
     set of TRVReportGeneratorEventId;

property SynchronizedEvents: TRVReportGeneratorEventIds;
```

Events included in this property are called in the context of the main process, even if reports are generated in a background thread (so you can work with user interface controls safely).

Value	Event
rvrgeOnDataQueryProgress	OnDataQueryProgress [68]
rvrgeOnError	OnError [70]

Default value:

[rvrgeOnDataQueryProgress, rvrgeOnError]

See also:

- Execute [66] method

1.3.1.1.6 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 [30].	'Unknown variable "%s"'
ErrorInvalidVariableValue	this text is inserted in place of a variable object field [30], 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 [27], if this data field refers to a non-existent table generation rule [126].	'Unknown rule "%s"'
ErrorUnknownField	this text is inserted in place of a data field [27], 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 [27], 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 [27], if this data field has a type unsupported by the report generator	'Type of field "%s" is not supported'

Property	Meaning	Default value
ErrorInExpression	this text is inserted in place of an expression field[41], if this expression is erroneous	'Error in expression'
RVF	an alternative name of rvf data field type[53]	'rvf'
TrueText, FalseText	default display values for boolean data fields[27]	'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[37] is inserted in a wrong context)	'–' (n-dash)

Call RWA_LocalizeReportGenerator[243] to localize these properties.

1.3.1.1.7 TCustomRVReportGenerator.Variables

A string list containing global variables[30] 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[127] in report tables.

See also:

- variables in templates[30]
- TRVReportTableItemInfo[124].Variables[127]
- TRVReportGenerationSession[251]

1.3.1.2 Methods

In TCustomRVReportGenerator

EscapeDataQuery[66]
Execute[66]

1.3.1.2.1 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[103].

See also

- data queries[13]

1.3.1.2.2 TCustomRVReportGenerator.Execute

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

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

The resulting document is not formatted, call **RV**.Format when report generation is complete.

Generating report in the context of the main process

If you call **Execute** with **UseThread** = *False*, the method generates a report and finishes when generation is completed.

If generation takes a long time, the application does not respond while this method is working. To solve this problem, you can call Application.ProcessMessages in OnDataQueryProgress[67] event. However, this method is not cross-platform, so it is not recommended to use it in FireMonkey applications. See also notes below.

In OnDataQueryProgress[67] event, you can provide some feedback (such as displaying a progress bar to show progress).

If generation is successful, the method returns *True*. There are may be some non-critical errors, though. You can control error handing using OnError[70] event.

Generating report in a background thread

If you call **Execute** with **UseThread** = *True*, the method starts a background thread for report generation. Then it immediately exits, returning *True*. When generation is completed, OnGenerated[78] event is called.

You must not call **Execute** while a report is being generated. In this case, **Execute** quickly finishes and returns *False* (and OnGenerated[78] will not be called for this nested call of **Execute**).

Special measures for using a thread and Application.ProcessMessages

If you generate a report in a background thread, or if you use Application.ProcessMessages, you must take the following measures:

1. Hide the TRichView control while a report is being generated, and show it only when finished. While generating, this TRichView is not ready for repainting, until you call its Format method.
2. Prevent the form closing while a report is being generated (check Generating[63] in the form's OnCloseQuery event).
3. Prevent subsequent calls of **Execute** while a report is being generated.

1.3.1.3 Events

In TCustomRVReportGenerator

- 🟨 OnCreateQueryProcessor[67]
- 🟩 OnDataQueryProgress[68]
- 🟩 OnError[70]
- 🟩 OnGenerated[78]
- 🟩 OnGetField[78]
- 🟩 OnProcessRecord[79]

1.3.1.3.1 TCustomRVReportGenerator.OnCreateQueryProcessor

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

```
type
  TRVCreateQueryProcessorEvent = procedure (
    Sender: TCustomRVReportGenerator[61];
    const DataQuery: TRVUnicodeString;
    var QueryProcessor: TRVReportQueryProcessor[252]) of object;

property OnCreateQueryProcessor: TRVCreateQueryProcessorEvent;
```

This is an optional event. Normally, query processors are created by a linked[62] data provider component[92]. 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.

If Execute[66] is called with parameter UseThread = *True*, this event is called in a thread context.

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[93] method of a data provider component.

See also:

- Definitions of Report Workshop terms [11]

1.3.1.3.2 TCustomRVReportGenerator.OnDataQueryProgress

Occurs while processing a data query.

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

  TRVDataQueryProgressEvent = procedure (
    Sender: TCustomRVReportGenerator[61];
    Rule: TRVRowGenerationCustomRule[155];
    Cell: TRVReportTableCellData[150];
    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[66].

By default, this event is called in the context of the main process (even if reports are generated in a background thread), to allow interacting with user interface controls. If you want to call it in a thread context, exclude *rvrgeOnDataQueryProgress* from SynchronizedEvents[63].

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 percentage (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 61 ;
  Rule: TRVRowGenerationRule 163 ; Cell: TRVReportTableCellData 150 ;
  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;
```

1.3.1.3.3 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,
    rvrgeFuncCannotBeCalculated,
    // expressions
    rvrExpressionParserError, rvrExpressionEvaluationError,
    // others
    rvrgeInternalError
    );

  TRVReportGeneratorErrorEvent = procedure (
    Sender: TCustomRVReportGenerator 61;
    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.

By default, this event is called in the context of the main process (even if reports are generated in a background thread), to allow interacting with user interface controls. If you want to call it in a thread context, exclude *rvrgeOnError* from SynchronizedEvents 63.

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. In this case, this error is treated as critical, and Execute 66 will return *False*.

Errors while checking table row generation rules for correctness

For this kind of errors, **RelatedObject** is TRVRowGenerationCustomRule 155.

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 155 or TRVReportTableCellData 150.

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[30]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[251].

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[32] 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[27]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[251].

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[32] cannot be evaluated as *True* or *False*, or a format of content of a Blob field cannot be detected	Field code or data field name
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[27], cross-tab header fields,[36] functions[37]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[251].

Error	Meaning	RelatedText
rvrgeFieldUnknownType	Unknown value in <field type>[53]	Field code
rvrgeInvalidTypeCast	The value cannot be converted to <field type>[53]	Field code
rvrgeInvalidFormatString	Error in <format string>[56]	Field code
rvrgeCustomFieldTypeError	An error was reported by a custom field type[17] handler. The reason may be because the field data are inappropriate, or a format string is erroneous.	Field code

Errors while processing commands[32]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession[251].

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 [130]

Error	Meaning	RelatedText	RelatedObject
rvrgeCrossTabInvalid	Table.CrossTabulation [126] is incorrect. Possible reasons: • empty Table.CrossTabulation [126] .Levels [139] [].FieldName [145]; • empty Table.CrossTabulation.Levels[ ].DataQuery [143], if they are required for the given ColumnGenerationType [135]; • invalid values (or a combination of values) for MinValue, MaxValue, Step [146], if ColumnGenerationType [135] = *rvcgtRange*	Empty	TRVReportTableItemInfo [124]
rvrgeCrossTabInvalidPosition	Position or structure of a cross-tab header [132] is incorrect.	Empty	TRVReportTableItemInfo [124]
rvrgeCrossTabNoRules	A cross tabulation is defined for a table, but this table has no row generation rules [126], or the first rule is invalid	Empty	TRVReportTableItemInfo [124]
rvrgeCrossTabUnknownField	Table.CrossTabulation [126] .Levels [139] [].FieldName [145] or CaptionFieldName [141] does not exists in the results of the respective data query (if ColumnGenerationType [135] <> *rvcgtRange*)	Field name	TRVCrossTabLevel [140]
	A cross-tab header field [36] refers to a non-existent data field	Field name	TRVReportGenerationSession [251]
rvrgeCrossTabWrongContext	A cross-tab header field [36] is found outside a cross-tab header cells	Field code	TRVReportGenerationSession [251]
rvrgeCrossTabRecordDoesNotMatch	A record in the result of a row generation rule's DataQuery [156]	Empty	TRVReportGenerationSession [251]

	contains a combination of values of column fields [145] not corresponding to any cross-tab data column		
rvrgeCrossTabDuplicateRecords	A record in the result of a row generation rule's DataQuery [156] contains the same combination of values of key fields [168] and column fields [145] as one of previous records.	Empty	TRVReportGenerationSession [251]
rvrgeCrossTabKeyFieldInCrossTabColumn	One ore more key fields [168] is included in column fields [145]	Field name	TRVRowGenerationRule [163]
rvrgeCrossTabNoCrossTabColumns	A cross tabulation contains 0 data columns	Empty	TRVReportTableItemInfo [124] or TRVCrossTabLevel [140], depending on ColumnGenerationType [135]
rvrgeCrossTabTooManyColumns	A group of columns in a cross-tab level exceeds level.MaxColCount [146]. In this case, only the first MaxColCount [146] columns are used.	Empty	TRVCrossTabLevel [140]
rvrgeCrossTabTooManyTotalColumns	A total number of column repetitions exceeds MaxColCount [139]	Empty	TRVReportTableItemInfo [124]

Errors while processing functions [37]

For this kind of errors, **RelatedObject** is TRVReportGenerationSession [251].

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

Errors while processing expressions 41

For this kind of errors, **RelatedObject** is TRVExpressionCalculator.

Error	Meaning	RelatedText
rvrExpressionParserError	An error occurs when parsing the expression	Expression text
rvrExpressionEvaluationError	An error occurs when evaluating the expression	Expression text

RelatedObject (TRVExpressionCalculator) allows receiving additional information about the reason of this error and the position in expression where it happened.

Other errors

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

Example 1

How to process error messages in OnError 70 event.

Before using RVReportGetErrorString 242, call RWA_LocalizeErrorMessages 242.

```
procedure TForm1.RVReportGenerator1Error(
  Sender: TCustomRVReportGenerator 61;
  ErrorCode: TRVReportGeneratorError;
  const RelatedText: TRVUnicodeString;
  RelatedObject: TObject; var Proceed: Boolean);
var
  Msg1, Msg2: TRVUnicodeString;
const
  MaxErrorCount = 1000;
begin
  if lstMessages.Items.Count = MaxErrorCount then
    lstMessages.Items.Add(RWA_GetS(rwm_log_TooManyErrors));
  if lstMessages.Items.Count > MaxErrorCount then
    exit;
  RVReportGetErrorString 242(ErrorCode, Msg1, Msg2, RelatedObject);
  if Msg1 <> '' then
    Msg2 := Msg1 + ': ' + Msg2;
  Msg2 := Format(String(Msg2), [RelatedText]);
  lstMessages.Items.Add(String(Msg2));
end;
```

Example 2

This example shows how to use custom error messages. This example is simplified, it does not provide detailed information about errors in expressions.

```
const ErrorMsg: array[TRVReportGeneratorError] of String =
 (
   'Rule error: invalid row or column range',
   'Rule error: overlapping rows',
   'Query error: can''t create query processor for %s',
   'Query error: execution error for %s',
   'Query error: this type of report requires a known record count before retri
   'Variable error: unknown variable %s',
   'Variable error: invalid value for %s',
   'Variable error: invalid name %s',
   'Variable error: unsupported object in %s',
   'Data field error: unknown field %s',
   'Data field error: invalid value for %s',
   'Data field error: unsupported field type for %s',
   'Data field error: invalid type in %s',
   'Data field error: unknown row generation rule in the field %s',
   'Field error: invalid type casting in %s',
   'Field error: invalid format string %s',
   'Field error: invalid data or format string for a custom field type',
   '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',
   'Cross-tab error: invalid cross-tab definition',
   'Cross-tab error: invalid cross-tab header location',
   'Cross-tab error: at least one row generation rule is required',
   'Cross-tab error: unknown field %s',
   'Cross-tab error: cross-tab field must be in a cross-tab header',
   'Cross-tab error: a record does not match the cross-tab columns (will be ski
   'Cross-tab error: two or more records match the same cross-tab cell (all but
   'Cross-tab error: the same field is used both as a key field as a cross-tab
   'Cross-tab error: no columns',
   'Cross-tab error: too many columns in a group',
   'Cross-tab error: too many total columns',
   'Function error: parenthesis is expected in %s',
   'Function error: unknown function %s',
   'Function error: bad parameters in %s',
   'Function error: function must be in a cross-tab summary rows or columns',
   'Function error: function cannot be calculated',
   'Expression parsing error in %s',
   'Expression evaluation error: %s',
   'Internal error'
```

```
    );

procedure TForm1.RVReportGenerator1Error(
  Sender: TCustomRVReportGenerator[61];
  ErrorCode: TRVReportGeneratorError;
  const RelatedText: TRVUnicodeString;
  RelatedObject: TObject; var Proceed: Boolean);
var S: TRVUnicodeString;
begin
  S := Format(ErrorMsg[ErrorCode], [RelatedText]);
  Memo1.Lines.Add(S);
end;
```

1.3.1.3.4 TCustomRVReportGenerator.OnGenerated

This event is called when report generation is finished.

```
type
  TRVReportGeneratedEvent = procedure (Sender: TObject;
    Res: Boolean) of object;

property OnGenerated: TRVReportGeneratedEvent;
```

If Execute[66] is called with the parameter UseThread = *False*, this event is called by the Execute method itself. In this case, processing this event is not necessary: you can do all processing after calling Execute.

If Execute[66] is called with the parameter UseThread = *True*, this method only starts report generation in a background thread, and exits immediately. This event is called when the thread finishes generating a report.

In all cases, this event is called in the context of the main process, so you can safely work with user interface in this event.

1.3.1.3.5 TCustomRVReportGenerator.OnGetField

Occurs before reading a data field value.

```
type
  TRVReportGeneratorGetFieldEvent = procedure (
    Sender: TCustomRVReportGenerator[61];
    const QueryProcessorName, FieldName: TRVUnicodeString;
    Session: TRVReportGenerationSession[251]) of object;

property OnGetField: TRVReportGeneratorGetFieldEvent;
```

This event may occur multiple times when a report is being generated. It is called when processing data fields[27] (for every data field in a report template, for every processed record).

If Execute[66] is called with parameter UseThread = *True*, this event is called in a thread context.

This is an informational event, it cannot be used to modify generated reports.

Parameters

QueryProcessorName – name of query processor[252], read from a data field[27];

FieldName – field name, read from a data field[27];

Session – an object representing a report generation session.

See also

- Definitions of Report Workshop terms[11]

1.3.1.3.6 TCustomRVReportGenerator.OnProcessRecord

Occurs before processing a record of a query processor.

```
type
  TRVProcessRecordEvent = procedure (Sender: TCustomRVReportGenerator[61];
    const DataQuery: TRVUnicodeString; Rule: TRVRowGenerationRule[163];
    Cell: TRVReportTableCellData[150];
    Session: TRVReportGenerationSession[251]; RecNo: Integer;
    QueryProcessor: TRVReportQueryProcessor[252]) of object;

property OnProcessRecord: TRVProcessRecordEvent;
```

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

If Execute[66] is called with parameter UseThread = *True*, this event is called in a thread context.

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[30] and data fields[27].

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[11]

1.3.2 TRVReportGenerator

A report generator component.

Unit RVReportGenerator;

Syntax

```
TRVReportGenerator = class (TCustomRVReportGenerator[61])
```

Hierarchy

TObject

TPersistent
TComponent
TCustomRVReportGenerator [61]

Description

Call Execute [66] 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 [62], or in OnCreateQueryProcessor [67] event.
While generating reports, the following events occur:

- OnDataQueryProgress [68] allows to show a progress indicator or to abort the generation;
- OnError [70] allows to process errors;
- OnProcessRecord [79] 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 [65].

See also

- Definitions of Report Workshop terms [11]

1.4 Data Provider Components

Report Workshop includes the following data provider components working with TRVReportGenerator [79].

Universal Data Providers

TRVReportBindSourceDataProvider [86] (Delphi XE3 or newer) [VCL, FMX]

Access to: data via LiveBindings

Supported types of data queries: names associated with TBaseObjectBindSource- or TBindSourceAdapter-based components

TRVReportDBDataProvider [94]

Access to: any data via TDataSet-based components

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

Data Providers Based on Standard Delphi Components

TRVReportADODataProvider [85] [VCL]

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 85 **[VCL]**

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

TRVReportDBXDataProvider 100 **[VCL, FMX]**

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

TRVReportFDDataProvider 102 **[VCL, FMX]**

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 103 **[VCL, FMX]**

Access to: ***MogoDB*** databases

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

Supported types of data queries: JSON

TRVReportIBDataProvider 105 **[VCL]**

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

Data Providers Based on Lazarus Components

TRVReportDbfDataProvider 97 **[LCL]**

Access to: DBF tables

Based on components: TDbf FCL components

Supported types of data queries: table name with optional filter

TRVReportSQLDataProvider 111 **[LCL]**

Access to: multiple databases, including including ***Firebird***, Microsoft SQL Server, Sybase ASE, MySQL.

Based on components: TSQLQuery FCL components

Supported types of data queries: SQL or table name

Data Providers Based on Third-Party Components

TRVReportAbsDataProvider 84 **[VCL]**

Access to: Absolute database

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

Supported types of data queries: SQL or table name

TRVReportDBISAMDataProvider 98 **[VCL]**

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

TRVReportDxMemDataProvider 101 **[VCL]**

Access to: any data stores available via TDataSet-based components; special support for TdxMemData by ***Developer Express Inc.***

Supported types of data queries:

- names of TDataSet-based components
- for TdxMemData, the query may contain a filter describing the required values of fields (useful to implement master/detail reports)

TRVReportEDBDataProvider 101 **[VCL]**

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

TRVReportIBCDataProvider 104 **[VCL]**

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

TRVReportIBODataProvider 107 **[VCL, FMX, LCL]**

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 107 **[VCL]**

Access to: ***MySQL*** databases

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

Supported types of data queries: SQL or table name

TRVReportMySQLDataProvider 107 **[VCL]**

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 108 **[VCL]**

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 109 **[VCL]**

Access to: ***PostgreSQL*** databases

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

Supported types of data queries: SQL or table name

TRVReportPSQLDataProvider 110 **[VCL]**

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 112 **[VCL]**

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 [113] **[VCL]**

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

1.4.1 TRVReportAbsDataProvider

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

Unit RVReportAbsDataProvider;

Syntax

```
TRVReportAbsDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider [92]
TRVReportCustomDBDataProvider [90]

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 [62] property of TRVReportGenerator [79].

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 [92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet [91] event. This DataSet is not necessary needed to be TABSQuery or TABSTable.

See also

- Definitions of Report Workshop terms [11]
- Data queries [13]

1.4.2 TRVReportADODataProvider

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

Unit RVReportAdoDataProvider;

Syntax

```
TRVReportADODataProvider = class (TRVReportCustomDBDataProvider 90)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 92
TRVReportCustomDBDataProvider 90

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 62 property of TRVReportGenerator 79.

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 92 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 91 event. This DataSet is not necessary needed to be TADOQuery or TADOTable.

See also

- Definitions of Report Workshop terms 11
- Data queries 13

1.4.3 TRVReportBDEDataProvider

A data provider receiving data from BDE databases.

Unit RVReportBDEDataProvider;

Syntax

```
TRVReportBDEDataProvider = class (TRVReportCustomDBDataProvider 90)
```

Hierarchy

TObject

TPersistent
TComponent
TRVReportDataProvider [92]
TRVReportCustomDBDataProvider [90]

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 [62] property of TRVReportGenerator [79].

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 [92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet [91] event. This DataSet is not necessary needed to be TQuery or TTable.

See also

- Definitions of Report Workshop terms [11]
- Data queries [13]

1.4.4 TRVReportBindSourceDataProvider

A data provider that gets data using LiveBindings sources.

Unit RVReportBindSourceDataProvider;

Syntax

```
TRVReportBindSourceDataProvider = class (TRVReportDataProvider [92])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider [92]

Description

Use this component to provide data using LiveBindings. To define sources, add items to BindSources[87] collection. Each item is a pair of (Name[90], BindSource[89]) or (Name[90], BindSourceAdapter[89]).

This data provider handles the following data queries:

- BindSource (or BindSourceAdapter) name (must be equal to one of BindSources[87][].Name[90])

A data provider receives data query strings from TRVReportGenerator[79] components, and creates TRVReportBindSourceAdapterQueryProcessor[246] objects to handle them. Normally, this process is invisible for you, TRVReportBindSourceAdapterQueryProcessor[246] is used internally. You only need to provide a BindSource/BindSourceAdapter object.

The design of this component is very similar to the design of TRVReportDBDataProvider[94]. While this component gets data from BindSources collection, TRVReportDBDataProvider[94] gets data from its DataSets collection.

See also

- Definitions of Report Workshop terms[11]

1.4.4.1 Properties

In TRVReportBindSourceDataProvider

🟩 BindSources[87]

1.4.4.1.1 TRVReportBindSourceDataProvider.BindSources

Allows using bind sources as sources of data for reports.

```
property BindSources: TRVBindSourceCollection[87];
```

This is a collection of pairs (Name[90], BindSource[89]) or (Name[90], BindSourceAdapter[89]). If data query[13] is equal to Name[90], the data provider creates a query processor using the corresponding BindSource[89] (or BindSourceAdapter[89]).

See also:

- Definitions of Report Workshop terms[11]

1.4.4.2 Classes of properties

Classes of TRVReportBindSourceDataProvider[86] Properties

- TRVBindSourceCollection[87] – collection of TRVBindSourceItem[88] items; a type of BindSources[87] property.

1.4.4.2.1 TRVBindSourceCollection

TRVBindSourceCollection is a collection of bind sources for use in report generator.

Unit RVReportBindSourceDataProvider;

Syntax

```
TRVBindSourceCollection = class (TCollection);
```

Hierarchy

TObject
TPersistent
TCollection

Description

TRVBindSourceCollection is a class of TRVReportBindSourceDataProvider[86].BindSources[87] property. It is a collection of TRVBindSourceItem[88] items.

Each item in this collection is a pair of (Name[90], BindSource[89]) or (Name[90], BindSourceAdapter[89]).

1.4.4.2.1.1 Properties

In TRVBindSourceCollection

Items[88]

Inherited from TCollection

▶ Count

Lists the items in the collection.

```
property Items[Index: Integer]: TRVBindSourceItem[88]; default;
```

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

1.4.4.2.2 TRVBindSourceItem

TRVBindSourceItem is an item in TRVBindSourceCollection[87] collection.

Unit RVReportBindSourceDataProvider;

Syntax

```
TRVBindSourceItem = class (TCollectionItem)
```

Hierarchy

TObject
TPersistent
TCollectionItem

Description

An object of this class allows using a bind source (BindSource[89] or BindSourceAdapter[89]) to provide data for the report generator[79]. It is referred by Name[90].

The number of records used in reports can be limited by MaxRecordCount[89] property.

1.4.4.2.2.1 Properties

In TRVBindSourceItem

- BindSource 89
- BindSourceAdapter 89
- MaxRecordCount 89
- Name 90

Specifies a bind source reference for TRVReportBindSourceDataProvider 86.

```
property BindSource: TBaseObjectBindSource;
```

The BindSource may be any TBaseObjectBindSource descendant, including:

- TAdapterBindSource component
- TPrototypeBindSource component.

It can be referred in report templates by Name 90.

There are two alternative ways to define a data source in TRVBindSourceItem: BindSourceAdapter 89 and **BindSource**. If the both of them are defined, BindSourceAdapter 89 is used, **BindSource** is ignored.

Note: only bind sources inherited from TBaseObjectBindSource are supported. For example, TBindSourceDB cannot be used in TRVReportBindSourceDataProvider 86 (use TRVReportDBDataProvider 94 or a data provider implemented for the specific set of DB components).

Specifies a bindsource adapter reference for TRVReportBindSourceDataProvider 86.

```
property BindSourceAdapter: TBindSourceAdapter;
```

The bindsource adapter may be any TBindSourceAdapter descendant, such as TDataGeneratorAdapter.

It can be referred in report templates by Name 90.

There are two alternative ways to define a data source in TRVBindSourceItem: **BindSourceAdapter** and BindSource 89. If the both of them are defined, **BindSourceAdapter** is used, BindSource 89 is ignored.

Specifies the maximum allowed count of data records.

```
property MaxRecordCount: Integer;
```

This property limits the count of records returned by BindSource 89 or BindSourceAdapter 89 and is used in a report generation.

Default value:

1000

Specifies a dataset name.

```
property Name: TRVUnicodeString;
```

If a report template contains a data query [13] equal to **Name**, data from BindSource [89] (or BindSourceAdapter [89]) are used.

Each item in TRVBindSourceCollection [87] must have an unique **Name**.

1.4.5 TRVReportCustomDBDataProvider

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

Unit RVReportDBDataProvider;

Syntax

```
TRVReportDBDataProvider = class (TRVReportDataProvider[92])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider [92]

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 [79] components, and creates TRVReportDBQueryProcessor [246] objects to handle them. Normally, this process is invisible for you, TRVReportDBQueryProcessor [246] is used internally.

See also

- Definitions of Report Workshop terms [11]

1.4.5.1 Properties

In TRVReportCustomDBDataProvider

- UseRecordCount [90]

1.4.5.1.1 TRVReportCustomDBDataProvider.UseRecordCount

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

```
property UseRecordCount: TRVReportRecordCountMode[235];
```

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 [94].DataSets [97], it has UseRecordCount [96] property in each item.

This property is used as an initial value of UseRecordCount parameter in OnCreateDataSet [91] event.

1.4.5.2 Events

In TRVReportDBDataProvider

- 🟨 AfterClose [92]
- 🟨 BeforeClose [92]
- 🟨 AfterOpen [92]
- 🟨 BeforeOpen [92]
- 🟨 AfterScroll [92]
- 🟨 BeforeScroll [92]
- 🟩 OnCreateDataSet [91]
- 🟨 OnDataSetCreated [92]

1.4.5.2.1 TRVReportCustomDBDataProvider.OnCreateDataSet

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

```
type
  TRVCreateDataSetEvent = procedure (
    Sender: TRVReportCustomDBDataProvider [90];
    const DataQuery: TRVUnicodeString; var DataSet: TDataSet;
      var DestroyAfterUse: Boolean;
      var AUseRecordCount: TRVReportRecordCountMode [235]) 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 [90].

See also:

- Definitions of Report Workshop terms [11]

1.4.5.2.2 TRVReportCustomDBDataProvider.OnDataSetCreated

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

```
type
  TRVDataSetCreatedEvent = procedure (Sender: TRVReportDBDataProvider[94];
    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[92]

1.4.5.2.3 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[94].DataSets[97]
- they are assigned to datasets returned in OnCreateDataSet[91] event, only if DestroyAfterUse = *True*
- they are assigned to datasets created by data providers automatically

1.4.6 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[94] and others.

Data providers are linked[62] to TRVReportGenerator[79] components. They create query processors for data queries.

Data provider receives data query strings from TRVReportGenerator[79] components, and creates TRVReportQueryProcessor[252] objects to handle them.

Inherited classes:

- TRVReportCustomDBDataProvider[90]

See also

- Definitions of Report Workshop terms[11]

1.4.6.1 Methods

In TRVReportDataProvider

CreateQueryProcessor[93]
GetFieldList[93]
GetTableList[94]

1.4.6.1.1 TRVReportDataProvider.CreateQueryProcessor

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

```
function CreateQueryProcessor(const DataQuery: TRVUnicodeString):
  TRVReportQueryProcessor[252]; virtual; abstract;
```

Normally, you do not need to call this method. Just assign this data provider component to TRVReportGenerator[79]'s DataProvider[62] property, and it will create query processors automatically.

You can use this method in TRVReportGenerator[79].OnCreateQueryProcessor[67] 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[11]

1.4.6.1.2 TRVReportDataProvider.GetFieldList

Fills **List** with names of fields for the given table.

```
procedure GetFieldList(const TableName: TRVUnicodeString;
  List: TStrings); virtual;
```

TRVReportDataProvider.GetFieldList simply clears **List**. Inherited classes may override this method to return a list of fields for the table **TableName**.

See also:

- GetTableList [94]

1.4.6.1.3 TRVReportDataProvider.GetTableList

Fills **List** with names of available database tables.

```
procedure GetTableList(List: TStrings); virtual;
```

TRVReportDataProvider.GetTableList simply clears **List**. Inherited classes may override this method to return a list of names of available tables.

See also:

- GetFieldList [93]

1.4.7 TRVReportDBDataProvider

An universal database-related data provider for report generators.

Unit RVReportDBDataProvider;

Syntax

```
TRVReportDBDataProvider = class (TRVReportCustomDBDataProvider)
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider [92]
TRVReportCustomDBDataProvider [90]

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

2. OnCreateDataSet [91] 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.

This data provider handles the following data queries:

- dataset name (must be equal to one of DataSets [97] [].Name [96])

A data provider receives data query strings from TRVReportGenerator [79] components, and creates TRVReportDBQueryProcessor [246] objects to handle them. Normally, this process is invisible for you, TRVReportDBQueryProcessor [246] is used internally. You only need to provide a dataset object.

The design of this component is very similar to the design of TRVReportBindSourceDataProvider[86]. While this component gets data from DataSets collection, TRVReportBindSourceDataProvider[86] gets data from its BindSources collection.

See also

- Definitions of Report Workshop terms[11]

1.4.7.1 Classes of properties

Classes of TRVReportDBDataProvider[94] Properties

- TRVDataSetCollection[95] – collection of TRVDataSetItem[96] items; a type of DataSets[97] property.

1.4.7.1.1 TRVDataSetCollection

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

Unit RVReportDBDataProvider;

Syntax

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

Hierarchy

TObject
TPersistent
TCollection

Description

TRVDataSetCollection is a class of TRVReportDBDataProvider[94].DataSets[97] property. It is a collection of TRVDataSetItem[96] items.

Each item in this collection is a pair of (Name[96], DataSet[96]).

1.4.7.1.1.1 Properties

In TRVDataSetCollection

Items[95]

Inherited from TCollection

▶ Count

Lists the items in the collection.

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

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

1.4.7.1.2 TRVDataSetItem

TRVDataSetItem is an item in TRVDataSetCollection[95] collection.

Unit RVReportDBDataProvider;

Syntax

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

Hierarchy

TObject
TPersistent
TCollectionItem

Description

An object of this class allows using an existing dataset DataSet[96] to provide data for the report generator[79]. It is referred by Name[96].

1.4.7.1.2.1 Properties

In TRVDataSetItem

- DataSet[96]
- Name[96]
- UseRecordCount[96]

Specifies a dataset reference for TRVReportDBDataProvider[94].

```
property DataSet: TDataSet;
```

The dataset may be any TDataSet descendant. It can be referred in report templates by Name[96].

Specifies a dataset name.

```
property Name: TRVUnicodeString;
```

If a report template contains a data query[13] equal to **Name**, data from DataSet[96] are used.

Each item in TRVDataSetCollection[95] must have an unique **Name**.

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

```
property UseRecordCount: TRVReportRecordCountMode[235];
```

This property specifies whether the report generator can use DataSet[96].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[91] event,

1.4.7.2 Properties

In TRVReportDBDataProvider

- 🟩 DataSets[97]

Inherited from TRVReportCustomDBDataProvider[90]

- 🟩 UseRecordCount[90]

1.4.7.2.1 TRVReportDBDataProvider.DataSets

Allows using existing datasets as sources of data for reports.

```
property DataSets: TRVDataSetCollection[95];
```

This is a collection of pairs (Name[96], DataSet[96]). If data query[13] is equal to Name[96], the data provider creates a query processor using the corresponding DataSet[96].

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

See also:

- Definitions of Report Workshop terms[11]

1.4.7.3 Events

Inherited from TRVReportCustomDBDataProvider[90]

- 🟩 OnCreateDataSet[91]
- 🟨 OnDataSetCreated[92]

1.4.8 TRVReportDbfDataProvider

A data provider receiving data from DBF tables, for Lazarus.

Unit RVReportDbfDataProviderLaz;

Syntax

```
TRVReportDbfDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

Description

This component provides data for a report generator component from DBF tables contained in the specified directory (FilePath property).

This data provider handles the following data queries:

- table name

- table name with filter

A table name must include extension, for example: *MyTable.dbf*

An optional filter expression may follow the table name after comma, for example: *Companies.dbf, COMPANYID=1*

For the syntax of filters, see ***TDbf Tutorial***

To use this component, assign it to DataProvider[62] property of TRVReportGenerator[79].

TRVReportDbfDataProvider introduces the properties:

```
property FilePath: String;
property FilterOptions: TFilterOptions;
```

Internally, the component uses temporal TDbf 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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TDbf.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.9 TRVReportDBISAMDataProvider

A data provider receiving data from DBISAM databases.

Unit RVReportDBISAMDataProvider;

Syntax

```
TRVReportDBISAMDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TDBISAMQuery or TDBISAMTable.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.10 TRVReportDBMemDataProvider

This is an ancestor class for data providers using simple in-memory datasets.

Unit RVReportMemDataProvider;

Syntax

```
TRVReportDBMemDataProvider = class (TRVReportDBDataProvider[94])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]
TRVReportDBDataProvider[94]

Description

This class is not used directly. The following components are inherited from this class:

- TRVReportDxMemDataProvider.

This data provider extends syntax of data queries implemented by TRVReportDBDataProvider[94].

This data provider handles the following data queries:

- dataset name (must be equal to one of DataSets[97][].Name[96])
- queries of the following syntax:

```
DataSetName, Field1 = Value1, Field2 = Value2, ...
```

where

- DataSetName is one of DataSets[97][].Name[96], identifies a dataset;
- Field1, Field2, ... are fields of this dataset;
- Value1, Value2, ... are required values of the corresponding fields.

The following types of values are supported:

- integer numbers;

- floating point numbers (using '.' as a decimal separator);
- true;
- false;
- strings (enclosed in single quotes; to include a single quote in the string, duplicate it).

The result of this query: all records of the specified dataset matched all the specified conditions.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.11 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[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TSQLQuery.

See also

- Definitions of Report Workshop terms[11]

- Data queries [13]

1.4.12 TRVReportDxMemDataProvider

An universal database-related data provider for report generators, that includes a special support for TdxMemData datasets.

Unit RVReportDXDataProvider;

Syntax

```
TRVReportDBMemDataProvider = class (TRVReportDBMemDataProvider[99])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider [92]
TRVReportCustomDBDataProvider [90]
TRVReportDBDataProvider [94]
TRVReportDBMemDataProvider [99]

Description

This component provides data for reports from datasets listed in the DataSets [97] property.

This data provider handles the following data queries:

- dataset name (must be equal to one of DataSets [97] [].Name [96])
- queries of the following syntax:

```
DataSetName, Field1 = Value1, Field2 = Value2, ...
```

This syntax is explained in the topic about TRVReportDBMemDataProvider [99]. It allows implementing master/detail reports.

See also

- Definitions of Report Workshop terms [11]
- Data queries [13]

1.4.13 TRVReportEDBDataProvider

A data provider receiving data from ElevateDB databases.

Unit RVReportEDBDataProvider;

Syntax

```
TRVReportEDBDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider [92]
TRVReportCustomDBDataProvider [90]

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TEDBQuery or TEDBTable.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.14 TRVReportFDDataProvider

A data provider receiving data from various databases via FireDAC.

Unit RVReportFDDataProvider;

Syntax

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

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]
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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TFDQuery or TFDTable.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.15 TRVReportFDMongoDataProvider

A data provider receiving data from MongoDB databases via FireDAC.

Unit RVReportFDMongoDataProvider;

Syntax

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

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]
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[66](
  '[{"$match":
     {"cuisine": "Italian", "address.zipcode": "10075"}},
    {"$limit":"10"}]')
// $match query syntax
RVReportGenerator1.EscapeDataQuery[66](
  '{"cuisine": "Italian", "address.zipcode": "10075"}');
// returning the whole collection:
RVReportGenerator1.EscapeDataQuery[66]('{}')
```

To use this component, assign it to DataProvider[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TFDMongoQuery.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.16 TRVReportIBCDataProvider

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

Unit RVReportIBCDataProvider;

Syntax

```
TRVReportIBCDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TIBCQuery or TIBCTable.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.17 TRVReportIBDataProvider

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

Unit RVReportIBDataProvider;

Syntax

```
TRVReportIBDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TIBQuery or TIBTable.

See also

- Definitions of Report Workshop terms [11]
- Data queries [13]

1.4.18 TRVReportIBODataProvider

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

Unit RVReportIBODataProvider;

Syntax

```
TRVReportIBODataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider [92]
TRVReportCustomDBDataProvider [90]

Description

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

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 [62] property of TRVReportGenerator [79].

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 [92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet [91] event. This DataSet is not necessary needed to be TIBOQuery or TIBOTable.

See also

- Definitions of Report Workshop terms [11]
- Data queries [13]

1.4.19 TRVReportMyDataProvider

A data provider receiving data from MySQL databases via MyDAC.

Unit RVReportMyDataProvider;

Syntax

```
TRVReportMyDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TMyQuery or TMyTable.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.20 TRVReportMySQLDataProvider

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

Unit RVReportMySQLDataProvider;

Syntax

```
TRVReportMySQLDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 92
TRVReportCustomDBDataProvider 90

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 62 property of TRVReportGenerator 79.

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 92 event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet 91 event. This DataSet is not necessary needed to be TMySQLQuery or TMySQLTable.

See also

- Definitions of Report Workshop terms 11
- Data queries 13

1.4.21 TRVReportNxDataProvider

A data provider receiving data from NexusDB databases.

Unit RVReportNxDataProvider;

Syntax

```
TRVReportNxDataProvider = class (TRVReportCustomDBDataProvider 90 )
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider 92
TRVReportCustomDBDataProvider 90

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TNxQuery and TNxTable.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.22 TRVReportPgDataProvider

A data provider receiving data from PostgreSQL databases via PgDAC.

Unit RVReportPgDataProvider;

Syntax

```
TRVReportPgDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TPgQuery or TPgTable.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.4.23 TRVReportPSQLDataProvider

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

Unit RVReportPSQLDataProvider;

Syntax

```
TRVReportPSQLDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

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[62] property of TRVReportGenerator[79].

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 [92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet [91] event. This DataSet is not necessary needed to be TPSQLQuery and TPSQLTable.

See also

- Definitions of Report Workshop terms [11]
- Data queries [13]

1.4.24 TRVReportSQLDataProvider

A data provider receiving data from various SQL databases, for Lazarus.

Unit RVReportSQLDataProviderLaz;

Syntax

```
TRVReportSQLDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider [92]
TRVReportCustomDBDataProvider [90]

Description

This component provides data for a report generator component from various SQL databases, including Firebird,Microsoft SQL Server, Sybase ASE, MySQL.

This data provider handles the following data queries:

- SQL select statement
- table name (without space characters)

To use this component, assign it to DataProvider [62] property of TRVReportGenerator [79].

TRVReportSQLDataProvider introduces the properties:

```
property Database: TDatabase;
property Transaction: TSQLTransaction;
```

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 [92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet [91] event. This DataSet is not necessary needed to be TSQLQuery.

See also

- Definitions of Report Workshop terms [11]

• Data queries [13]

1.4.25 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[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TUniQuery or TUniTable.

See also

• Definitions of Report Workshop terms [11]
• Data queries [13]

1.4.26 TRVReportZEOSDSDataProvider

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

Unit RVReportZEOSDSDataProvider;

Syntax

```
TRVReportZEOSDSDataProvider = class (TRVReportCustomDBDataProvider[90])
```

Hierarchy

TObject
TPersistent
TComponent
TRVReportDataProvider[92]
TRVReportCustomDBDataProvider[90]

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[62] property of TRVReportGenerator[79].

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[92] event.

In addition to the default query processing, you can provide another DataSet component in OnCreateDataSet[91] event. This DataSet is not necessary needed to be TZQuery or TZTable.

See also

- Definitions of Report Workshop terms[11]
- Data queries[13]

1.5 Additional Components

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

TRVShape[114]. This component draws a shape, such as a circle, a polygon, a star, a flag, an emoticon, etc.

1.5.1 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[174].

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[117].

It is colored using Color[115], StartColor[117], GradientType[116], Opacity[115] properties. It is outlined using LineColor[116], LineWidth[117], LineUsesFillColor[116] properties.

The component background is filled using BackgroundColor and BackgroundOpacity[115] properties.

Additional properties affecting the layout are: Margin, Padding[117], EqualSides[116].

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

See also

- TRVShapeItemInfo[169] – shape object for TRichView documents

1.5.1.1 Properties

In TRVShape

- 🟩 BackgroundColor[115]
- 🟩 BackgroundOpacity[115]
- 🟩 Color[115]
- 🟩 EqualSides[116]
- 🟩 GradientType[116]
- 🟩 LineColor[116]
- 🟩 LineUsesFillColor[116]
- 🟩 LineWidth[117]
- 🟩 Margin[117]
- 🟩 Opacity[115]
- 🟩 Padding[117]
- 🟩 ShapeProperties[117]

- StartColor 117

Inherited from TGraphicControl

- Align
- Anchors;
- DragCursor
- DragKind
- DragMode
- Enabled
- Constraints;
- ParentShowHint
- ShowHint
- Touch (Delphi 2010+)
- Visible

1.5.1.1.1 TRVShape.BackgroundColor, BackgroundOpacity

Fill color and opacity for the component.

```
property BackgroundColor: TRVColor;
property BackgroundOpacity: TRVOpacity;
```

These properties define the component background fill. For shape fill, see Color and Opacity 115 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+ and optionally in Lazarus).

Default value

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

1.5.1.1.2 TRVShape.Color, Opacity

Fill color and opacity for the shape.

```
property Color: TRVColor;
property Opacity: TRVOpacity;
```

If GradientType 116 = *rvgtNone*, the shape is filled with this color. Otherwise, this color is used as an ending gradient color (the starting color is StartColor 117).

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

VCL and LCL: Gradient and opacity are used only if shapes are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Default value

- Color: $C68E63 for VCL and LCL; $FF638EC6 for FMX
- Opacity: 100000 (i.e. 100%)

1.5.1.1.3 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

1.5.1.1.4 TRVShape.GradientType

Type of gradient fill for shapes.

```
property GradientType: TRVGradientType(233);
```

Gradient is drawn from StartColor(117) to Color(115).

If **GradientType** = *rvgtNone*, the shape is filled with Color(115).

VCL and LCL: Gradient is used only if shapes are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Default value

rvgtNone

1.5.1.1.5 TRVShape.LineColor

Line color

```
property LineColor: TRVColor
```

Line color can be auto-calculated basing on Color(115), see LineUsesFillColor(116).

Default value

rvclBlack

See also

- LineWidth(117)

1.5.1.1.6 TRVShape.LineUsesFillColor

Specifies how shapes are outlined.

```
property LineUsesFillColor: Boolean;
```

If **LineUsesFillColor** = *False*:

Shapes are with LineColor(116).

If **LineUsesFillColor** = *True*:

If Color(115) = *rvclNone*, LineColor(116) is used. Otherwise, if GradientType(116) <> *rvgtNone*, shapes are outlined with Color(115); for a flat fill, they are outlined with twice darker color.

Default value

False

1.5.1.1.7 TRVShape.LineWidth

Line width, in pixels

```
property LineWidth: Integer;
```

Assign 0 to hide lines.

Default value

1

See also

- LineColor[116]

1.5.1.1.8 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[115].

Default values

0

1.5.1.1.9 TRVShape.ShapeProperties

Specifies main properties of a shape.

```
property ShapeProperties: TRVReportShapeProperties[254];
```

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

1.5.1.1.10 TRVShape.StartColor

Start color for gradient fill.

```
property StartColor: TRVColor;
```

Gradient uses **StartColor** and Color[115].

VCL and LCL: Gradient is used only if shapes are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Default value

rvclWhite

1.6 Extensions

Field Types

Barcodes with Zint for Delphi

Barcodes with Zint for Delphi[118] allows to display text or integer values as barcodes.

Input: text or integer value

Output: image containing barcode

Field types: barcode, qrcode

1.6.1 Field Types

Barcodes with Zint for Delphi

Barcodes with Zint for Delphi[118] allows to display text or integer values as barcodes.

Input: text or integer value

Output: image containing barcode

Field types: barcode, qrcode

1.6.1.1 Barcodes with Zint for Delphi

This extension adds two new data field types[53]: **barcode** and **qrcode**.

Input: text or integer value.

Output: image containing barcode.

Field types **barcode** and **qrcode** are almost identical, except for the default barcode type[120]: EANX for **barcode**, QRCODE for **qrcode**.

How to use

1. Download ***Zint Barcode Generator for Delphi***, free open source barcode generator for Delphi and Lazarus.
2. *Delphi:* add path to Zint Barcode Generator's PAS-files to Delphi library. *Lazarus:* include Zint Barcode Generator's PAS-files to your project (all PAS-files except for zint_render_svg, zint_qr_epc, zint_render_wmf, zint_render_fmx_canvas, zint_render_fmx_bmp)
3. *VCL and Lazarus:* Include RVReportZintBarcode.pas in your project
   FireMonkey: Include fmxRVReportZintBarcode.pas in your project
4. *Lazarus:* add the path <TRichView Dir>\TRichView\Source\Include to "Compiler Options | Paths | Include files" in your project's options.
5. *VCL and Lazarus:* Add RVReportZintBarcode to "uses" of the main form unit. This unit is in <TRichView Dir>\ThirdParty\Barcode\Zint\Source\ folder.
   FireMonkey: Add fmxRVReportZintBarcode to "uses" of the main form unit. This unit is in <TRichView Dir>\ThirdParty\Barcode\Zint.FMX\Source\ folder.

Example syntax

{VALUE qrcode}

{VALUE qrcode "ecc=H color=darkred"}

{VALUE barcode "type=upca width=300 height=50 showtext=no"}

Format string

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

<property name>='<property value>' |<property name>=<property value>

<property name> is one of: color, backcolor, size, width, height, type, imageformat, ecc, fontname, fontsize, fontcolor, showtext, padding, borderwidth, vspace, hspace, bordercolor, align.

<property value> may be enclosed in single quotes. It is necessary for values containing space characters (such as font names).

Properties are processed from left to right, so earlier values can be overridden.

The most important property is "type". It defines the type of generated barcode.

Common properties

Property	Meaning	Default value
color	color of barcode symbol	black
backcolor	background color	white
size	width and height of the resulting image, pixels	200
width	width of the resulting image, pixels	200
height	height of the resulting image, pixels	200
imageformat	format of the resulting image (wmf, bmp, png, etc.)	wmf for VCL png for Lazarus and FireMonkey
fontname	font name for text	RVDefaultLoadProperties .DefaultFontName
fontsize	font size for text, points	8
textcolor	color of text	black
showtext	turn text displaying on/off (text is shown for values: y, yes, t, true, 1; text is hidden for values: n, no, f, false, 0)	true
type	barcode type, see Types of Barcodes 120.	EANX for barcode type QRCODE for qrcode type

Note: if size of only one side (width or height) is defined, the same size is assigned to another side.

Additionally, the following properties of format strings for images [56] are supported:

- padding
- borderwidth
- vspace
- hspace
- bordercolor
- align

Properties for specific types of barcodes

Property	Meaning	Default value
ecc	Error correction level for QR codes Possible values: • 0 or L • 1 or M • 2 or Q • 3 or H	auto

1.6.1.1.1 Types of Barcodes

See **https://zint.org.uk/manual/chapter/6/1** for details.

In the tables below, the first column contains values that can be used for **type** property in format string for barcode and qrcode field type [118].

One-Dimensional Barcodes

Value	Meaning
CODE11	Code 11
C25STANDARD	Standard Code 2 of 5
C25IATA	IATA Code 2 of 5
C25IND	Industrial Code 2 of 5
C25INTER	Interleaved Code 2 of 5 (ISO 16390)
C25LOGIC	Code 2 of 5 Data Logic
ITF14 CASE	TF-14, also known as UPC Shipping Container Symbol or Case Code
DPLEIT	Deutsche Post Leitcode
DPIDENT	Deutsche Post Identcode
UPCA	UPC Version A

UPCE	UPC Version E
EANX EAN8 JAN8 EAN13 JAN13	EAN-2, EAN-5, EAN-8 and EAN-13
ISBNX	SBN, ISBN and ISBN-13
PLESSEY	UK Plessey
MSI_PLESSEY	MSI Plessey
TELEPEN	Telepen Alpha
TELEPEN_NUM	Telepen Numeric
CODE39	Standard Code 39 (ISO 16388)
EXCODE39	Extended Code 39
CODE93	Code 93
PZN	PZN (Pharmazentralnummer)
LOGMARS	LOGMARS
CODE32	Code 32
HIBC_39	HIBC Code 39
CODABAR NW7	Codabar (EN 798), also known as NW-7, Monarch, ABC Codabar, USD-4, Ames Code and Code 27
PHARMA	Pharmacode
CODE128	Standard Code 128 (ISO 15417)
CODE128B	Code 128 Subset B
GS1_128	GS1-128, previously known as UCC/EAN-128
EAN14	EAN-14
NVE18	NVE-18 (SSCC-18), also known as SSCC-18 (Serial Shipping Container Code)
HIBC_128	HIBC Code 128

DBAR_OMN	GS1 DataBar Omnidirectional and GS1 DataBar Truncated, previously known as RSS-14
DBAR_LTD	GS1 DataBar Limited, previously known as RSS Limited
DBAR_EXP	GS1 DataBar Expanded, previously known as RSS Expanded
KOREAPOST	Korea Post Barcode
CHANNEL	Channel Code

Stacked Barcodes

Value	Meaning
CODE16K	Code 16K (EN 12323)
PDF417	PDF417 (ISO 15438)
PDF417COMP	Compact PDF417 (ISO 15438), previously known as Truncated PDF417
MICROPDF417	MicroPDF417 (ISO 24728)
CODE49	Code 49

Two-Track Symbols

Value	Meaning
PHARMA_TWO	Two-Track Pharmacode
POSTNET	POSTNET
PLANET	PLANET

4-State Postal Codes

Value	Meaning
AUSPOST	Australia Post 4-State Symbols: Customer Barcodes
AUSREPLY	Australia Post 4-State Symbols: Reply Paid Barcode
AUSROUTE	Australia Post 4-State Symbols: Routing Barcode

AUSREDIRECT	Australia Post 4-State Symbols: Redirect Barcode
KIX	Dutch Post KIX Code
RM4SCC	UK Royal Mail 4-State Customer Code (RM4SCC)
JAPANPOST JPPOST	Japanese Postal Code
DAFT	DAFT Code

Matrix Symbols

Value	Meaning
HIBC_DM	Data Matrix (ISO 16022)
QRCODE QR	QR Code (ISO 18004)
MICROQR	Micro QR Code (ISO 18004)
AZTEC	Aztec Code (ISO 24778)
AZRUNE	Aztec Runes (ISO 24778)
CODEONE	Code One
GRIDMATRIX	Grid Matrix
DOTCODE	DotCode

Other Barcode-Like Markings

Value	Meaning
FIM	Facing Identification Mark (FIM)

1.7 Document Items

Document Items in ReportWorkshop

Report Workshop includes several special objects that can be inserted in TRichView documents.

Reporting items:

- TRVReportTableItemInfo[124] – a report table; a table that may have associated data queries[13] (in row generation rules and cells)

Other items:

- TRVShapeItemInfo[169] displays a shape.

1.7.1 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[126] – a collection of a table row generation rules, allowing to apply data queries to certain ranges of table rows.
- Variables[127] – a list of local variables[30] for this table
- BackgroundColorChangers[125] – a collections of color changers allowing to change colors and opacities of cells according to values associated with these cells.
- BackgroundVisualizers[125] – a collections of visualizers allowing to display values associated with cells on these cells' backgrounds.

A class of cells for report tables is TRVReportTableCellData[150], 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[126][] property:

```
Table.ReportCells[126][Row, Col].DataQuery[152] := 'SELECT * FROM MyTable';
```

See also

- Definitions of Report Workshop terms[11]

1.7.1.1 Properties

In TRVReportTableItemInfo

🟩 BackgroundColorChangers[125]
BackgroundVisualizers[125]
CrossTabSelected[127]

- 🟩 CrossTabulation [126]
- 🟩 Processed [126]
- 🟩 ReportCells [126]
- 🟩 RowGenerationRules [126]
- SelectedRule [127]
- 🟩 Variables [127]

Inherited from TRVTableItemInfo

(many properties, see the TRichView manual)

1.7.1.1.1 TRVReportTableItemInfo.BackgroundColorChangers

A collection containing background color changers for table cells.

```
property BackgroundColorChangers: TRVReportColorChangers[148];
```

If a report cell [150] is linked [151] to a color changer, this color changer changes the cell color and opacity according to a value [152] 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 [128] to assign a new value to this property as an editing operation.

See also

- Definitions of Report Workshop terms [11]

1.7.1.1.2 TRVReportTableItemInfo.BackgroundVisualizers

A collection containing value visualizers [174] for table cells.

```
property BackgroundVisualizers: TRVReportValueVisualizers[154];
```

If a report cell [150] is linked [153] to a visualizer, this visualizer displays a value [153] 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 [191].

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 [124] (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 [128] to assign a new value to this property as an editing operation.

See also

- Definitions of Report Workshop terms [11]

1.7.1.1.3 TRVReportTableItemInfo.CrossTabulation

A property defining parameters for cross-tab reports.

```
property CrossTabulation: TRVCrossTab[130];
```

A direct assignment to this property cannot be undone by the user. Use SetCrossTabulation[129] to assign a new value to this property as an editing operation.

1.7.1.1.4 TRVReportTableItemInfo.Processed

Specifies if this table was processed by a report generator[79].

```
property Processed: Boolean;
```

Initially, **Processed** = *False*. When a report generator's Execute[66] 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[247].HighlightRules[249]

1.7.1.1.5 TRVReportTableItemInfo.ReportCells

Lists cells in the table.

```
property ReportCells[Row, Col: Integer]: TRVReportTableCellData[150];
```

This property returns the same values as Cells[Row, Col], but the result is typecasted TRVReportTableCellData[150] (i.e. to the actual class of cells).

Default value

False

1.7.1.1.6 TRVReportTableItemInfo.RowGenerationRules

A collection of row generation rules for this report table.

```
property RowGenerationRules: TRVRowGenerationRules[168];
```

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[129] to assign a new value to this property as an editing operation.

See also

- Definitions of Report Workshop terms[11]
- item in this collection: TRVRowGenerationRule[163]

1.7.1.1.7 TRVReportTableItemInfo.SelectedRule

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

```
property SelectedRule: TRVRowGenerationCustomRule 155;
property CrossTabSelected: Boolean;
```

Assign one of RowGenerationRules 126 or their SubRules 158 to **SelectedRule** to draw a border around it.

Assign *True* to **CrossTabSelected** to draw a border around the cross tabulation 126 header.

These properties are used only when highlighting 249 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 165;
- when drawing a border for a nested rule, it includes the leftmost and rightmost cells 161, while background filling does not.

See also

- TRVRowGenerationCustomRule 155.HighlightColor 157
- TRVCrossTab 130.HighlightColor 138

1.7.1.1.8 TRVReportTableItemInfo.Variables

A string list containing local variables 30 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 129 to assign a new value to this property as an editing operation.

See also:

- variables in templates 30
- TCustomRVReportGenerator 61.Variables 65
- TRVReportGenerationSession 251

1.7.1.2 Methods

In TRVReportTableItemInfo

SetBackgroundColorChangers 128
SetBackgroundVisualizers 128
SetCellColorChanger 128
SetCellDataQuery 128
SetCellHighlightColor 129
SetCellName 129
SetCellVisualizer 129

SetCrossTabulation 129
SetRowGenerationRules 129
SetVariables 129

Inherited from TRVTableItemInfo

(many methods, see the TRichView manual)

1.7.1.2.1 TRVReportTableItemInfo.SetBackgroundColorChangers

Assigns **ColorChangers** to BackgroundColorChangers 125 as an editing operation.

```
procedure SetBackgroundColorChangers(
  ColorChangers: TRVReportColorChangers 148);
```

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 194 properties of items.

1.7.1.2.2 TRVReportTableItemInfo.SetBackgroundVisualizers

Assigns **Visualizers** to BackgroundVisualizers 125 as an editing operation.

```
procedure SetBackgroundVisualizers(
  Visualizers: TRVReportValueVisualizers 154);
```

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 194 properties of items.

1.7.1.2.3 TRVReportTableItemInfo.SetCellColorChanger

Assigns **ColorChangerId** to ReportCells 126 [**Row**, **Col**].ColorChangerId 151 as an editing operation.

```
procedure SetCellColorChanger(
  ColorChangerId: TRVValueVisualizerId 240;
  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.

1.7.1.2.4 TRVReportTableItemInfo.SetCellDataQuery

Assigns **Value** to ReportCells 126 [**Row**, **Col**].DataQuery 152 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.

1.7.1.2.5 TRVReportTableItemInfo.SetCellHighlightColor

Assigns **Value** to ReportCells[126][**Row**, **Col**].HighlightColor[152] as an editing operation.

```
procedure SetCellHighlightColor(
  Value: TRVColor; 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.

1.7.1.2.6 TRVReportTableItemInfo.SetCellName

Assigns **Value** to ReportCells[126][**Row**, **Col**].Name[153] as an editing operation.

```
procedure SetCellName(
  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.

1.7.1.2.7 TRVReportTableItemInfo.SetCellVisualizer

Assigns **VisualizerId** to ReportCells[126][**Row**, **Col**].VisualizerId[153] as an editing operation.

```
procedure SetCellVisualizer(
  VisualizerId: TRVValueVisualizerId[240];
  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.

1.7.1.2.8 TRVReportTableItemInfo.SetCrossTabulation

Assigns **CrossTab** to CrossTabulation[126] as an editing operation.

```
procedure SetCrossTabulation(CrossTab: TRVCrossTab[130]);
```

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.

1.7.1.2.9 TRVReportTableItemInfo.SetRowGenerationRules

Assigns **Rules** to RowGenerationRules[126] as an editing operation.

```
procedure SetRowGenerationRules(Rules: TRVRowGenerationRules[168]);
```

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.

1.7.1.2.10 TRVReportTableItemInfo.SetVariables

Assigns **AVariables** to Variables[127] 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[127].Objects[] are owned by Variables[127], this method assigns Variables[127].Objects[] and clears all **AVariables**.Objects[].

1.7.1.3 Classes of properties

Classes of TRVReportTableItemInfo[124] Properties

- TRVCrossTab[130] – cross-tab properties;
- TRVReportColorChangers[148] – a collection of TRVReportColorChangerItem[184] items; a type of BackgroundColorChangers[125] property; allows changing colors of cells depending on values;
- TRVReportValueVisualizers[154] – a collection of value visualizer[174] items; a type of BackgroundVisualizers[125] property; allows drawing diagrams on cells backgrounds;
- TRVRowGenerationRules[168] – a collection of TRVRowGenerationRule[163] items; A type of RowGenerationRules[126] property; associates a data query with table rows.
- TRVReportTableCellData[150] – a table cells, accessible as ReportCells[,][126] property.

Classes of Properties of Properties

Classes of TRVCrossTab[130] properties:

- TRVCrossTabLevels[148] – a collection of TRVCrossTabLevel[140]; a type of Levels[139] property; specifies cross-tab levels;

Classes of TRVRowGenerationRule[163] (and TRVRowGenerationNestedRule[160]) properties:

- TRVRowGenerationNestedRules[162] – a collection of TRVRowGenerationNestedRule[160] items; A type of SubRules[158] property; associates a data query with several cells of the parent rule.

1.7.1.3.1 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[124].CrossTabulation[126] property.

It defines the position of the cross-tab header in the table (Column and FirstRow[132] properties) and its levels (Levels[139] property).

If Levels[139] 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[139][].FieldName[145]. 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[126]. *Row fields* are listed in Table.RowGenerationRules[126][].KeyFieldNames[168], *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[132] 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[132] 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[37] 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[156], the report generator compares values of KeyFieldNames[168] 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[168] fields.

Value fields are written in the intersection of this row and a cross-tab column** corresponding to the set of values of Items[139][].FieldName[145], 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[139][].FieldName[145] fields, not corresponding to any *column fields* (it is not possible if ColumnGenerationType[135] = *rvcgtAutoSeparate* and *rvcgtAutoGroup*, but possible with other column generation types); OnError[70] event occurs with *rvrgeCrossTabRecordDoesNotMatch* parameter, and this record is ignored;
- a record contains values of Levels[139][].FieldName[145] and KeyFieldNames[168] fields, equal to values of another record (i.e., two or more records corresponds to the same cell*** of a cross-tab report); OnError[70] 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[156] 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[156] 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[132] properties

*** strictly speaking, it may be a group of cells having the number of rows and columns described in "*" and "**" notes

1.7.1.3.1.1 Properties

In TRVCrossTab

🟩 Column[132]
🟩 FirstRow[132]
Levels[139]
🟩 MaxColCount[139]
🟩 TextForEmptyCells[139]

Inherited from TCollection

Count

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** + Levels[139].Count - 1
- columns from **Column** to Table.Cells[**FirstRow**, **Column**].**ColSpan** - 1

Structure of a cross-tab header

The cross-tab header has Levels[139].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[139].

Template

	{#Person}
{Year}	${Salary}

Result Sample

	Alice	Bob	Carol
2014	$1000	$2000	$3000
2015	$4000	$5000	$6000

Example 2

One item in Levels[139], two data columns

Template

	{#Person}	
	Salary	Days
{Year}	${Salary}	{WorkDays}

Result Sample

	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[139]

Template

	{#Question}
	{#Option}

Result Sample

	Do you like porridge?		Have you ever seen a mouse?		
	Yes	No	Yes	Never	Not sure

{Age}	{Percent}%

Below 20	10%	90%	10%	80%	0%
20 and older	50%	50%	10%	0%	80%

Example 3

Two items in Levels [139], a summary column. Additionally, the rightmost table column is used as a grand summary.

Template

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

Result Sample

	Fruits			
	Apples	Oranges	Total for Fruits	Tomato
2014	100	200	300	300
2015	150	250	400	350

Example 3

Three items in Levels [139], two summary columns in each level.

Template

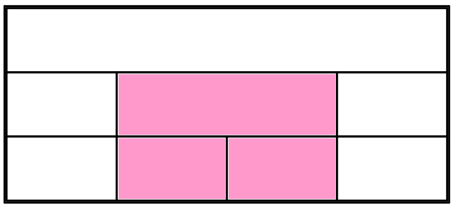
	{#Category}					Total	Average
	{#Product}			Total for	Average f		
	{#Year	Total f	Averag				
{Region}	{Sales	{sum(S	{avera	{sum(Sale	{average(	**{sum(Sales)}**	**{average(Sales)}**

Position of a cross-tab header in a table

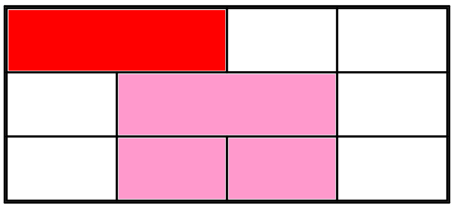
The whole cross-tab header must be inside a rectangular area, i.e. no cells may span across its sides.

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.

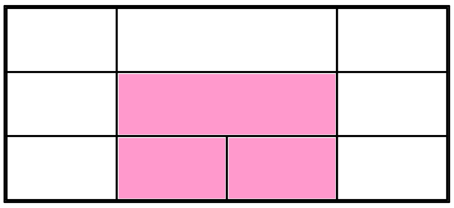
Good:

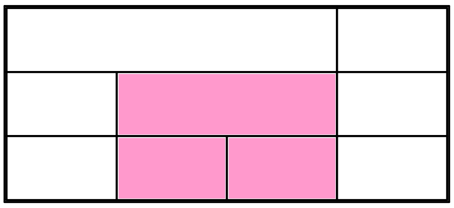


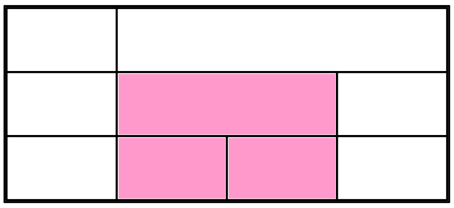
Bad:

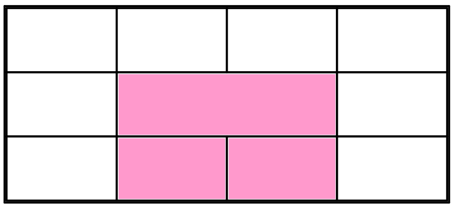


Good:

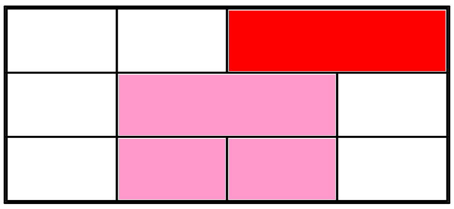


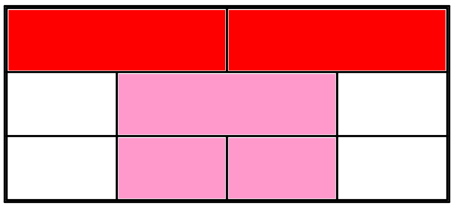






Bad:





The cells below the cross-tab header may not span across cross-tab columns.

Default values:

0

See also

- Definitions of Report Workshop terms[11]

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

	rvcgtAutoSeparate	*rvcgtAutoGroup*	*rvcgtDataQuerySeparate*	*rvcgtDataQueryCascade*	*rvcgtRange*
Columns are generated basing on...	report data (results of Table.RowGenerationRules[126][0].DataQuery[156])		special data query (results on Levels[139][].DataQuery[143])		range of values
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)	–	Yes	–	Yes	–
Can use special data field for captions[141]	Yes				–
Can use integer, boolean, floating point, date-time data fields for values	Yes				
Can use text data fields for values	Yes				–
Sort order	defined in Levels[139][].SortType[147]		as returned by a data query		ascending or descending, depending on Levels[139][].Step[146]

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[126][0].DataQuery[156]. 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[139]):

- values for column generation are taken from FieldName[145] field;
- additionally, captions may be taken from CaptionFieldNameField[141];
- columns are sorted according to SortType[147].

rvcgtDataQuerySeparate and rvcgtDataQueryCascade

For *rvcgtDataQuerySeparate* and *rvcgtDataQueryCascade*, columns are generated from data queries specified in cross-tab levels, i.e in Levels[139][].DataQuery[143].

ColumnGenerationType	Comments
rvcgtDataQuerySeparate	All Table.Levels[139][].DataQuery[143] 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[139][*N*].DataQuery[143] may refer[36] to data queries of higher levels, i.e. the results of execution of Levels[139][*M*].DataQuery[143]-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[139][].DataQuery[143] for an example.

For each cross-tab level (i.e., for each item in Levels[139]):

- values for column generation are taken either from DataQueryFieldName[145] or from FieldName[145] field;
- additionally, captions may be taken from CaptionFieldNameField[141];

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[139]), this enumeration is defined by MinValue, MaxValue, and Step[146] properties.

Default value

rvcgtAutoGroup

Specifies the color to highlight the cross-tab header in the report template.

```
property HighlightColor: TRVColor;
```

The cross-tab header is highlighted if:

- the first TRVReportDocObject[247] item in TRichView.DocObjects has HighlightRules[249] = *True*;
- the owner table's Processed[126] = *False*
- FirstRow[132] and Column[132] define the position of the cross-tab header inside the table
- the owner table's Cells[FirstRow[132], Column[132]] is not *nil*.

The header is shaded with this color.

Default value:

$FF0099 for VCL and LCL; $FF9900FF for FMX

See also

- Definitions of Report Workshop terms [11]
- TRVReportTableCellData [152].HighlightColor [152]
- TRVRowGenerationCustomRule [155].HighlightColor [157]
- TRVReportTableItemInfo [124].CrossTabSelected [127]

Collection of levels of cross-tab columns.

```
property Levels: TRVCrossTabLevels; default;
```

[148]

This is a collection of TRVCrossTabLevel [140] items.

A cross tabulation is defined, if this collection has at least one item.

Levels[0] corresponds to ReportTable.Cells[FirstRow, Column [132]].
Levels[1] corresponds to ReportTable.Cells[FirstRow + 1, Column].
Levels[2] corresponds to ReportTable.Cells[FirstRow + 2, Column].

and so on.

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 [70], 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 [140].MaxColCount [146]

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 [143] = True.

Default value:

'' (empty string)

See also

- Definitions of Report Workshop terms [11]

1.7.1.3.2 TRVCrossTabLevel

TRVCrossTabLevel is an item in TRVCrossTabLevels[148] collection.

Unit RVReportTable;

Syntax

```
TRVCrossTabLevel = class (TCollectionItem);
```

Hierarchy

TObject
TPersistent
TCollectionItem

Description

TRVCrossTabLevel is a class of items in the collection TRVCrossTabLevels[148], available as TRVCrossTab[130].Levels[139] 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[124].CrossTabulation[126].ColumnGenerationType[135].

Different properties of **TRVCrossTabLevel** are used for different values of ColumnGenerationType.

ColumnGenerationType	Properties of TRVCrossTabLevel
rvcgtAutoSeparate, *rvcgtAutoGroup*	• FieldName[145] • CaptionFieldName[141] • SortType[147] • CaseSensitive[142]
rvcgtDataQuerySeparate, *rvcgtDataQueryCascade*	• DataQuery[143] • FieldName[145] • DataQueryFieldName[145] • CaptionFieldName[141] • CaseSensitive[142]
rvcgtRange	• MinValue, MaxValue, Step[146]

ClearEmptySummaryCols[143] affects summary columns started on the next cross-tab level.

MaxColCount[146] allows restricting the count of columns at this level.

1.7.1.3.2.1 Properties

In TRVCrossTabLevel

- CaptionFieldName[141]
- CaseSensitive[142]
- ClearEmptySummaryCols[143]
- DataQuery[143]
- DataQueryFieldName[145]

- FieldName[145]
- MaxColCount[146]
- MaxValue[146]
- MinValue[146]
- SortType[147]
- Step[146]

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[36] syntax).

This property is used if TRVReportTableItemInfo[124].CrossTabulation[126].ColumnGenerationType[135] 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[147] = *rvrstCaptionsAscending* or *rvrstCaptionsDescending*, and TRVReportTableItemInfo[124] .CrossTabulation[126].ColumnGenerationType[135] = *rvcgtAutoSeparate* or *rvcgtAutoGroup*, captions are used when sorting columns at this level of cross-tab (see also CaseSensitive[142] 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[126].ColumnGenerationType[135] :=
  rvcgtDataQuerySeparate;
Table.CrossTabulation.Column[132] := 1;
Table.CrossTabulation.FirstRow[132] := 0;
with Table.CrossTabulation.Levels[139].Add do
begin
  DataQuery :=
    'SELECT ItemId, ItemName FROM ItemsTable ORDER BY ItemName';
  FieldName[145] := 'ItemId';
  CaptionFieldName := 'ItemName';
end;
with Table.RowGenerationRules[126].Add do
begin
  FirstRow[156] := 1;
  RowCount[156] := 1;
  DataQuery[156] := 'SELECT * FROM OrdersTable';
  KeyFieldNames[168] := '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[11]

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[124].CrossTabulation[126].ColumnGenerationType[135] = *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[143], values are taken from DataQueryFieldName[145] or FieldName[145] field)
- if TRVReportTableItemInfo[124].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[124].RowGenerationRules[126][0].DataQuery[143], values are taken from FieldName[145] field)

2. if SortType[147] = *rvrstCaptionsAscending* or *rvrstCaptionsDescending*, and TRVReportTableItemInfo[124].CrossTabulation.ColumnGenerationType = *rvcgtAutoSeparate* or *rvcgtAutoGroup,* this property is used when sorting columns (captions are taken from CaptionFieldName[141] field)

Default value

False

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[124].CrossTabulation[126].Levels[139][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[139].

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

A data query for cross-tab columns generation.

```
property DataQuery: TRVUnicodeString;
```

This property is used only if TRVReportTableItemInfo[124].CrossTabulation[126].ColumnGenerationType[135] = *rvcgtDataQuerySeparate* or *rvcgtDataQueryCascade.*

If there is only one cross-tab level (i.e. TRVReportTableItemInfo[124].CrossTabulation.Levels[139].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[139][].**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[139][].**DataQuery** may refer[36] 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[145] field of the result of application of **DataQuery**. If DataQueryFieldName[145] is empty, FieldName[145] field is used.

This field must be of one of the following types[234]:

- *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.ColumnGenerationType := rvcgtDataQueryCascade;
Table.CrossTabulation.Column := 1;
Table.CrossTabulation.FirstRow := 0;
with Table.CrossTabulation.Levels.Add do
begin
  DataQuery := 'SELECT QuestionText, QuestionId FROM QuestionsTable';
  FieldName := 'QuestionId';
  CaptionFieldName := 'QuestionText';
end;
with Table.CrossTabulation.Levels.Add do
begin
  DataQuery := 'SELECT OptionText, OptionId FROM QuestionsOptionsTable WHERE Ques
  FieldName := 'QuestionId';
  CaptionFieldName := '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[11]
- Information about data queries[13]
- TRVReportDocObject[247].DataQuery[249]
- TRVRowGenerationRule[163].DataQuery[156]
- TRVReportTableCellData[150].DataQuery[152]

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[124].CrossTabulation[126].ColumnGenerationType[135] = *rvcgtDataQuerySeparate* or *rvcgtDataQueryCascade.*

This field must be of one of the following types[234]:

- *rvrftText,*
- *rvrftInteger,*
- *rvrftFloat,*
- *rvrftBoolean,*
- *rvrftDateTime, rvrftDate, rvrftTime*

If this property is empty, FieldName[145] is used instead.

Default value:

'' (empty string)

See also

- Definitions of Report Workshop terms[11]

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[156]. 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[36] syntax); see also CaptionFieldName[141];
- when generating columns of this level of the cross-tab table in the following cases:
  - if TRVReportTableItemInfo[124].CrossTabulation[126].ColumnGenerationType[135] = *rvcgtDataQuerySeparate* or *rvcgtDataQueryCascade*, and DataQueryFieldName[145] is empty;
  - if TRVReportTableItemInfo.CrossTabulation.ColumnGenerationType = *rvcgtAutoSeparate* or *rvcgtAutoGroup*.

This field must be of one of the following types[234]:

- *rvrftText,*
- *rvrftInteger,*
- *rvrftFloat,*
- *rvrftBoolean,*
- *rvrftDateTime, rvrftDate, rvrftTime*

Default value:

'' (empty string)

See also

- Definitions of Report Workshop terms[11]

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[70], 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[130].MaxColCount[139]

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[124].CrossTabulation[126].ColumnGenerationType[135] = *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 [11]

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 [124].CrossTabulation [126].ColumnGenerationType [135] is one of:

- *rvcgtDataQuerySeparate*
- *rvcgtDataQueryCascade.*

In these column generation modes, values (and, optionally, captions) are taken from the results of TRVReportTableItemInfo [124].RowGenerationRules [126][0].DataQuery [143].

Values are in FieldName [145] field, captions are in CaptionFieldName [141] 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 [142] property.

Default value:

rvrstAscending

See also

- Definitions of Report Workshop terms [11]

1.7.1.3.3 TRVCrossTabLevels

TRVCrossTabLevels is a collection of cross-tab levels for a report table [124].

Unit RVReportTable;

Syntax

```
TRVCrossTabLevels = class (TOwnedCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection

Description

TRVCrossTabLevels is a class of TRVCrossTab [130].Levels [139] property.

In other words, it is a class of TRVReportTableItemInfo [124].CrossTabulation [126].Levels [139] property.

It is a collection of TRVCrossTabLevel [140] items.

1.7.1.3.3.1 Properties

In TRVCrossTabLevels

Items [148]

Inherited from TCollection

▶ Count

Lists the items in the collection.

```
property Items[Index: Integer]: TRVCrossTabLevel ; default;
```

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

1.7.1.3.4 TRVReportColorChangers

TRVReportColorChangers is a collection of color changers for a report table [124].

Unit RVReportColorChanger;

Syntax

```
TRVReportColorChangers = class (TCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection
TRVReportCustomValueVisualizerCollection 149

Description

TRVReportColorChangers is a class of TRVReportTableItemInfo 124.BackgroundColorChangers 125 property. It is a collection of TRVReportColorChangerItem 184 items.

See also

- TRVReportTableCellData 150.ColorChangerId 151

1.7.1.3.4.1 Properties

In TRVReportColorChangers

Items 149

Inherited from TCollection

▶ Count

Lists the items in the collection.

```
property Items[Index: Integer]: TRVReportColorChangerItem 184; default;
```

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

1.7.1.3.5 TRVReportCustomValueVisualizerCollection

TRVReportCustomValueVisualizerCollection is a parent class for TRVReportValueVisualizers 154 and TRVReportColorChangers 148.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportValueVisualizers = class (TCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection

Description

This class is not used directly.

Use TRVReportValueVisualizers 154 and TRVReportColorChangers 148.

1.7.1.3.5.1 Properties

In TRVReportCustomValueVisualizerCollection

MaxValue[150]

Inherited from TOwnedCollection

...

A list of forbidden values for Id[194] 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[194] to this list. It prevents reusing this value for new items.

1.7.1.3.5.2 Methods

In TRVReportCustomValueVisualizerCollection

MaxValue[150]

Inherited from TOwnedCollection

...

Assigns Source to the collection.

```
procedure AssignWithId(Source: TRVReportCustomValueVisualizerCollection[149]);
```

Unlike Assign method, AssignWithId copies all properties of items, including Id[194]

Searches for the item with the specified value of Id[194] property.

```
function FindById(Id: TRVValueVisualizerId[240]): Integer;
function FindItemById(Id: TRVValueVisualizerId[240]):
  TRVReportCustomValueVisualizerBase[193];
```

FindById returns the index of the item, or -1 if not found.

FindItemById returns the item, or *nil* if not found.

1.7.1.3.6 TRVReportTableCellData

TRVReportTableCellData is a class of cell of a report table (TRVReportTableItemInfo[124]).

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[152] and Name[153]. If DataQuery is specified, the content of this cell is repeated for each record returned by the execution of this query. Name allows to refer this cell from data fields[27].

Also, this cell may be linked to a color changer (ColorChangerId[151]) or a value visualizer (VisualizerId[153]). They allow changing a background color and displaying a diagram according to the value calculated for this cell.

Cells are accessible using TRVReportTableItemInfo[124].ReportCells[126] property.

1.7.1.3.6.1 Properties

In TRVReportTableCellData

🟩 ColorChangerId[151]
ColorValue[152]
🟩 DataQuery[152]
🟩 HighlightColor[152]
🟩 Name[153]
🟩 Value[153]
🟩 VisualizerId[153]

Inherited from TRVTableCellData

(many properties, see the TRichView manual)

This property allows linking this cell with a color changer.

```
property ColorChangerId: TRVValueVisualizerId[240];
```

This property allows referring to the item in the report table's BackgroundColorChangers[125] collection by its Id[194] property. (or -1, if this cell is not linked to a color changer).

When TRVReportGenerator[79] generates a report, it calculates a value of this item's ValueString[195] for this cell and assigns it to ColorValue[152].

After applying table's RowGenerationRules[126] and CrossTabulation[126], 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[128] to assign a new value to this property as an editing operation.

Default value:

-1

See also

- Definitions of Report Workshop terms [11]

This property contains a value used for the cell's background color changer.

```
property ColorValue: Variant;
```

When TRVReportGenerator [79] generates a report, it calculates values of table.BackgroundColorChangers [125].FindItemById [150](ColorChangerId [151]).ValueString [195] 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 [11]

A string containing a data query.

```
property DataQuery: TRVUnicodeString;
```

An example of a DataQuery is a SQL query, like 'SELECT * FROM MyTable'.

TRVReportGenerator [79] creates a query processor [252] 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 [128] to assign a new value to this property as an editing operation.

Default value:

'' (empty string)

See also

- Definitions of Report Workshop terms [11]
- Information about data queries [13]
- TRVReportDocObject [247].DataQuery [249]
- TRVRowGenerationCustomRule [155].DataQuery [156]
- TRVCrossTabLevel [140].DataQuery [143]

Specifies the color to highlight this cell in the report template.

```
property HighlightColor: TRVColor;
```

The cell is highlighted if:

- its DataQuery [152] is not empty
- the first TRVReportDocObject [247] item in TRichView.DocObjects has HighlightRules [249] = *True*;
- the owner table's Processed [126] = *False*

The cell is shaded with this color.

A direct assignment to this property cannot be undone by the user. Use Table.SetCellHighlightColor [129] to assign a new value to this property as an editing operation.

Default value:

$41C589 for VCL and LCL; $FF89C541 for FMX

See also

- Definitions of Report Workshop terms [11]
- TRVRowGenerationCustomRule [155].HighlightColor [157]
- TRVCrossTab [130].HighlightColor [138]

A name of this cell.

```
property DataQuery: TRVUnicodeString;
```

A name allows data fields [27] to refer to data produced by an application this cell's DataQuery [152]. Also, it makes it easier to identify a cell in the report generator events, such as OnDataQueryProgress [68].

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.

A direct assignment to this property cannot be undone by the user. Use Table.SetCellName [129] to assign a new value to this property as an editing operation.

Default value:

'' (empty string)

See also

- Definitions of Report Workshop terms [11]
- TRVRowGenerationCustomRule [155].Name [157]

This property contains a value used for the cell's value visualizer.

```
property Value: Variant;
```

When TRVReportGenerator [79] generates a report, it calculates values of table.BackgroundVisualizers [125].FindItemById [150](VizualizerId [153]).ValueString [195] 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 [11]

This property allows linking this cell with a value visualizer.

```
property VisualizerId: TRVValueVisualizerId [240];
```

This property allows referring to the item in the report table's BackgroundVisualizers [125] collection (or -1, if this cell is not linked to a background visualizer).

When TRVReportGenerator [79] generates a report, it calculates a value of this item's ValueString [195] for this cell and assigns it to Value [153].

After applying table's RowGenerationRules [126] and CrossTabulation [126], 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 [129] to assign a new value to this property as an editing operation.

Default value:

-1

See also

- Definitions of Report Workshop terms [11]

1.7.1.3.7 TRVReportValueVisualizers

TRVReportValueVisualizers is a collection of value visualizers for a report table [124].

Unit RVReportValueVisualizer;

Syntax

```
TRVReportValueVisualizers = class (TCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection
TRVReportCustomValueVisualizerCollection [149]

Description

TRVReportValueVisualizers is a class of TRVReportTableItemInfo [124].BackgroundVisualizers [125] property. This collection may include items of different classes, inherited from TRVReportCustomValueVisualizer [191].

See also

- TRVReportTableCellData [150].VisualizerId [153]

1.7.1.3.7.1 Properties

In TRVReportValueVisualizers

Items [155]

Inherited from TCollection

▶ Count

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.

1.7.1.3.8 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 [251] – a class of items in the collection TRVReportTableItemInfo [124] .RowGenerationRules [126].
- TRVReportGenerationCustomRule [251] – a class of items in the collection SubRules [158].

Main properties:

- DataQuery [156] – a data query to apply to the rows
- FirstRow, RowCount [156] – a range of table rows that are replicated when applying the rule
- Name [157] – a name of the rule, allowing to refer to it from data fields [27]
- SubRules [158] – rules applied to subsets of cells belonging to the rows defined with FirstRow, RowCount [156]

1.7.1.3.8.1 Properties

In TRVRowGenerationCustomRule

- DataQuery [156]
- FirstRow [156]
- HighlightColor [157]
- Name [157]
- RowCount [156]
- SubRules [158]

A string containing a data query.

```
property DataQuery: TRVUnicodeString;
```

An example of a DataQuery is a SQL query, like 'SELECT * FROM MyTable'.

TRVReportGenerator[79] creates a query processor[252] for this data query. This query processor returns data for this query, and the report generator replicates rows specified in FirstRow and RowCount[156] 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[126].ColumnGenerationType[135] = *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[11]
- Information about data queries[13]
- TRVReportDocObject[247].DataQuery[249]
- TRVReportTableCellData[150].DataQuery[152]
- TRVCrossTabLevel[140].DataQuery[143]

The properties defining a range of rows of the report table[124] 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[160], **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[163], this range is inside 0..Table.RowCount-1;
- for a nested rule[160], 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

Specifies the color to highlight row(s) belonging to this rule in the report template.

```
property HighlightColor: TRVColor;
```

The row(s) are highlighted if:

- the first TRVReportDocObject[247] item in TRichView.DocObjects has HighlightRules[249] = *True*;
- the owner table's Processed[126] = *False*
- FirstRow[156] and RowCount[156] 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[124] .SelectedRule[127].

Default value:

$B18419 for VCL and LCL; $FF1984B1 for FMX

See also

- Definitions of Report Workshop terms[11]
- TRVReportTableCellData[152].HighlightColor[152]
- TRVCrossTab[130].HighlightColor[138]
- TRVReportTableItemInfo[124].SelectedRule[127]

A name of this row generation rule.

```
property Name: TRVUnicodeString;
```

A name allows data fields[27] 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[68].

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 [11]
- TRVReportTableCellData [150].Name [153]

A collection of nested rules for this rule.

```
property SubRules: TRVRowGenerationNestedRules[162];
```

Nested rules are applied only if:

- if cross-tabulation is not defined (i.e. the table's CrossTabulation [130].Levels [139].Count = 0)
- copying is not defined in the rule (i.e. CopyColCount [165] = 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[126].Add do
begin
  FirstRow[156] := 1;
  RowCount[156] := 2;
  DataQuery[156] := 'SELECT Category FROM Products';
  with SubRules[158].Add do
  begin
    FirstRow[156] := 1; // the actual first row is 1 + 1 = 2
    RowCount[156] := 1;
    DataQuery[156] :=
      'SELECT Product FROM Products WHERE Category=''{Category}''';
  end;
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[126].Add do
begin
  FirstRow[156] := 1;
  RowCount[156] := 1;
  DataQuery[156] := 'SELECT Category FROM Products';
  with SubRules[158].Add do
  begin
    FirstRow[156] := 0; // the actual first row is 1 + 0 = 1
    RowCount[156] := 1;
    SkipLeftColCount[161] := 1;
    DataQuery[156] :=
      'SELECT Product FROM Products WHERE Category=''{Category}''';
  end;
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 of 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[161] 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

1.7.1.3.9 TRVRowGenerationNestedRule

TRVRowGenerationNestedRule is an item in TRVRowGenerationRules[162] collection.

Unit RVReportTable;

Syntax

```
TRVRowGenerationNestedRule = class (TRVRowGenerationCustomRule[155])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVRowGenerationCustomRule[155]

Description

TRVRowGenerationNestedRule is a class of items in the collection TRVRowGenerationCustomRule[155].SubRules[158].

The report table[124] has RowGenerationRules[126] property, a collection of TRVRowGenerationRule[163] items. Each item has SubRules[158] property, a collection of **TRVRowGenerationNestedRule**. Each item in this collection has its own SubRules[158], 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[156] properties. However, rows are counted from the parent rule's first row, not from the table beginning.

This class introduces new properties allowing to define outermost columns: SkipLeftColCount and SkipRightColCount[161]. Cells of these columns are not replicated when applying this rule. Instead, they are merged vertically.

1.7.1.3.9.1 Properties

In TRVRowGenerationNestedRule

- MergeWithPrevious[161]
- SkipLeftColCount[161]
- SkipRightColCount[161]

Inherited TRVRowGenerationCustomRule 155

- DataQuery 156
- FirstRow 156
- HighlightColor 157
- Name 157
- RowCount 156
- SubRules 158

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 156 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 156 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 158). If MergeWithPrevious 161 = *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

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 161.

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
	2
Vegetables	tomatoes
	cucumbers
	2

Default value:

False

1.7.1.3.10 TRVRowGenerationNestedRules

TRVRowGenerationNestedRules is a collection of row generation sub-rules for a report table[124].

Unit RVReportTable;

Syntax

```
TRVRowGenerationNestedRules = class (TOwnedCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection

Description

TRVRowGenerationRules [168] is a class of TRVReportTableItemInfo [124].RowGenerationRules [126] property. It is a collection of TRVRowGenerationRule [163] items.

Each item in this collection has SubRules [158] property of **TRVRowGenerationNestedRules** class. It is a collection of TRVRowGenerationNestedRule [160] items. Each item in its order has its own SubRules [158] property.

1.7.1.3.10.1 Properties

In TRVRowGenerationNestedRules

Items [163]

Inherited from TCollection

▶ Count

Lists the items in the collection.

```
property Items[Index: Integer]: TRVRowGenerationRule[163]; default;
```

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

1.7.1.3.11 TRVRowGenerationRule

TRVRowGenerationRule is an item in TRVRowGenerationRules [168] collection.

Unit RVReportTable;

Syntax

```
TRVRowGenerationRule = class (TRVRowGenerationCustomRule[155])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVRowGenerationCustomRule [155]

Description

TRVRowGenerationRule is a class of items in the collection TRVReportTableItemInfo [124].RowGenerationRules [126].

In addition to properties inherited from TRVRowGenerationCustomRule [155], this class introduces the properties listed below.

Properties used when building a cross-tab report:

- KeyFieldNames[168] – a list of fields allowing to distinguish records when grouping them in table rows
- CaseSensitive[165] specifies how values of text key fields are compared
- AllowSummaryCols, AllowSummaryRows[165] allow generating reports that do not have summary rows and columns more efficiently.

Properties used for copying columns:

- CopyFirstCol, CopyColCount, CopySpacingColCount, CopyMaxCount[165]

Scripts:

- Script_OnStart, Script_BeforeRecord, Script_AfterRecord, Script_OnEnd[168] are executed when this rule is applied.

Special report features

There are three special report features that can be used in rules:

- cross-tabulation (mainly defined in the table's CrossTabulation[130] 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[158], 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[130].Levels[139].Count > 0), neither column copying nor nested rules are applied;
- if column copying is defined in the rule (i.e. CopyColCount[165] > 0), nested rules are not applied; however, different rules in the same table may have column copying and nested rules.

1.7.1.3.11.1 Properties

In TRVRowGenerationRule

- CopyColCount[165]
- CopyFirstCol[165]
- CopyMaxCount[165]
- CopySpacingColCount[165]
- CopySpacingColCount[165]
- Essential[167]
- Script_BeforeRecord[168]
- Script_OnEnd[168]
- Script_OnStart[168]

Inherited TRVRowGenerationCustomRule[155]

- DataQuery[156]
- FirstRow[156]
- HighlightColor[157]
- Name[157]
- RowCount[156]

- SubRules[158]

The properties allow generating cross-tab reports[130] 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[124] 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

Defines how values of text fields in KeyFieldNames[168] are compared when building a cross-tab[130] report.

```
property CaseSensitive: Boolean;
```

Default value:

False

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[126].Levels[139].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[156], 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[126].Levels[139].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[158] 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[156] = 1, RowCount[156] = 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[156] = 1, RowCount[156] = 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 (Developmen t)	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 [11]

Specifies whether results of this row generation rule are required for this table.

```
property Essential: Boolean;
```

If this property is *True*, if this rule produces 0 records when applied, the whole table is deleted from the final report.

The table is always deleted, if a rule produces 0 records and this table consists only of rows belonging to this rule. However, this property is important if the table has rows before and/or after this rule's rows.

Default value:

False

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[156]) 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[156] execution, this field is ignored, and OnError[70] event occurs with the parameter *rvrgeCrossTabUnknownField*.

Fields listed in the property may have one of the following types[234]: *rvrftInteger, rvrftBoolean, rvrftDateTime, rvrftDate, rvrftTime, rvrftText, rvrftFloat*. Otherwise, this field is ignored, and OnError[70] event occurs with the parameter *rvrgeFieldUnsupportedType*.

If a field listed in this property has *rvrftText* type, its values are compared using CaseSensitive[165] property.

Default value:

'' (empty string)

Example:

'id1;id2'

See also:

- information about building a cross-tab report in the topic on TRVCrossTab[130]

Scripts[15] that are executed when this row generation rule is applied.

```
property Script_OnStart: TStrings;
property Script_BeforeRecord: TStrings;
property Script_AfterRecord: TStrings;
property Script_OnEnd: TStrings;
```

Script_OnStart is executed before the rule is applied, just after the DataQuery[156] is executed.

Script_BeforeRecord is executed before processing each record of DataQuery results.

Script_AfterRecord is executed after processing each record of DataQuery results.

Script_OnEnd is executed at the end of the rule application.

Assignment to these properties copies TString object.

See also

- scripts associated with the whole report[250]

1.7.1.3.12 TRVRowGenerationRules

TRVRowGenerationRules is a collection of row generation rules for a report table[124].

Unit RVReportTable;

Syntax

```
TRVRowGenerationRules = class (TOwnedCollection);
```

Hierarchy

TObject
TPersistent
TCollection
TOwnedCollection

Description

TRVRowGenerationRules is a class of TRVReportTableItemInfo[124].RowGenerationRules[126] property. It is a collection of TRVRowGenerationRule[163] items.

Each item in this collection has SubRules[158] property of TRVRowGenerationNestedRules[162] class. It is a collection of TRVRowGenerationNestedRule[160] items. Each item in its order has its own SubRules[158] property.

1.7.1.3.12.1 Properties

In TRVRowGenerationRules

Items[169]

Inherited from TCollection

▶ Count

Lists the items in the collection.

```
property Items[Index: Integer]: TRVRowGenerationRule; default;
```

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

1.7.2 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[174].

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[173]. Its size is specified in Width and Height[173] and EqualSides[171] properties.

The shape is colored using Color[171], StartColor[173], GradientType[172], Opacity[171] properties. It is outlined using LineColor[172], LineWidth[173], LineUsesFillColor[172] properties.

The object background is filled using BackgroundColor and BackgroundOpacity[171] properties.

The StyleNo of this item is rvsShape (-211).

See also

- TRVShape[114] component

1.7.2.1 Properties

In TRVShapeItemInfo

Alt[170]
BackgroundOpacity[171]
Color[171]
EqualSides[171]
GradientType[172]
Height[173]
LineColor[172]
LineUsesFillColor[172]
LineWidth[173]
Opacity[171]
ShapeProperties[173]
StartColor[173]
Width[173]

1.7.2.1.1 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)

1.7.2.1.2 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+ and optionally in Lazarus).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcBackgroundOpacity* 240).

Default value

- 100000 (i.e. 100%)

1.7.2.1.3 TRVShapeItemInfo.Color, Opacity

Fill color and opacity for the shape.

```
Color: TRVColor;
Opacity: TRVOpacity;
```

If GradientType 172 = *rvgtNone*, the shape is filled with this color. Otherwise, this color is used as an ending gradient color (the starting color is StartColor 173).

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

VCL and LCL: Gradient and opacity are used only if shapes are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of these properties as an editing operation (*rveipcShapeColor, rveipcShapeOpacity* 240).

Default values

- Color: $C68E63 for VCL and LCL; $FF638EC6 for FMX
- Opacity: 100000 (i.e. 100%)

1.7.2.1.4 TRVShapeItemInfo.EqualSides

Specifies whether the shape has the same width and height.

```
EqualSides: Boolean;
```

If *False*, the shape is drawn in the rectangle Width 173 × Height 173.

If *True*, the shape is drawn in the square having the side min(Width 173, Height 173).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcEqualSides* 240).

Default value

True

1.7.2.1.5 TRVShapeItemInfo.GradientType

Type of gradient fill for shapes.

```
GradientType: TRVGradientType(233);
```

Gradient is drawn from StartColor(173) to Color(171).

If **GradientType** = *rvgtNone*, the shape is filled with Color(171).

VCL and LCL: Gradient is used only if shapes are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcGradientType*(240)).

Default value

rvgtNone

1.7.2.1.6 TRVShapeItemInfo.LineColor

Line color

```
LineColor: TRVColor
```

Line color can be auto-calculated basing on Color(171), see LineUsesFillColor(172).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcLineColor*(240)).

Default value

rvclBlack

See also

- LineWidth(173)

1.7.2.1.7 TRVShapeItemInfo.LineUsesFillColor

Specifies how shapes are outlined.

```
property LineUsesFillColor: Boolean;
```

If **LineUsesFillColor** = *False*:

Shapes are with LineColor(172).

If **LineUsesFillColor** = *True*:

If Color(171) = *rvclNone*, LineColor(172) is used. Otherwise, if GradientType(172) <> *rvgtNone*, shapes are outlined with Color(171); 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*(240)).

Default value

False

1.7.2.1.8 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* [240]).

Default value

1

See also

- LineColor [172]

1.7.2.1.9 TRVShapeItemInfo.ShapeProperties

Specifies main properties of a shape.

```
property ShapeProperties: TRVReportShapeProperties [254];
```

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* [240]).

1.7.2.1.10 TRVShapeItemInfo.StartColor

Start color for gradient fill.

```
property StartColor: TRVColor;
```

Gradient uses **StartColor** and Color [171].

VCL and LCL: Gradient is used only if shapes are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx to change value of this property as an editing operation (*rveipcStartShapeColor* [240]).

Default value

rvclWhite

1.7.2.1.11 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 [171]

1.8 Data Visualizers

Data visualizers may be associated with cells [150] of report tables [124].

There are two group of visualizers:

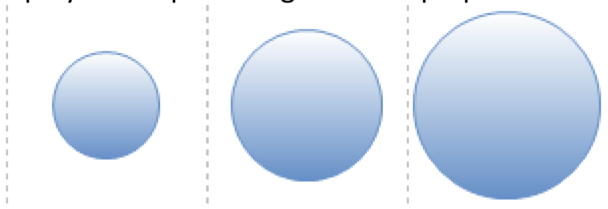
- for changing background colors depending on values
- for displaying diagrams that visualize values (a diagram is drawn on top on the standard cell background, but below the cell content)

The same cell may have both a color changer and a diagram.

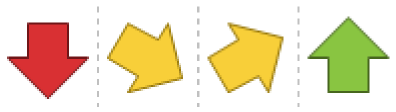
This feature is similar to conditional formatting that can be found in Microsoft Excel, however, there are important differences:

- 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 an 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

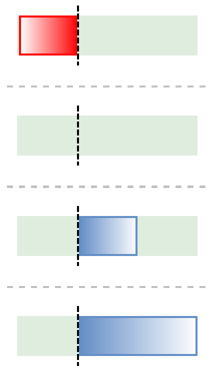
All visualizers are inherited from TRVReportCustomValueVisualizerBase [193]. It defines the string that must be evaluated to get values for each cell, and specify how minimal and maximal values are calculated.
The first group of visualizers (color changers) are represented by a single class: TRVReportColorChangerItem [184].
The second group of visualizers (value visualizers displaying diagrams) include:
TRVReportAreaSizeVisualizer [175]: displays a shape having the area proportional to the value



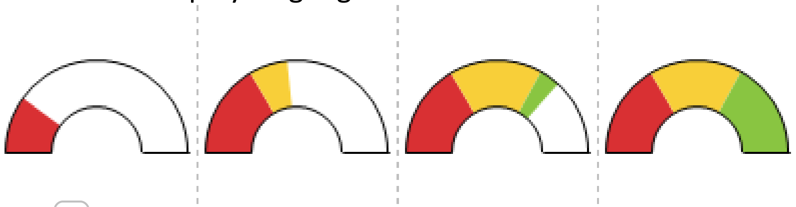
TRVReportColoredShapeVisualizer [186]: displays a shape having colors and rotation depending on the value



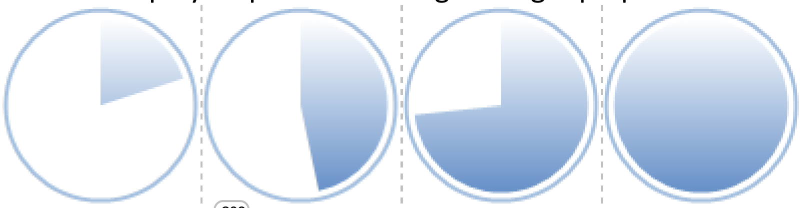
TRVReportBarVisualizer [178]: displays a horizontal or vertical bar having length proportional to the value



TRVReportGaugeVisualizer 196: displays a gauge

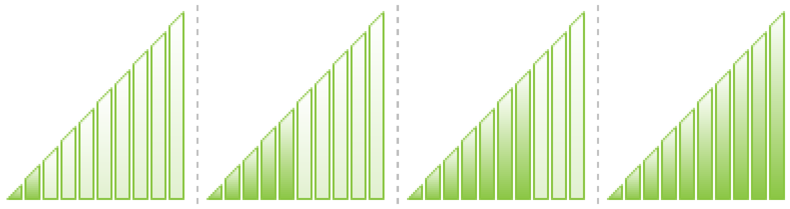


TRVReportPieVisualizer 202: displays a pie slice having an angle proportional to the value



TRVReportShapeRepeaterVisualizer 206: displays the count of shapes proportional to the value

TRVReportSignalStrengthVisualizer 211: displays a diagram that is usually used to show a signal strength or a volume



1.8.1 TRVReportAreaSizeVisualizer

TRVReportAreaSizeVisualizer visualizes positive numeric values by displaying shapes having areas proportional to these values.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportAreaSizeVisualizer = class (TRVReportCustomValueVisualizer 191)
```

Hierarchy

TObject
TPersistent

TCollectionItem
TRVReportCustomValueVisualizerBase[193]
TRVReportCustomValueVisualizer[191]

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[178] property.

Shapes are drawn using Color[193], LineColor[193], and Gradient[177] properties.

Sizes for the minimal and the maximal values can be specified in MinSize and MaxSize[177] properties.

Examples

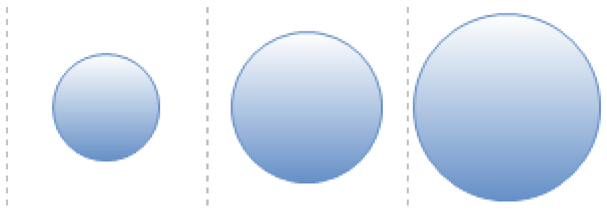
These examples display the following values:

0.0	33.3	66.7	100.0

MinValue[195] = 0, MaxValue[195] = 100.

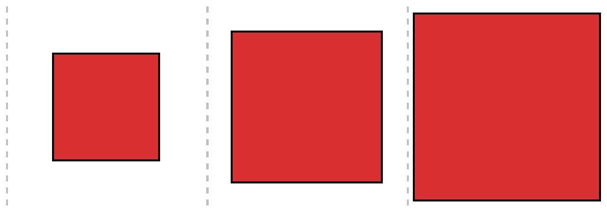
Example1:

Shape[178] = *rvrsshCircle*, Gradient[177] = *True*, Color[193] = LineColor[193] = $C68E63



Example 2

Shape[178] = *rvrsshSquare*, Gradient[177] = *False*, Color[193] = $3330D9, LineColor[193] = *clBlack*.



See also

- TRVReportTableItemInfo[124].BackgroundVisualizers[125]

1.8.1.1 Properties

In TRVReportAreaSizeVisualizer

🟩 Gradient[177]
🟩 MaxSize[177]

- MinSize 177
- Shape 178

Inherited from TRVReportCustomValueVisualizer 191

- Color 193
- HAlign 193
- LineColor 193
- Margin 193
- VAlign 193

Inherited from TRVReportCustomValueVisualizerBase 193

- AutoMaxValue 195
- AutoMinValue 195
- MaxValue 195
- MinValue 195
- ValueString 195

1.8.1.1.1 TRVReportAreaSizeVisualizer.Gradient

Specifies whether shapes must be painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, shapes are filled with Color 193.

If *True*, shapes are filled with a gradient from Color 193 (at bottom) to a twice lighter color (at top).

VCL and LCL: Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Default value

False

1.8.1.1.2 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) or in EMU, depending on Units property of TRVStyle component.

MinSize specifies the size of a shape corresponding to MinValue 195 (or to 0, if MinValue 195 is negative).

The maximal shape size (corresponding to MaxValue 195) 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[195], and maximal possible shape size for MaxValue[195])

1.8.1.1.3 TRVReportAreaSizeVisualizer.Shape

A shape to display.

```
type
  TRVReportSimpleShape = (rvrsshCircle, rvrsshSquare);
property Shape: TRVReportSimpleShape;
```

Shape	Meaning
rvrsshCircle	circle
rvrsshSquare	square

Default value

rvrsshCircle

1.8.2 TRVReportBarVisualizer

TRVReportBarVisualizer visualizes numeric values by displaying bars having lengths proportional to these values.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportBarVisualizer = class (TRVReportCustomValueVisualizer[191])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[193]
TRVReportCustomValueVisualizer[191]

Description

This visualizer shows bars having lengths proportional to visualized values.

Bar lengths are calculated for absolute values.

AxisPosition[180] 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[182].

Bars for positive values are drawn with Color and LineColor[193], bars for negative values are drawn using NegativeColor and NegativeLineColor[184]. Background can be solid or gradient[183].

Optionally, a background is drawn using BackgroundColor and BackgroundOpacity[181] properties.

An axis, if visible, is drawn using AxisColor [180].

Widths of bars are defined in FitWidth [182] and MaxWidth [183] properties.

Examples

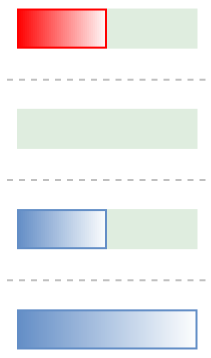
These examples display the following values:

-50	0	50	100

MinValue [195] = -50, MaxValue [195] = 100.

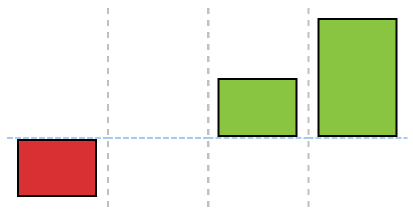
Example 1:

AxisPosition [180] = *rvrbapSide*, Gradient [183] = *True*, BarDirection [182] = *rvrbdRight* (default), Color [193] = $C68E63 (default), NegativeColor [184] = *clRed* (default), LineColor = Color [193], NegativeLineColor = NegativeColor [184], Margin [193] = 5, BackgroundColor [181] = *clMoneyGreen*



Example 2

AxisPosition [180] = *rvrbapAuto* (default), Gradient [183] = *False* (default), BarDirection [182] = *rvrbdUp*, Color [193] = $41C589, NegativeColor [184] = $3330D9, LineColor [193] = *clBlack* (default), NegativeLineColor [184] = *clNone* (default), Margin [193] = 5, , BackgroundColor [181] = *clNone* (default), AxisColor [180] = *clSkyBlue*



See also

- TRVReportTableItemInfo [124].BackgroundVisualizers [125]

1.8.2.1 Properties

In TRVReportBarVisualizer

- AxisColor [180]
- AxisPosition [180]
- BackgroundColor [181]

- BackgroundPosition 181
- BarDirection 182
- FitWidth 182
- Gradient 183
- MaxWidth 183
- NegativeColor 184
- NegativeLineColor 184

Inherited from TRVReportCustomValueVisualizer 191

- Color 193
- HAlign 193
- LineColor 193
- Margin 193
- VAlign 193

Inherited from TRVReportCustomValueVisualizerBase 193

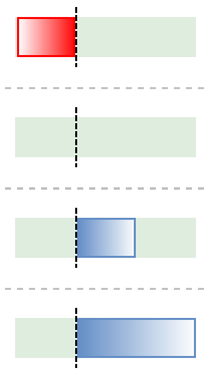
- AutoMaxValue 195
- AutoMinValue 195
- MaxValue 195
- MinValue 195
- ValueString 195

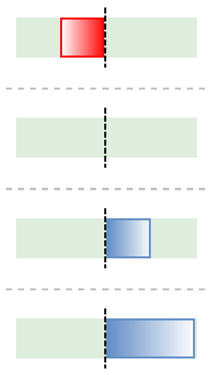
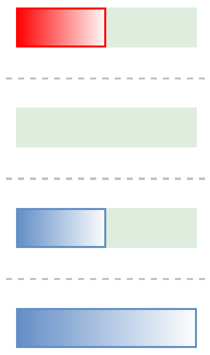
1.8.2.1.1 TRVReportBarVisualizer.AxisPosition, AxisColor

These properties specify the axis position and color.

```
type
  TRVReportBarAxisPosition =
    (rvrbapAuto, rvrbapMiddle, rvrbapSide);
property AxisPosition: TRVReportBarAxisPosition;
property AxisColor: TRVColor;
```

AxisPosition is taken into account only if MinValue 195 < 0. Otherwise, all bars are started from a side (for example, if BarDirection 182 = *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 195 and MaxValue 195 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** = *rvclNone*, an axis is not drawn. Axis is drawn on margins [193] as well.

Default values:

- AxisPosition: *rvrbapAuto*
- AxisColor: *rvclBlack*

1.8.2.1.2 TRVReportBarVisualizer.BackgroundColor, BackgroundOpacity

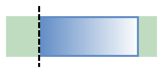
A background color for the visualizer.

```
property BackgroundColor: TRVColor;
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 [193] are not filled with background.

Default value

- BackgroundColor: *rvclNone* (no background)
- BackgroundOpacity: 50000 (i.e. 50%), ignored for BackgroundColor = *rvclNone*

1.8.2.1.3 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[180].

Default value

rvrbdRight

1.8.2.1.4 TRVReportBarVisualizer.FitWidth

Specifies whether all bars have the same width.

```
property FitWidth: Boolean;
```

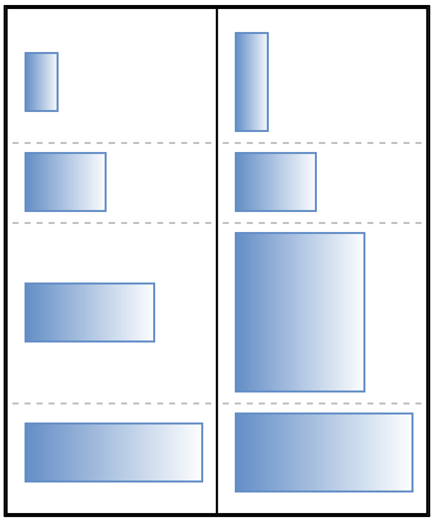
If **FitWidth** = *False*, all bars have the same width, to fit the narrowest cell.

If **FitWidth** = *True*, width of each bar is calculated individually, to fit the container cell.

Bar widths may be limited by MaxWidth[183].

Example:

FitWidth = False	FitWidth = True



In this example, Margin[193] = 5

Default value

True

1.8.2.1.5 TRVReportBarVisualizer.Gradient

Specifies whether bars must be painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, bars are filled with Color[193].

If *True*, bars are filled with a gradient from Color[193] to a twice lighter color, in the direction specified in BarDirection[182].

For negative values, NegativeColor[184] is used instead, and the direction may be opposite, see AxisPosition[180].

VCL and LCL: Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Default value

False

1.8.2.1.6 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) or in EMU, depending on Units property of TRVStyle component.

Default value

0

1.8.2.1.7 TRVReportBarVisualizer.NegativeColor, NegativeLineColor

These properties specify colors for bars displayed for negative values.

```
property NegativeColor: TRVColor;
property NegativeLineColor: TRVColor;
```

NegativeColor is used to fill bars displayed for negative values.

NegativeLineColor is used to draw a rectangle around these bars. If **NegativeLineColor** = *rvclNone*, LineColor[193] is used.

Note: for bars displayed for positive values, Color and LineColor[193] are used

Default value

- NegativeColor: *rvclRed*
- NegativeLineColor: *rvclNone*

1.8.3 TRVReportColorChangerItem

TRVReportColorChangerItem is a class for color changers.

Unit RVReportColorChanger;

Syntax

```
TRVReportColorChangerItem = class (TRVReportCustomValueVisualizerBase[193])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[193]

Description

TRVReportColorChangerItem is a class of items in TRVReportTableItemInfo[124] .BackgroundColorChangers[125].

If a report cell[150] is linked[151] to a color changer, this color changer changes the cell color and opacity according to a value[152] 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[186] – colors for defining a color scale
- ColorBegin, ColorMiddle, ColorEnd[185] – the same for opacity
- PercentMiddle[186] defines the midpoint.

The following properties are inherited:

- ValueString[195] – expression to evaluate for cells.
- MinValue, MaxValue, AutoMinValue, AutoMaxValue[195] – properties for calibration of the color scale

See also

- Definitions of Report Workshop terms [11]

1.8.3.1 Properties

In TRVReportColorChangerItem

- 🟩 ColorBegin [185]
- 🟩 ColorEnd [185]
- 🟩 ColorMiddle [185]
- 🟩 OpacityBegin [186]
- 🟩 OpacityEnd [186]
- 🟩 OpacityMiddle [186]
- 🟩 PercentMiddle [186]

Inherited from TRVReportCustomValueVisualizerBase [193]

- 🟩 AutoMaxValue [195]
- 🟩 AutoMinValue [195]
- 🟩 MaxValue [195]
- 🟩 MinValue [195]
- 🟩 ValueString [195]

1.8.3.1.1 TRVReportColorChangerItem Colors

These properties specify colors.

```
property ColorBegin: TRVColor
property ColorMiddle: TRVColor
property ColorEnd: TRVColor;
```

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 [195] (all lesser values are treated as MinValue)

ColorEnd is used for values equal to MaxValue [195] (all greater values are treated as MaxValue).

For other values:

- If **ColorMiddle** = *rvclNone*, the cell is shaded with gradations of **ColorBegin** and **ColorEnd**, proportionally to (value - MinValue) / (MaxValue - MinValue).
- If **ColorMiddle** <> *rvclNone*, 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 for VCL and LCL; $FF63BE7B for FMX
- ColorMiddle: $84EBFF for VCL and LCL; $FFFFEB84 for FMX
- ColorEnd: $6B69F8 for VCL and LCL; $FFF8696B for FMX

See also

- OpacityBegin, OpacityMiddle, OpacityEnd [186]
- PercentMiddle [186]

1.8.3.1.2 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[185] <> *rv*clNone.

An opacity corresponding the given value is calculated similarly to colors[185].

Default value

100000 (100%)

See also

- ColorBegin, ColorMiddle, ColorEnd[185]
- PercentMiddle[186]

1.8.3.1.3 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[185] <> *rvclNone*.

The midpoint value is calculated as MinValue[195] + (MaxValue[195] - MinValue[195]) * **PercentMiddle** / 100. This value corresponds to ColorMiddle[185] and OpacityMiddle[186].

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[186]
- ColorBegin, ColorMiddle, ColorEnd[185]

1.8.4 TRVReportColoredShapeVisualizer

TRVReportColoredShapeVisualizer visualizes numeric values by applying color and rotation to a shape.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportColoredShapeVisualizer = class (TRVReportCustomShapeVisualizer[190])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[193]
TRVReportCustomValueVisualizer[191]
TRVReportCustomShapeVisualizer[190]

Description

The shape is defined in ShapeProperties[191].

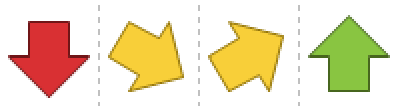
A color and a rotation angle of this shape depend on the visualized value. The visualizer searches in ConditionalShapeProperties[189] for the item corresponding to the visualized value. If none of the items corresponds to the visualized value (it may happen for low values), the default color and rotation angle is used.

Shape size may be limited by MaxSize[189].

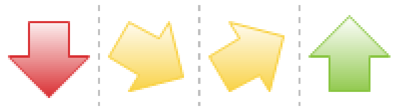
By default, the shape is inscribed in a square. You can stretch it by changing ShapeScaleX[191].

Examples

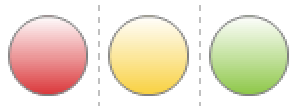
Example 1: ShapeProperties[191].Shape[255] = *rvrshArrow1*, ShapeProperties[191].StartAngle[255] = 180, angles in ConditionalShapeProperties[189] are from -60 to -180 (so arrows are rotated by 180°, 120°, 60°, 0°), Gradient[191] = *False*



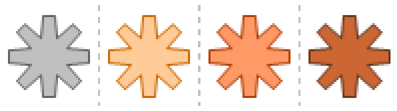
Example 2: The same settings, Gradient[191] = *True*



Example 3: ShapeProperties[191].Shape[255] = *rvrshCircle*, LineUsesFillColor[191] = *False*, LineColor[193] = *clSilver*



Example 4: ShapeProperties[191].Shape[255] = *rvrshBluntPointStar*, ShapeProperties[191].PointCount[255] = 8, Gradient[191] = *False*



Note: in all these examples, a visualized value is increased from left to right. The leftmost shape is drawn using Color[193] and ShapeProperties[191].StartAngle[255]. Colors and rotations of other shapes are defined in ConditionalShapeProperties[189]

See also

- TRVReportTableItemInfo 124 .BackgroundVisualizers 125

1.8.4.1 Properties

In TRVReportColoredShapeVisualizer

- AbsoluteValues 188
- ConditionalShapeProperties 189
- MaxSize 189

Inherited from TRVReportCustomShapeVisualizer 190

- Gradient 191
- LineUsesFillColor 191
- ShapeProperties 191
- ShapeScaleX 191

Inherited from TRVReportCustomValueVisualizer 191

- Color 193
- HAlign 193
- LineColor 193
- Margin 193
- VAlign 193

Inherited from TRVReportCustomValueVisualizerBase 193

- AutoMaxValue 195
- AutoMinValue 195
- MaxValue 195
- MinValue 195
- ValueString 195

1.8.4.1.1 TRVReportColoredShapeVisualizer.AbsoluteValues

Specifies whether values in ConditionalShapeProperties 189 are defined as absolute values or as percentage.

```
property AbsoluteValues: Boolean;
```

If *False*, all ConditionalShapeProperties 189 [].Value 246 s are specified in percentage.

If *True*, all ConditionalShapeProperties 189 [].Value 246 s are specified in absolute values

Default value

False

1.8.4.1.2 TRVReportColoredShapeVisualizer.ConditionalShapeProperties

Specifies a set of shape properties that are applied depending on the visualized value.

```
property ConditionalShapeProperties: TRVConditionalShapePropertiesCollection[244];
```

The programmer must ensure that items in this collection are sorted in an ascending order by **ConditionalShapeProperties**[].Value[246].

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[246] and the visualized value returns *True*, this item is used. If all comparisons return *False*, Color[193] and ShapeProperties[191].StartAngle[255] are applied.

If the *Index*-th item of this collection is used, a shape is displayed using **ConditionalShapeProperties**[*Index*].Color[245] and ShapeProperties[191].StartAngle[255] + **ConditionalShapeProperties**[*Index*].DeltaAngle[245].

Comparisons with the visualized value depend on AbsoluteValues[188] property.

- If AbsoluteValues[188] = *True*, values are compared as:
  Visualized_Value >= **ConditionalShapeProperties**[].Value[246]
- If AbsoluteValues[188] = *False*, values are compared as:
  (*Visualized_Value* - MinValue[195]) *100 / (MaxValue[195] - MinValue[195]) >= **ConditionalShapeProperties**[].Value[246])

Values less than MinValue[195] are treated as MinValue[195], values greater than MaxValue[195] are treated as MaxValue[195].

Example

Let this collection has two items:

- **ConditionalShapeProperties**[0].Value[246] = 3
- **ConditionalShapeProperties**[1].Value[246] = 7

AbsoluteValues[188] = *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.

1.8.4.1.3 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) or in EMU, depending on Units property of TRVStyle component.

Default value

0

1.8.5 TRVReportCustomShapeVisualizer

TRVReportCustomShapeVisualizer is a base class for visualizers that draw shapes.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportCustomShapeVisualizer = class (TRVReportCustomValueVisualizer 191)
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase 193
TRVReportCustomValueVisualizer 191

Description

This class is not used directly. The following visualizers are inherited from this class:

- TRVReportColoredShapeVisualizer 186
- TRVReportShapeRepeaterVisualizer 206

TRVReportCustomShapeVisualizer introduces properties:

- ShapeProperties 191 – shape type and other properties,
- LineUsesFillColor 191, Gradient 191 specify how shapes are outlined and filled.

See also

- TRVReportTableItemInfo 124.BackgroundVisualizers 125

1.8.5.1 Properties

In TRVReportCustomShapeVisualizer

- 🟩 Gradient 191
- 🟩 LineUsesFillColor 191
- 🟩 ShapeProperties 191
- 🟩 ShapeScaleX 191

Inherited from TRVReportCustomValueVisualizer 191

- 🟩 Color 193
- 🟩 HAlign 193
- 🟩 LineColor 193
- 🟩 Margin 193
- 🟩 VAlign 193

Inherited from TRVReportCustomValueVisualizerBase 193

- 🟩 AutoMaxValue 195
- 🟩 AutoMinValue 195
- 🟩 MaxValue 195
- 🟩 MinValue 195

- ValueString 195

1.8.5.1.1 TRVReportCustomShapeVisualizer.Gradient

Specifies whether shapes are painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, shapes are filled with Color 193.

If *True*, shapes are filled with a gradient from Color 193 (at bottom) to a twice lighter color (at top).

VCL and LCL: Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Default value

True

1.8.5.1.2 TRVReportCustomShapeVisualizer.LineUsesFillColor

Specifies how shapes are outlined.

```
property LineUsesFillColor: Boolean;
```

If **LineUsesFillColor** = *False*:

Shapes are with LineColor 193.

If **LineUsesFillColor** = *True*:

If the corresponding fill color is equal to *rvclNone*, LineColor 193 is used. Otherwise, if Gradient 191 = *True*, shapes are outlined with Color 193; if Gradient 191 = *False*, they are outlined with twice darker colors.

Default value

True

1.8.5.1.3 TRVReportCustomShapeVisualizer.ShapeProperties

Specifies main properties of a shape

```
property ShapeProperties: TRVReportShapeProperties 254;
```

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

1.8.5.1.4 TRVReportCustomShapeVisualizer.ShapeScaleX

Horizontal scale ratio for the shape, percent

```
property ShapeScaleX: Integer;
```

"Shape width" is calculated as "shape height" * ShapeScaleX / 100.

1.8.6 TRVReportCustomValueVisualizer

TRVReportColorChangerItem is a base class for value visualizers.

Unit RVReportColorChanger;

Syntax

```
TRVReportCustomValueVisualizer = class (TRVReportCustomValueVisualizerBase[193])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[193]

Description

This class is not used directly. All value visualizers are inherited from this class.

TRVReportCustomValueVisualizer is a base class for items in TRVReportTableItemInfo[124].BackgroundVisualizers[125].

This class introduces new properties, common for all value visualizers:

- HAlign, VAlign[193] – alignment of images displayed by the visualizers relative to respective cells.
- Margin[193] – space around the image.
- Color, LineColor[193] – colors of shapes drawn by the visualizer.

The following properties are inherited:

- ValueString[195] – expression to evaluate for cells.
- MinValue, MaxValue, AutoMinValue, AutoMaxValue[195] – properties for calibration of the visualizer.

See also

- Definitions of Report Workshop terms[11]

1.8.6.1 Properties

In TRVReportCustomValueVisualizer

- Color[193]
- HAlign[193]
- LineColor[193]
- Margin[193]
- VAlign[193]

Inherited from TRVReportCustomValueVisualizerBase[193]

- AutoMaxValue[195]
- AutoMinValue[195]
- MaxValue[195]
- MinValue[195]
- ValueString[195]

1.8.6.1.1 TRVReportCustomValueVisualizer.Color, LineColor

These properties specify colors.

```
property Color: TRVColor
property LineColor: TRVColor
```

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** <> *rvc*lNone, 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 for VCL and LCL; $FF638EC6 for FMX
- LineColor: *rvclBlack*

1.8.6.1.2 TRVReportCustomValueVisualizer.HAlign, VAlign

These properties define positions of images displayed by the visualizer, relative to the respective cells.

```
property HAlign: 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[193])

Default values

rvcCenter, rvcMiddle

1.8.6.1.3 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) or in EMU, depending on Units property of TRVStyle component.

Default value

0

1.8.7 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[184]. This is a class of items in TRVReportTableItemInfo[124].BackgroundColorChangers[125]. Color changers allow modifying background color and opacity for cells depending on the value of expression[195] calculated for these cells.

There are many *value visualizers* available, all of them are inherited from TRVReportCustomValueVisualizer[191]. They are classes of items in TRVReportTableItemInfo[124].BackgroundVisualizers[125]. 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[195] – expression to evaluate for cells
- MinValue, MaxValue, AutoMinValue, AutoMaxValue[195] – properties for calibration of visualizers

See also

- Definitions of Report Workshop terms[11]

1.8.7.1 Properties

In TRVReportCustomValueVisualizerBase

- 🟩 AutoMaxValue[195]
- 🟩 AutoMinValue[195]
- ▶ Id[194]
- 🟩 MaxValue[195]
- 🟩 MinValue[195]
- 🟩 ValueString[195]

1.8.7.1.1 TRVReportCustomValueVisualizerBase.Id

An unique identifier of this item in the collection.

```
property Id: TRVValueVisualizerId[240];
```

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).

1.8.7.1.2 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)

1.8.7.1.3 TRVReportCustomValueVisualizerBase.ValueString

An expression to calculate.

```
property ValueString: TRVUnicodeString;
```

When generating reports, TRVReportGenerator[79] evaluates an expression stored in this string and store the result[153] in the corresponding report table cell[150]. Later, the result of this evaluation will be visualized on this cell background.

The syntax is similar to data fields[27], but:

- curly brackets are not used;
- format string is not supported.

Syntax:

<value string> ::= [']<full data field name>[27]['][<type>] | [']**%**<variable name>[30]['][<type>] | [']
#<cross-tab field>[36]['][<type>] |
[']<aggregate function name>**(**<param>**)**[37]['][<type>] **|** [']**=**<expression text>[41]['][<type>]

<type>, if specified, may be one of: **int**, **float**, **bool**.

Examples:

- 'Myquery:Myfield' – value of 'Myfield' field from the result of 'Myquery' row generation rule 126
- '%count' – value of 'count' variable
- 'sum(salary)' – sum of 'salary' field values for cross-tab 126 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)

1.8.8 TRVReportGaugeVisualizer

TRVReportGaugeVisualizer visualizes numeric values by displaying them on a gauge.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportGaugeVisualizer = class (TRVReportCustomValueVisualizer 191 )
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase 193
TRVReportCustomValueVisualizer 191

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" categories. 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 200 property.

You can define relative sizes 198 and colors 198 of these areas.

This visualizer overrides initial values of MinValue, MaxValue, AutoMinValue, AutoMaxValue 200 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 201 .

Gauge size may be limited by MaxSize 200 .

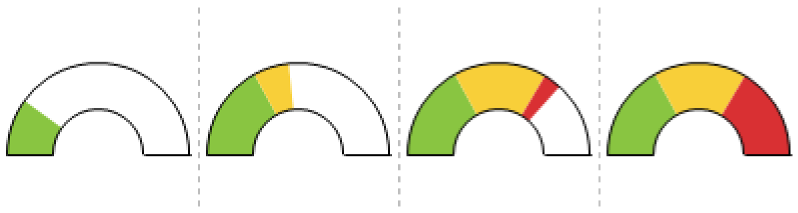
Examples

The examples use the following data:

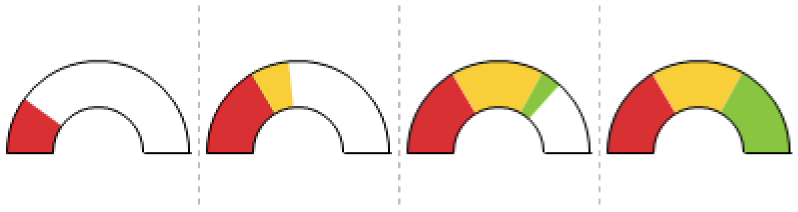
20	46.7	73.3	100

MinValue[200] = 0, MaxValue[200] = 100.

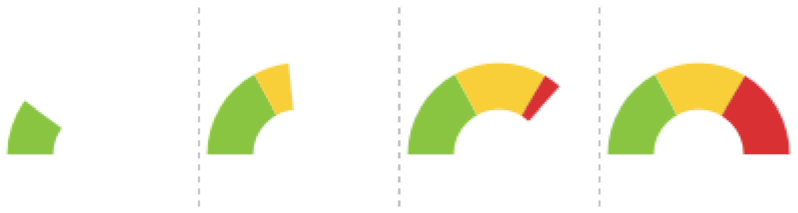
Example 1: Default property values



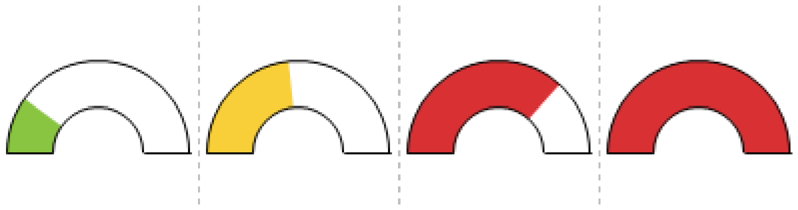
Example 2: RedLowValues[200] = *True*



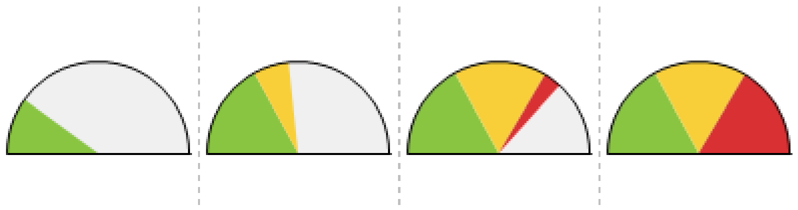
Example 3: LineColor[193] = *clNone*



Example 4: SingleColorMode[201] = *True*



Example 5: MiddleColor[199] = *clNone*, Color[199] = $F0F0F0



See also

- TRVReportTableItemInfo[124].BackgroundVisualizers[125]

1.8.8.1 Properties

In TRVReportGaugeVisualizer

- AutoMaxValue[200]
- AutoMinValue[200]
- Color[199]

- GreenAreaColor 198
- MaxSize 200
- MaxValue 200
- MiddleColor 199
- MinValue 200
- RedAreaColor 198
- RedAreaPercent 198
- RedLowValues 200
- SingleColorMode 201
- YellowAreaColor 198
- YellowAreaPercent 198

Inherited from TRVReportCustomValueVisualizer 191

- HAlign 193
- LineColor 193
- Margin 193
- VAlign 193

Inherited from TRVReportCustomValueVisualizerBase 193

- ValueString 195

1.8.8.1.1 TRVReportGaugeVisualizer Area Colors

The properties define colors of the gauge areas.

```
property GreenAreaColor: TRVColor;
property YellowAreaColor: TRVColor;
property RedAreaColor: TRVColor;
```

The range of values from MinValue to MaxValue 200 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 for VCL and LCL; $FF89C541 for FMX
- YellowAreaColor: $39CFF8 for VCL and LCL; $FFF8CF39 for FMX
- RedAreaColor: $3330D9 for VCL and LCL; $FFD93033 for FMX

See also

- RedLowValues 200
- SingleColorMode 201
- RedAreaPercent, YellowAreaPercent 198

1.8.8.1.2 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 [200] 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 [200].

Default values:

33, 33 (so the green area is 34%)

1.8.8.1.3 TRVReportGaugeVisualizer.Color, MiddleColor

Background colors for gauges.

```
property Color: TRVColor;
property MiddleColor: TRVColor;
```

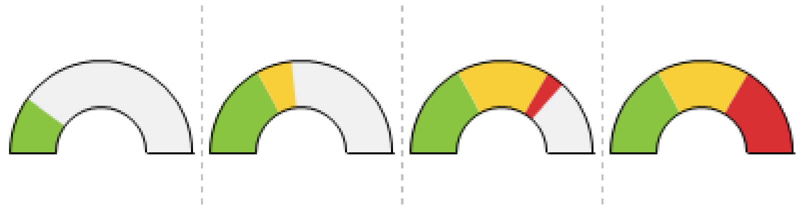
Color is inherited from TRVReportCustomValueVisualizer [191]. See the description of the original property [193]. TRVReportGaugeVisualizer overrides the initial value of this property.

Color is used to fill the rest of a gauge, not filled by colors corresponding to a value.

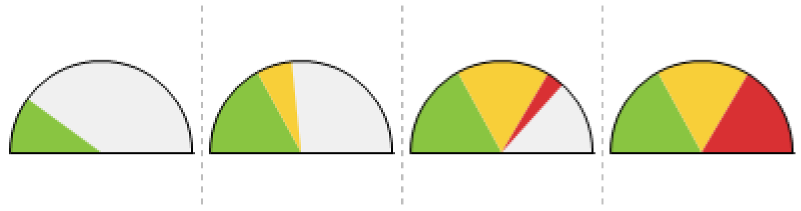
MiddleColor is used to fill the center of a gauge. If **MiddleColor** = *rvclNone*, gauges look like a semicircles, otherwise they look like arcs of semicircles.

Example (for VCL or LCL):

MiddleColor = *clWhite*, **Color** = $F0F0F0



MiddleColor = *clNone*, **Color** = $F0F0F0



Default values

- Color: *rvclWhite*
- MiddleColor: *rvclWhite*

1.8.8.1.4 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) or in EMU, depending on Units property of TRVStyle component.

Default value

0

1.8.8.1.5 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 [193]. See the description of the original properties [195].

TRVReportGaugeVisualizer overrides the initial values of these properties.

Default values

0, 100, False, False

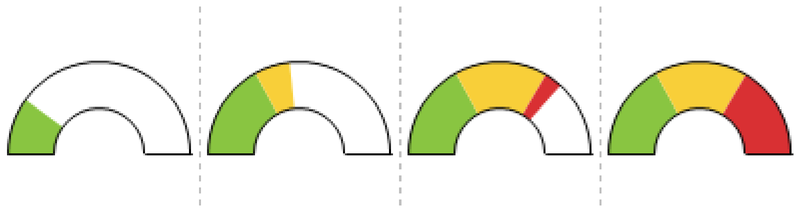
1.8.8.1.6 TRVReportGaugeVisualizer.RedLowValues

This property defines the order of areas of a gauge.

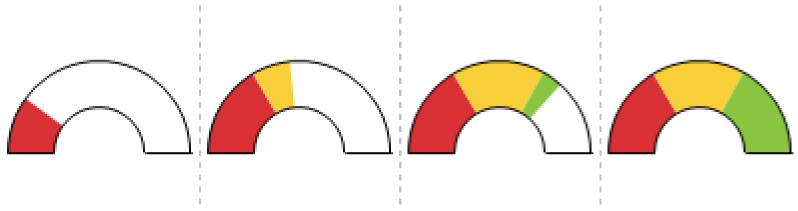
```
property RedLowValues: Boolean;
```

The range of values from MinValue to MaxValue [200] 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 [200] = 0, MaxValue [200] = 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 [198]
- RedAreaColor, YellowAreaColor, GreenAreaColor [198]
- SingleColorMode [201]

1.8.8.1.7 TRVReportGaugeVisualizer.SingleColorMode

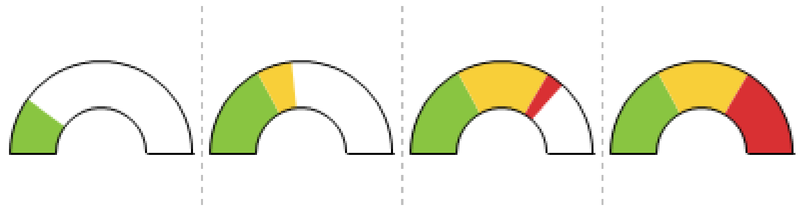
This property specifies whether values are displayed using all colors or a single color.

```
property SingleColorMode: Boolean;
```

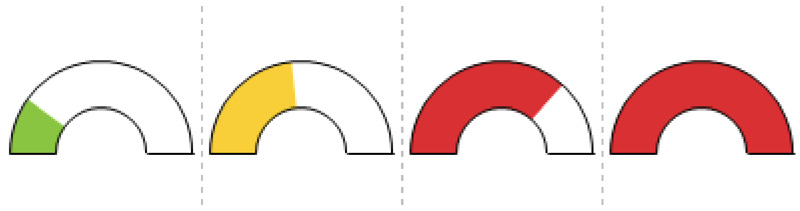
The range of values from MinValue to MaxValue [200] are separated into three areas: "green", "yellow", "red".

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 [200] = 0, MaxValue [200] = 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 [198]
- RedAreaColor, YellowAreaColor, GreenAreaColor [198]
- RedLowValues [200]

1.8.9 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[191])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase [193]
TRVReportCustomValueVisualizer [191]

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 [195], full circle corresponds to MaxValue [195]. Values less than MinValue are displayed as MinValue, values greater than MaxValue are displayed as MaxValue.

Pie slices are filled with Color [193] and Gradient [204]. Circles around slices are drawn using LineColor [204] and LineWidth [204]. Gap [203] 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 [205] property.

Size of these diagrams may be limited by MaxSize [205].

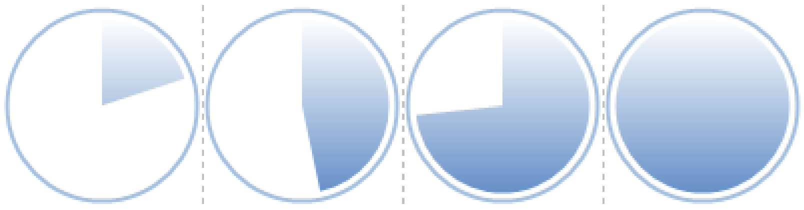
Example

The example uses the following data:

20	46.7	73.3	100

MinValue [200] = 0, MaxValue [200] = 100.

Gradient [204] = True, LineColor [204] is lighter than Color [193] by 1.3, LineWidth [204] = 2, Gap [203] = 4.



See also

- TRVReportTableItemInfo 124 .BackgroundVisualizers 125

1.8.9.1 Properties

In TRVReportPieVisualizer

- Gap 203
- Gradient 204
- LineColor 204
- LineWidth 204
- MaxSize 205
- PartsCount 205
- PartsCount 205

Inherited from TRVReportCustomValueVisualizer 191

- Color 193
- HAlign 193
- Margin 193
- VAlign 193

Inherited from TRVReportCustomValueVisualizerBase 193

- AutoMaxValue 195
- AutoMinValue 195
- MaxValue 195
- MinValue 195
- ValueString 195

1.8.9.1.1 TRVReportPieVisualizer.Gap

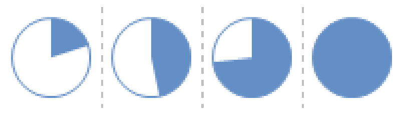
A difference between the radii of the pie itself and of the surrounding circle.

```
property MaxSize: TRVStyleLength;
```

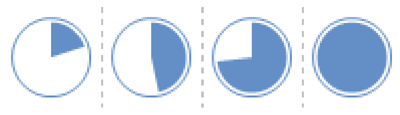
This value is measured in screen pixels or in twips (1/20 of point) or in EMU, depending on Units property of TRVStyle component.

Examples:

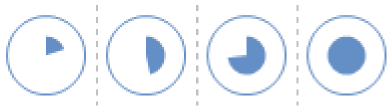
Gap = 0



Gap = 2



Gap = 10



Default value

2

1.8.9.1.2 TRVReportPieVisualizer.Gradient

Specifies whether pie slices are painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, pie slices are filled with Color[193].

If *True*, pie slices are filled with a gradient from Color[193] (at bottom) to a twice lighter color (at top).

VCL and LCL: Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Default value

False

1.8.9.1.3 TRVReportPieVisualizer.LineColor

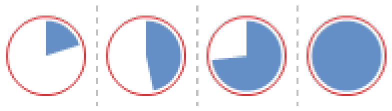
A color of a circles surrounding pies.

```
property LineColor: TRVColor
```

LineColor is inherited from TRVReportCustomValueVisualizer[191]. See the description of the original property[193]. TRVReportPieVisualizer overrides the initial value of this property: by default, it is the same as Color[193].

This circle is drawn if **LineColor** <> *rvclNone* and LineWidth[204] > 0.

Example: red line color



Default value

$C68E63 for VCL and LCL; $FF638EC6 for FMX

1.8.9.1.4 TRVReportPieVisualizer.LineWidth

A line width of a circles what surround pies.

```
property LineWidth: TRVStyleLength;
```

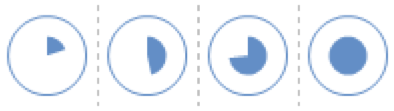
This circle is drawn if LineColor[204] <> *rvclNone* and **LineWidth** > 0.

This value is measured in screen pixels or in twips (1/20 of point) or in EMU, depending on Units property of TRVStyle component.

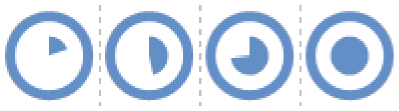
Examples:

In these examples, Gap[203] = 10.

LineWidth = 1



LineWidth = 5



Default value

1

1.8.9.1.5 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) or in EMU, depending on Units property of TRVStyle component.

Default value

0

1.8.9.1.6 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**.

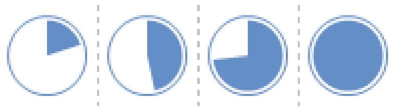
Examples

The examples use the following data:

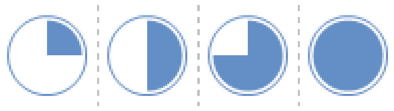
20	46.7	73.3	100

MinValue [195] = 0, MaxValue [195] = 100.

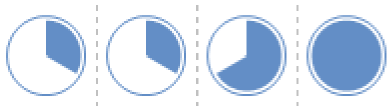
PartsCount = 0



PartsCount = 4



PartsCount = 3



Default value

0

1.8.10 TRVReportShapeRepeaterVisualizer

TRVReportShapeRepeaterVisualizer visualizes numeric values by displaying the count of shapes proportional to these values.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportShapeRepeaterVisualizer = class (TRVReportCustomShapeVisualizer[190])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[193]
TRVReportCustomValueVisualizer[191]
TRVReportCustomShapeVisualizer[190]

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[195], displaying all full shapes corresponds to MaxValue[195]. Values less than MinValue are displayed as MinValue, values greater than MaxValue are displayed as MaxValue.

Shapes are shown in RowCount[208] rows and ColCount[208] columns, the filling direction is specified in Direction[209].

The shape is defined in ShapeProperties[191].

Full shapes are drawn using Color[208] and LineColor[193], empty shapes are drawn using ColorEmpty and LineColorEmpty[209]. Empty shapes can be semitransparent[209]

The following properties affect shape drawing: Gradient[191] and LineUsesFillColor[191].

Optionally, background can be filled using BackgroundColor[208].

Distance between shapes is specified in Spacing[210], space around the whole diagram is specified in Padding[210] and Margin[193].

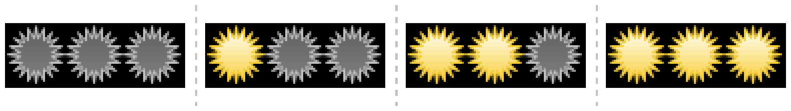
Shape size may be limited by MaxShapeSize[209].

By default, shapes are inscribed in squares. You can stretch them by changing ShapeScaleX[191].

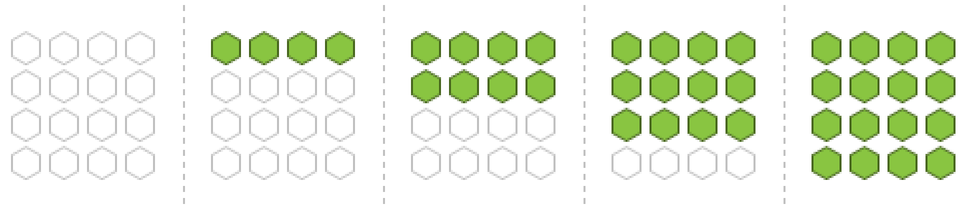
Examples

Example 1: ColCount[208] = 4; ShapeProperties[191]: Shape[255] = *rvrshStar*, PointCount[255] = 5, MiddlePercent[255] 40; Gradient[191] = *False*

Example 2: ColCount 208 = 3; ShapeProperties 191: Shape 255 = *rvrshStar*, PointCount 255 = 20, MiddlePercent 255 60; BackgroundColor 208 = *clBlack*, Padding 210 = 2



Example 3: ColCount 208 = RowCount 208 = 4; ShapeProperties 191: Shape 255 = *rvrshPolygon*, PointCount 255 = 6, Spacing 210 = 2, Gradient 191 = *False*, ColorEmpty 209 = *clNone*, LineColorEmpty 209 = *clSilver*, Color 208 = $41C589:



See also

- TRVReportTableItemInfo 124.BackgroundVisualizers 125

1.8.10.1 Properties

In TRVReportShapeRepeaterVisualizer

- BackgroundColor 208
- ColCount 208
- Color 208
- ColorEmpty 209
- Direction 209
- LineColorEmpty 209
- MaxShapeSize 209
- OpacityEmpty 209
- Padding 210
- RowCount 208
- Spacing 210

Inherited from TRVReportCustomShapeVisualizer 190

- Gradient 191
- LineUsesFillColor 191
- ShapeProperties 191
- ShapeScaleX 191

Inherited from TRVReportCustomValueVisualizer 191

- HAlign 193
- LineColor 193
- Margin 193

- VAlign 193

Inherited from TRVReportCustomValueVisualizerBase 193

- AutoMaxValue 195
- AutoMinValue 195
- MaxValue 195
- MinValue 195
- ValueString 195

1.8.10.1.1 TRVReportShapeRepeaterVisualizer.BackgroundColor

Background color

```
property BackgroundColor: TRVColor;
```

This property defines the color of a rectangle shown below shapes.

Example: BackgroundColor = *rvclGray*

Default value

rvclNone

See also

- Padding 210

1.8.10.1.2 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

1.8.10.1.3 TRVReportShapeRepeaterVisualizer.Color

Color for full shapes.

```
property Color: TRVColor;
```

Color is inherited from TRVReportCustomValueVisualizer 191. See the description of the original property 193. TRVReportShapeRepeaterVisualizer overrides the initial value of this property.

Default value

$39CFF8 for VCL and LCL; $FFF8CF39 for FMX.

See also

- ColorEmpty 209

1.8.10.1.4 TRVReportShapeRepeaterVisualizer.ColorEmpty, LineColorEmpty, OpacityEmpty

These properties define colors and opacity for empty shapes.

```
property LineColorEmpty: TRVColor;
property ColorEmpty: TRVColor;
property OpacityEmpty: TRVOpacity;
```

Empty shapes are filled with **ColorEmpty** and **OpacityEmpty**, and outlined with **LineColorEmpty** (unless LineUsesFillColor[191] = *True*)

Full shapes are filled with Color[208] (fully opaque), and outlined with LineColor[193] (unless LineUsesFillColor[191] = *True*).

Default values

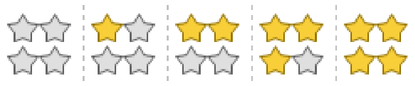
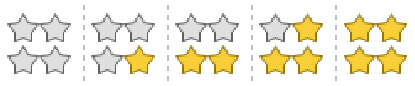
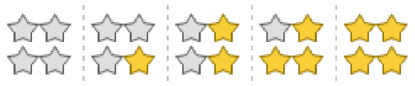
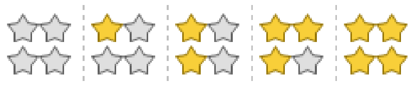
- ColorEmpty: *rvclSilver*
- LineColorEmpty: *rvclBlack*
- OpacityEmpty: 50000 (i.e. 50%)

1.8.10.1.5 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

1.8.10.1.6 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) or in EMU, 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[208] + Spacing[210] * (ColCount[208] - 1) + Padding[210] * 2
- height: **MaxShapeSize** * RowCount[208] + Spacing[210] * (RowCount[208] - 1) + Padding[210] * 2

(additionally, there is space around the diagram specified in Margin[193])

1.8.10.1.7 TRVReportShapeRepeaterVisualizer.Padding

Sets a colored space around images displayed by the visualizer.

```
property Padding: TRVStyleLength;
```

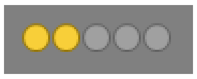
This value is measured in screen pixels or in twips (1/20 of point) or in EMU, depending on Units property of TRVStyle component.

If BackgroundColor[208] = *rvclNone*, this value is simply added to Margin[193]. Otherwise, this space is colored, while margins are transparent.

Examples

Padding = 2

Padding = 10



Default value

1

See also

- Spacing[210]

1.8.10.1.8 TRVReportShapeRepeaterVisualizer.Spacing

A distance between shapes

```
property Spacing: TRVStyleLength;
```

This value is measured in screen pixels or in twips (1/20 of point) or in EMU, depending on Units property of TRVStyle component.

Default value

1

See also

- Padding[210]

1.8.11 TRVReportSignalStrengthVisualizer

TRVReportSignalStrengthVisualizer visualizes numeric values by a drawing a "signal strength" diagrams.

Unit RVReportValueVisualizer;

Syntax

```
TRVReportSignalStrengthVisualizer = class (TRVReportCustomValueVisualizer[191])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TRVReportCustomValueVisualizerBase[193]
TRVReportCustomValueVisualizer[191]

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[213] property. For simplicity, we will call them "bars".

The count of bars is specified in PartsCount[214], a distance between them is specified in Spacing[215].

Values are visualized by filling bars. The count of full bars is proportional to the visualized value. Displaying all empty bars corresponds to MinValue[195], displaying all full bars corresponds to MaxValue[195]. Values less than MinValue are displayed as MinValue, values greater than MaxValue are displayed as MaxValue.

Full bars are drawn using Color and LineColor[193], empty bars are drawn using ColorEmpty and LineColorEmpty[213]. Empty bars can be semitransparent[213].

The following properties affect bar drawing: Gradient[213] and LineUsesFillColor[214].

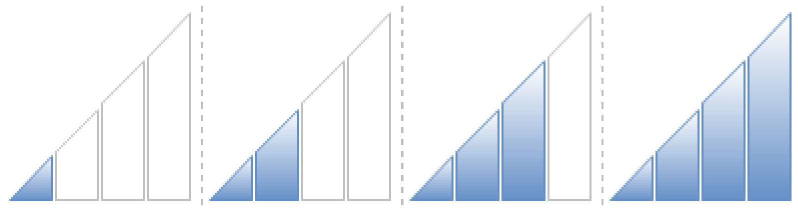
Examples

The examples use the following data:

20	46.7	73.3	100

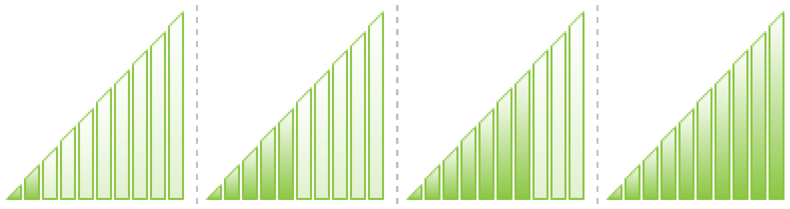
MinValue[200] = 0, MaxValue[200] = 100.

Example 1: default values



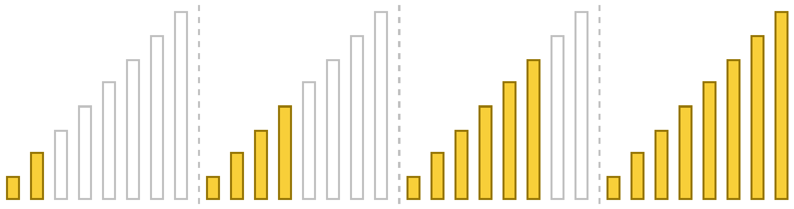
Example 2:

Color [193] = ColorEmpty [213] = $41C589, OpacityEmpty [213] = 25000, PartsCount [214] = 10



Example 3:

Color [193] = $39CFF8, PartsCount [214] = 8, Spacing [215] = 5, Gradient [213] = *False*, DisplayStyle [213] = *rvrsssBars*



See also

- TRVReportTableItemInfo [124].BackgroundVisualizers [125]

1.8.11.1 Properties

In TRVReportSignalStrengthVisualizer

- ColorEmpty [213]
- DisplayStyle [213]
- Gradient [213]
- LineColorEmpty [213]
- LineUsesFillColor [214]
- MaxSize [214]
- OpacityEmpty [213]
- PartsCount [214]
- Spacing [215]

Inherited from TRVReportCustomValueVisualizer [191]

- HAlign [193]
- LineColor [193]
- Margin [193]
- VAlign [193]

Inherited from TRVReportCustomValueVisualizerBase [193]

- AutoMaxValue [195]
- AutoMinValue [195]
- MaxValue [195]
- MinValue [195]
- ValueString [195]

1.8.11.1.1 TRVReportSignalStrengthVisualizer.ColorEmpty, LineColorEmpty, OpacityEmpty

These properties define colors and opacity for empty bars or wedges.

```
property ColorEmpty: TRVColor;
property LineColorEmpty: TRVColor;
property OpacityEmpty: TRVOpacity;
```

Empty bars are filled with **ColorEmpty** and **OpacityEmpty**, and outlined with **LineColorEmpty** (unless LineUsesFillColor[214] = *True*)

Full bars are filled with Color[193] (fully opaque), and outlined with LineColor[193] (unless LineUsesFillColor[214] = *True*).

Default values

- ColorEmpty: *rvclNone*
- LineColorEmpty: *rvclSilver*
- OpacityEmpty: 50000 (i.e. 50%)

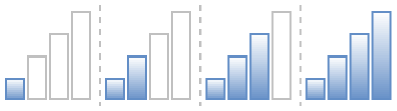
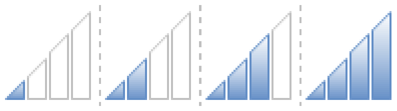
See also:

- Gradient[213]

1.8.11.1.2 TRVReportSignalStrengthVisualizer.DisplayStyle

Specifies shapes used in diagrams.

```
type
  TRVReportSignalStrengthStyle = (rvrsssBars, rvrsssWedges);
property DisplayStyle: TRVReportSignalStrengthStyle;
```

Value	Example
rvrsssBars	
rvrsssWedges	

Default value

rvrsssWedges

1.8.11.1.3 TRVReportSignalStrengthVisualizer.Gradient

Specifies whether bars (or wedges) are painted using a linear gradient fill.

```
property Gradient: Boolean;
```

If *False*, bars are filled with Color[193] / ColorEmpty[213].

If *True*, bars are filled with a gradient from Color[193] / ColorEmpty[213] (at bottom) to a twice lighter color (at top).

VCL and LCL: Gradient is used only if visualizers are drawn using GDI+ (in Delphi XE2+ and optionally in Lazarus).

Default value

True

1.8.11.1.4 TRVReportSignalStrengthVisualizer.LineUsesFillColor

Specifies how bars (or wedges) are outlined.

```
property LineUsesFillColor: Boolean;
```

If **LineUsesFillColor** = *False*:

Full bars are outlined with LineColor[193], empty bars are outlined with LineColorEmpty[213].

If **LineUsesFillColor** = *True*:

If the corresponding fill color is equal to *rvclNone*, LineColor[193]/LineColorEmpty[213] is used. Otherwise, if Gradient[213] = *True*, bars are outlined with Color[193]/ColorEmpty[213]; if Gradient[213] = *False*, they are outlined with twice darker colors.

Default value

True

1.8.11.1.5 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) or in EMU, depending on Units property of TRVStyle component.

Default value

0

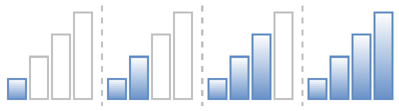
1.8.11.1.6 TRVReportSignalStrengthVisualizer.PartsCount

A count of bars (or wedges) in each diagram displayed by the visualizer.

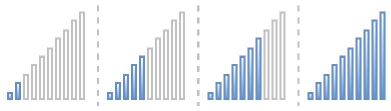
```
property PartsCount: Integer;
```

Examples

PartsCount = 4



PartsCount = 10



Default value

4

1.8.11.1.7 TRVReportSignalStrengthVisualizer.Spacing

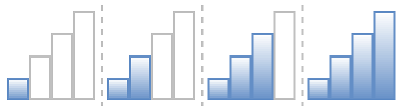
A distance between bars (or wedges)

```
property Spacing: TRVStyleLength;
```

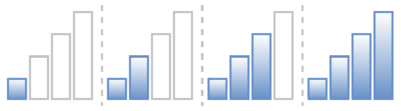
This value is measured in screen pixels or in twips (1/20 of point) or in EMU, depending on Units property of TRVStyle component.

Examples

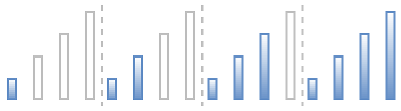
Spacing = 0



Spacing = 1



Spacing = 8



Default value

1

1.9 Actions

VCL and Lazarus versions of Report Workshop include a set of actions for editing report templates. They can be used in addition to other RichViewActions.

Actions related to the whole report

TrvrActionReportWizard [227] generates a new master-detail report template basing on available data.

TrvrActionDocProperties [218] edits report-related document properties.

Actions for editing properties of report tables

TrvrActionRowGenerationRules [231] edits row generation rules [126].

TrvrActionCrossTab [217] edits a cross tabulation [126].

TrvrActionCellProperties [216] edits report table cell [150] properties, as well as BackgroundVisualizers [125] and BackgroundColorChangers [125] collections

The actions above may be applied to normal tables: they convert them to report tables (after a confirmation)

Actions related to report tables

TrvrActionConvertToReportTable[217] converts the selected normal table to a report table.

TrvrActionInsertTable[224] creates a new a report table (either a blank table or a table showing the a data table chosen by the user).

Note: additionally, the actions for editing report table properties may convert normal tables to report tables.

Other actions

TrvrActionInsertShape[219] inserts a geometric shape into the editor.

1.9.1 TrvrActionCellProperties

TrvrActionCellProperties is an action for editing report table cell[150] properties.

Unit RVReportActions;

Syntax

```
TrvrActionCellProperties =
  class (TrvrCustomActionReportTableWithDataProvider[233]);
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionTableCell
TrvrCustomActionReportTable[232]
TrvrCustomActionReportTableWithDataProvider[232]

Description

This action displays a dialog window for editing properties of report table cell[150] 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[152]
- VisualizerId[153]
- ColorChangerId[151]

Additionally, it edits BackgroundVisualizers[125] and BackgroundColorChangers[125] properties of the table.

If DataProvider[233] property of this action is assigned, the dialog displays lists of tables and fields to help editing data queries associated with the cells.

1.9.2 TrvrActionConvertToReportTable

TrvrActionConvertToReportTable converts the selected normal table to a report table[124].

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[124].

This action is redundant: the actions for editing report table properties may convert normal tables to report tables themselves.

1.9.3 TrvrActionCrossTab

TrvrActionCrossTab is an action for editing a cross tabulation[126] for the selected report table[124].

Unit RVReportActions;

Syntax

```
TrvrActionCrossTab = class (TrvrCustomActionReportTableWithDataProvider[233]);
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionTableCell

TrvrCustomActionReportTable [232]
TrvrCustomActionReportTableWithDataProvider [232]

Description

This action displays a dialog window for editing cross tabulation [126] for the report table [124] 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 [231].

If DataProvider [233] property of this action is assigned, the dialog displays lists of tables and fields to help editing data queries for column generation.

1.9.4 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 [247] item in DocObjects property of the target editor:

- DataQuery [249]
- PageBreaksBetweenCopies [250]

1.9.4.1 Properties

In TrvrActionDocProperties

🟩 DataProvider [219]

1.9.4.1.1 TrvrActionDocProperties.DataProvider

Allows linking the action with a data provider.

```
property DataProvider: TRVReportDataProvider[92];
```

This data provider is used to get a list of tables and fields, to help users writing data queries.

1.9.5 TrvrActionInsertShape

TrvrActionInsertShape is an action for inserting a geometric shape[169] at the caret position of the target editor.

This action is not related to reporting, it simply reuses shape drawing functions implemented for value visualizers[174].

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[223] = *False*, the specified shape is simply inserted in the editor.

If UserInterface[223] = *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.

1.9.5.1 Properties

In TrvrActionInsertShape

1.9.5.1.1 TrvrActionInsertShape.BackgroundColor, BackgroundOpacity

Specify the item background color and opacity

```
property BackgroundColor: TRVColor;
property BackgroundOpacity: TRVOpacity;
```

These properties are assigned to BackgroundColor and BackgroundOpacity 171 of the inserted item.

Default value

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

1.9.5.1.2 TrvrActionInsertShape.BorderColor

Color of border around the item.

```
property BorderColor: TRVColor;
```

This property is assigned to BorderColor of the inserted item.

Default value

rvclBlack

1.9.5.1.3 TrvrActionInsertShape.BorderWidth

Width of border around the item, in pixels, twips or EMU (depending on ControlPanel.UnitsProgram)

```
property BorderWidth: TRVStyleLength;
```

This property is assigned to BorderWidth of the inserted item. A conversion of measure units is performed, if necessary.

Default value

0

1.9.5.1.4 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.

1.9.5.1.5 TrvrActionInsertShape.Color, Opacity

Fill color and opacity for the shape.

```
property Color: TRVColor;
property Opacity: TRVOpacity;
```

These properties are assigned to Color and Opacity [171] of the inserted item.

Default values

- Color: $C68E63 for VCL and LCL; $FF638EC6 for FMX
- Opacity: 100000 (i.e. 100%)

1.9.5.1.6 TrvrActionInsertShape.EqualSides

Specifies whether the shape has the same width and height.

```
property EqualSides: Boolean;
```

This property is assigned to EqualSides [171] of the inserted item.

Default value

True

1.9.5.1.7 TrvrActionInsertShape.GradientType

Type of gradient fill for shapes.

```
property GradientType: TRVGradientType[233];
```

This property is assigned to GradientType [172] of the inserted item.

Default value

rvgtNone

1.9.5.1.8 TrvrActionInsertShape.LineColor

Line color

```
property LineColor: TRVColor;
```

This property is assigned to LineColor[172] of the inserted item.

Default value

rvclBlack

1.9.5.1.9 TrvrActionInsertShape.LineUsesFillColor

Specifies how shapes are outlined.

```
property LineUsesFillColor: Boolean;
```

This property is assigned to LineUsesFillColor[172] of the inserted item.

Default value

False

1.9.5.1.10 TrvrActionInsertShape.LineWidth

Line width, in pixels, twips or EMU (depending on ControlPanel.UnitsProgram)

```
property LineWidth: TRVStyleLength;
```

This property is assigned to LineWidth[173] of the inserted item. A conversion of measure units is performed, if necessary.

Default value

1

1.9.5.1.11 TrvrActionInsertShape.OuterHSpacing, OuterVSpacing

Horizontal and vertical space around the item (around the border, if it exists), in pixels, twips or EMU (depending on ControlPanel.UnitsProgram)

```
property OuterHSpacing: TRVStyleLength;
property OuterVSpacing: TRVStyleLength;
```

This property is assigned to OuterHSpacing and OuterVSpacing of the inserted item. A conversion of measure units performed, if necessary.

Default value

0

1.9.5.1.12 TrvrActionInsertShape.ShapeProperties

Specifies main properties of a shape.

```
property ShapeProperties: TRVReportShapeProperties[254];
```

This property is assigned to ShapeProperties[173] of the inserted item.

1.9.5.1.13 TrvrActionInsertShape.StartColor

Start color for gradient fill.

```
property StartColor: TRVColor;
```

This property is assigned to StartColor [173] of the inserted item.

Default value

rvclWhite

1.9.5.1.14 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 [223] method. If the action was initiated by a button, the window is displayed below this button (for old version of Delphi, use CallerControl [221]), otherwise it is displayed at the position of the mouse pointer.

Default value

True

1.9.5.1.15 TrvrActionInsertShape.Width, Height

Shape size, in pixels, twips or EMU (depending on ControlPanel.UnitsProgram)

```
property Width: TRVStyleLength;
property Height: TRVStyleLength;
```

These properties are assigned to Width and Height [173] of the inserted item. A conversion of measure units is performed, if necessary.

Default value

48

1.9.5.2 Methods

In TrvrActionInsertShape

ShowInsertDialog [223]

1.9.5.2.1 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.

1.9.6 TrvrActionInsertTable

TrvrActionInsertTable is an action for inserting a report table[124] at the caret position of the target editor.

Unit RVReportActions;

Syntax

```
TrvrActionInsertTable = class (TrvActionInsertTable)
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionInsertTable

Description

Inserting a blank table

By default, this action works exactly like RichViewActions' TrvActionInsertTable, but inserts TRVReportTableItemInfo[124] instead of TRVTableItemInfo.

In this mode, this action is redundant: you can use TrvActionInsertTable instead, and the actions for editing report table properties may convert normal tables to report tables.

Generating from a data table

If you assign DataProvider[225] property, this action works in a different mode: generating a report table by a data table (i.e., from a dataset). The action displays a dialog window where the user can:

- select a data table (from a list of data tables returned by DataProvider)[225]
- choose which fields of this data table will be included in a report table (see OnIsIDFieldName[227] event)
- rearrange these fields
- define the report table heading lines.

Each column in the resulting report table corresponds to a chosen data field.

The table has from 1 to 3 rows (1 data row and up to 2 heading rows).

Heading rows

The first heading row is added if the user types a table heading text. This row has a single cell that occupies all columns.

The second optional heading row contains names of fields.

If the target editor uses style templates, HeadingTextStyleTemplateName and HeadingParaStyleTemplateName[226] are applied to cells in heading rows.

HeadingRowColor property is used only if the table has heading row(s).

Data row

The action adds a row generation rule[126] to the table. The rule name is equal to the name of the selected table.

The last row contains field codes[27]. They can be short form ({FIELDNAME}) or in the full form ({RULENAME:FIELDNAME}) depending on the value of FullFieldNames[226] property.

1.9.6.1 Properties

In TrvrActionInsertTable

- DataProvider[225]
- FullFieldNames[226]
- HeadingParaStyleTemplateName[226]
- HeadingTextStyleTemplateName[226]
- UseDataTables[226]

1.9.6.1.1 TrvrActionInsertTable.DataProvider

A data provider that is used to generate a new report table.

```
property DataProvider: TRVReportDataProvider[92];
```

This property is used only if UseDataTables[226] = *True*.

If **DataProvider** is defined, and it can list available tables[94], and if there is at least one data table, this action shows a special dialog to create a report table basing on the selected data table. The user can choose which fields to include in a report table.

Otherwise, a standard table insertion dialog is used to insert a blank report table.

1.9.6.1.2 TrvrActionInsertTable.FullFieldNames

Specifies whether to use the full or the short field names in field codes.

```
property FullFieldNames: Boolean;
```

If *True*: fields are inserted as {ROWGENERATIONRULE:FIELDNAME}

If *False*: fields are inserted as {FIELDNAME}

Default value:

True

See also:

- Data field format [27]

1.9.6.1.3 TrvrActionInsertTable.HeadingTextStyleTemplateName, HeadingParaStyleTemplateName

Specify names of style templates for applying to cells of heading rows.

```
property HeadingTextStyleTemplateName: TRVStyleTemplateName;
property HeadingParaStyleTemplateName: TRVStyleTemplateName;
```

Heading rows can be added if DataProvider [225] is defined and UseDataTables [226] = *True*.

These properties are used only if the target editor's UseStyleTemplates property = *True*.

HeadingParaStyleTemplateName is applied to paragraphs.

HeadingTextStyleTemplateName is applied to text.

Default value:

- HeadingTextStyleTemplateName: 'Strong'
- HeadingParaStyleTemplateName: ''

1.9.6.1.4 TrvrActionInsertTable.UseDataTables

Allows using DataProvider [225] to get a list of tables and their fields.

```
property UseDataTables: Boolean;
```

If *True*, and DataProvider [225] is defined, the action displays a dialog for creating a report table basing on a data table.

Otherwise, the action displays the standard table insertion dialog to insert a blank table.

Default value:

True

1.9.6.2 Events

In TrvrActionInsertTable

- OnIsIDFieldName [227]

1.9.6.2.1 TrvrActionInsertTable.OnIsIDFieldName

The event where the user can identify "id" fields.

```
type
  TRVIsIDFieldNameEvent = procedure (Sender: TObject;
    const FieldName: TRVUnicodeString;
    var IsID: Boolean) of object;
```

```
property OnIsIDFieldName: TRVIsIDFieldNameEvent;
```

When a list of fields is displayed for inclusion in a report table, all fields are selected by default, except for "id" fields (because normally they should not be shown to end-users).

By default, the action treats as "id" fields all fields with names that end with 'id' or 'code' (case insensitive). You can modify this behavior in this event.

Input parameters:

FieldName – field name.

Output parameters:

IsID – "is this an id-field?"

1.9.7 TrvrActionReportWizard

TrvrActionReportWizard generates a new master-detail report template.

Unit RVReportActions;

Syntax

```
TrvrActionReportWizard = class (TrvAction)
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction

Description

Report wizard

The action displays a wizard dialog to generate a new master-detail report template.

In this dialog, the user selects a master data table, chooses fields of this data table, and how they are displayed. Then the process is repeated for a detail table, then for a detail table of this detail table, and so on. The wizard finishes when:

- the user chooses to finish

- there are no more detail tables
- at the last level, the user chose a way of data representation that does not support details.

The resulting report template can have any number of master-detail levels; but there can be only one detail at each level. This limitation comes from the step-by-step nature of a wizard dialog.

Data tables (datasets)

Data for a report template are provided by DataProvider 230 (required).

The following types of data providers can be used:

- data providers with predefined master-detail relationships (such as TRVReportDBDataProvider 94);
- data providers that support SQL data queries (see the list of data providers 80); see also AllowSelfReferences 230 ;
- TRVReportDbfDataProvider 97 for Lazarus

Clearing the editor

The action clears the target editor and resets its properties before the report template generation. It uses ActionNew 229 for this task (required).

Visual appearance

If the report template contains tables, their appearance is defined by ActionInsertTable 229 and FullWidthTables 230 properties.

If the target editor's **UseStyleTemplates** = *True*, appearance of text and paragraph is defined by style templates (otherwise, the action uses Style.TextStyles[0] and Style.ParaStyles[0] of the target editor for all text).

Depending on the value of UseCurrentStyleTemplates 231 , the action uses either style templates of the target editor, or ActionNew 229 .StyleTemplates.

Some names of used style templates are hard-coded ('Normal', 'heading 1', 'List Paragraph'), some names are defined in properties. See the topic about AccentedTextStyleTemplateName, ReportFooterStyleTemplateName 229 for additional information.

Tables can have either 100% width, or a fixed size that depends on the current width of the target editor window, see FullWidthTables 230 .

1.9.7.1 Properties

In TrvrActionReportWizard

- AccentedTextStyleTemplateName 229
- ActionInsertTable 229
- ActionNew 229
- AllowSelfReferences 230
- DataProvider 230
- FullWidthTables 230
- ReportFooterStyleTemplateName 229
- StartFromAllDataSets 230
- UseCurrentStyleTemplates 231

1.9.7.1.1 TrvrActionReportWizard.AccentedTextStyleTemplateName, ReportFooterStyleTemplateName

Specify names of style templates for report generation.

```
property AccentedTextStyleTemplateName: TRVStyleTemplateName;
property ReportFooterStyleTemplateName: TRVStyleTemplateName;
```

These properties are used only if the target editor's UseStyleTemplates property = *True*.

AccentedTextStyleTemplateName is applied to emphasized text.

ReportFooterStyleTemplateName is applied to report footer's paragraph.

The action use the following style templates:

- 'heading 1' for report title (paragraph)
- **ReportFooterStyleTemplateName** for report footer (paragraph)
- 'Normal' for paragraphs of normal text (paragraph)
- 'List Paragraph' for paragraphs of text that should be compact (paragraph)
- **AccentedTextStyleTemplateName** for emphasized text (text)
- ActionInsertTable 229 .HeadingTextStyleTemplate 226 for heading rows of tables (text)
- ActionInsertTable 229 .HeadingParaStyleTemplate 226 for heading rows of tables (paragraph)

Default value:

- AccentedTextStyleTemplateName: 'Strong'
- ReportFooterStyleTemplateName: 'Normal'

See also:

- UseCurrentStyleTemplates 231

1.9.7.1.2 TrvrActionReportWizard.ActionInsertTable

A link to the action that defines appearance of report tables 124 generated by this action.

```
property ActionInsertTable: TrvrActionInsertTable 224 ;
```

If this property is not defined, this action searches for the first TrvrActionInsertTable action on the same form/datamodule, and if it is found, uses it.

ActionInsertTable is optional: if it is not linked, generated report tables have default appearance.

ActionInsertTable.BestWidth is ignored.

1.9.7.1.3 TrvrActionReportWizard.ActionNew

A link to the action that is used to clear the target editor and to reset its properties (before the report template generation).

```
property ActionNew: TrvActionNew;
```

If this property is not defined, this action searches for the first TrvActionNew action on the same form/datamodule, and if it is found, uses it.

ActionNew is requires: if it is not linked, the action cannot generate a report template.

If UseCurrentStyleTemplates 231 = *True*, ActionNew.StyleTemplates are not used.

1.9.7.1.4 TrvrActionReportWizard.AllowSelfReferences

Specifies whether the report wizard supports self-referential tables.

```
property AllowSelfReferences: Boolean;
```

This property specifies how the report wizard works for data providers that do not support predefined master-detail relationships (i.e. for data providers that generate SQL queries).

If **AllowSelfReferences** = *False:* the wizard does not allow to specify the same table as a master and as a detail; the wizard does not allow choosing tables that were already chosen on master levels.

If **AllowSelfReferences** = *True:* the wizard allows choosing any table on any step.

Note: if DataProvider[230] supports predefined master-detail relationships, the action always work as if **AllowSelfReferences** = *False*.

Default value:

False

1.9.7.1.5 TrvrActionReportWizard.DataProvider

A data provider that is used to generate a new report.

```
property DataProvider: TRVReportDataProvider[92];
```

This property is equired: without a data provider, this action cannot generate a report template.

he following types of data providers can be used:

- data providers with predefined master-detail relationships (such as TRVReportDBDataProvider[94])
- data providers that support SQL data queries (see the list of data providers[80]).

See also:

- StartFromAllDataSets[230]

1.9.7.1.6 TrvrActionReportWizard.FullWidthTables

Specifies width of generated report tables[124].

```
property FullWidthTables: Boolean;
```

This property is used instead of ActionInsertTable[229].BestWidth.

If **FullWidthTables** = *True*: -100 is assigned (100% width)

If **FullWidthTables** = *False*: 0 is assigned (widths of tables is calculated basing on widths of their columns; the resulting table has width approximately equal to the client width of the target editor)

Default value:

True

1.9.7.1.7 TrvrActionReportWizard.StartFromAllDataSets

Specifies how the report template generation wizards starts.

```
property StartFromAllDataSets: Boolean;
```

This property is used only if DataProvider[230] uses data tables with predefined master-detail relationship (an example of such a dataprovider is TRVReportDBDataProvider[94]: it uses MasterSource properties of DataSets)

If **StartFromAllDataSets** = *False:* at the first page of the report wizard, the end-user can choose only a data table that is not detail table.

If **StartFromAllDataSets** = *True:* the end-user start from any available table.

Default value:

False

1.9.7.1.8 TrvrActionReportWizard.UseCurrentStyleTemplates

Specify whether the target editor's StyleTemplates must be reset before the report template generation.

```
property UseCurrentStyleTemplates: Boolean;
```

This property is used only if the target editor's UseStyleTemplates property = *True*.

If **UseCurrentStyleTemplates** = *True:* the report is generated using the current values of StyleTemplates collection of the target editor.

If **UseCurrentStyleTemplates** = *False:* StyleTemplates of the target editor are replaced with ActionNew[229].StyleTemplates (if ActionNew[229].ApplyStyleTemplates = *True*).

Default value:

False

See also:

- AccentedTextStyleTemplateName, ReportFooterStyleTemplateName[229]

1.9.8 TrvrActionRowGenerationRules

TrvrActionRowGenerationRules is an action for editing row generation rules[126] for the selected report table[124].

Unit RVReportActions;

Syntax

```
TrvrActionRowGenerationRules =
  class (TrvrCustomActionReportTableWithDataProvider[233]);
```

Hierarchy

TObject
TPersistent
TComponent
TBasicAction
TContainedAction
TCustomAction
TAction
TrvCustomAction
TrvAction
TrvActionTableCell
TrvrCustomActionReportTable[232]
TrvrCustomActionReportTableWithDataProvider[232]

Description

This action displays a dialog window for editing row generation rules[126] for the report table[124] selected in the target editor.

This action can also be applied to normal tables (it may convert them to report tables).

If DataProvider[233] property of this action is assigned, the dialog displays lists of tables and fields to help editing data queries for row generation.

1.9.9 TrvrCustomActionReportTable

Basic reporting actions for editing properties of report tables[124].

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[231]
- TrvrActionCrossTab[217]
- TrvrActionCellProperties[216]

TrvrCustomActionReportTableWithDataProvider introduces DataProvider[233] property.

1.9.9.1 Properties

In TrvrCustomActionReportTableWithDataProvider

🟩 DataProvider[233]

1.9.9.1.1 TrvrCustomActionReportTableWithDataProvider.DataProvider

Allows linking the action with a data provider.

```
property DataProvider: TRVReportDataProvider[92];
```

This data provider is used to get a list of tables and fields, to help users writing data queries.

1.10 Types

Other Types Declared in ReportWorkshop

- TRVGradientType[233] – a gradient fill type (used in shapes)
- TRVReportField[234] – a field identifier in query processors
- TRVReportFieldType[234] – a type of field in query processors
- TRVReportRecordCountMode[235] specifies how a record count is used in query processors
- TRVReportShape[236] – a shape type
- TRVValueVisualizerId[240] – an identifier of a value visualizer[174]

See also

- Definitions of Report Workshop terms[11]

1.10.1 TRVAControlPanel

VCL and LCL

TRVAControlPanel is a special component defined in RichViewActions unit.

FMX

Unit fmxRVReportGenerator;

Syntax

```
type
  TRVAControlPanel = TCustomRVReportGenerator[61];
```

This is a temporary solution to provide UI localization in FireMonkey version of ReportWorkshop.

See also

- TCustomRVReportGenerator[61].Language[63]

1.10.2 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

1.10.3 TRVReportField

Identifies a data field in TRVReportQueryProcessor[252].

Unit RVReportDataProvider;

Syntax

```
type
  TRVReportField = Pointer;
```

1.10.4 TRVReportFieldType

Identifies a type of a data field of TRVReportQueryProcessor[252].

Unit RVReportDataProvider;

Syntax

```
type
 TRVReportFieldType = (rvrftUnknown,
    rvrftText, rvrftInteger, rvrftFloat, rvrftBoolean,
    rvrftDateTime,
      rvrftDate,
      rvrftTime,
    rvrftBlob,
      rvrftAnsiMemo,
      rvrftUnicodeMemo,
      rvrftDocument,
        rvrftRTF,
        rvrftDocX,
        rvrftRVF,
        rvrftMarkdown,
      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[53]

1.10.5 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 126 .CopyColCount 165 > 1, or for cross-tab reports 126); 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 68 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.

1.10.6 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
    );
```

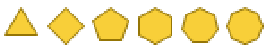
A shape of each type is defined by the following parameters:

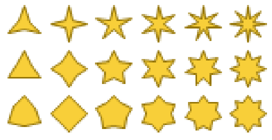
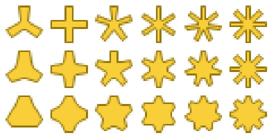
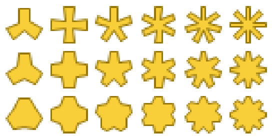
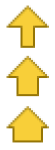
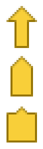
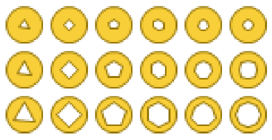
- diameter of an imaginary circle surrounding the shape
- diameter of an imaginary internal circle
- count of points

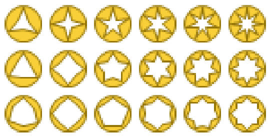
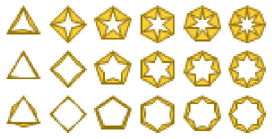
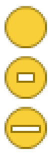
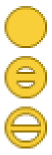
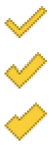
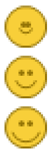
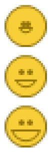
In the examples below, the same shape is displayed with different parameters. Horizontally, the count of points is increased from 3 to 7. Vertically, shapes are displayed with internal radii 30%, 50%, 70% of the external radius.

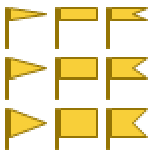
Note: the shapes may have unequal width and height. In this case, circles become ellipses, squares become rectangles, etc.

All shapes are shown unrotated.

Value	Meaning	Rotatabl e	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.		

1.10.7 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[194] property of value visualizers[191] and color changers[184]; this property is an unique identifier of items in their collections;
- VisualizerId[153] property of report table cell[150]; this property links the cell with the item of the report table[124]'s BackgroundVisualizers[125] collection;
- ColorChangerId[151] property of report table cell[150]; this property links the cell with the item of the report table[124]'s BackgroundColorChangers[125] collection.

1.11 Constants

Constants in ReportWorkshop

- rveipc*** constants[240] identify properties TRVShapeItemInfo[169]

1.11.1 rveipc*** 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[169]

These properties are defined in RVReportShapeItem unit.

Constant	Value	Property
rveipcShapeColor	200	Color[171]
rveipcStartShapeColor	201	StartColor[173]
rveipcLineColor	202	LineColor[172]
rveipcLineWidth	203	LineWidth[173]
rveipcLineUsesFillColor	204	LineUsesFillColor[172]
rveipcEqualSides	205	EqualSides[171] *

rveipcGradientType	206	GradientType[172]*
rveipcBackgroundOpacity	207	BackgroundOpacity[171]
rveipcShapeOpacity	208	Opacity[171]
rveipcShapeType	209	ShapeProperties.Shape[255]*
rveipcPointCount	210	ShapeProperties.PointCount[255]
rveipcMiddlePercent	211	ShapeProperties.MiddlePercent[255]
rveipcStartAngle	212	ShapeProperties.StartAngle[255] **

* the property is type-casted to Integer

** the property is multiplied by 100 and rounded

1.12 Procedures and functions

Procedures for report generation

GenerateReport[241] generates a report in ScaleRichView.

Procedures for localization

RVReportGetErrorString[242] returns a localized error message for the specified error.

RWA_LocalizeErrorMessages[242] initializes data necessary for RVReportGetErrorString procedure.

RWA_LocalizeReportGenerator[243] localize text properties of TRVReportGenerator[79] component.

1.12.1 GenerateReport

Generates error for ScaleRichView editor **Editor**.

Unit RVReportSRVGenerator;

Syntax

```
function GenerateReport(Editor: TSRichViewEdit;
 Generator: TRVReportGenerator[79];
 IncludeInvisibleHeaders: Boolean = False;
 UseThread: Boolean = False): Boolean;
```

This method calls **Generator**.Execute[66] for **Editor**.SubDocuments[] (headers and footers) and then for Editor.**RichViewEdit** (main document).

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.

Headers and footers are always processed in the context of the main process. The main document can be processed in a background thread, if **UseThread** = *True*.

Generator.OnGenerated[78] event is called only for the last generation (for the main document).

Return value:

False, if at least one call of **Generator**.Execute[66] returns *False*.

1.12.2 RVReportGetErrorString

Returns a localized error message for the specified error code.

Unit RVReportErrors;

Syntax

```
procedure RVReportGetErrorString(ErrorCode: TRVReportGeneratorError[104];
  out ErrTypeStr, ErrStr: TRVUnicodeString; RelatedObject: TObject;
  ErrorTypes: PRVReportGeneratorErrorTypeStr = nil;
  ErrorMessages: PRVReportGeneratorErrorStr = nil;
  ParserErrorMessages: PRVReportParserErrorStr = nil;
  ExprEvalErrorMessages: PRVReportExprEvalErrorStr = nil
  );
```

This procedure can be used in TRVReportGenerator[79].OnError[70] event. Pass **ErrorCode** and **RelatedObject** parameters of this event as an input parameters of this procedure.

There are two output string parameters: **ErrTypeStr** and **ErrStr**.

ErrTypeStr receives a string that describes a type of this error (e.g. like 'Variable error' or 'Cross-tabulation error').

ErrStr receives a string that describes the specific error. This is usually a format string. To get a final string, call Format(**ErrStr**, [**RelatedText**]), where **RelatedText** is a parameter of OnError event.

The rest of parameters are pointers to arrays of error messages. Normally, you can omit them, and default global variables will be used. Before using these arrays (including default global arrays), you need to initialize them using RWA_LocalizeErrorMessages[242].

See the example in the topic about TRVReportGenerator[79].OnError[70].

1.12.3 RWA_LocalizeErrorMessages

Localizes error messages.

Unit RVReportLocalize;

Syntax

```
procedure RWA_LocalizeErrorMessages(
  ControlPanel: TRVAControlPanel[233] = nil;
  ErrorTypes: PRVReportGeneratorErrorTypeStr = nil;
  ErrorMessages: PRVReportGeneratorErrorStr = nil;
  ParserErrorMessages: PRVReportParserErrorStr = nil;
  ExprEvalErrorMessages: PRVReportExprEvalErrorStr = nil
  );
```

This procedure localizes error messages used in Report Workshop according to **ControlPanel**.Language. If **ControlPanel** parameter is omitted, the default control panel is used.

Call this procedure when this application starts, and after each language change (i.e after each call of RVA_ChooseLanguage).

The rest of parameters are pointers to arrays of error messages. Normally, you can omit them, and default global variables will be used.

Localized error messages are used by RVReportGetErrorString[242] procedure.

See also

- RWA_LocalizeReportGenerator[243]

1.12.4 RWA_LocalizeReportGenerator

Localizes text properties of **Generator**.

Unit RVReportLocalize;

Syntax

```
procedure RWA_LocalizeReportGenerator(Generator: TRVReportGenerator[79];
  ControlPanel: TRVAControlPanel[233] = nil);
```

This procedure localizes **Generator**.Texts[64] according to **ControlPanel**.Language. If **ControlPanel** parameter is omitted, the default control panel is used.

Call this procedure when this application starts, and after each language change (i.e after each call of RVA_ChooseLanguage).

See also

- RWA_LocalizeErrorMessages[242]

1.13 Classes

Other Classes in Report Workshop

Properties of Value Visualizers:

- TRVConditionalShapePropertiesCollection[244] – collection of TRVConditionalShapePropertiesItem[244] items; a collection of conditional shape properties;
- TRVReportShapeProperties[254] – shape properties.

Query processors:

- TRVReportQueryProcessor[252] – basic query processor;
- TRVReportDBQueryProcessor[246] – db-related query processor.
- TRVReportBindSourceAdapterQueryProcessor[246] – LiveBindings-related query processor.

Report template properties:

- TRVReportDocObject 247 – report template properties, can be stored in RVF documents.

Report generation:

- TRVReportGenerationSession 251 stores information about a report generation session.

1.13.1 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 244.

Value 246 properties of all items 244 must be assigned, and items must be sorted by Value 246 in ascending order.

1.13.1.1 Properties

In TRVConditionalShapePropertiesCollection

Items 244

Inherited from TCollection

▶ Count

1.13.1.1.1 TRVConditionalShapePropertiesCollection.Items

Lists the items in the collection.

```
property Items[Index: Integer]: TRVConditionalShapePropertiesItem 244;
  default;
```

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

1.13.2 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 244 .

This item contains the following shape properties:

- Color 245
- DeltaAngle 245

Additionally, it has Value 246 property. An owner of this item's collection compares a visualized value with Value 246 property of all items in the collection to find the proper item.

1.13.2.1 Properties

In TRVConditionalShapePropertiesItem

🟩 Color 245
🟩 DeltaAngle 245
🟩 Value 246

1.13.2.1.1 TRVConditionalShapePropertiesItem.Color

Shape fill color

```
property Color: TRVColor;
```

Default value

rvclWhite

1.13.2.1.2 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

1.13.2.1.3 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)

1.13.3 TRVReportBindSourceAdapterQueryProcessor

The class of an object processing a database-related data query.

unit RVReportBindSourceDataProvider;

```
TRVReportTValueQueryProcessor = class (TRVReportQueryProcessor[252]);
TRVReportBindSourceAdapterQueryProcessor =
  class (TRVReportTValueQueryProcessor);
```

Hierarchy

TObject
TRVReportQueryProcessor[252]
TRVReportTValueQueryProcessor

Description

This class is a wrapper for TBindSourceAdapter-based components.

It provides data of the following types: String, integer and floating point values, Boolean, TDate, TTime, TDateTime, bitmaps (from TBitmap), memos (from TStringList).

This query processor is created by TRVReportBindSourceDataProvider[86] component.

Normally, this class is used internally in TRVReportGenerator[79] 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[79] event.

See also

- Definitions of Report Workshop terms[11]
- TRVReportGenerationSession[251].GetQueryProcessor[252]

1.13.4 TRVReportDBQueryProcessor

The class of an object processing a database-related data query.

unit RVReportDBDataProvider;

```
TRVReportDBQueryProcessor = class (TRVReportQueryProcessor[252]);
```

Hierarchy

TObject
TRVReportQueryProcessor[252]

Description

This class is a wrapper for TDataSet-based components processing database-related data queries, normally, SQL SELECT statements.

Normally, this class is used internally in TRVReportGenerator[79] 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[79] event.

See also

- Definitions of Report Workshop terms[11]
- TRVReportGenerationSession[251].GetQueryProcessor[252]

1.13.5 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);

function RVReportGetPageBreaksBetweenCopies(
  rv: TCustomRichView): Boolean;
procedure RVReportSetPageBreaksBetweenCopies(
  rv: TCustomRichView; Value: Boolean; EditMode: Boolean);

type
  TRVReportScript = (rvrsOnStart, rvrsAfterRecord, rvrsOnEnd);
function RVReportGetScript(rv: TCustomRichView;
  Script: TRVReportScript): TStrings;
procedure RVReportSetScript(rv: TCustomRichView;
  Script: TRVReportScript;
  Value: TStrings; EditMode: Boolean);
procedure RVReportSetScriptAsString(rv: TCustomRichView;
  Script: TRVReportScript;
  const ScriptText: TRVUnicodeString; EditMode: 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 249 property of the first TRVReportDocObject item in **rv**.DocObjects, or empty string if not found.

RVReportSetRootDataQuery assigns a new value to the DataQuery 249 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 249 property,
- **RVReportGetPageBreaksBetweenCopies** and **RVReportSetPageBreaksBetweenCopies** provide access to PageBreaksBetweenCopies 250 property.
- **RVReportGetScript, RVReportSetScript, RVReportSetScriptAsString** provide access to Script_OnStart, Script_AfterRecord, Script_OnEnd 250 properties. **RVReportGetScript** returns a reference to the property, do not free it. **RVReportSetScript** copies the Value parameter to the property.

1.13.5.1 Properties

In TRVReportDocObject

- DataQuery 249
- HighlightRules 249
- PageBreaksBetweenCopies 250
- Script_AfterRecord 250
- Script_BeforeRecord 250
- Script_OnEnd 250
- Script_OnStart 250

1.13.5.1.1 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[79] creates a query processor[252] for this data query. This query processor returns data for this query, and the report generator replicates the 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.

By default, the whole content of TRichView is replicated. To replicate a fragment instead, use {$HEADER} and {$FOOTER} commands[32].

TRVReportGenerator[79] starts processing this data query before any other data queries.

TRVReportGenerator[79] processes this property only for the first object of TRVReportDocObject class in TRichView.DocObjects collection.

Default value:

'' (empty string)

See also

- RVReportGetRootDataQuery[247] function
- RVReportSetRootDataQuery[247] procedure

See also

- Definitions of Report Workshop terms[11]
- Information about data queries[13]
- PafeBreaksBetweenCopies[250]
- TRVRowGenerationCustomRule[155].DataQuery[156]
- TRVReportTableCellData[150].DataQuery[152]
- TRVCrossTabLevel[140].DataQuery[143]

1.13.5.1.2 TRVReportDocObject.HighlightRules

Allows/disallows highlighting in all report tables[124].

```
property HighlightRules: Boolean;
```

You can highlight:

- report cells using TRVReportTableCellData[152].HighlightColor[152] (if they have a data query associated)
- row generation rules using TRVRowGenerationCustomRule[155].HighlightColor[157]
- cross-tab header using TRVCrossTab[130].HighlightColor[138]

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 [247] function
- RVReportSetHighlightRules [247] procedure

1.13.5.1.3 TRVReportDocObject.PageBreaksBetweenCopies

Specifies whether the report generator must add page breaks between copies of data generated when applying DataQuery [249].

```
property PageBreaksBetweenCopies: Boolean;
```

This property is used only if DataQuery [249] is not empty.

Default value:

False

See also

- RVReportGetPageBreaksBetweenCopies [247] function
- RVReportSetPageBreaksBetweenCopies [247] procedure

1.13.5.1.4 TRVReportDocObject's Scripts

Scripts [15] that are executed when a report is generated.

```
property Script_OnStart: TStrings;
property Script_BeforeRecord: TStrings;
property Script_AfterRecord: TStrings;
property Script_OnEnd: TStrings;
```

If DataQuery [249] is not empty:

- Script_OnStart is executed at the beginning of report generation, just after the DataQuery is executed.
- Script_BeforeRecord is executed before processing each record of DataQuery results.
- Script_AfterRecord is executed after processing each record of DataQuery results.
- Script_OnEnd is executed at the end of report generation.

If DataQuery [249] is empty:

- Script_OnStart is executed at the beginning of report generation
- Script_OnEnd is executed at the end of report generation.

For empty scripts, these properties may return *nil*.

Assignment to these properties copies TString object.

Default value:

Nil

See also

- RVReportGetScript, RVReportSetScript, RVReportSetScriptAsString [247] functions and procedures
- scripts associated with table generation rules [168]

1.13.6 TRVReportGenerationSession

The class of an object containing information about a report generation session. It can be used to get values of variables [30] and data fields [27].

Unit RVReportGenerator;

Syntax

```
TRVReportGenerationSession = class (TRVList)
```

Hierarchy

TObject
TList
TRVList

Description

Methods:

- GetQueryProcessor [252] returns a query processor; it can be used to get values of data fields [27].
- GetVariableValue [251] returns a value of a variable [30].

See also

- Definitions of Report Workshop terms [11]

1.13.6.1 Methods

In TRVReportGenerationSession

GetQueryProcessor [252]
GetVariableValue [251]

1.13.6.1.1 TRVReportGenerationSession.GetVariableValue

Returns a value and an associated object for the given variable [30].

```
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 [127] of the most deeply nested report table, then in the table containing it, and so on. Finally, it looks in the TRVReportGenerator's Variables [65].

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 11

1.13.6.1.2 TRVReportGenerationSession.GetQueryProcessor

Returns a query processor identified by **Name**.

```
function GetQueryProcessor(const Name: TRVUnicodeString):
  TRVReportQueryProcessor 252;
```

Input parameter

Name can be one of the following values:

- *table row generation rule name* 157. A data query 156 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 11

1.13.7 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 79 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 79 event.

See also

- Definitions of Report Workshop terms 11
- TRVReportGenerationSession 251.GetQueryProcessor 252

1.13.7.1 Methods

In TRVReportQueryProcessor

GetAsStringDateTime [253]
GetAsFloat [253]
GetAsInteger [253]
GetAsStream [253]
GetAsString [253]
GetField [253]
GetFieldType [253]
GetRecordCount [253]
IsFieldEmpty [254]

1.13.7.1.1 TRVReportQueryProcessor.GetAs*

Returns a value for the specified field.

```
function GetAsString(Field: TRVReportField[234]): TRVUnicodeString;
function GetAsInteger(Field: TRVReportField[234]): Int64;
function GetAsFloat(Field: TRVReportField[234]): Extended;*
function GetAsDateTime(Field: TRVReportField[234]): TDateTime;
function GetAsStream(Field: TRVReportField[234]): TStream;
```

* for Delphi 6-2007, this function returns Double

To get a value for the **Field** parameter, use GetField [253].

1.13.7.1.2 TRVReportQueryProcessor.GetField

Returns an object identifying a field having the specified name.

```
function GetField(const FieldName: TRVUnicodeString): TRVReportField[234];
```

If a field is not found, the function returns *nil*.

1.13.7.1.3 TRVReportQueryProcessor.GetFieldType

Returns a type for the specified field.

```
function GetFieldType(Field: TRVReportField[234]): TRVReportFieldType[234];
```

To get a value for the **Field** parameter, use GetField [253].

1.13.7.1.4 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.

1.13.7.1.5 TRVReportQueryProcessor.IsFieldEmpty

Returns: is the field value empty/undefined?

```
function IsFieldEmpty(Field: TRVReportField): Boolean;
```

To get a value for the **Field** parameter, use GetField[253].

This method is used by a report generator to check a parameter in "IfDef" and "IfNDef" commands[32].

1.13.8 TRVReportShapeProperties

TRVReportShapeProperties contains shape properties.

Unit RVReportShapes;

Syntax

```
TRVReportShapeProperties = class (TPersistent)
```

Hierarchy

TObject
TPersistent

Description

This item contains the following shape properties:

- MiddlePercent[255] defines the radius of an internal circle for the shape
- PointCount[255] – count of points in the shape
- Shape[255] – shape type (circle, regular polygon, star, arrow, etc.)
- StartAngle[255] – rotation angle for the shape

This class is used for the following properties:

- TRVReportCustomShapeVisualizer.ShapeProperties[191]
- TRVShape.ShapeProperties[117]
- TRVShapeItemInfo.ShapeProperties[173]

1.13.8.1 Properties

In TRVReportShapeProperties

- 🟩 MiddlePercent[255]
- 🟩 PointCount[255]
- 🟩 Shape[255]
- 🟩 StartAngle[255]

1.13.8.1.1 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 236)

Some shapes (regular polygons) ignore this property. Meaning of this property for other shapes is described in TRVReportShape 236.

Default value

50

1.13.8.1.2 TRVReportShapeProperties.PointCount

A count of points in the shape.

```
property PointCount: Integer;
```

The meaning of this property depends on the shape type (Shape 236)

Some shapes (circle, arrows) ignore this property. Meaning of this property for other shapes is described in TRVReportShape 236.

Default value

5

1.13.8.1.3 TRVReportShapeProperties.Shape

A shape type

```
property Shape: TRVReportShape 236;
```

Default value

rvrshCircle

1.13.8.1.4 TRVReportShapeProperties.StartAngle

An angle of the shape rotation.

```
property StartAngle: Double;
```

This value is measured in degrees.

In the topic on TRVReportShape 236 you can find examples of unrotated shapes (for **StartAngle** = 0).

Positive values rotate the shape clockwise, negative values rotate it counterclockwise.

Default value

0

Index