SharpShooter OLAP Documentation

Last modified on: September 7, 2011

Table of Content

Destination and Basic Features 3

Getting Started 4

Basic Information 9

Concept 9

Data Sources 9

Working with the DataCubeGrid Component (Pivot Table Presentation, Manipulation) 11

Expressions 19

Destination and General Principles of Expression Use 19

Description of the Expression Language Syntax 19

Work in Windows Forms Applications 25

Components Used in Windows Forms Applications 25

The Use and Capabilities of the DataCube 25

The Use and Capabilities of the DataCubeGrid 27

Working with Groups 32

The Work with Large Body of Data 38

Custom Cell Styles 39

Custom Drawing 40

The Use and Capabilities of DataCubePrintDocument 41

The Interaction of the SharpShooter OLAP with Third Party Components 42

The Interaction with SharpShooter OLAP by Means of ADO.NET Objects 43

The Interaction with SharpShooter OLAP by Means of the DataCubeView Component 44

Destination and Basic Features

SharpShooter OLAP is intended for developing and processing cross reports based on high-volume statistical data. Nowadays, managing data comprised by stacks of statistical information seems to be an unsolvable task without any automation. At the same time, the demands for dynamic change of data presentation based on one and the same data set often lead into a dead end. SharpShooter OLAP is a convenient solution to deal with such a challenge.

A full-featured graphical designer destined for pivot table retrofitting allows visualized indication of the data which is to be used for report creation. In order to get a ready-made report it is often enough to simply drag the required fields given within a certain dataset into the corresponding fields in the designer and hit a couple of keys to set a representation style.

Viewing the same data taking into account all possible dimensional groups is simple as well. The only thing that has to be done is dragging a group header into a required table field.

In order to assign value calculation regulations for both table dimensions and cross dimensional cells one can use various expressions. It allows adjusting a table in accordance with the information contained in a data set. The use of expressions affords a wide range of opportunities from basic summation of several data set field values up to various sorts of groups within a desired report (like fragmentation of date into years, months, quarters and days of week or indication of value intervals within a group).

To calculate pivot table cross-dimensional values there can be used one of aggregate functions (e.g. mean value calculation or the search for maximal/minimal value).

You are given an opportunity to choose one of preset pivot table presentation styles or utilize XP style. You can also create your own styles which can be saved to a file for further use.

The use of a generated pivot table is not limited by the borders of your application. It is possible to print a cross report or export it into an Excel table, PDF or HTML document where all current pivot table settings (such as dimensional groups minimized by a user, data sorting and filtering, or presentation style) will be taken into account

All pivot table settings can be saved to a file and this makes it possible to quickly go back to the unconfigured state of a table at any moment.

Getting Started

The DataCube component is intended for organizing the data from an outer data source as a pivot table. To specify a table, fields for each dimension (by rows and columns) should be specified. One should also specify the fields (facts) from which the data are taken to calculate values on intersections of rows and columns.

A regulation of data access is assigned in a field with the help of an expression.

Simple arithmetical and logical operations are available in expressions. Source fields can also be used as variables in expressions.

For the fact fields there can also be assigned an aggregation function used to aggregate data on rows and columns crossovers.

The DataCubeGrid component is intended for displaying the data presented in the DataCube. To do this, it is necessary to specify the Source property for the DataCubeGrid.

The current section contains a simple step-by-step example of SharpShooter OLAP utilization. Let us create a pivot table representing the data related to sales total of a certain product for each of several companies.

The initial data table structure is shown below.

Field name	Type	Description
CategoryName	String	Name of food category.
ProductName	String	Name of product
CompanyName	String	Name of company
UnitPrice	Currency	Unit price
Quantity	Number	Quantity
Discount	Number	Discount

The grouping of data in the resulting table is presented in the image below.

		CompanyName	
CategoryName	**ProductName**	Company1	Company2
Category1	Product1	*SalesSum11*	*SalesSum12*
	Product2	*SalesSum21*	*SalesSum22*
Category2	Product3	*SalesSum31*	*SalesSum32*
	Product4	*SalesSum41*	*SalesSum42*

SalesSum = UnitPrice * Quantity * (1 - Discount)

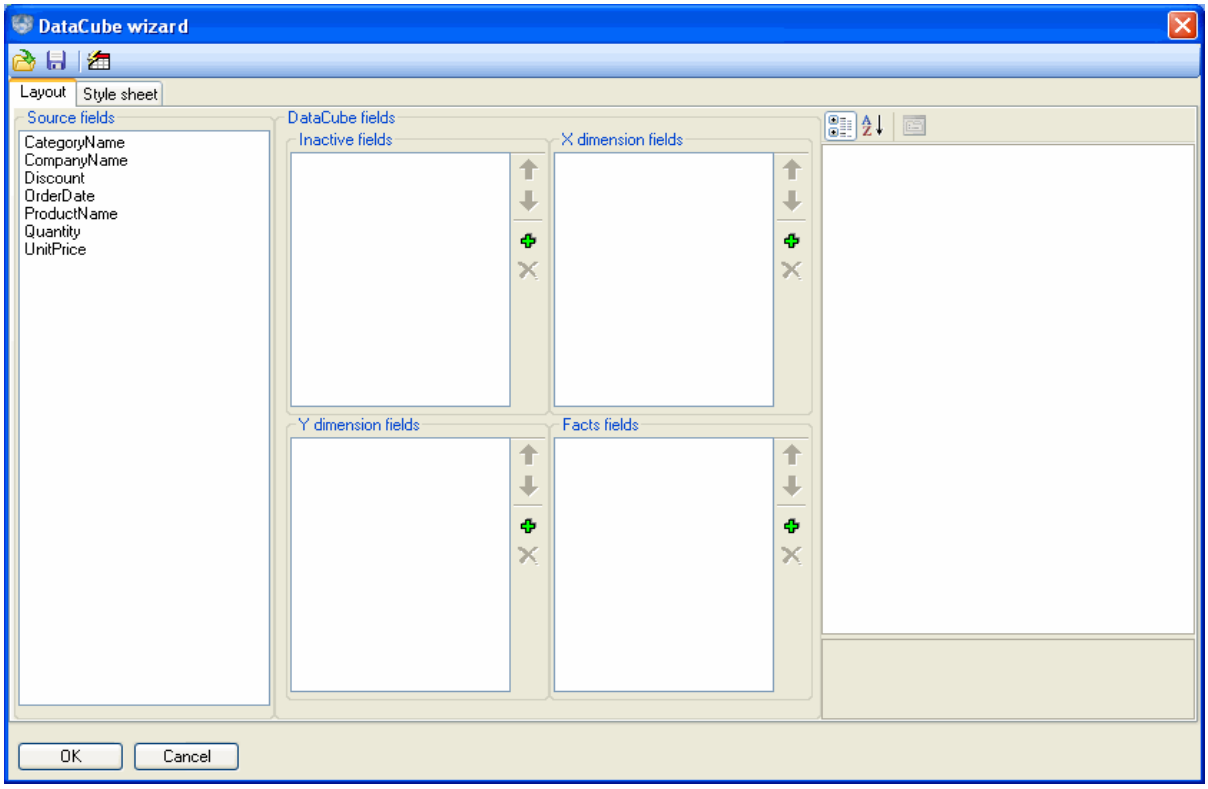
The columns will contain the names of supplier companies and the rows will contain product categories and denominations. The intersections will provide total sales information subject to products prices, quantity and discounts.

We shall use the demonstrational database Access NWind.mdb as source data.

Run Microsoft Visual Studio and create a new C# Windows Application project.

For a start let's create a data source for our report. To do that, add the System.Data.DataSet object onto the form and create a Sales table with fields corresponding to the initial data table structure.

Add the DataCube object onto the form. Indicate the Sales table from the DataSet as the DataSource value. Run the DataCube object designer by means of clicking Run Designer in its property window.



Drag the CompanyName caption from the Source fields list into the X dimension fields list. Add CategoryName and ProductName from the Source fields list into the Y dimension fields list.

Click the "Add" button in the Facts fields list. Indicate the following properties for a new field:

Expression UnitPrice * Quantity * (1 - Discount)

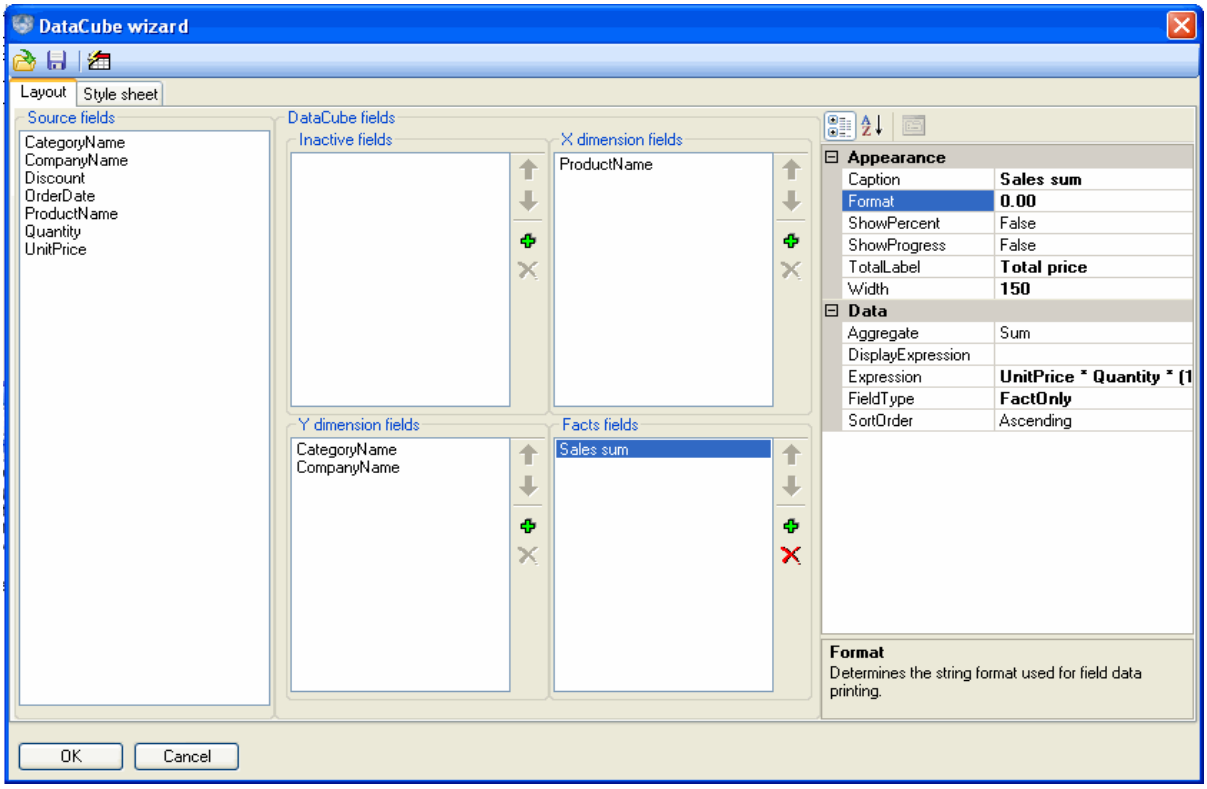
Caption Sales sum

Field type FactOnly

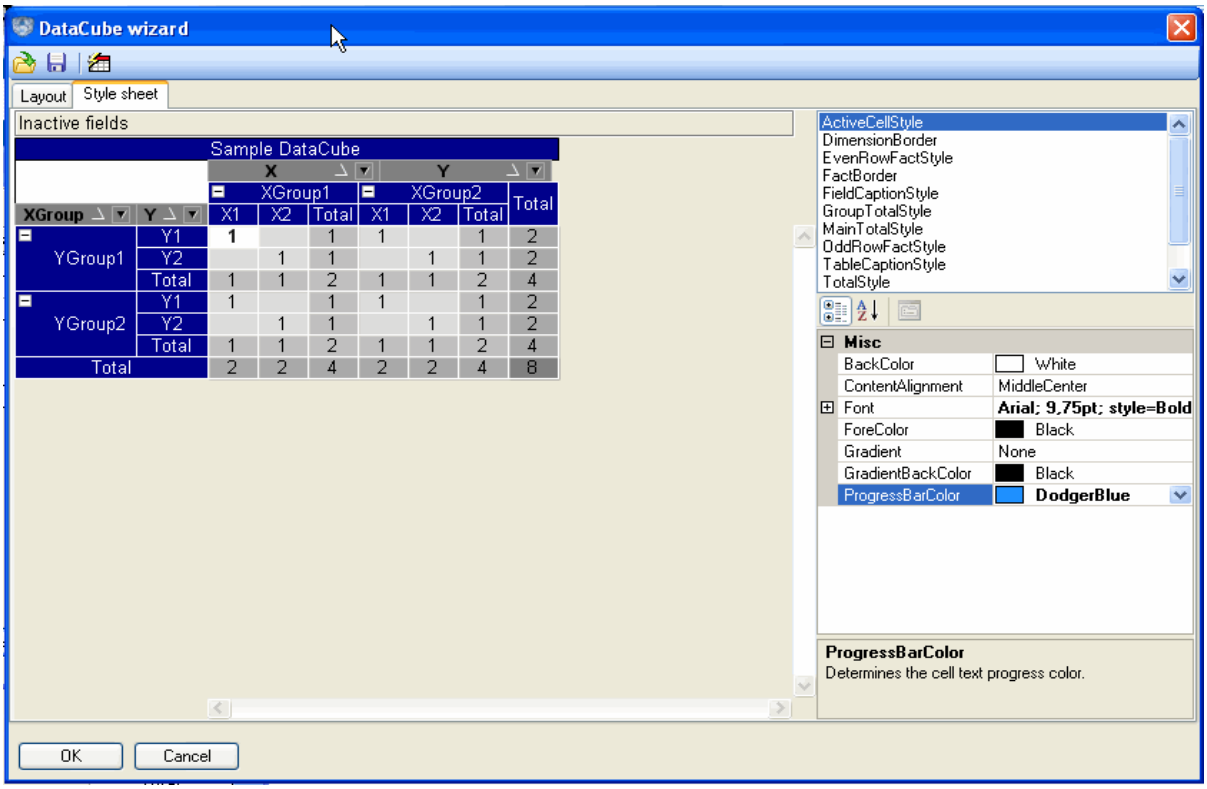
Format 0.00

Width 150

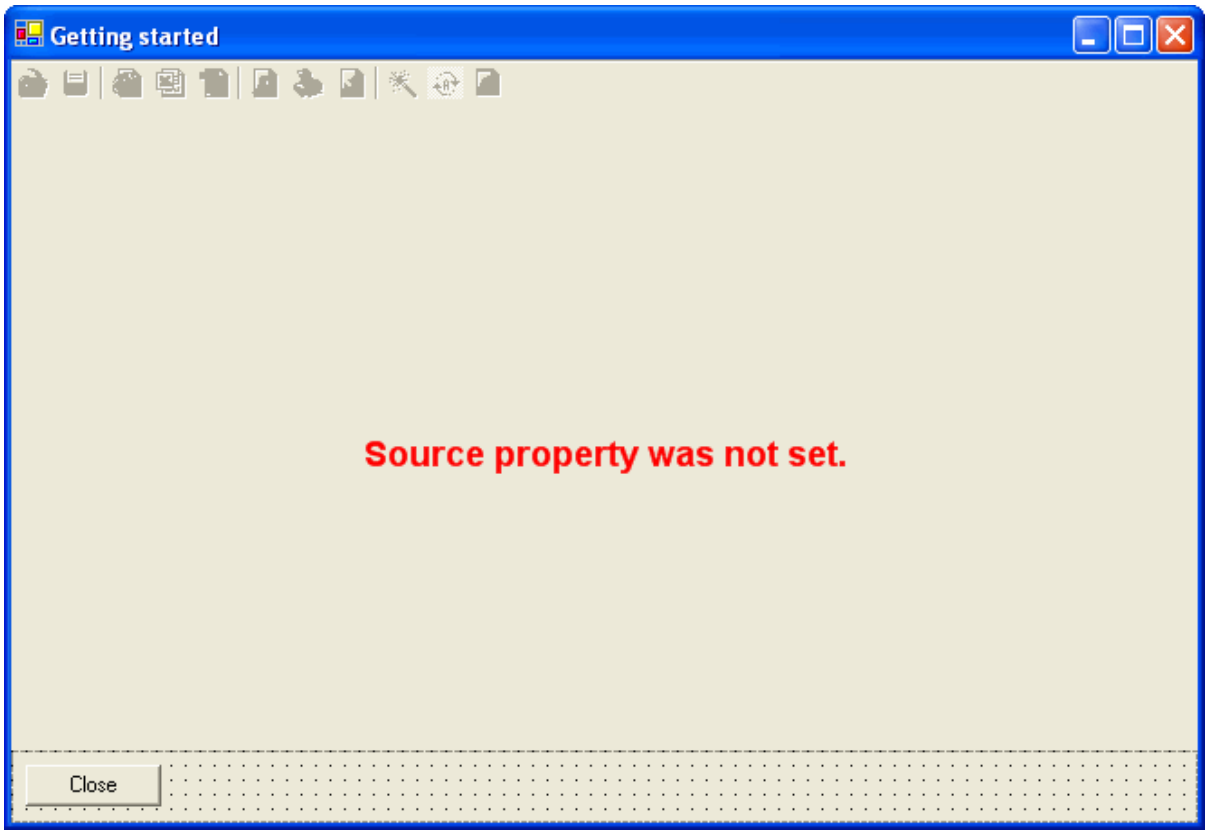
Total label Total price



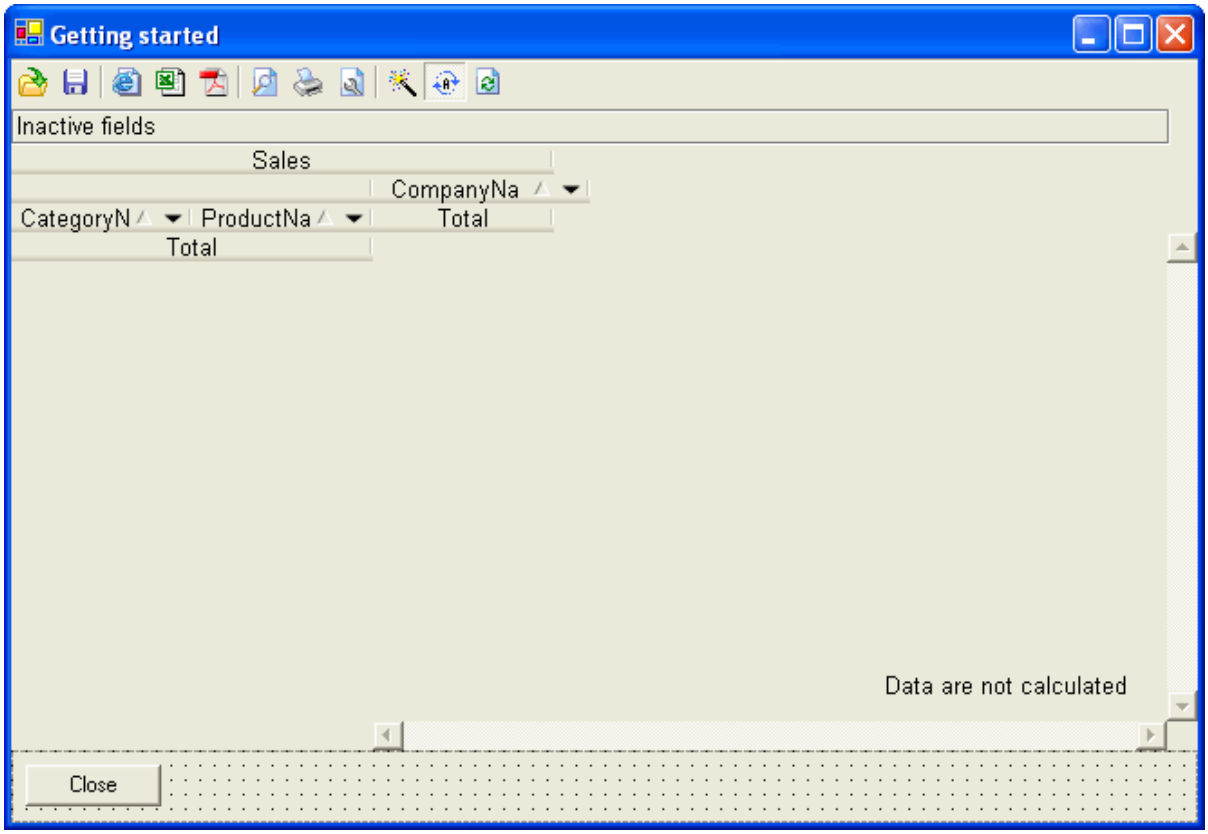
On the Style sheet tab specify a presentation style of your pivot table. It is possible to select one of the existing styles by clicking the "Auto format" button.



Add the DataCubeGrid object onto the form.

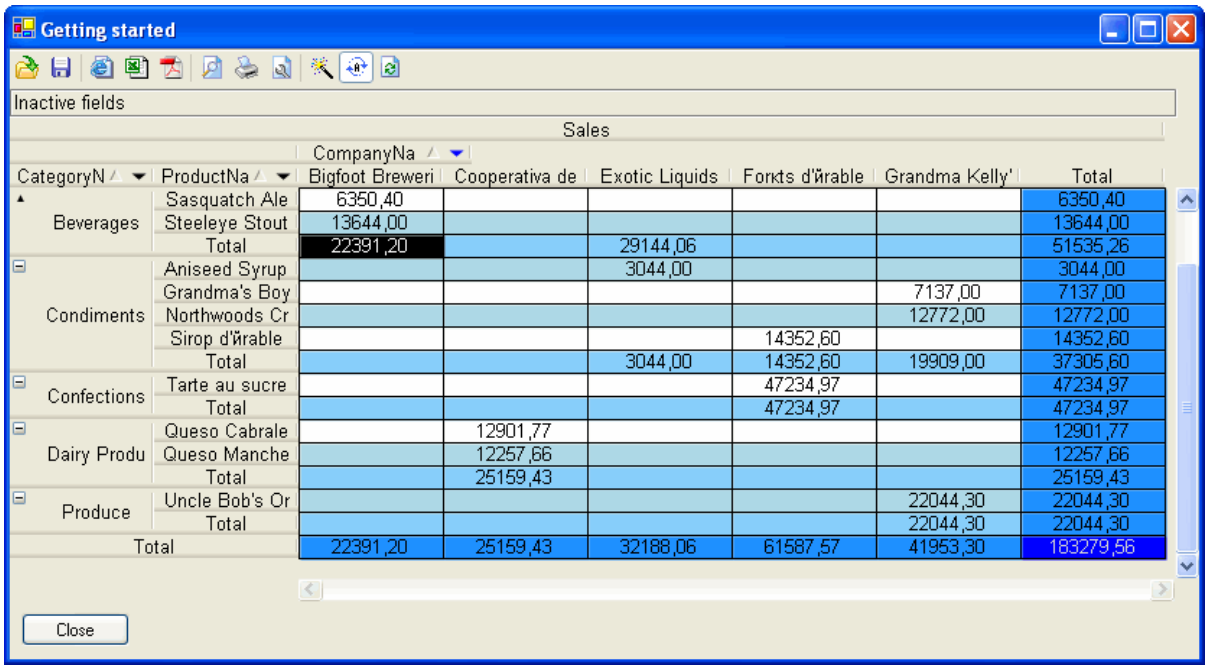


Indicate the adjusted DataCube object as the DataCubeGrid Source property value.



Add the code for filling the Sales table from the DataSet into the form uploading event handler.

Run your application and in its form you will see the pivot table created by you.



In this table you will be able to change the layout by dragging field headers. You will also be able to assign sorting and filtering by value and condition.

Basic Information

Concept

SharpShooter OLAP pivot table is a kind of an interactive table designed to be used for statistical analysis of multiple data. The rows and columns of such interactive table are based on the data taken from several columns of an initial table or another data source.

The information in pivot table cells is the data aggregated by the corresponding row and column.

Generally, an OLAP cube represents a structure containing multidimensional OLAP data, i.e. dimensions – the descriptive data that form the axes of a multidimensional cube, and facts – the computational quantitative values. Dimensions can contain multilevel hierarchies of values, and measures are aggregated data (sums, mean, maximal or minimal value, number of records) based on the data source fields that are of interest in respect to statistical analysis.

The formation of a pivot table presupposes the execution of the following operations:

Data grouping;

Calculation of sub-group subtotals;

Calculation of totals.

The field objects are used to define dimensions and facts. In a field, one can specify the initial table data to be used for a pivot table calculation and the way this pivot table will be provided to a user. The field indication order for one dimension also assigns the grouping of data within this dimension.

A user is the one who determines the assigning of field resultant values (for both dimensions and facts). It can be implemented by means of assigning an expression to a field. An aggregation function is assigned for facts. Thus, the resultant value for facts is the aggregation of their expressions results. So, the process of pivot table formation is being conducted as follows:

- At the data grouping stage the tree structure of table dimensions is being formed. For each tree node (dimension element value) the calculation of a corresponding field expression in respect to the data source information takes place. Here the grouping is being taken into account: group – tree node (a corresponding field from the dimension element list is being inserted), sub-group – the leaves of this tree node (the following field in the dimension elements list).

- The calculation of sub-group subtotals stage is the stage at which the direct filling of a resultant pivot table is being conducted. Fact values for dimension intersections are being calculated through corresponding fields' expressions. The calculation of expressions is performed subject to the values of all dimension groups formed at the data grouping stage.

- The calculation of totals occurs concurrently with filling a table for sub-groups in general. It is being held taking into account the aggregate functions specified in fields on the basis of fact values already calculated.

In addition, a user can alter the sorting and conduct a filtering by random data combination, display data as percentage and perform various useful conversions of a pivot table.

Data Sources

The following classes can be used as data sources for SharpShooter OLAP:

- ADO.NET objects – System.Data.DataSet, System.Data.DataView, System.Data.DataTable;

- user-defined classes (Business Objects) that implement System.ComponentModel.IListSource or System.Collections.Ilist interface.

Data sources for pivot tables are specified through the DataSources property of the DataCube class. This property is of System.Object type.

In order to specify a data source for a pivot table via the property editor, select the DataCube object and activate the DataSource property editor in the Properties window. There will appear a list containing the names of objects that are present on the form and fit to be used as data sources. It is necessary to select the name of an object intended to be used as a data source.

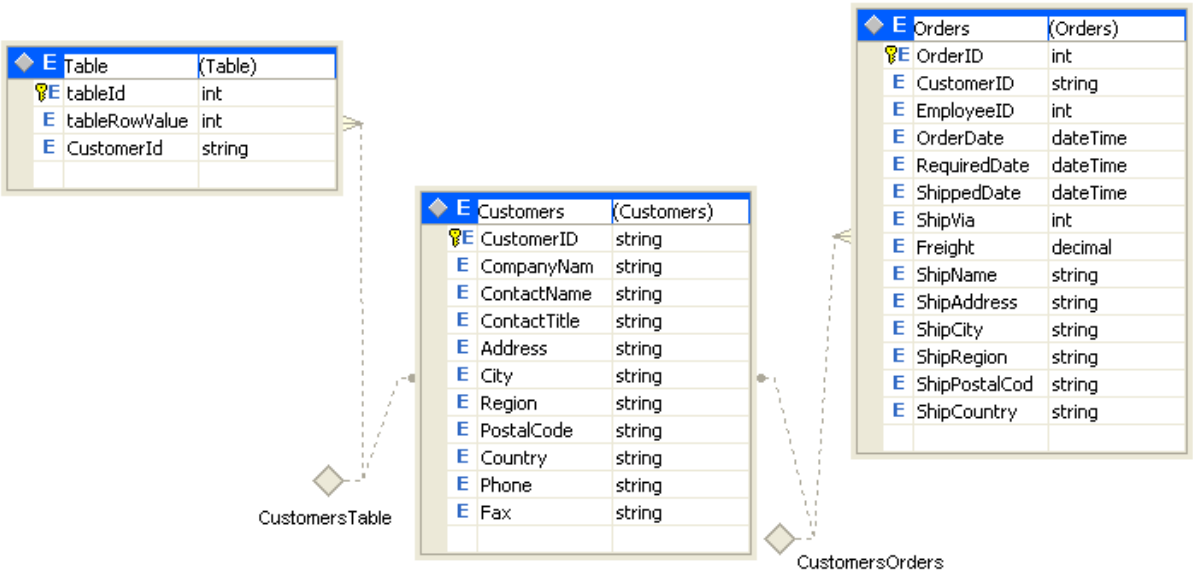
After that, the properties of an object selected as a data source will be available in the DataCube object for pivot table adjustment.

Nested data sources.

SharpShooter OLAP supports nested data sources. Now there is no need to prepare a special data access meeting the required calculations. It is enough to simply upload data form several connected tables into a single DataSet. Then it is necessary to specify a parent table to the DataCube object as a DataSource property. The JointFields property (string array) is filled in such a way that the property contains a list of data source fields through which a connection with the data source child elements is executed.

For example, in case one uses data schemas displayed in the screenshot below, the Customers table should be specified to the DataCube object's DataSource property and the JiontFields set should be filled in the following way:

```
"CustomersTable"
"CustomersOrders"
```

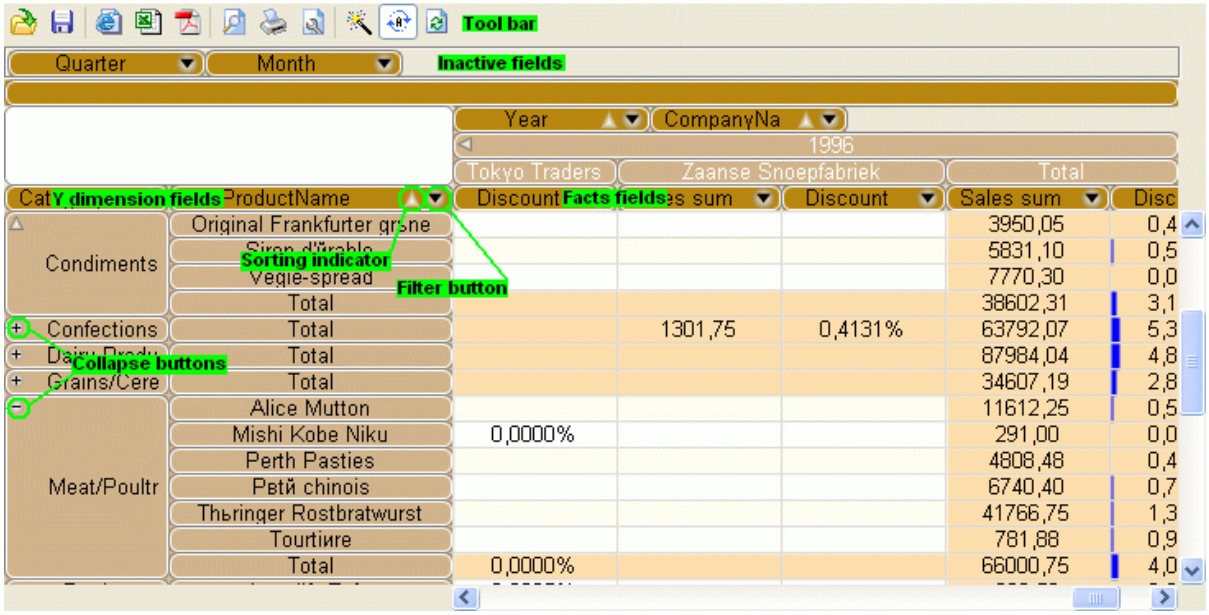


In dimension and fact elements' expressions it is possible to refer to nested data source fields specifying a path to the source. E.g., "CustomersTable.tableRowValue" or "CustomersOrders.OrderId"

Working with the DataCubeGrid Component (Pivot Table Presentation, Manipulation)

The DataCubeGrid object is an interactive control which displays a pivot table on the application form. There a user can view a final pivot table, modify its appearance and content, print or export a table into another file format.

The image below shows the appearance of DataCubeGrid.



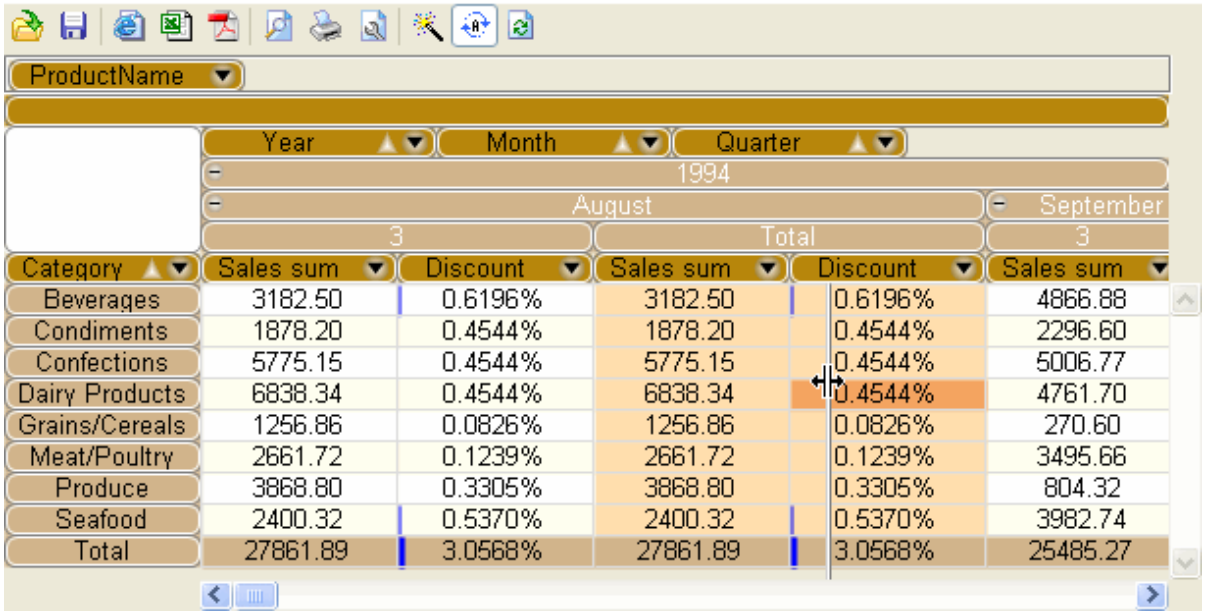
The `Inactive fields` scope displays headers of the fields that are specified in DataCube, but not engaged in pivot table formation.

The `X dimension fields` scope displays headers of the fields that define pivot table columns.

The `Y dimension fields` scope displays headers of the fields that define pivot table rows.

The `Fact fields` scope contains headers of the fields that are the facts of a pivot table.

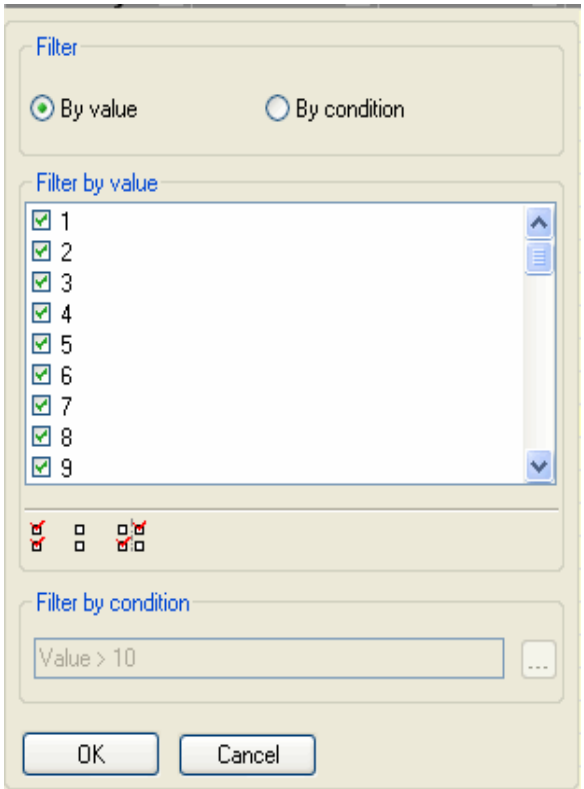
The fields in `X dimension fields` scope and in `Y dimension fields` scope correspondingly specify columns and rows of a pivot table. The width of any `DataCubeGrid` column can be altered by dragging its borders with a mouse.



Directly below the dimension field headers, the values of these dimensions (subject to their grouping) are displayed. A grouping is defined by the DataCube fields indication order. It is also possible to drill down/roll up group values (⊟ / ⊞ buttons) on the stipulation that this group contains sub-groups with values different from the Total.

It is possible to drag field headers. They can be moved between Inactive fields, X dimension fields, Y dimension fields and Facts fields scopes. It is also possible to change their order within one scope.

The field headers are interactive. Each of them has a filter button (▼), which helps a user to enter a filter setup dialogue for a respective field



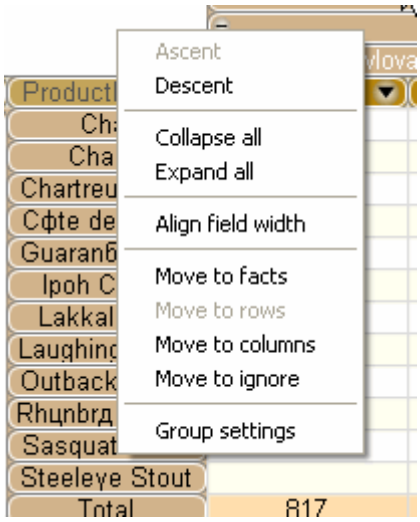
There are two possible ways of filtering field values: filtering by value and filtering by condition.

The setting of filtering by value presumes selection of those field values which will be displayed. It is also possible to set a filter for all field values by means of clicking the "Select all" button ( ). To cancel filtering a user should click the "Unselect all" button ( ). To invert filter selection a user can click the "Invert" button ( ).

To filter data by condition it is necessary to specify an expression, which result would satisfy the desired field value selection. A press of the "OK" button leads to saving filter condition and pivot table recalculation subject to the preset filtering. The "Cancel" button leads to ignoring all filter alterations and closing the dialogue window.

Field headers used in pivot table calculation allow specifying pivot table sorting. When the field header of the dimension is clicked the sorting by this dimension is carried out taking into account the grouping. When the field header that defines table fact is clicked, the sorting by the fact values column is conducted taking into account its dimensions. The sequence of displaying strings is changed. This capability of sorting by total columns is intended to facilitate quantitative data analysis.

When the right mouse button is pressed, the following contextual menu appears.



The contextual menu contains the following items:

Ascent – is used to set ascending sorting;

Descent – is used to set descending sorting;

Collapse all – is used to roll all field groups in;

Expand all – is used to roll all field groups out;

Align field width – is used to align field width by maximal width of its value;

Move to facts – is used to move a field into facts;

Move to rows – is used to move a field into row dimensions;

Move to columns – is used to move a field into column dimensions;

Move to ignore – is used to ungroup a field from pivot table calculation.

Group settings – is used to open the window for the field groups manipulation.

The DataCubeGrid tool set also contains ToolBox.

It has the following buttons:

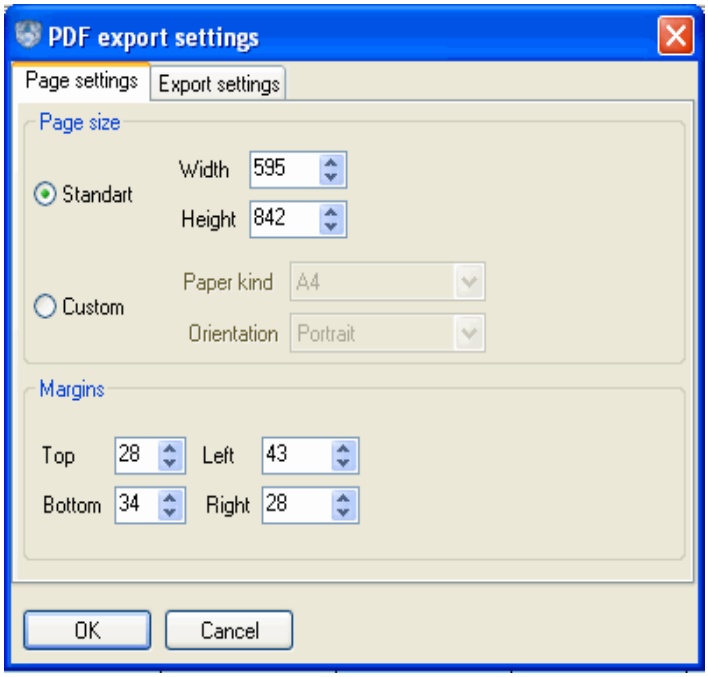
The "Open" button ( ). A press on this button allows a user to select a file, from which the DataCube layout and/or presentation style will be downloaded.

The "Save" button ( ). A press on this button calls the file saving dialogue. Here one can indicate what should be saved: a DataCube layout, a pivot table presentation style or both.

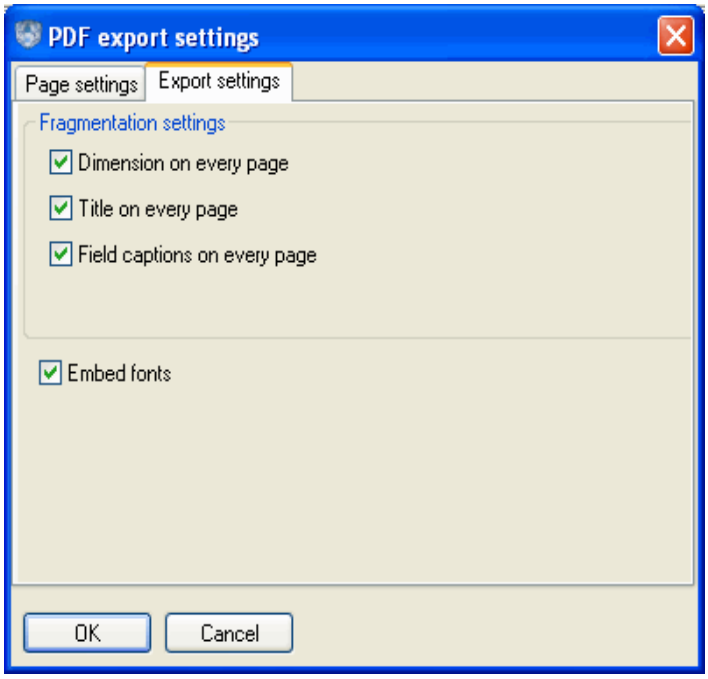
The "Export to HTML" button ( ). A press on this button will allow a user to enter the export to HTML dialogue box. At that, the current state of a pivot table (layout, styles, column width, sorting, and filters, the dimension element groups collapse) is being taken into account.

The "Export to Excel" button ( ). A press on this button opens the export to MS Excel dialogue box. A file is saved including the current state of a pivot table.

The "Export to PDF" button ( ). A press on this button opens the PDF export settings dialogue box. On the Page Settings tab a user is able to specify final PDF document settings: page size, orientation and indents.



On the Export Settings tab a user will see the pivot table appearance settings in a PDF document. Here, a user can specify whether the pivot table dimensions, its header and fields' names for dimensions and facts will be repeated on each document page. In addition, a user can specify whether to include the fonts used in a pivot table into the document.



The “Print setting” button ( ). This button opens the print settings dialogue.

The “Print preview” button ( ). This button allows previewing a pivot table before printing. At that, the settings specified in the Print setting dialogue window are retained and a pivot table is split into pages.

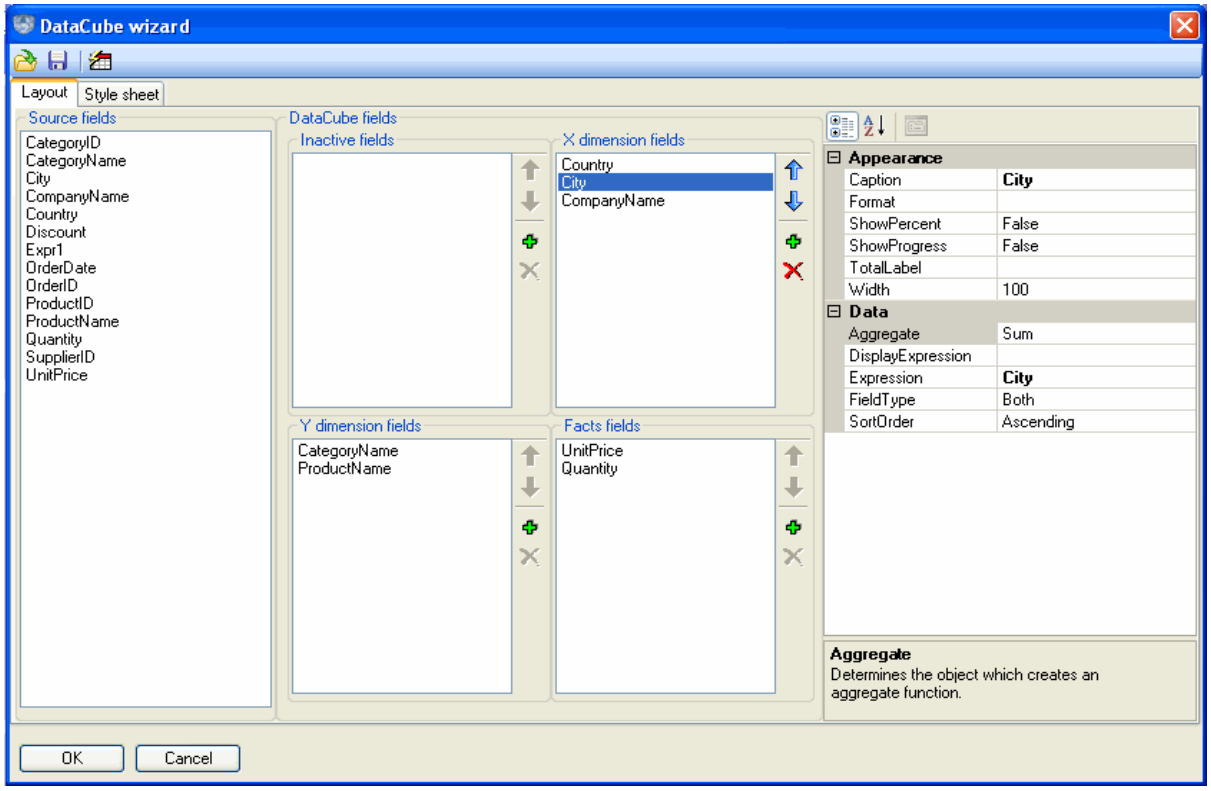
The “Print button” ( ). This button allows printing a pivot table.

The “Recalculate” button ( ). A press on this button calls forced recalculation of a pivot table.

The “Auto recalculate” button ( ). A press on this button turns on/off pivot table automatic recalculation while changing table layout and filtering conditions. This button usage is reasonable while working with large body of data. In order to recalculate a pivot table after specifying the settings, you should use the Recalculate button together with the Auto recalculate button pressed.

The “Wizard” button ( ). When pressed, this button runs the DataCube wizard for pivot table retrofitting.

The "DataCube wizard appears as follows.



The DataCube wizard allows altering a pivot table layout and a style of its display. The DataCube wizard provides four lists containing fields for pivot table formation:

Source fields – the list that provides a user with all data source fields;

Inactive fields – the list of fields selected by a user for pivot table formation, but not used for pivot table calculation;

X dimension fields – the list of fields selected from a data source or created by a user for column dimension formation;

Y dimension fields – the list of fields selected from a data source or created by a user for row dimension formation;

Facts fields – the list of fields, selected from a data source or created by a user for pivot table facts formation.

A user is capable of dragging fields from one list into another or within one and the same list, thereby changing the order of fields. In order to make pivot table field lists contents alteration a more convenient process, each list has a button panel.

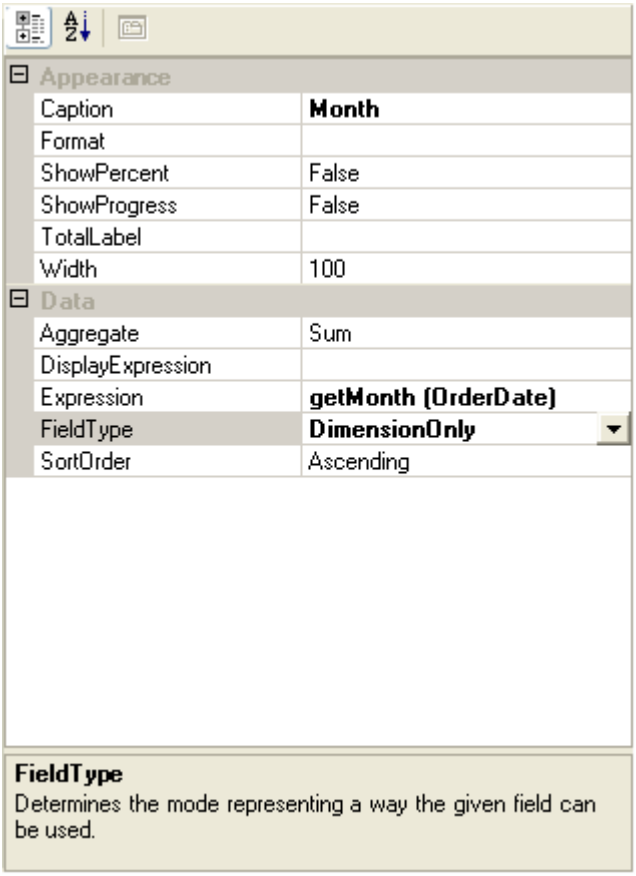
The "Move up" button ( ) allows moving a selected field one position up within a list;

The "Move down button ( ) allows moving a selected field one position down within a list.

When pressed, the "Add field" button ( ) creates a new field which is added in a list.

The "Remove field" button ( ) allows removing selected fields from a list.

The DataCube wizard window contains a selected field property editor (Fields properties).



The Expression property assigns an expression for a field;

The Caption property assigns an inscription on a field caption;

The Aggregate property assigns a data aggregation type for a field (makes sense for facts only);

The Field type property assigns a field type (Fact only, Dimension only, Both);

The Format property assigns a field data output format string;

The Width property assigns a field width;

The Show percent property defines, whether a field data will be presented as percentage;

The Show progress property defines whether a cell will be presented as progress. To set this property, it is necessary to set the Show percent property (makes sense for facts only);

The Total label property assigns a header for a field total;

The Sort order property assigns a field data sorting order.

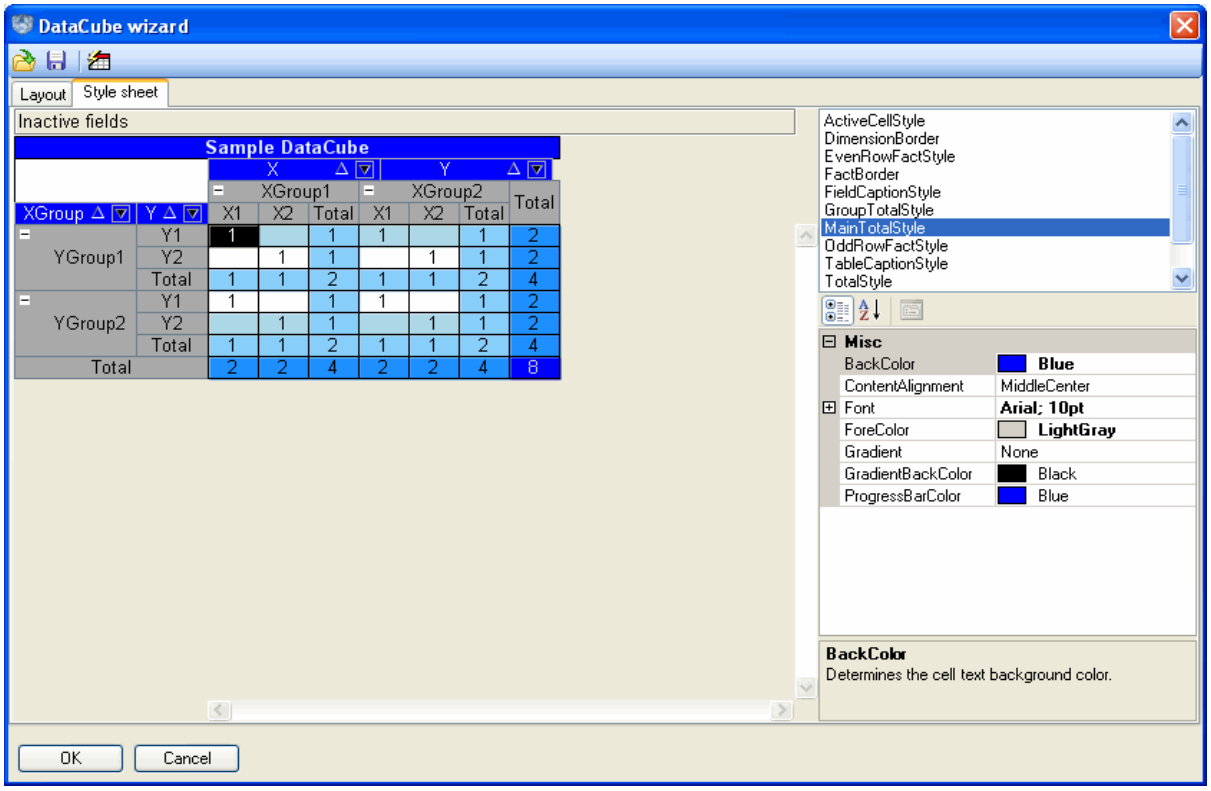
The DataCube wizard also has the following buttons:

The "Open" button ( ); it allows uploading pivot table settings.

The "Save" button ( ); it allows saving pivot table settings. It is possible to save a pivot table layout, its presentation style or both.

The "Auto format" button ( ); it opens a style selection dialogue box.

DataCube wizard has the Style sheet tab.



Here a user can specify pivot table appearance settings. In particular, a user can define displaying styles for X and Y dimensions, fields' headers, even/odd strings of facts table, totals crossings etc. In order to do this one should select a table element from the list and specify the corresponding properties to this element in the Property grid.

Expressions

Destination and General Principles of Expression Use

It is possible to assign an expression for all fields specified in a pivot table (facts and dimensions). The expression calculation result defines a field value, and therefore the dimension elements values and fact values. As for facts, values defined by their expression results take part in aggregation.

The order of expression use is as follows: during a pivot table formation a field responsible for a dimension element or a fact is assigned for dimension element or fact cell value calculation. There takes place a calculation of such field expression taking into consideration the dimension intersection for facts and the data source value for dimension elements. The calculated value is recorded in a table.

If an expression contains syntax or semantic errors or an exception appears during the calculation process, the component will generate a corresponding exception message and provide a user with information on the error type and location.

Description of the Expression Language Syntax

An expression is a task on formula calculation. The expression value is a calculation result. The expression is built out of constants, objects and their properties and methods, operators, function calls and round brackets.

Data types used:

In the expressions all data types, available in .NET Framework, are used. Also there is the special support for fractional and logical values, strings and the DataTime type.

The following operations are available in expressions:

Arithmetic operations:

`'+' - addition`. It is defined for numerical values;

`'-' - subtraction`. It is defined for numerical values;

`'-' - unary minus`. It is defined for numerical values;

`'*' - multiplication`. It is defined for numerical values;

`'/' - division`. It is defined for numerical values.

`'%' - residue of division`. It is defined for numerical values.

Logical operations:

`'or' - logical OR`. It is defined for logical values;

`'and' - logical AND`. It is defined for logical values;

`'=' - equality`. It is defined for numerical, string and logical values;

`'! =' - inequality`. It is defined for numerical, string and logical values;

`'<' - less`. It is defined for numerical values;

`'<=' - less or equal`. It is defined for numerical values;

`'>' - more`. It is defined for numerical values;

`'>='` – `more or equal`. It is defined for numerical values;

`'not'` – `logical negation`. It is defined for logical values.

String operations:

`'&'` – `strings concatenation`. It is defined for string values.

The descending priority operations order.

'–'(unary), 'not'

'*', '/', '%'

'+', '–', '&'

'>', '>=', '<', '<='

'=', '!–'

'and'

'or'

Constants assigning:

`< decimal numeral >*` - the integral value.

`< decimal numeral >*.< decimal numeral >*` - value.

For example, 1234 is an integral value, 1234.1234 is a fractional value.

String constants are assigned with the help of a double quotation mark (")

For example, `"It is a string value"`.

Logical constants are assigned with the help of two key words: `'true'` and `'false'`.

'Null' constant. This constant is used for indeterminate value assigning.

Access to variables, objects, their fields and methods.

In the expressions one can apply to variables, available in a field by their names. The search for variables and objects by their names is executed in the following way: in case an address by the object's name is present in an expression, the GetObject function that should return the desired object is called for a field the given expression is written for. A field as an object is available by "value" name.

A dot '.' is used to apply to object fields and methods.

For example:

`Value.Year` – a reference to the field Year property the value of which is of the DateTime type;

`Field1.ValueToString()` – a call for the field Field1 ValueToString() method;

ATTENTION! It is impossible to call the object methods, if they are overloaded.

As the expression language is not typified, it is impossible to define what method should be called.

In the expressions, one can apply to the functions available in a field by their names. The search is executed similarly to the objects search by the names.

The following functions are available:

`The functions of conditional choice:`
`if (condition, value1, value2)`, the 1st parameter should be of logical type; the 2nd and the 3rd parameters should be of object type. If the 1st parameter is true, the function returns the 2nd parameter value, otherwise it returns the 3rd parameter value.

`switch (condition[0], value[0], … , value by default)`, if the [i] condition is set to true, the function returns the [i] value, if the [i] condition in any i is false, the value returns by default.

Formatting Functions:

`format (object, mask)`, the object is of object type, the mask is of string type. It returns the object's string presentation according to the mask.

Mathematical Functions

`sin (argument)` – calculates the argument sine. The argument is of the double type, the calculation result is of the double type.

`cos (argument)` – calculates the argument cosine. The argument is of the double type, the calculation result is of the double type.

`tan (argument)` – calculates the argument tangent. The argument is of the double type, the calculation result is of the double type.

`atan (argument)` – calculates the argument arctangent. The argument is of the double type, the calculation result is of the double type.

`sqr (argument)` – calculates the argument to square. The argument is of the double type, the calculation result is of the double type.

`sqrt (argument)` – calculates the argument's square root. The argument is of the double type, the calculation result is of the double type.

`log (argument)` – calculates the argument's natural logarithm. The argument is of the double type, the calculation result is of the double type.

`exp (argument)` – raise E number to the argument degree. The argument is of the double type, the calculation result is of the double type.

`sign (argument)` – returns:-1 if the argument is negative, 0 – if the argument is 0, 1 – if the argument is positive. The argument is of the double type, the calculation result is of the int type.

`abs (argument)` – returns the argument's absolute value. The argument is of the double type, the calculation result is of the double type.

`round (argument1, argument2)` – rounds the 1st argument value to the symbols amount after the comma, assigned by the 2nd argument. The 1st argument is of the double type, the 2nd argument is of the int type, the calculation result is of the double type.

The expression can be the argument of any function and method. If the expression calculation result is not appropriate to the argument type, the expression will not be calculated.

Strings Processing Functions:

`concat(string1, string2)` – returns concatenation of two strings assigned by parameters;

`indexOf(string1, string2[, start])` – returns index of occurrence of the substring string2 to the string1 string from the Start position;

`insert (string, position, subString)` – pastes subString into the String from the Position;

`length(string1)` – returns string1 length;

`padLeft(string, length, char)` – returns string complemented by the char character on the left up to the specified length;

`padRight(string, length, char)` – returns string complemented by the char character on the right up to the specified length;

`remove (string1, start, length)` – removes the length substring from string1 beginning from the Start position;

`replace(string, subString, newSubString)` – replaces substring with newSubString in the string;

`subString(subString (string, start, length))` – returns the string substring with the specified length beginning from the Start position;

`toLower(string)` – casts string to the lower registry;

`toUpper(string)` – casts string to the upper registry;

`trim(string)` – removes white space characters from the string;

`trimEnd(string)` – removes white space characters from the end of the string;

`trimStart(string)` – removes white space characters from the beginning of the string;

`regIsMatch(string, expressionString)` – returns match char of the string to the expressionString regular expression;

`regReplace(string, expressionString, newString)` – replaces substring, corresponding to the expressionString regular expression, with newString in the string;

Date Processing Functions:

`getMonth (argument)` – returns the string month name by the date assigned by an argument. An argument should be of the System.DateTime type.

`getDayOfWeek (argument)` – returns the string day of week name by the date assigned by an argument. An argument should be of the System.DateTime type.

`getQuarter (argument)` – returns the quarter number of a date specified as an argument. An argument should be of the System.DateTime type.

`addDays(date, n)` – adds N number value of days to the Date object;

`addHours(date, n)` – adds N number of hours value to the Date object;

`addMinutes(date, n)` – adds N number of minutes value to the Date object;

`addMonths(date, n)` – adds N number of months value to the Date object;

`addSeconds(date, n)` – adds N number of seconds value to the Date object;

`addYears(date, n)` – adds N number of years value to the Date object;

`getDate(date)` – returns the Date object without time;

`getDayOfMonth(date)` – returns the day of the month value of the Date object;

`getDayOfYear(date)` – returns the day of the year value of the Date object;

`getHour(date)` – returns the hours of the specified date value of the Date object;

`getMinute(date)` – returns the minutes of the specified date value of the Date object;

`getSecond(date)` – returns the seconds of the specified date value of the Date object;

`getTime(date)` – returns the time of the specified date value of the Date object;

`getYear(date)` – returns the year of the specified date value of the Date object;

`toShortDateString(date)` – returns string representation of the date;

`toShortTimeString(date)` – returns string representation of the time.

The ranking function:

`Range (currentValue, range, zeroPoint)` – extracts data intervals. All arguments should be of numeral type. currentValue – the current value; range – the range; zeroPoint – the zero point.

Example:

Let us create a new field and specify `"Range(OrderDate.Year, 4, 0)"` as the Expression property value. Here the OrderDate is a System.DateTime type data source field. The result is presented in the image below.

OrderDate | Quarter | Month | Quantity

Purchases

	RangeField / Year					
	1992 - 1996			1996 - 2000		Total
Category	1994	1995	Total	1996	Total	
Beverages	1693.40	4587.00	6280.40	5531.25	5531.25	11811.65
Condiments	626.30	2077.20	2703.50	1901.80	1901.80	4605.30
Confections	1047.90	3388.50	4436.40	3112.90	3112.90	7549.30
Dairy Produ	1435.80	5006.20	6442.00	3433.80	3433.80	9875.80
Grains/Cere	422.40	2192.15	2614.55	1549.75	1549.75	4164.30
Meat/Poultr	973.40	3450.42	4423.82	2993.51	2993.51	7417.33
Produce	526.20	2216.10	2742.30	2044.15	2044.15	4786.45
Seafood	755.50	3100.04	3855.54	2435.24	2435.24	6290.78
Total	7480.90	26017.61	33498.51	23002.40	23002.40	**56500.91**

Realization of custom functions and variables is available in the expressions.

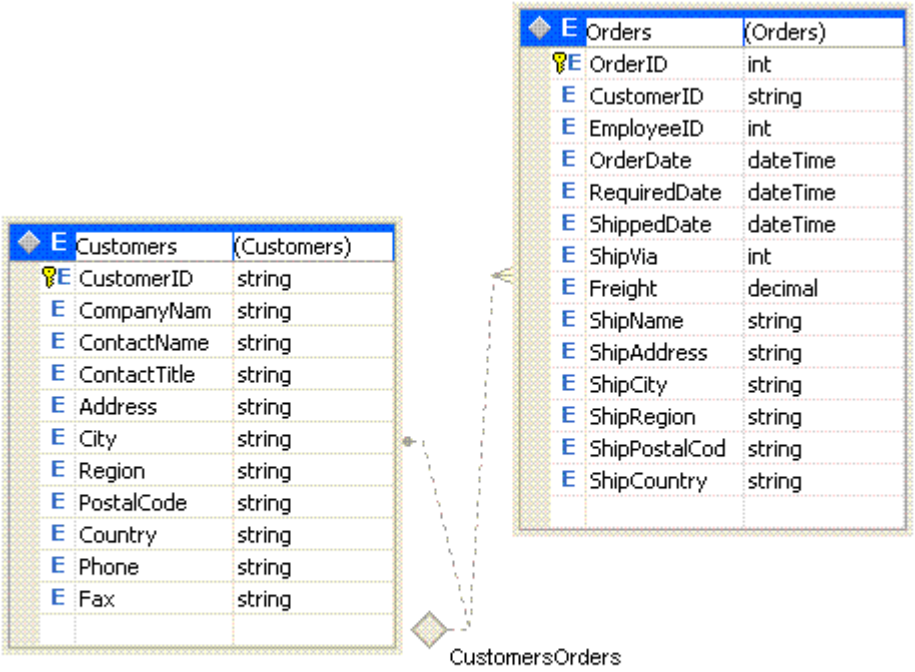
To make custom functions and variables available in the elements expressions it is necessary to implement custom element that will realize the PerpetuumSoft.Framework.Expressions.IExpressionSite interface. After that it is necessary to register custom function. The registration is carried out the following way:

```
PerpetuumSoft.Framework.Expressions.BuiltInFunctions.RegisterFunction(<Function name>, <inplemented IFunction object>);
```

Access to the nested fields of a data source.

In expressions it is possible to refer to the data source nested fields. The reference itself is executed with specification (specifying) of the complete path to a field relative to the main source specified as a value to the `DataCube.DataSource` property. E.g., in case a data schema displayed in the screenshot below is used the access code for the Orders table's `OrderDate` field and `Freight` field should look in the following way:

"CustomersOrders.OrderDate" and "CustomersOrders.Freight", correspondingly.


Orders (Orders)
OrderID int
CustomerID string
EmployeeID int
OrderDate dateTime
RequiredDate dateTime
ShippedDate dateTime
ShipVia int
Freight decimal
ShipName string
ShipAddress string
ShipCity string
ShipRegion string
ShipPostalCod string
ShipCountry string
Customers (Customers)
CustomerID string
CompanyNam string
ContactName string
ContactTitle string
Address string
City string
Region string
PostalCode string
Country string
Phone string
Fax string
CustomersOrders

Work in Windows Forms Applications

Components Used in Windows Forms Applications

The four components used for displaying and managing pivot table in Windows Forms applications are the DataCube, DataCubeGrid, DataCubeView and DataCubePrintDocument.

The DataCube ( ) component serves for organizing the data from an outer source into a pivot table. DataCube stores pivot table layouts, its presentation style and allows pivot table print and export.

The DataCubeGrid ( ) component is intended for displaying the data presented in the DataCube. The DataCubeGrid enables a user to interactively shift pivot table layout and presentation style as well as to manage its print and export.

The DataCubePrintDocument ( ) component is responsible for splitting a pivot table into pages and leading it to a printing device.

The DataCubeView ( ) component allows creating presentations of data stored within the DataCube object. It is necessary to provide an interface for access to data stored in the DataCube by other components that use the same information (like chart and diagram components). The DataCubeView component "expands" the pivot table dimension cube providing the data as a linear table, which columns are facts of the initial cube and a field of one of dimensions. DataCubeView allows reorganizing a cube into a linear table by any dimension.

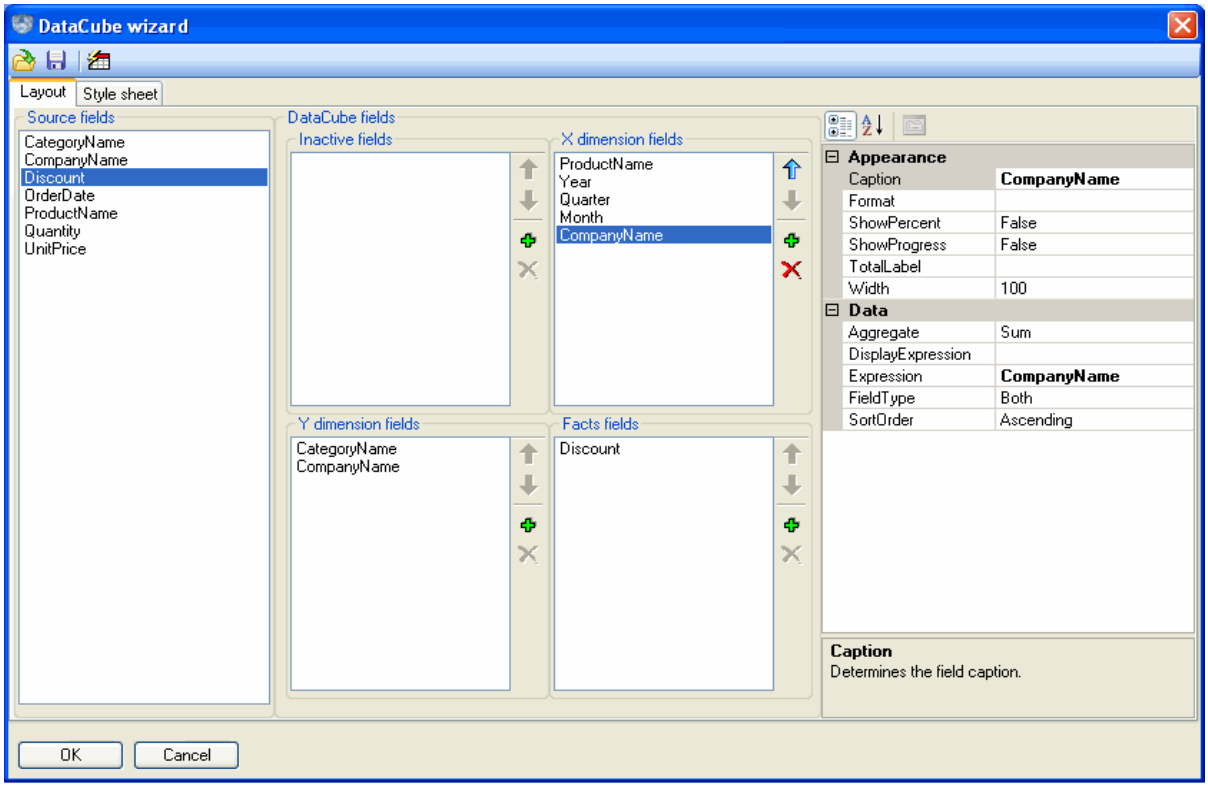
The Use and Capabilities of the DataCube

Let us examine the use of the DataCube component by the following example.

Let us create a Windows Forms application and add the DataCube component object situated on the SharpShooter OLAP tab onto the form. Let us name the newly-created object salesDataCube. It is necessary to specify a data source (the DataSource property). To do that, we shall add the DataSet object onto the form and create a Sales table for this object. The Sales table structure is shown in the image below.

Field name	Type	Description
CategoryName	String	Name of food category
ProductName	String	Name of product
CompanyName	String	Name of company
UnitPrice	Currency	Unit price
Quantity	Number	Quantity
Discount	Number	Discount
OrderDate	DateTime	Date fo sale

In order to adjust the desired pivot table layout let us call the DataCube wizard designer by double clicking the salesDataCube object or by selecting the "Run Designer" option in the salesDataCube property window.



The designer Source fields list contains the Sales table field names. Let us assign a pivot table row dimension. To do that, we shall drag the CategoryName and ProductName fields from the Source fields list into the Y dimension field list (category and product names). This will be a row dimension, in which the products are grouped by name and category. Now we shall assign column dimension. Let it be the date of sale indicating year, month, day of week and the name of a producer company. Let us create three fields within the X dimensions fields list and set the Expression property for the newly-created fields:

```
Year    -  OrderDate.Year
Quarter -  getQuarter(OrderDate)
Month   -  getMonth(OrderDate)
```

Also we shall drag the CompanyName field from the Source fields list into the X dimension fields list. Thus we have created a column dimension. Here the data will be grouped by company name and by date identifying year, quarter, and month.

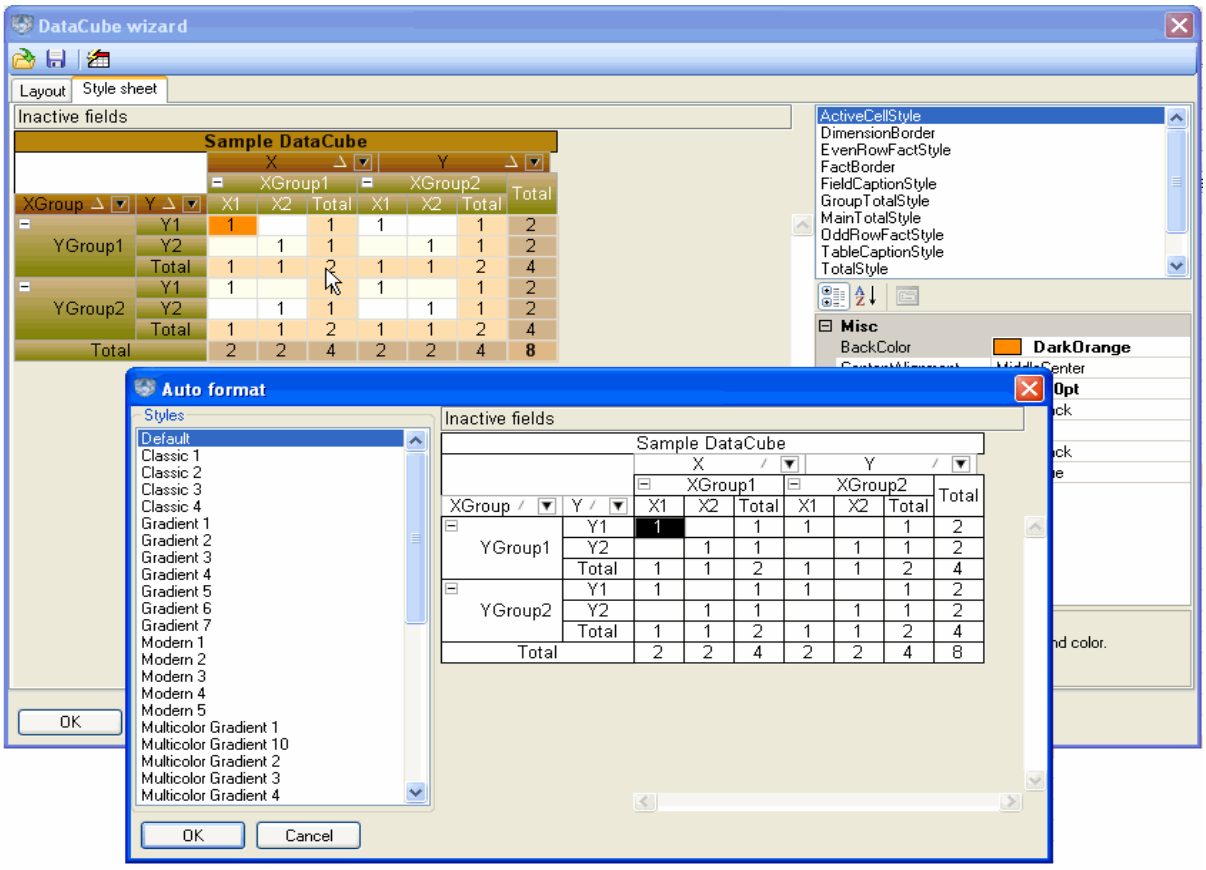
Let us specify the Field type property for all dimension fields as Dimension only.

Let us specify the facts for our pivot table. First of all, we shall create a field in the Facts fields list and name a new field by specifying the "`Sales sum`" as the Caption property value. We shall indicate "`UnitPrice*Quantity*(1 - Discount)`" as the Expression property value for this field. This fact will represent the final sales total for a product of a specific company on a specific date taking in consideration the unit price, the amount sold and discounts. We shall also drag the Discount field from the Source fields list into the Facts fields list. Then let us set the Show percent and Show progress properties to true. And after that, let us select an aggregate function for the given fact. Specify the Avg value for the Aggregate property. The given fact will indicate the mean discount value for the given product as percentage. Let us also set the Format property to "0.00" for all fields from the Facts fields list.

The pivot table layout is now assigned.

		Year	Quarter	Month	CompanyName				
		Year1							
		Quarter1						Quarter2	
		Month1		Month2				Month3	
CategoryName	ProductName	Company1		Company1		Company2		Company3	
Category1	Product1	SalesSum11	Discount11	SalesSum12	Discount12	SalesSum13	Discount13	SalesSum14	Discount14
	Product2	SalesSum21	Discount21	SalesSum22	Discount22	SalesSum23	Discount23	SalesSum24	Discount24
Category2	Product3	SalesSum31	Discount31	SalesSum32	Discount32	SalesSum33	Discount33	SalesSum34	Discount34
	Product4	SalesSum41	Discount41	SalesSum42	Discount42	SalesSum43	Discount43	SalesSum44	Discount44

Now let us assign the pivot table presentation style in the Style sheet designer tab.



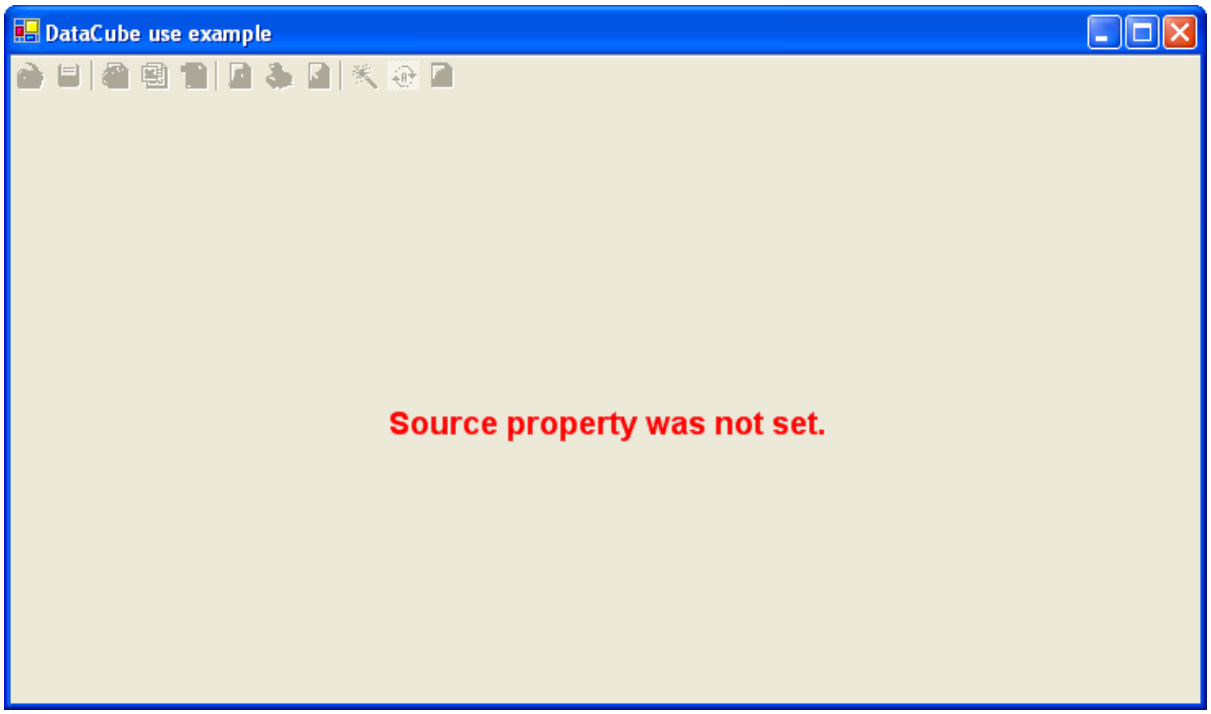
To do that, we shall launch the auto format call dialogue and select one of the predefined pivot table presentation styles. Then we shall save the pivot table settings having specified the "Layout and sheet" file type in the saving dialogue box.

You can find this example in the DataCubeUseExample folder.

The Use and Capabilities of the DataCubeGrid

We shall demonstrate some DataCubeGrid capabilities by an example.

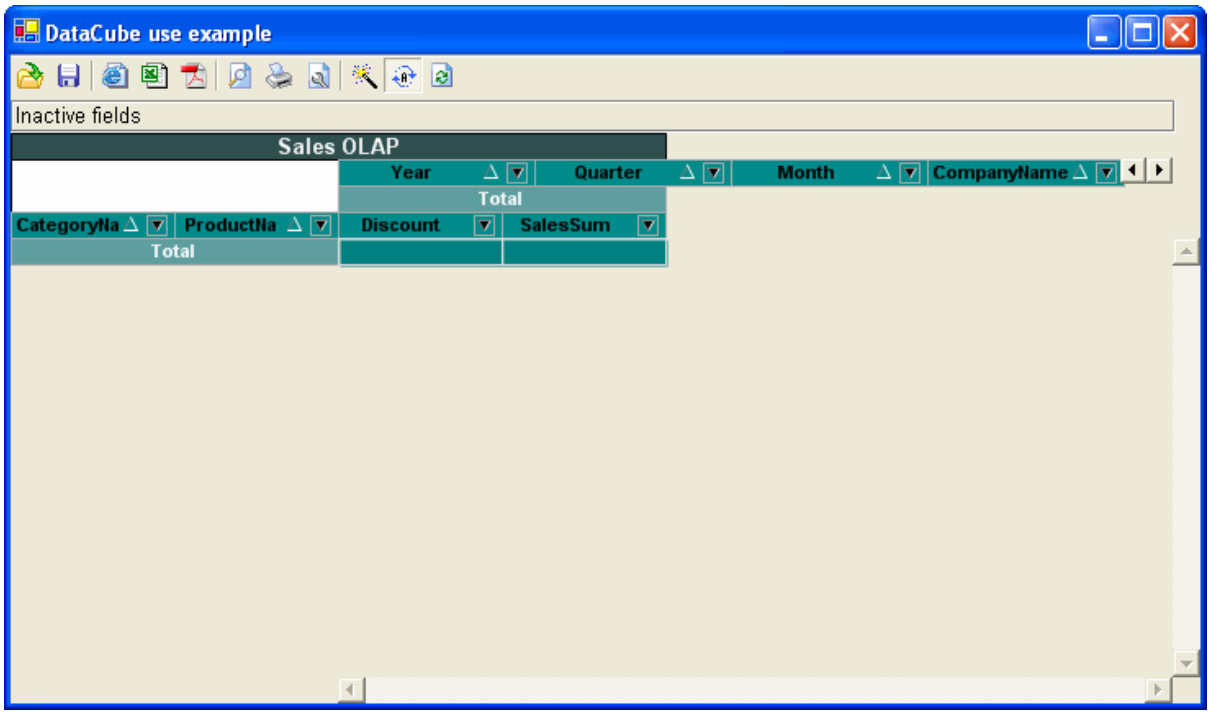
Let us create a Windows Forms application and add the DataCubeGrid component object situated on SharpShooter OLAP tab onto the form.



In order to work with DataCubeGrid, it is necessary to specify a DataCube to it. Let us create a DataSet and a DataCube by adding them onto the form. Adjust the DataCube and the DataSet as it is shown in the previous example of this User Guide.

Let us set the DataCube ShowSingleFact property to true. This will ensure the display of a fact header provided that the quantity of fact is equal to 1.

Let's specify the obtained DataCube as the DataCubeGrid Source property value.



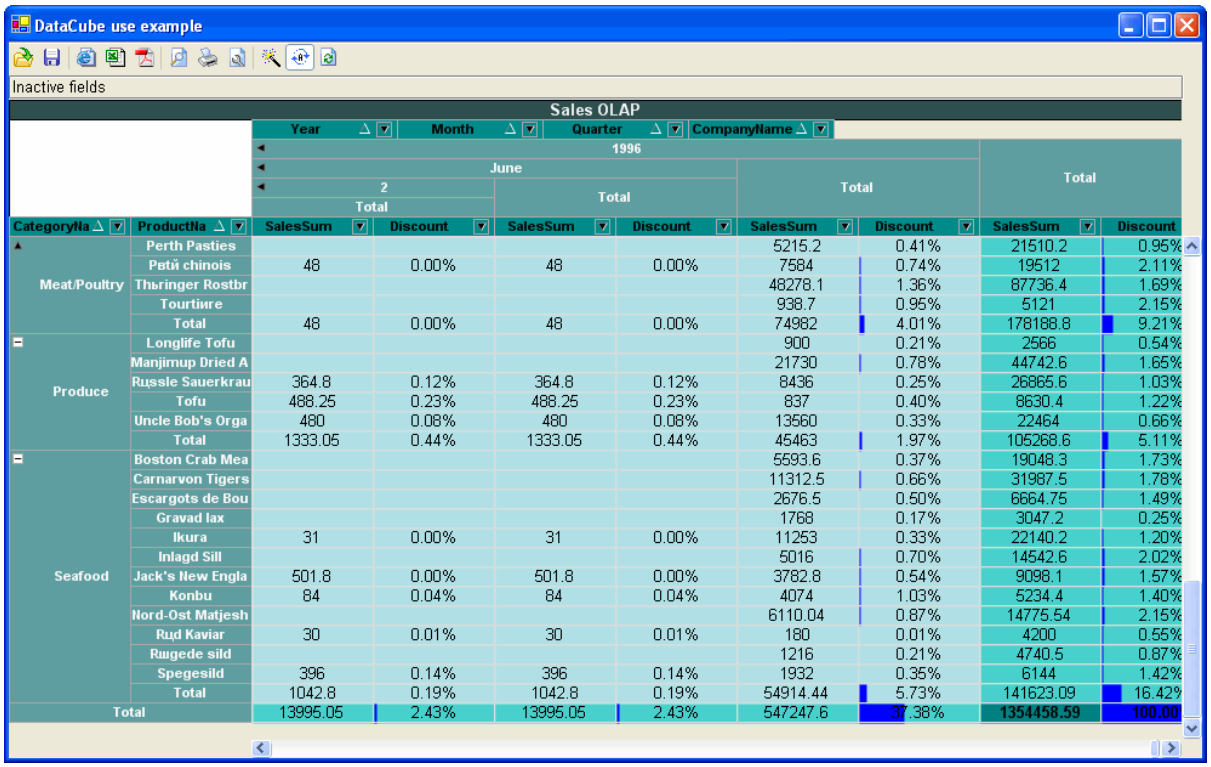
It is necessary to add the code for filling the Sales table from the DataSet into the form load event.

It is also necessary to write the following code in the form load event handler:

salesDataCube.RaiseDataCubeChanged();

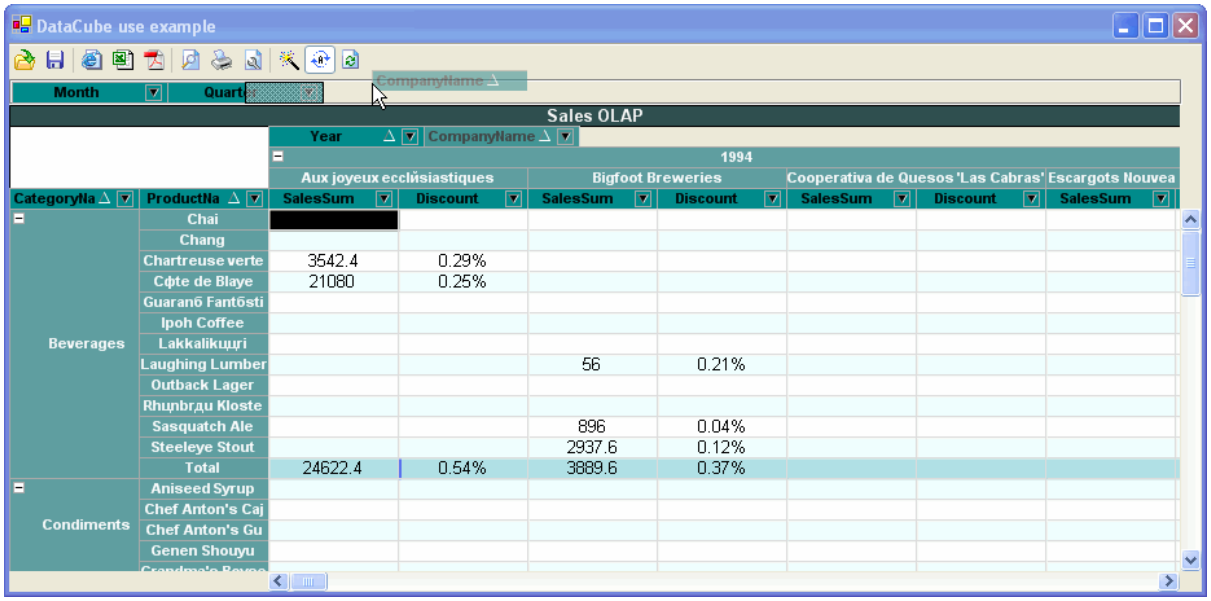
If you change DataCube settings from the code (not from DataCubeGrid), it is necessary to invoke the DataCube.RaiseDataCubeChanged() to notify DataCubeGrid about settings change. Only in this case the correct displaying of the changed pivot table will be provided.

Let us run our application. The required pivot table will be displayed on the form.

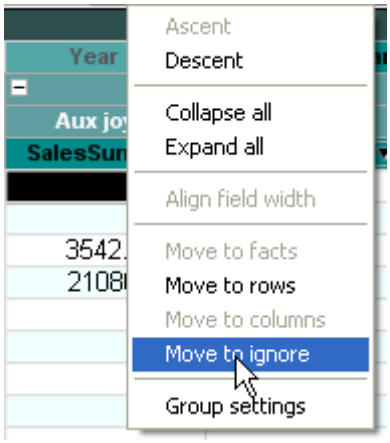


DataCubeGrid allows a user to implement visual control over a pivot table.

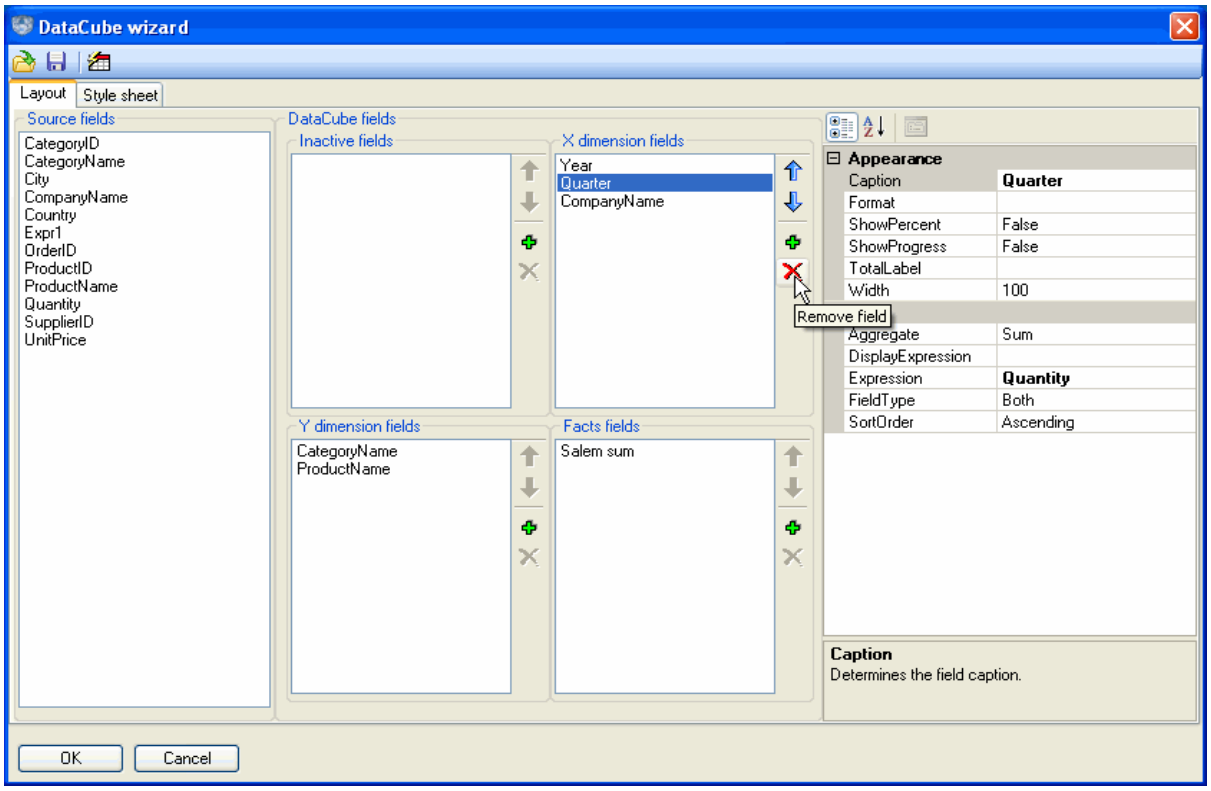
Let us save the pivot table settings to a file having specified the “Layout and sheet” file type in the saving dialogue box. Our task will be to form a pivot table to represent the sales total for a specific product, for a specific time period not taking any discounts into account. For a start, let us exclude the unnecessary fields from facts and dimensions. Here, these fields are Month, Quarter, CompanyName and Discount. We can exclude these fields from our pivot table calculation by dragging their headers into the DataCubeGrid Inactive fields’ scope.



Another way of doing that is to click on a field with the right mouse button and select the Move to ignore item from the contextual menu.

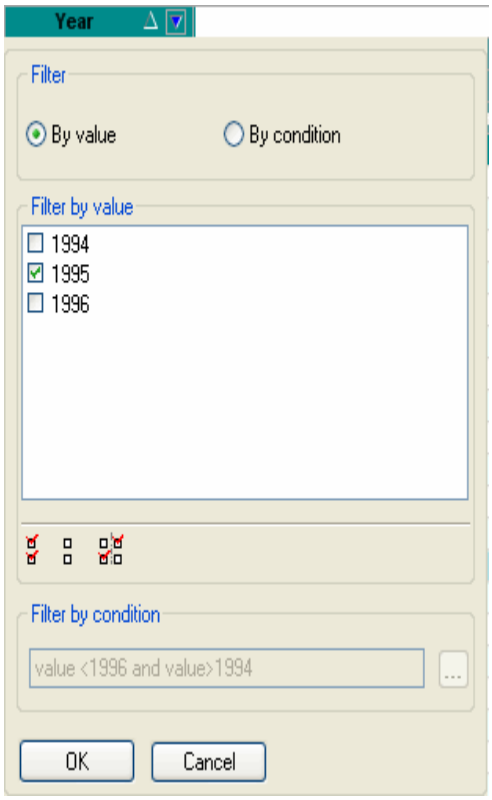


If the specified fields are no longer needed for calculations, they can be excluded from the table by means of launching the DataCube wizard designer and deleting these fields from the corresponding field lists.

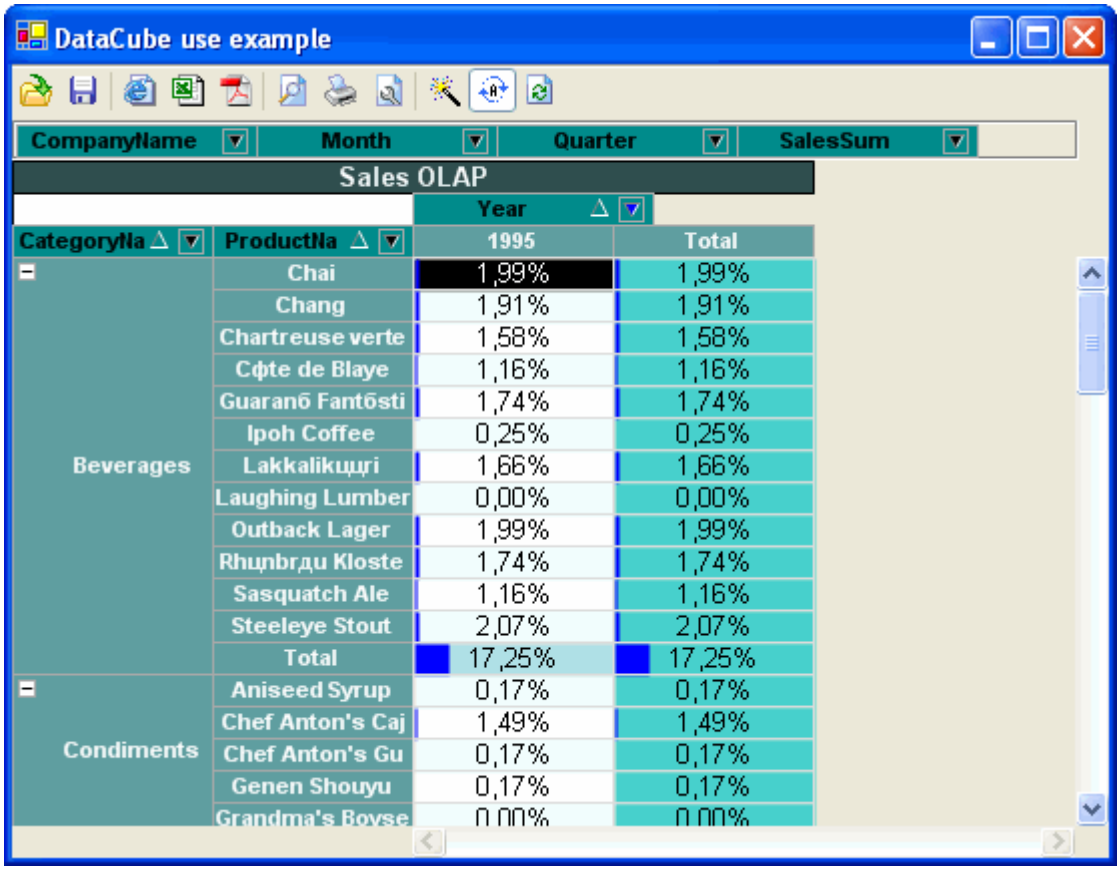


Let us also modify the Expression property for the Sales sum field by assigning the “UnitPrice * Quantity” value to it. This will expel the participation of the discount value in calculations. Close the designer by clicking “OK”.

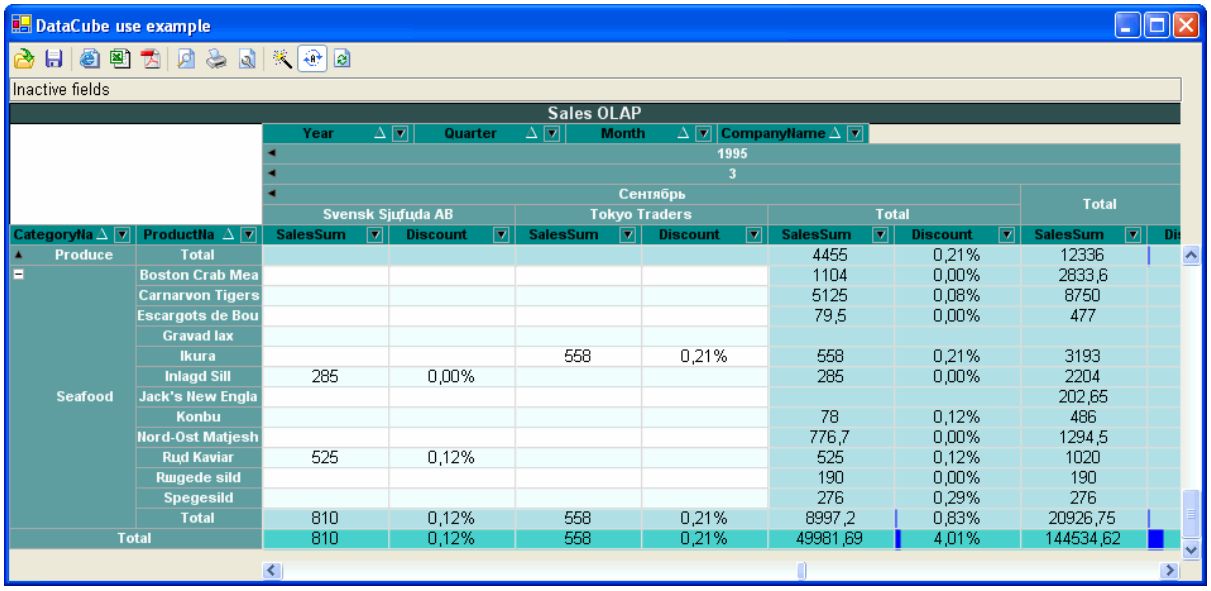
Let us use the filters capabilities to specify the pivot table formation cycle. Click the ( ) button on the Year field header to call the filter dialogue. Here, let us select filtering by condition. For example: “value < 1996 and value > 1994”.



As a result, we shall get the required pivot table.

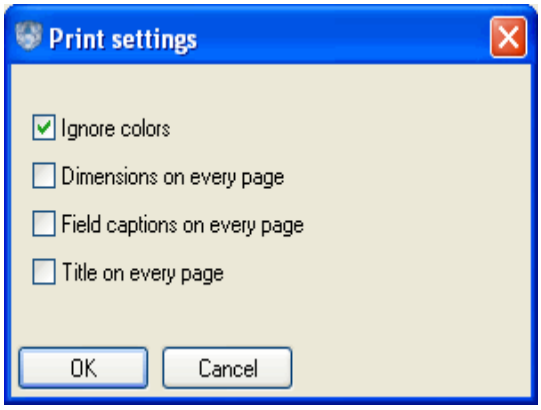


Now let us open the pivot table settings previously saved. We can make any necessary changes of our pivot table appearance, i.e. roll the unnecessary groups up and change the sorting. We can do that by using the corresponding buttons or by selecting the required item from a field's contextual menu.



This table with its current settings can be printed and/or exported into Excel table, PDF or HTML document.

The table printing occurs subject to current print settings. Let's change these settings by calling the Print settings dialogue.



We shall change the settings in the following way:

We shall set the IgnoreColor (the printing occurs ignoring the preset pivot table presentation style) and Field captions on every page (the pivot table fields captions will appear on every page of a document).

Also we shall cancel Dimensions on every page (the dimension groups will be printed only once) and Title on every page (the pivot table header will be printed only once).

After we do that, the print preview and the printing itself will be performed subject to these settings.

This example can be found in the DataCubeGridUseExample catalog.

Working with Groups

The capability of data grouping in pivot table dimensions allows getting more detailed reporting. The SharpShooter OLAP enables to group data in dimensions by default taking into consideration sequence of assigning the fields representing these data. In order to provide extra flexibility the SharpShooter OLAP grants the capability to set customs rules for groupings. Let's consider the example demonstrating this functionality. Let us assume that we need to get the detailed information on purchases by supplying countries and part of the world these countries belong to.

Let's create a WindowsForms application and add the DataCube component's object onto the form. Let us name a newly-created object salesDataCube. Let's adjust the salesDataCube object so that a pivot table layout looks like it is shown below:

Sales					
	CategoryName	ProductName			
	CategoryName1				Total
Country	ProductName1	ProductName2	ProductName3	Total	
Country1					
Country2					
Total					

As a result we should get the following pivot table:

City	Quantity	Sum				
Sales						
	CategoryNam	ProductName				
	Beverages					
Country	Chai	Chang	Chartreuse vert	Côte de Blaye	Guaraná Fantá	Ipoh Coffee
Australia						
Brazil					216,00	
Canada						
Denmark						
Finland						
France			500,40	5902,40		
Germany						
Italy						
Japan						
Netherlands						
Norway						
Singapore						1205,20
Spain						
Sweden						
UK	651,60	786,60				
USA						
Total	651,60	786,60	500,40	5902,40	216,00	1205,20

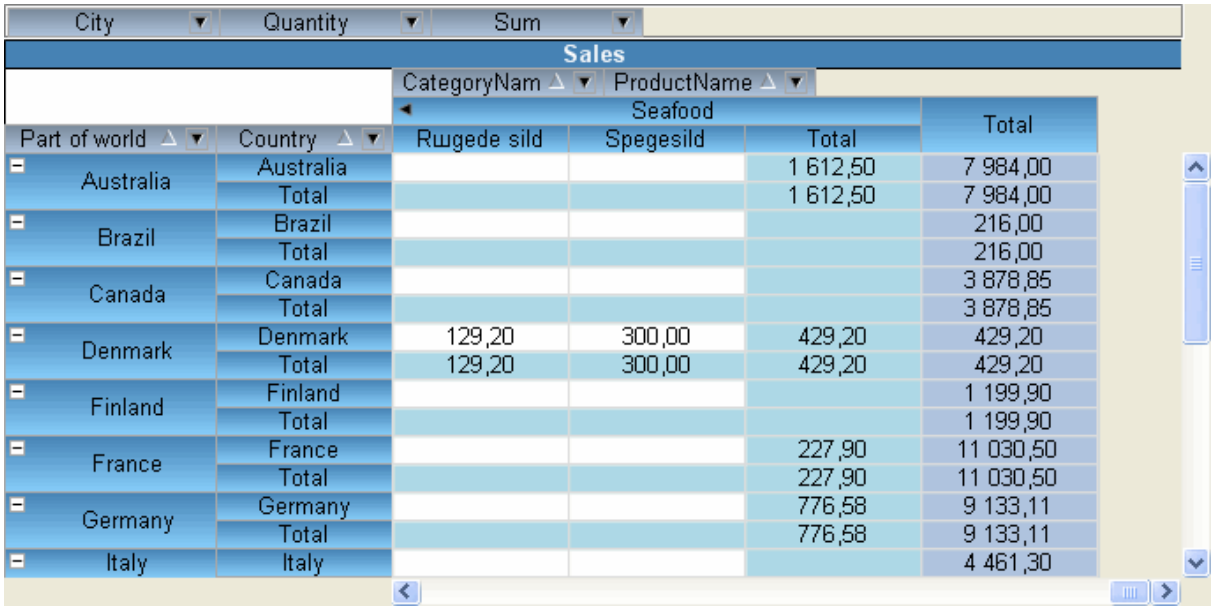
To specify the required grouping let's create one more field for a pivot table with the help of DataCube wizard. Let us assign the following properties for a newly-created field:

Caption = "Part of world"

Expression = "Country"

Appearance	
Caption	**Part of world**
Format	
ShowPercent	False
ShowProgress	False
TotalLabel	
Width	**120**
Data	
Aggregate	Sum
DisplayExpression	
Expression	**Country**
FieldType	Both
SortOrder	Ascending

We get the following pivot table

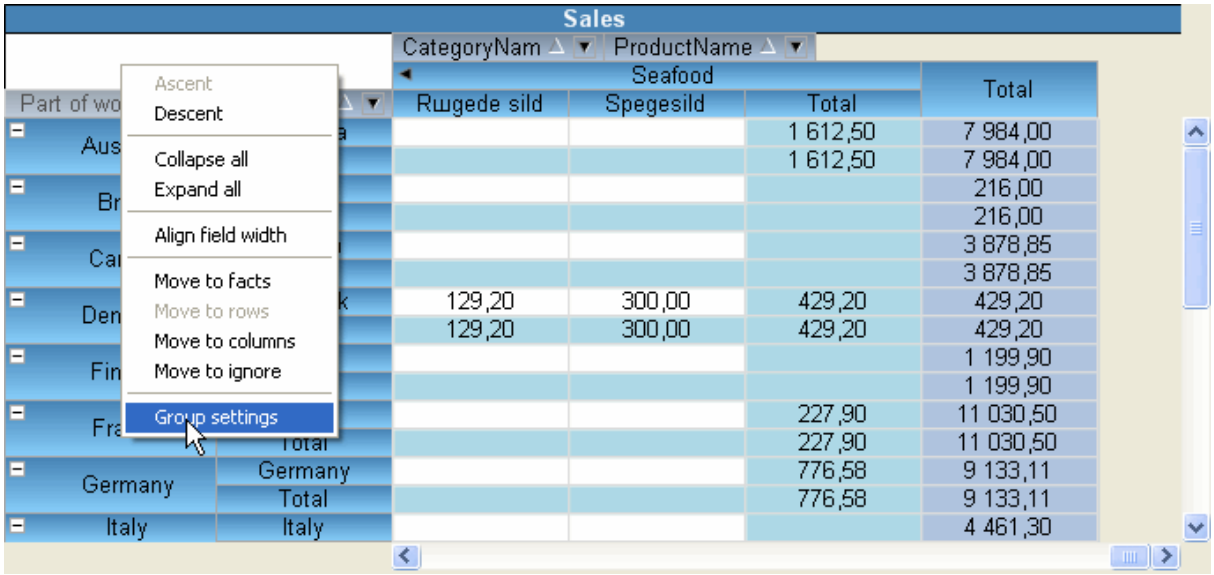


City	Quantity	Sum

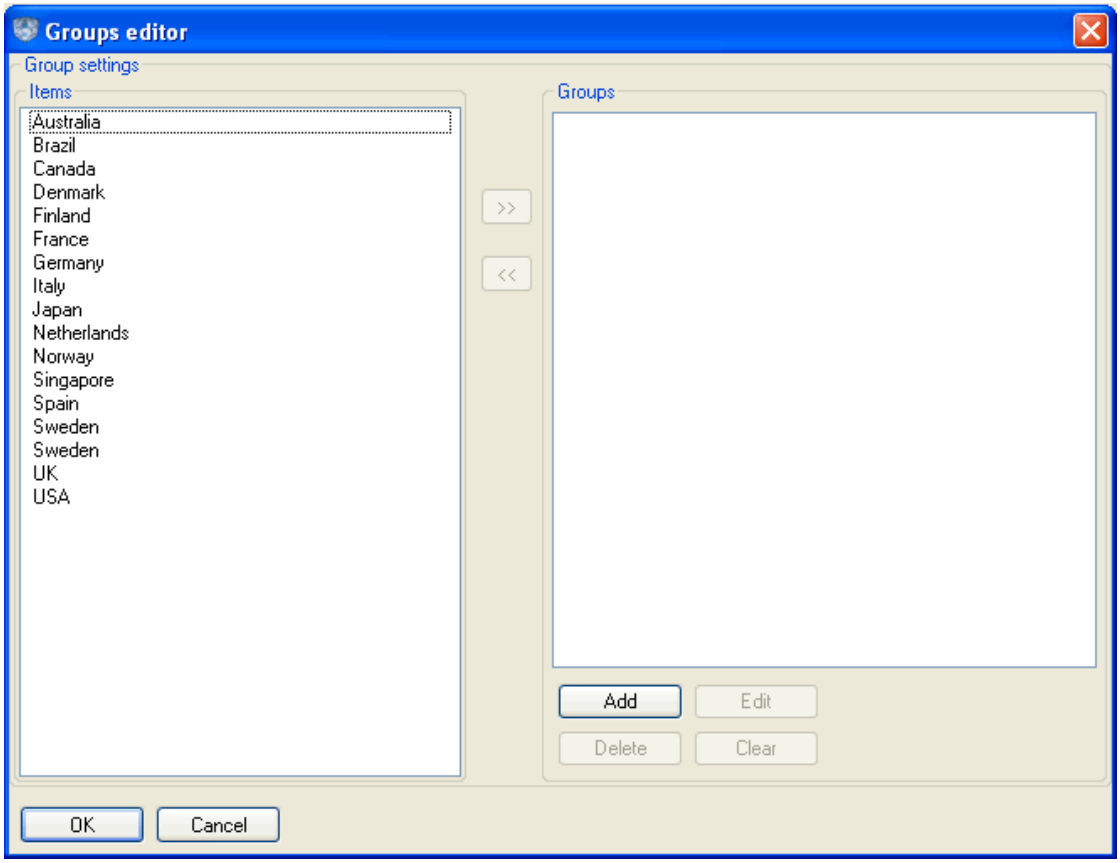
Sales

		CategoryNam / ProductName			
		Seafood			Total
Part of world	Country	Rugede sild	Spegesild	Total	
Australia	Australia			1 612,50	7 984,00
	Total			1 612,50	7 984,00
Brazil	Brazil				216,00
	Total				216,00
Canada	Canada				3 878,85
	Total				3 878,85
Denmark	Denmark	129,20	300,00	429,20	429,20
	Total	129,20	300,00	429,20	429,20
Finland	Finland				1 199,90
	Total				1 199,90
France	France			227,90	11 030,50
	Total			227,90	11 030,50
Germany	Germany			776,58	9 133,11
	Total			776,58	9 133,11
Italy	Italy				4 461,30

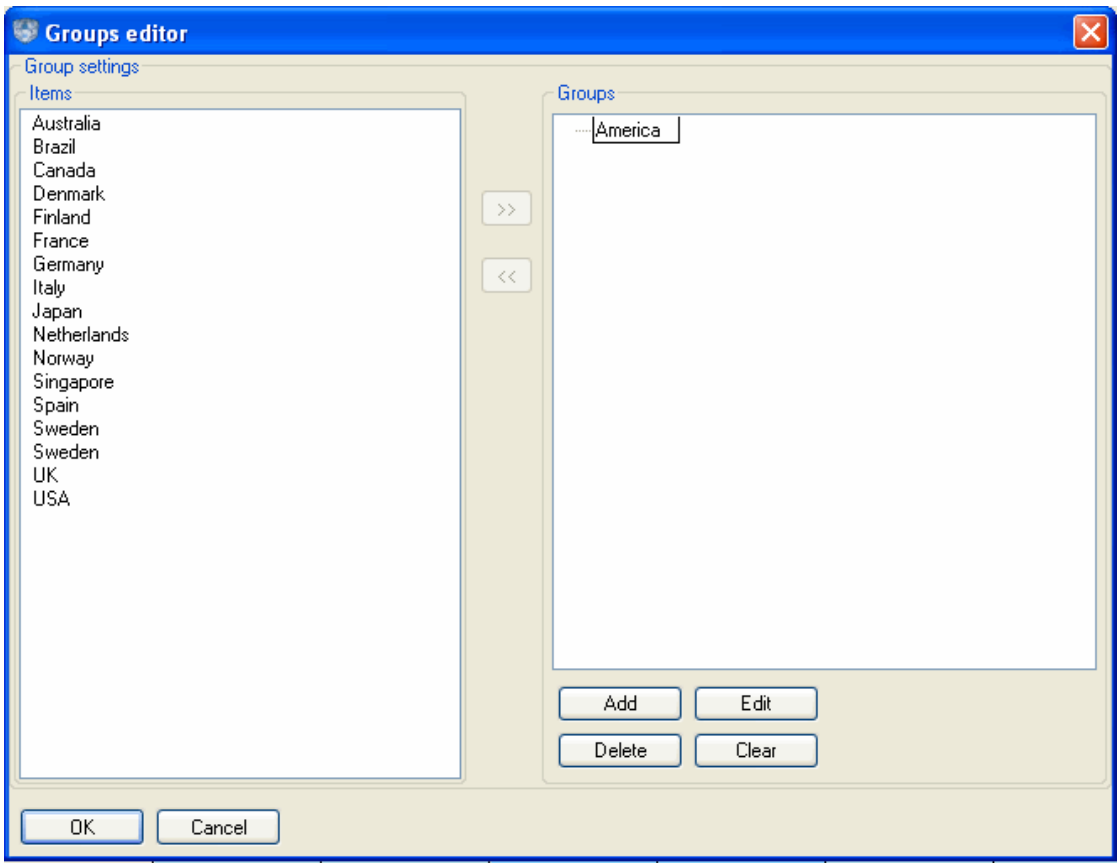
Now let's select the Group Settings item in the contextual menu of the Part of World field.



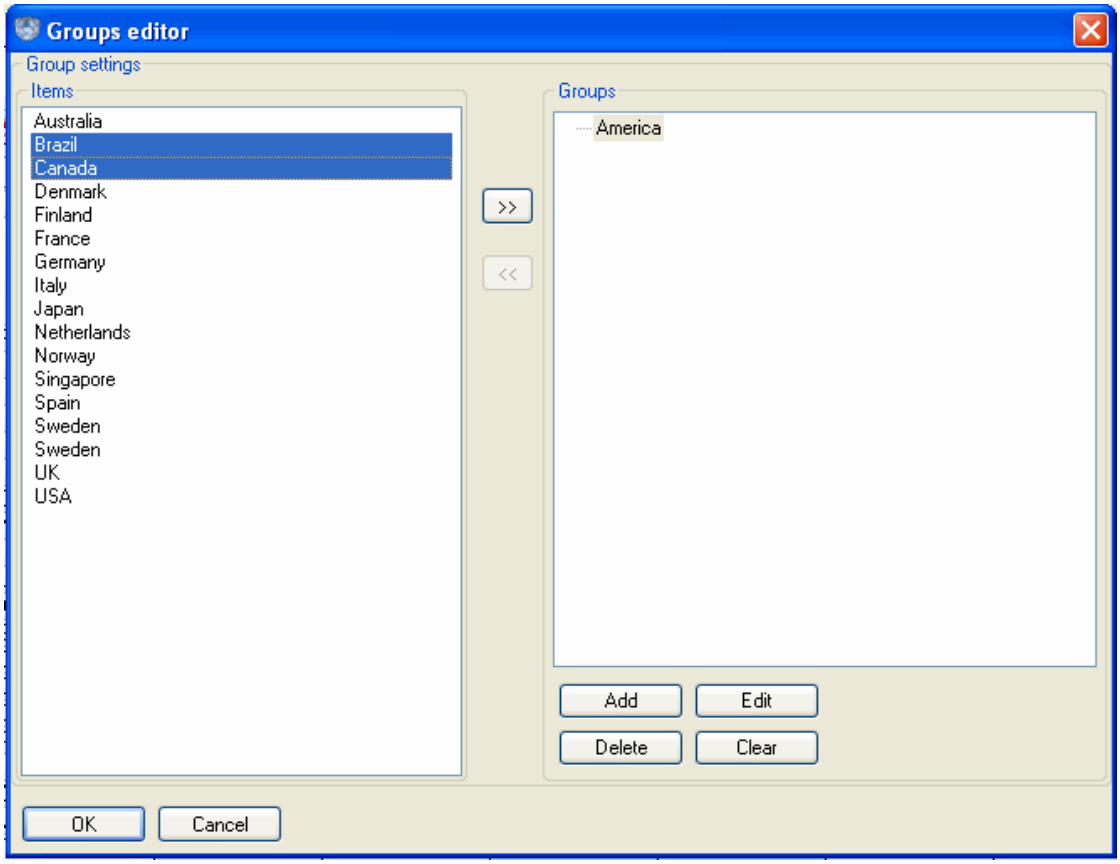
This action opens the group settings dialogue box for this field.



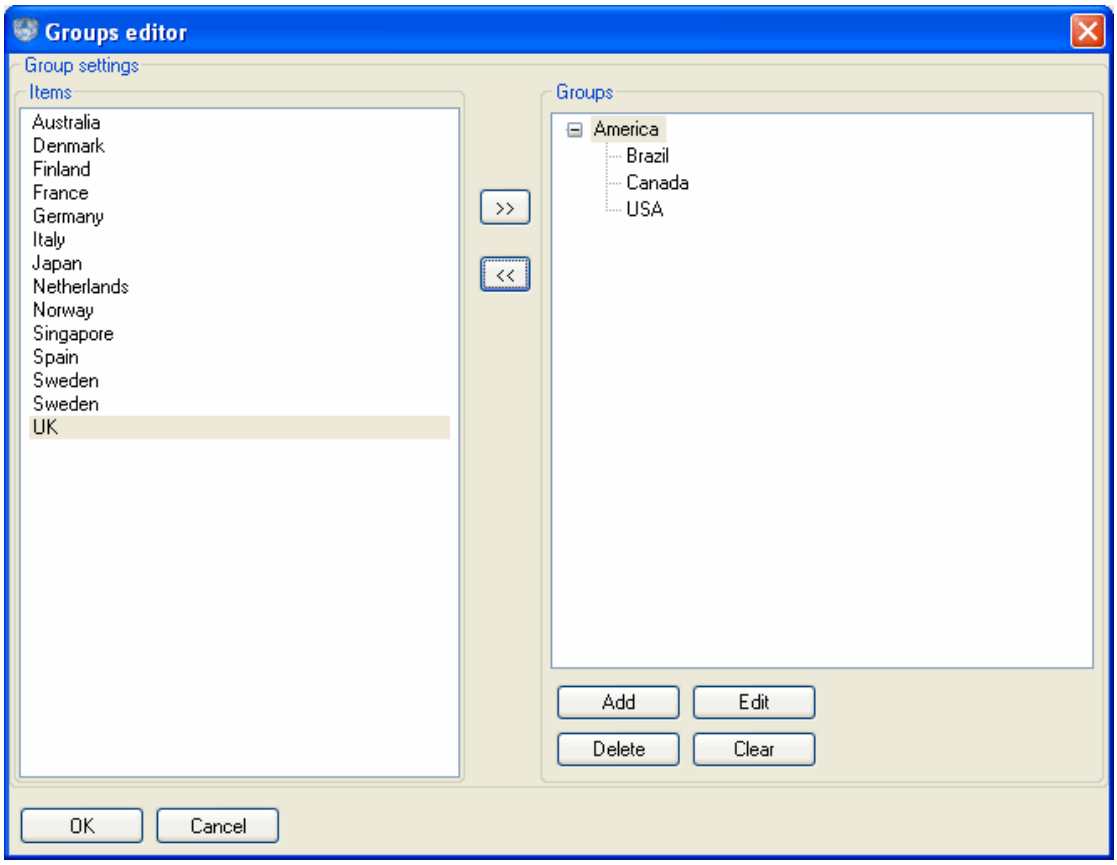
In the Items list all field values obtained as a result of its expression calculation are shown. In this particular case they are the countries. In order to group these countries by parts of the world it is necessary to add the corresponding groups. Let's press the "Add" button and specify group's name, for example America.



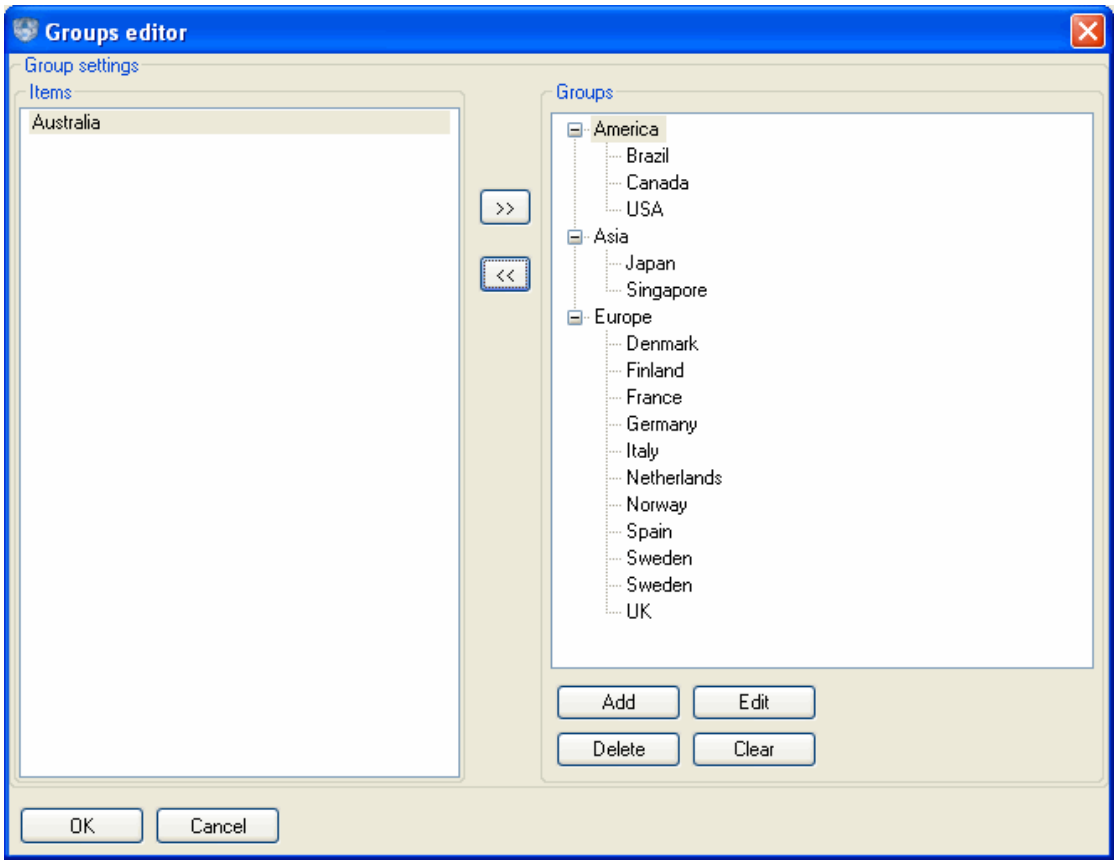
Let's add Brazil, Canada and USA into the group. To do that we should select the required countries and drag them by means of mouse to the required group as it is shown on the picture below.



After the values are added to the group they are eliminated from the list and displayed in the groups' tree like the leaves.



Let's add Asia and Europe in the same way.



Press the button "OK" to confirm the settings. As a result the pivot table will look like it is shown below:

City	Quantity	Sum

Sales

		CategoryNam / ProductName				
		Seafood				Total
Part of world	Country	Röd Kaviar	Røgede sild	Spegesild	Total	
America	Brazil					216,00
	Canada					3878,85
	USA				1138,40	5128,70
	Total				1138,40	9223,55
Asia	Japan				1210,00	2346,05
	Singapore					2036,60
	Total				1210,00	4382,65
Australia	Australia				1612,50	7984,00
	Total				1612,50	7984,00
Europe	Denmark		129,20	300,00	429,20	429,20
	Finland					1199,90
	France				227,90	11030,50
	Germany				776,58	9133,11
	Italy					4461,30
	Netherlands					263,30
	Norway					1941,20
	Spain					1261,60
	Sweden	201,00			896,20	1349,20
	UK					3841,40
	Total	201,00	129,20	300,00	2329,88	34910,71
Total		201,00	129,20	300,00	6290,78	**56500,91**

You can find this example in the CustomGrouppingExample folder.

Extra flexibility in creating group can be obtained by means of expressions.

For instance, the countries grouping shown in the previous example can be carried out by the first letters of the countries names. To do that it is necessary to create Product Group field and specify "subString(ProductName, 0, 1)" value to the Expression property. The result of such grouping is shown below.

City	Quantity	Sum

Sales

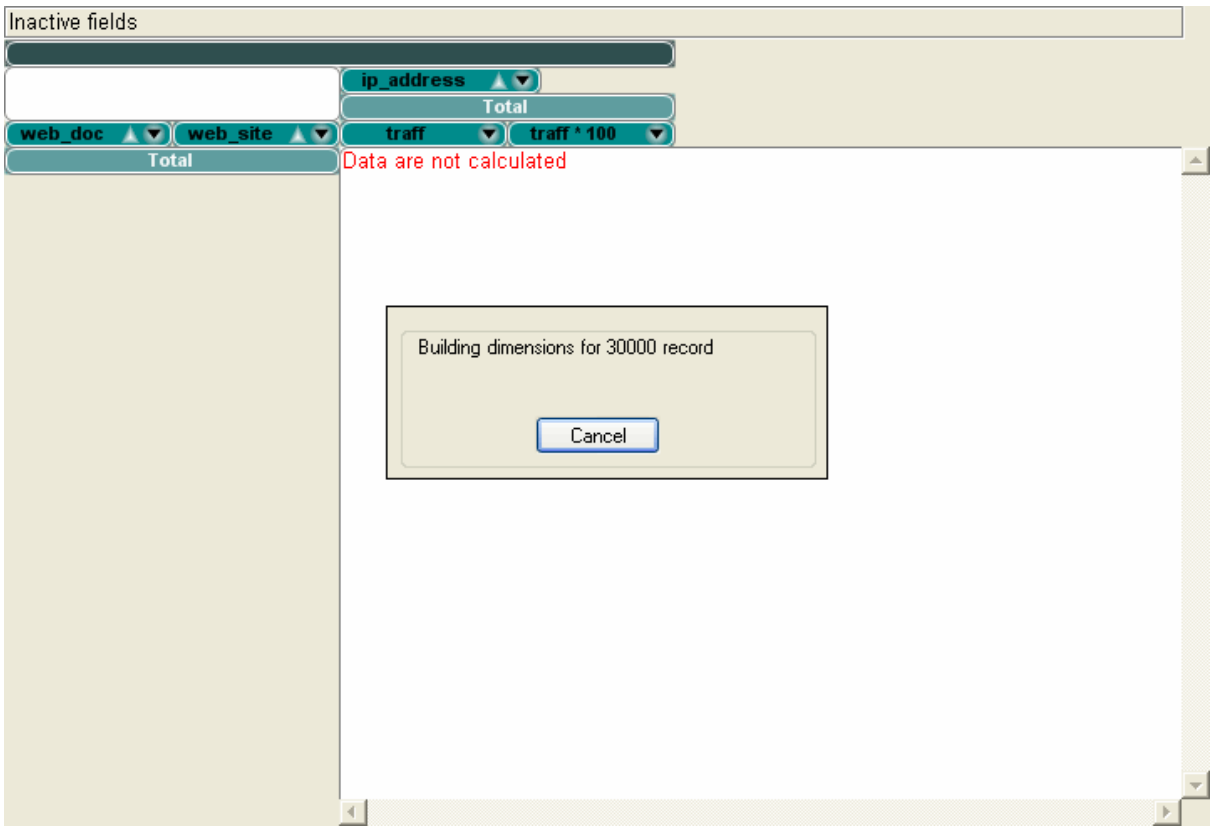
		CategoryNam / ProductName				
		Beverages				
Country grou	Country	Chai	Chang	Chartreuse vert	Côte de Blaye	Guaraná Fantá
A	Total					
B	Total					216,00
C	Total					
D	Total					
F	Total			500,40	5902,40	
N	Netherlands					
	Norway					
	Total					
S	Singapore					
	Spain					
	Sweden					
	Total					
U	UK	651,60	786,60			
	USA					
	Total	651,60	786,60			
Total		651,60	786,60	500,40	5902,40	216,00

To specify groups it is also possible to use the Range function defined in the expression language (see *The Description of Expression Language Syntax* section).

The Work with Large Body of Data

SharpShooter OLAP allows creating pivot tables using large body of data. The SharpShooter OLAP is supplied with a number of specific instruments to make the product use in such cases more convenient.

Pivot table calculation can be carried out in a standalone thread. It allows a user to interrupt calculation at any time. While working with DataCubeGrid it is carried out the following way: when pivot table recalculation is started there appears a dialogue box displaying calculation progress. The dialog box also has the "Cancel" button by pressing on which a user can correctly interrupt the calculation.



In order to on/off simultaneous calculation of the pivot table, use the DataCubeGrid.CalculationMode property.

DataCubeGrid interactivity can be temporarily turned off. So, when you specify any setting that requires table recalculation, calculation is not executed. As a result, while working with large body of data a user can adjust pivot table layout, set the required filters, group the values and only after that call forced recalculation. To make this operation available the Recalculate ( ) and Auto recalculate ( ) buttons are added on the DataCubeGrid Toolbar. The last button is responsible for turning off/on interactivity.

Custom Cell Styles

When using the DataCubeGrid component, there is the possibility to selectively change the sell style set in the style sheet for the displayed DataCube. In order to change the cell style, one should use the DrawCell event of the DataCubeGrid object.

The DrawCell event is called each time a DataCubeGrid element is drawn. At that, the event handler receives the DrawCellEventArgs object as an argument.

Use the following DrawCellEventArgs object properties:

CellStyle – the PerpetuumSoft.Olap.CellStyle type object which contains the information on the current style of a table element drawn. Change this property to specify a custom cell display style;

CellInfo –the information on a table cell. You can learn more about the PerpetuumSoft.OlapCellInfo type in the Class reference;

IsActiveCell –indicates whether the sell specified is an active table cell.

A sample of the DrawCell event handler is given below.

```
private PerpetuumSoft.Olap.CellStyle customStyle = new CellStyle(Color.Purple,
Color.Black);

private void dataCubeGrid1_DrawCell(object sender,
PerpetuumSoft.Olap.DrawCellEventArgs e)
{
      if (e.CellInfo.XFieldItem != null && e.CellInfo.YFieldItem != null)
      {
            bool contains = CanChangeStyle(e.CellInfo.XFieldItem, textBox1.Text);
            if (contains && !e.IsActiveCell)
                  e.CellStyle = customStyle;
            contains = CanChangeStyle(e.CellInfo.YFieldItem, textBox2.Text);
            if (contains && !e.IsActiveCell)
                  e.CellStyle = customStyle;
      }
}

private bool CanChangeStyle (PerpetuumSoft.Olap.FieldItem item, string text)
{
      PerpetuumSoft.Olap.FieldItem fi = item;
      while(fi != null && fi.Value != null)
      {
            if ((text.Length > 0) && (fi.Value.ToString().IndexOf(text) >= 0))
                  return true;
            fi = fi.Parent;
      }
      return false;
}
```

You can find a sample of the program which implements a custom cell style in the DataCubeGridCustomDrawing catalogue.

Custom Drawing

Along with the ability to change styles of table elements while using the DataCubeGrid component, a user can draw either one or another table element with his/her own hands. At an attempt to draw a grid cell, the DataCubeGrid.DrawGridItem event is called. So, in order to change a specific table cell drawing method, one should implement a handler for this event. The DrawGridItemEventArgs object is passed to the event handler as an argument.

Use the following DrawGridItemEventArgs object properties:

Graphics – the System.Drawing.Graphics type object actual for table drawing;

Bounds – the System.Drawing.Rectangle type object which contains the information on the position of an element within a table;

CellStyle – the PerpetuumSoft.Olap.CellStyle, type object which contains the information on the current style of a table element drawn;

CellInfo – the information on a table cell. You can learn more about the PerpetuumSoft.OlapCellInfo type in the Class reference;

Handled – if this property is set to true, the DataCubeGrid will not draw an element itself.

A sample of the DrawGridItemCell event handler is given below.

```
private void dataCubeGrid1_DrawGridItem(object sender,
PerpetuumSoft.Olap.DrawGridItemEventArgs e)
{
```

```
      if ((e.CellInfo.XFieldItem != null && e.CellInfo.YFieldItem == null &&
e.CellInfo.XFieldItem.Value.ToString().IndexOf(textBox1.Text) >= 0 &&
textBox1.Text.Length > 0)
            ||(e.CellInfo.XFieldItem == null && e.CellInfo.YFieldItem != null &&
e.CellInfo.YFieldItem.Value.ToString().IndexOf(textBox2.Text) >= 0 &&
textBox2.Text.Length > 0))
      {
            Brush b = new System.Drawing.Drawing2D.LinearGradientBrush(e.Bounds,
Color.White, Color.Purple, 30);
            e.Graphics.FillRectangle(b, e.Bounds);
            e.Graphics.DrawString(e.Value, new Font("Arial", 8, FontStyle.Bold),
Brushes.Black, e.Bounds);
            e.Handled = true;
      }
}
```

You can find a sample of the program which implements the use of custom drawing in the DataCubeGridCustomDrawing folder.

The capabilities supplied by the aforementioned events (i.e. DataCubeGrid.DrawCell and DataCubeGrid.DrawGridItem) are enough to provide your table with custom-made appearance suitable for a particular task. However, in case these instruments are not enough, you can assign a completely custom drawing to the DataCubeGrid object. To do that, you need to create a class derived from the abstract PerpetuumSoft.Olap.Painter class and specify it as the primary pivot table painter (myDataCubeGrid.Painter = myCustomPainter).

The Use and Capabilities of DataCubePrintDocument

The DataCubePrintDocument component is responsible for pivot table output onto a printing device.

Let us create a Windows Forms application and add the DataCubePrintDocument component object situated on the SharpShooter OLAP tab onto the form.

It is necessary to specify a DataCube to the `DataCubePrintDocument` in order to work with it latter. Let us create a DataSet and a DataCube by adding them onto the form and adjust them as it is shown in the "The Use and Capabilities of DataCube" section of this User Guide.

Let's specify the obtained DataCube as the DataCubePrintDocument DataCube property value.

We shall add the following code into the DataCubePrintDocument object BeginPrint event:

```
MessageBox.Show("Start of printing.");
```

And the following for BeginPrint:

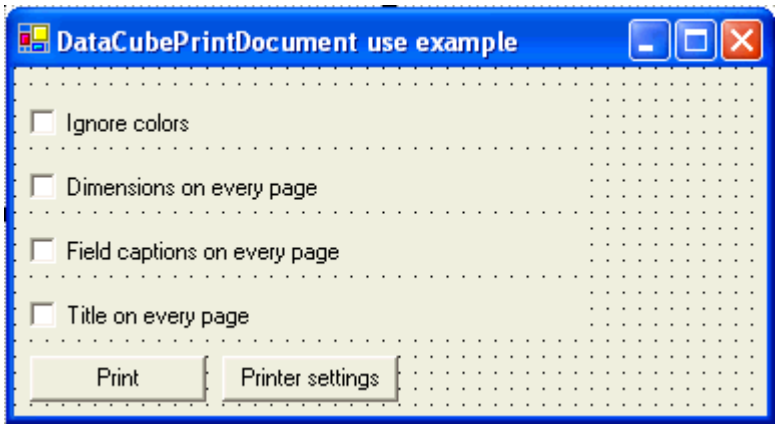
```
MessageBox.Show("End of printing.");
```

It is necessary to add the code for filling the Sales table from the DataSet into the form filling event.

We shall also add the following four System.Windows.Forms.CheckBox elements onto the form

```
isMultiColorCheckBox;
dimensionsOnEveryPageCheckBox;
titleOnEveryPageCheckBox;
fieldCaptionsOnEveryPageCheckBox;
```

And specify their properties as it is shown in the image below.



Now let us add the `System.Windows.Forms.PrintDialog` component onto the form and set the `DataCubePrintDocument` object as the Document property value.

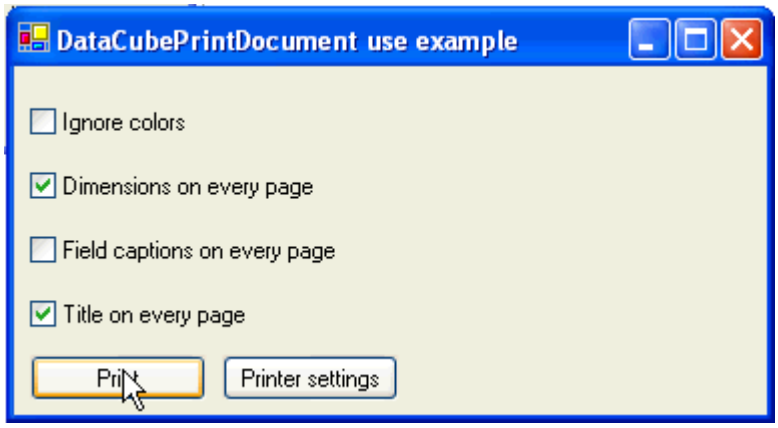
Add the `System.Windows.Forms.Button` component `printButton` object onto the form. Then write the following code in the button press event handler:

```
PerpetuumSoft.Olap.DataCubePrintSettings settings = new
PerpetuumSoft.Olap.DataCubePrintSettings();
settings.IgnoreColors = isMultiColorCheckBox.Checked;
settings.DimensionsOnEveryPage = dimensionsOnEveryPageCheckBox.Checked;
settings.TitleOnEveryPage = titleOnEveryPageCheckBox.Checked;
settings.FieldCaptionsOnEveryPage = fieldCaptionsOnEveryPageCheckBox.Checked;
dataCubePrintDocument.PrintSettings = settings;
dataCubePrintDocument.PrinterSettings = printDialog.PrinterSettings;
dataCubePrintDocument.Print();
```

Add the `System.Windows.Forms.Button` component `printerSettingsButton` object onto the form and write the following code in the button press event handler:

```
printDialog.ShowDialog();
```

Run the application.



Now let us specify all necessary pivot table print settings. Press the "Printer settings" button to specify the required printer settings. After that, click the "Print" button. The pivot table will be printed.

This example can be found in the DataCubePrintDocumentUseExample folder.

The Interaction of the SharpShooter OLAP with Third Party Components

The SharpShooter OLAP is intended for creating and utilizing cross reports based on multiple statistical data of large volumes. The SharpShooter OLAP provides an opportunity to represent the final report data not only as a table; it also allows organizing the information so to make it

available for use by third party chart components. Presently, there are lots of such components available. These are various components for charts, diagrams, reports and grids creation.

The DataCubeView component which is included in the SharpShooter OLAP package can be used for implementing an interface similar for a pivot table and graphic visualization components. DataCubeView allows creating presentations of data stored within the DataCube object. The DataCubeView component "expands" a pivot table dimension cube presenting the data as a linear table the columns of which are facts of the initial cube and a field of one of dimensions. DataCubeView makes it possible to move from one dimension into another and from one group into another within one and the same dimension.

It is more preferable to use the DataCubeView component in order to organize the presentation of pivot table data to other components. However, there are many graphic visualization components that do not correctly implement the interfaces necessary for using presentations as data sources. Due to that fact, SharpShooter OLAP supplies a tool for interaction with other components with the help of ADO.NET objects.

The DataCubeView component is able to partially rollout a pivot table presented by it into ADO.NET objects which an end user can use at his own discretion.

The Interaction with SharpShooter OLAP by Means of ADO.NET Objects

In the DataCube object a user has access to the Fill() open method. The Fill() open method implements pivot table data rollout into ADO.NET data objects.

The Fill() method is overloaded. Let us scrutinize each way of its use.

`PerpetuumSoft.Olap.DataCube.Fill(PerpetuumSoft.Olap.ViewDimension primaryDimension, System.Data.DataSet dataSet, int depth)` executes pivot table data rollout by the dimension that is defined by the primaryDimension list value (X - by columns, Y - by rows). Data rollout is conducted in the dataSet object subject to the max group nesting depth within a dimension (the depth parameter). If necessary, here takes place the creation of tables and adding these tables into the dataset. The name of each table corresponds to the name of a field responsible for group value calculation. The depth parameter value should not exceed the number of groups in a specified dimension.

`PerpetuumSoft.Olap.DataCube.Fill(PerpetuumSoft.Olap.ViewDimension primaryDimension, System.Data.DataSet dataSet, params string[] tableNames)` executes pivot table data rollout into the dataSet by the dimension that is defined by the primaryDimension list value (X - by columns, Y - by rows). At that, the tables assigned by the string name array are created in the dataset. The number of table names in the array shall not exceed the number of groups in a specified dimension.

`PerpetuumSoft.Olap.DataCube.Fill(PerpetuumSoft.Olap.ViewDimension primaryDimension, System.Data.DataSet dataSet, params System.Data.DataTable[] tables)` executes pivot table data rollout into DataTable objects contained in the tables array by the dimension that is defined by the primaryDimension list value (X - by columns, Y - by rows). The necessary requirement is the belonging of all tables to one DataSet object. The number of tables in the array shall not exceed the number of groups in a specified dimension.

A set of bound and filled tables represents the result of this or that Fill() method implementation result. The tables contain:

- group values in a specified dimension (a table column name corresponds to a name of a field responsible for group value calculation);

- fact values (a column name corresponds to a name of a field responsible for fact value calculation);

- primary and foreign key values which are used for organizing bonds between various tables.

At this method realization, the tables are being bound by the "one-to-many" type relation in such a way that the parent table corresponds to the dimension group with the nesting level lower than a group corresponding to a subordinate table.

Let us consider the pivot table shown in the image below.

			XGroup		
YGroup	Y	X	Xg1	Xg2	Total
Yg1	Y1	X1	25		25
		X2		15	15
		X3		30	30
		Total	25	45	70
	Total		25	45	70
Yg2	Y1	X2		55	55
		Total		55	55
	Y3	X3		10	10
		Total		10	10
	Total			65	65
Total			25	110	**135**

The `Fill(PerpetuumSoft.Olap.ViewDimension.Y, System.DataSet dataSet, 3) method call (the dataSet does not contain tables and bonds)` will lead to filling the dataSet object as follows:

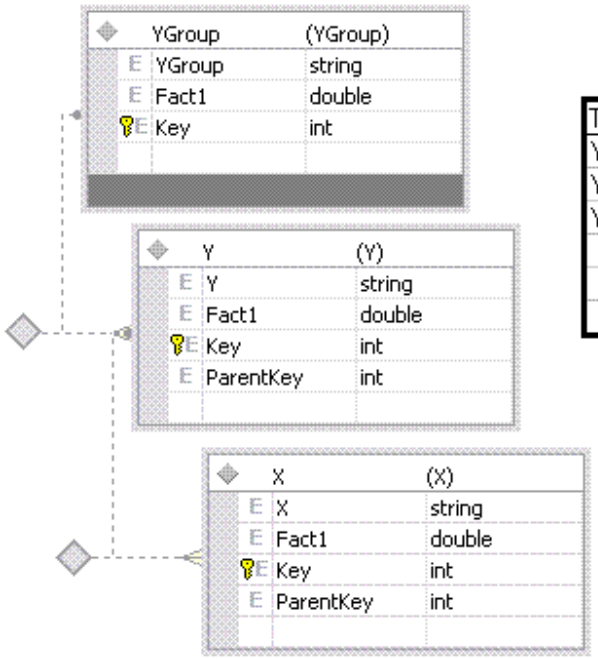


Table 'YGroup'

Ygroup	Fact1	Key
Yg1	70	1
Yg2	65	2

Table 'Y'

Y	Fact1	Key	ParentKey
Y1	70	1	1
Y2	55	2	2
Y3	10	3	2

Table 'X'

X	Fact1	Key	ParentKey
X1	25	1	1
X2	15	2	1
X3	30	3	1
X2	55	4	2
X3	10	5	3

The Interaction with SharpShooter OLAP by Means of the DataCubeView Component

The DataCubeView object creates a presentation of a table specified to it as the DataCube property value. The PrimaryDimension property defining a value by which the cube "straightening" takes place is specified to the DataCubeView object.

While using the DataCubeView component a user has the ability to jump from one dimension group (parent) to nested subgroups (children).

Attention! In order to ensure the correct work of the DataCubeView object make sure that your components correctly support the following interfaces: IList, IBindingList, ICollection, IEnumerable, ITypedList.

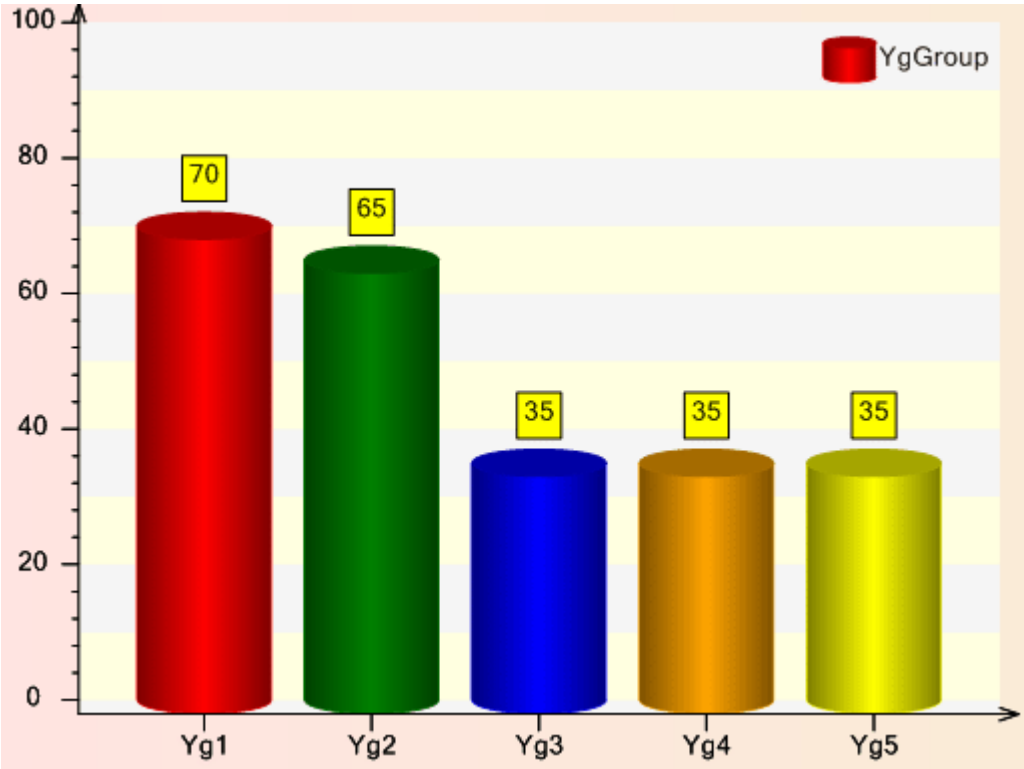
Example:

Let us add the DataCubeView object onto the form. As the DataCube property value we shall specify a pivot table. Let us examine the pivot table presented in the image below.

			XGroup		
YGroup	Y	X	Xg1	Xg2	Total
Yg1	Y1	X1	25		25
		X2		15	15
		X3		30	30
		Total	25	45	70
	Total		25	45	70
Yg2	Y1	X2		55	55
		Total		55	55
	Y3	X3		10	10
		Total		10	10
	Total			65	65
Total			25	110	**135**

Than we shall specify the PrimaryDimension property to the DataCubeView object. We shall specify Y. Now we can use the DataCubeView object for pivot table data presentation.

The result presented below is the output of Perpetuum Software SharpShooter Charts component which has used DataCubeView of a required pivot table as a data source.



If you have any questions regarding the integration with SharpShooter OLAP don't hesitate to contact us at support@perpetuumsoft.com