FastCube.Net

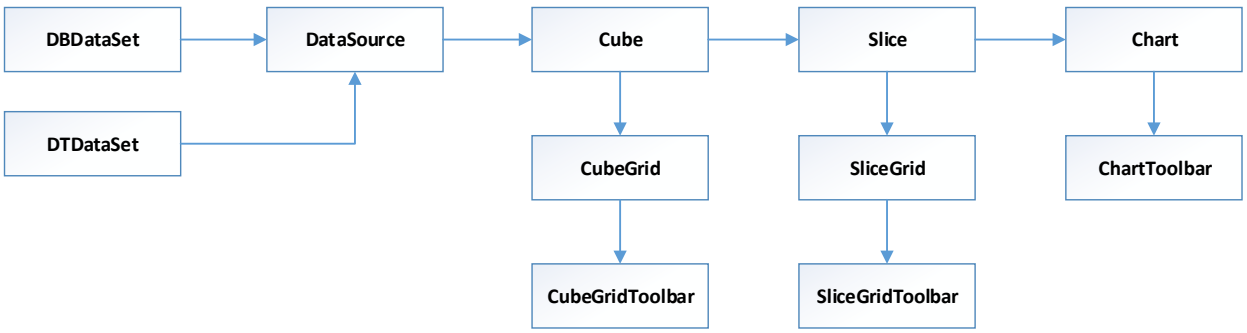
Developer's Guide

FastCube.Net Components

FastCube.Net is a library of components for Visual Studio .Net. Components can be divided into Visual and not Visual.
This guide is designed to help you create and edit a cube, slice and other components and their properties in the application code.

List of FastCube.Net components
• Cube - non-visual component that is responsible for loading the fields with unique values and rows of data from files, databases, and other sources, storing and processing structures in application memory.
• CubeGrid - visual component that displays Cube data in tabular form with the possibility of manipulation by the user.
• CubeGridToolbar - visual component, toolbar with CubeGrid management functions.
• Slice - non-visual component responsible for executing OLAP operations on the data Cube, the calculation of the measures and the preparation of views for the PivotTable and chart.
• SliceGrid - visual component that displays data Slice in the form of a PivotTable, with the possibility of manipulation by the user.
• SliceGridToolbar - visual component, a toolbar with control functions for SliceGrid.
• Char - a visual component that displays data Slice in the chart.
• ChartToolbar - visual component, a toolbar with control functions for the Chart.
• DataSource - non-visual component, the data source for the cube.
• DBDataSet - non-visual component that provides data in the DataSource from the database.
• DTDataSet - non-visual component that provides data in the DataSource from the DataTable.

Now, let's consider the scheme of the relationship of components.



Consider the diagram from left to right. The DataSource object has a property of a DataSet. The value of this property is a reference to one of two objects: DBDataSet, or DTDataSet.
A Cube object is the DataSource property. CubeGrid and Slice are associated with the Cube.
In turn, SliceGrid and Chart associated with the Slice. In the future, this scheme helps us to configure connection between components.

Cube

1) Cube Component is the Foundation of all FastCube. It stores the data in a cube and performs as the data provider and data schema for other components.

The data in the cube can be loaded from the database from the source, or from a saved cube.

Below you will find basic properties and methods of this component.

Basic properties:

Property	Description
`public bool Active`	Cube activity (data workload) - true or false. After the Cube opening the Active property is true. Only in this case visual components display data.
`public string Caption`	Cube caption. It's visible in CubeGrid.
`public bool CompressCubeFile`	Whether to use compression when saving the cube data for later use or not
`public string Description`	Description.
`public bool SkipFieldsWithErrors`	Specifies whether to ignore data source fields with errors (an unsupported type, etc.) or output Exception
`public SourceType SourceType`	The data source type specifies where to take data to populate the cube. Can be one of the following values: • Empty • DataSource • File • Stream • Manual

Main methods:

Method	Description
`public void ClearGroups()`	Clear grouping. While all groups applied will be cleared.
`public void Close()`	Close cube. The data cube and its structure will be cleared.
`public int GetFieldsCount()`	Get the number of fields in the cube.
`public int GetSourceRecordsCount()`	Get the number of source data records in the cube.
`public Utils.Variant GetSourceValue(int recordIndex, int fieldIndex)`	Get the value of data source field in specified record. Parameters – the record index and the index of the field.
`public Utils.Variant GetSourceValue(int recordIndex, CubeField cubeField)`	Get the value of data source field in specified record. Parameters – the record field index and the cube index.
`public string GetSourceValueAsString(int recordIndex, int fieldIndex)` `public string`	Get the value of a field in a data source to the specified record in the form of a string. Parameters - the index of the record and the index of the field.
`GetSourceValueAsString(int recordIndex, CubeField cubeField)`	Get the value of a field in a data source to the specified record in the form of a string. Parameters – the record index and the cube field.

`public int GetSourceValueId(int recordIndex, int fieldIndex)`	Get the unique value of a field in a data source to the specified record. Parameters - the index of the record and the index of the field.
`public int GetSourceValueId(int recordIndex, CubeField cubeField)`	Get the unique value of a field in a data source to the specified record.
`public int GetSourceValueIdAndVariant(int recordIndex, CubeField cubeField, ref Utils.Variant value)`	Receive a unique identifier of data source field value and the value itself in specified record. Parameters – the record field index and the value of the cube.
`public int GetSourceValueIdAndVariant(int recordIndex, int fieldIndex, ref Utils.Variant value)`	Receive a unique identifier of data source field value and the value itself in said recording. Parameters – the record Index, the index field and the value.
`void Load(Stream stream)`	Loads stored cube data from the stream. Cube is cleaned before loading.
`void Load(string fileName)`	Loads saved cube data from the specified file. Cube is cleaned before loading.
`public void Open()`	Open Cube. Loads cube from a data source specified in the settings.
`public void Save(Stream s, bool compress = true, object filter = null)`	Writes the cube structure and data to the stream.
`public void Save(string fileName)`	Writes the cube structure and data to the specified file.

Loading of the cube from saved file

```
FastReport.Olap.Cube.Cube cube = new FastReport.Olap.Cube.Cube();
cube.Load("C:\\Program Files (x86)\\FastReports\\FastCube.Net
Professional\\Demos\\Data\\Cubes\\2_0_sample_en1.mdc");
```

Loading of the cube from data source

```
cube.SourceType = SourceType.DataSource;
cube.DataSource = DataSource1;
cube.Open();
```

CubeGrid

2) CubeGrid Component represents a table that is filled with data from the cube. Simply saying it is the Visual representation of the cube.

Properties:

Property	Description
`public Cube.Cube Cube`	Cube reference.
`ublic CubeDataZone DataZone`	Grid data zone reference.

Methods:

Method	Description
`public bool Export(ExportBase export)`	Cube export to one of the following formats: • HTML; • DBF; • CSV; • XML; • Open Document Spreadsheet; • Excel;

	• Excel 2007. Returns success or failure
`public override string GetClipboardText()`	Returns a string representation of the selected data in grid.

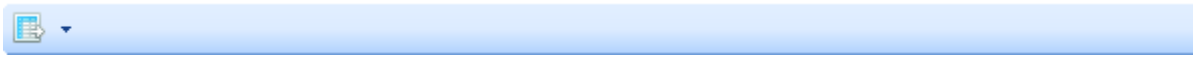
CubeGrid setup

```
CubeGrid cubeGrid = new CubeGrid();
cubeGrid.Dock = DockStyle.Fill;
cubeGrid.Parent = this;
cubeGrid.Cube = cube;
```

The following example shows how to create CubeGrid from the application code. The newly created object should be placed on the form (the Parent property), and location (the Dock property). In addition, you must specify a cube from which data should be taken.

CubeGridToolbar

CubeGridToolbar component is a toolbar that works in conjunction with CubeGrid component.



This toolbar provides only one item - export. In the table below you will find available export formats.
Properties:

Property	Discriptin
`public CubeGrid Grid`	Grid for which toolbar acts.

CubeGridToolbar setup

```
CubeGridToolbar cubeGridToolbar = new CubeGridToolbar();
cubeGridToolbar.Dock = DockStyle.Top;
cubeGridToolbar.Parent = this;
cubeGridToolbar.Grid = cubeGrid;
```

Slice

Slice component contains settings of the slice of the cube and performs operations on grouping and evaluation of data. You can manage slice through a connected component SliceGrid, from your code, or you can load a previously saved schema from a file or stream.

Properties:

Property	Description
`public bool` AutoUniqueValuesFilter	Sets instant/batch mode filter applying. Relevant to the drop-down list of unique values in the SliceGrid
`public Cube.Cube` Cube	Cube object reference.
`public FieldComparerType` FieldsOrder	The order of displaying the list of fields: ByIndex, ByName, ByCaption.

`public int ColCount`	Returns the number of columns in the pivot table.
`public bool HideColZeros`	Hide empty columns (columns in which all values are equal to 0 or empty).
`public bool HideRowZeros`	Hide empty rows (rows in which all values are equal to 0 or empty).
`public bool HideTotalForSingleValue`	Hide the total if the node contains only one value.
`public int RowCount`	Returns the number of rows in the pivot table.

`public Types.Language ScriptLanguage`	The script language. Represents an enumeration with values:
	`CSharp = 0,`
	`Vb = 1`

`public string ScriptText`	The script text.
`public PermissionSet ScriptRestrictions`	Script restriction settings.
`public SliceFields SliceFields`	The container that contains all fields of the slice.
`public AxisContainer XAxisContainer`	The container for the fields located on the x-axis. For dimensions.
`public AxisContainer YAxisContainer`	The container for the fields located on the y-axis. For dimensions.
`public FiltersContainer FiltersContainer`	The container for the fields of the filters zone.
`public MeasuresContainer MeasuresContainer`	The container, which contains measures.

Main methods:

Method	Description
`public void BeginUpdate()`	Enable update mode (batch schema changes).
`public void EndUpdate()`	Finish update mode (batch schema changes).
`public void Clear()`	Clean the slice. Reset all settings.
`public void Save(Stream stream, SliceSaveExtras extras = SliceSaveExtras.None)`	Writes the schema of the slice to the stream. SliceSaveExtras parameter specifies additional save settings (None, Filters, Groups, Charts).
`public void Save(XmlDocument doc, SliceSaveExtras extras = SliceSaveExtras.None)`	Writes the schema of the slice to the XML document. SliceSaveExtras parameter specifies additional save settings (None, Filters, Groups, Charts).
`public void Save(string fileName, SliceSaveExtras extras = SliceSaveExtras.None)`	Writes the schema of the slice to the specified file. SliceSaveExtras parameter specifies additional save settings (None, Filters, Groups, Charts).
`public bool Load(string fileName)`	Loads slice scheme from a file with the specified name. Slice schema, group settings, filters, and charts are reset before loading. On success returns true.
`public bool Load(XmlDocument doc)`	Loads slice scheme from XML document. Slice schema, group settings, filters, and charts are reset before loading. On success returns true.
`public bool Load(Stream stream)`	Loads slice scheme from a stream. Slicer schema, group settings, filters, and charts are reset before loading. On success returns true.
`public void SetColsWidth(int value)`	Set the width of all columns. The value in pixels.
`public void SetColWidth(int columnIndex, int value)`	Set the width of the specified column. The value in pixels.
`public void SetRowHeight(int rowIndex, int value)`	Set the height of the specified row. The value in pixels.
`public void SetRowsHeight(int value)`	Set the height of all rows. The value in pixels.
`public void Transpose()`	Transpose slice (swap axis).

Slice setup

```
FastReport.Olap.Slice.Slice slice1 = new FastReport.Olap.Slice.Slice();
slice1.Cube = cube;
```

Slice structure setup

Customization of the slice structure is to add fields and indicators in appropriate containers.

To add or remove fields in containers (XAxisContainer, YAxisContainer), use the following methods:

`public int AddSliceField(SliceField sliceField)`	Add a field to a slice. Returns the index of the added element.
`public int InsertSliceFieldToPosition(SliceField sliceField, int index)`	Insert slicer field at the specified position. Returns the index of the added element.
`public void RemoveAxisField(AxisField axisField)`	Remove a field from the container of the axis.
`public int AddMeasuresField()`	Adds the field "Measures" to the axis.
`public void DeleteMeasuresField()`	Removes the field "Measures" from the axis.

To add and remove measures from container (MeasuresContainer), use the following methods:

`public int AddMeasure(MeasureField measureField)`	Add a measure in the container. Returns the index of the added element.
`public void DeleteMeasure(MeasureField measureField)`	Remove measure from the container. The field is transferred to the filter area.
`public void DeleteMeasures()`	Delete all measures from the container.
`public void DeleteMeasure(int measureIndex)`	Remove measure from the container by index.
`public void InsertMeasure(MeasureField measureField, int index)`	Insert measure at the specified position.

```
//Start changing the structure of the slice
slice1.BeginUpdate();
//Add a field to a container to the X axis
slice1.YAxisContainer.AddSliceField(slice1.SliceFields.GetFieldByIndex(0));
//Insert the SliceField1 into the container in position 5
slice1.YAxisContainer.InsertSliceFieldToPosition(
                  slice1.SliceFields.GetFieldByName("SliceField1"), 5);
//Add the field "Measures" in the container X axis
slice1.XAxisContainer.AddMeasuresField();
//Add a field to the container "Measures"
slice1.MeasuresContainer.AddMeasure(new MeasureField(
                    slice1,
                    FastReport.Olap.Types.AggregateFunction.Sum,
                    slice1.SliceFields.GetFieldByIndex(1), null, null, "Measure1",
                    "Measure1", false));
//Finish updating the structure of the slice
slice1.EndUpdate();
```

Measures can be moved in the list to determine the order in which they are displayed using the public bool MoveMeasure (int fromIndex, int toIndex). Parameters: the index of the item that you want to move and the index where to move. On success returns true.

The methods SetAllVisible () and SetNoneVisible () can hide all or show all the measures.
To hide or display a specific measure, use:
slice1. MeasuresContainer. GetMeasureFieldByIndex (1). Visible = true;

SliceGrid

SliceGrid component displays the slice in the form of cross-table. This is the basic analytics tool. Allows you to customize the layout of the fields of dimensions and measures add new, sort, group, and other.

Properties:

Property	Description
`public SliceDataZone DataZone`	Data zone.
`public Slice.Slice Slice`	Specifies a slice object reference.
`public XAxisZone XAxisZone`	X-axis zone.
`public YAxisZone YAxisZone`	Y-axis zone.
`public SliceItemsZone FilterFieldsZone`	Filters zone
`public SliceItemsZone XFieldsZone`	X-axis dimensions zone.
`public SliceItemsZone YFieldsZone`	Y-axis dimensions zone.
`public SliceFieldsZone FieldsZone`	Field list zone.

Methods:

Method	Description
`public bool Export(ExportBase export)`	Export method of the report in one of the formats: • HTML; • DBF; • CSV; • XML; • Open Document Spreadsheet; • Excel; • Excel 2007.
`public void ShowFieldsEditor()`	Open a window with a list of slice fields.
`public override string GetClipboardText()`	Get text representation of the selected cells.

SliceGrid setup

```
SliceGrid sliceGrid = new SliceGrid();
sliceGrid.Dock = DockStyle.Fill;
sliceGrid.Parent = this;
sliceGrid.Slice = slice1;
```

This example shows how to create a sliceGrid in your application code. If there is no such necessity, all settings can be made in the property inspector.

SliceGridToolbar

SliceGridToolbar component represents a visualized toolbar for SliceGrid:

The toolkit is as follows:
1. Save:
• Cube;
• Scheme.
2. Open:
• Cube;
• Additional cube (addition data in the active cube);
• Scheme.
3. Clear settings;
4. Export to:
• HTML;
• DBF;
• CSV;
• XML;
• Open Document Spreadsheet;
• Excel;
• Excel 2007.
5. Transpose-swap X and Y ;
6. Hide empty rows;
7. Hide empty columns;
8. Sort rows:
• Sort by value axis;
• Sort by dimension values;
• Sort by the selected column.
9. Sort by:
• Sort by value axis;
• Sort by dimension values;
• Sort by the selected row.
10. Edit measures – Measures settings. This includes configuring conditional data selection;
11. The display format of the data - sets for the selected columns or rows;
12. List of fields - a list of all fields available in the slice;
13. The Script Editor - the script editor in C# or VB;
14. Information - information about the slice;

Properties:

Property	Description
`public SliceGrid Grid`	Grid, for which the toolbar works
`public override List<MetaItem> ToolItems`	The list of elements of the toolbar
`public string DialogsDefaultPath`	Default path for open/save dialogs

SliceGridToolbar setup

```
FastReport.Olap.Controls.SliceGridToolbar toolbar = new
FastReport.Olap.Controls.SliceGridToolbar();
toolbar.Grid = sliceGrid1;
toolbar.Parent = sliceGrid1;
toolbar.Dock = DockStyle.Top;
```

Chart

A component Chart is a chart based on data from slice. It is built automatically, you just set the Slice property.

Properties:

Property	Description
`public MarksShowStyle MarksShowStyle`	Marks display style
`public SeriesType SeriesType`	Series type (bar, pie, etc.)
`public bool SkipNullPoints`	Ignore empty points on the diagram
`public AxisDataType BaseAxisDataType`	Base axis data type
`public int MeasureFieldIndex`	The index of the measure field to show
`public int SeriesFieldCount`	The number of fields that are used as a source for Series
`public int CategoriesFieldCount`	The number of fields that are used as a source for Categories
`public AxisRegion SeriesAxis`	Axis – the source for Series.
`public AxisRegion CategoriesAxis`	Axis – the source for Categories.
`public ChartDataType DataType`	The way of retrieving data in a chart: ByAxisAxis - Categories and Series of dimensions. Used to display a single measure; ByAxisMeasures - Categories of dimensions, Series of measures; ByMeasuresAxis - categories of measures, Series of dimensions;
`public Slice.Slice Slice`	A reference to the slice object
`public bool Frozen`	Freezing allows you to fix the status of graphics, so that subsequent changes in the slice will not be reflected on it

Methods:

Method	Description
`public void BeginUpdate()`	Enable batch edit mode
`public void EndUpdate()`	Complete batch editing
`public void Load(XmlItem item)`	Load chart settings from an XML element
`public void Save(XmlItem item)`	Save chart settings to the XML element

Chart setup

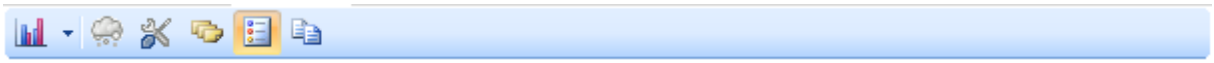
To setup Chart component set available slice to the Slice property. To create and configure the target objects in the code applications use the code.

```
Chart chart = new Chart();
chart.Dock = DockStyle.Fill;
chart.Parent = tabPage3;
chart.Slice = slice1;
```

In this case, we need to create an object, customize its appearance, bind to the parent object, and set the slice.

ChartToolbar

ChartToolbar component provides tools to customize the view of the chart.



Composition:
1) Chart style-style of the chart:

• Bar;
• Line;
• Point;
• Area;
• Pie;
• Horiz Bar (horizontal bar).

2) Frozen chart-freeze the current state of the chart;
3) Chart properties-show properties of the chart in a separate window;
4) Marks- chart marks;
5) Legend-the legend of a chart;
6) Copy-copy the chart as a picture.

Properties:

Property	Description
`public Chart Chart`	The Chart object for which this toolbar acts
`public override List<MetaItem> ToolItems`	List of the Toolbox items
`public string DialogsDefaultPath`	

ChartToolbar setup

When you configure a chart toolbar, you must set the Chart property - a chart which would be attached the toolbar.

```
ChartToolbar chartToolbar = new ChartToolbar();
chartToolbar.Dock = DockStyle.Top;
chartToolbar.Parent = this;
chartToolbar.Chart = chart;
```

DataSource

DataSource component is the source of data for the cube.

Properties:

Property	Description
`public IBaseDataSet DataSet`	Data Set-DBDataSet, or DTDataSet
`public BaseFields<DataSourceField> Fields`	List the fields of a data source

Methods:

Method	Description
`public void AddFields()`	Loads cube fields from the data source
`public bool Check(StringBuilder msg, bool skipFieldsWithErrors)`	Checks fields on duplication and other errors
`public void Close()`	Resets the data source
`public void DeleteFields()`	Clears the list of fields
`public void InitFields(bool loaded = false)`	Field initializing
`public bool Open()`	Open data source. The data will be loaded.

In the settings of this component, you must define a DataSet. This can be a DBDataSet, or DTDataSet.
Below, these components will be displayed configuration of the application code from the entire chain of receiving data.

DataSource field setup

Configuration of the fields for the data source is needed when you want to convert the field to another data type, clear the field, or you simply want to download certain fields.
DataSource.Fields property stores the description of the source. The field itself describes the object DataSourceField.
DataFieldProperties source properties depend on the type AttributeType:

```
public enum AttributeType
{
    None = 0,
    Custom = 1,
    Reference = 2,
    DateTime = 3
}
```

Property DataField describes the data type, name and the name of the field in the data source with an indication of the destination data type, name and the name of the field in the cube.
Examples:
// Load description fields
dataSource1.AddFields ();
// Change the name of the field in the cube for the field with index 2
dataSource1.Fields [2] .DataField.CubeFieldCaption = "Customer";
// Set conversion rule for the field named Poplation
dataSource1.Fields.GetByName ("Population") DataField.Convert = true.;
dataSource1.Fields.GetByName ("Population") DataField.CubeFieldType =
FastReport.Olap.Types.DataType.Int.;

DataSource attribute setup

You can split data fields to attributes. This works for fields of time and date type and fields with dependent sources.

To split fields to attributes, use the SplitProperty field. The attribute can have nested attributes. The maximum nesting level is not restricted.

For a field, you can specify a name attribute containing the value CaptionSource and the name of the attribute to sort OrderSource.

```
FastReport.Olap.Cube.DataSourceField aAtribute;


            dataSource1.Fields.GetByName("123").SplitProperty.SplitPaths =
FastReport.Olap.Types.DateTimePart.Day;
            aAtribute = new
FastReport.Olap.Cube.DataSourceField(dataSource1.Fields.GetByName("Client").SplitProperty
.Attributes.Add());
```

DBDataSet

DBDataSet - a set of data for the DataSource, obtained from the database.

Properties:

Property	Description
public IDbCommand DbCommand	Database command which contains SQL query

Methods:

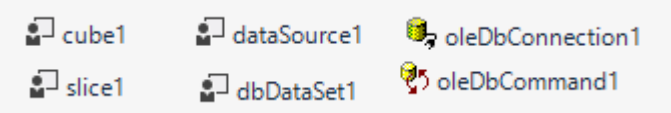
Method	Description
`public override bool AssignedSource()`	Checks for the linked source (DbCommand).
`public override void Close()`	Closes the DataSet.
`public override bool Open()`	Opens the DataSet for read operation.

Cube connection to a database setup

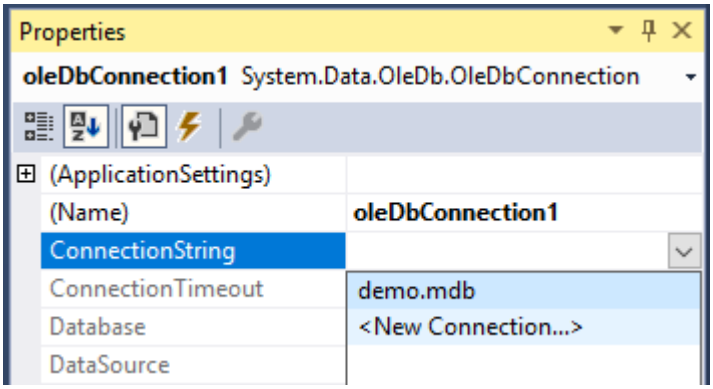
1) Using Visual components:

To configure the data connection through the DBSataSet you need to create DBCommand object using the oleDBCommand. In turn, for the oleDBCommand component you need to set the connection to the database by using the oleDBConnection.

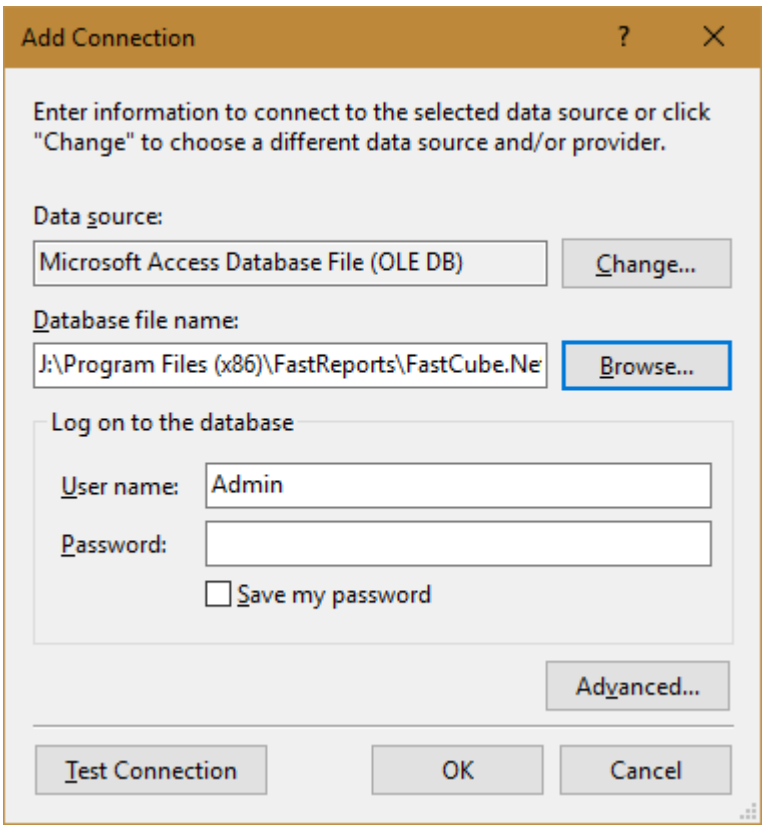
To configure a connection to a database, you need the following components:



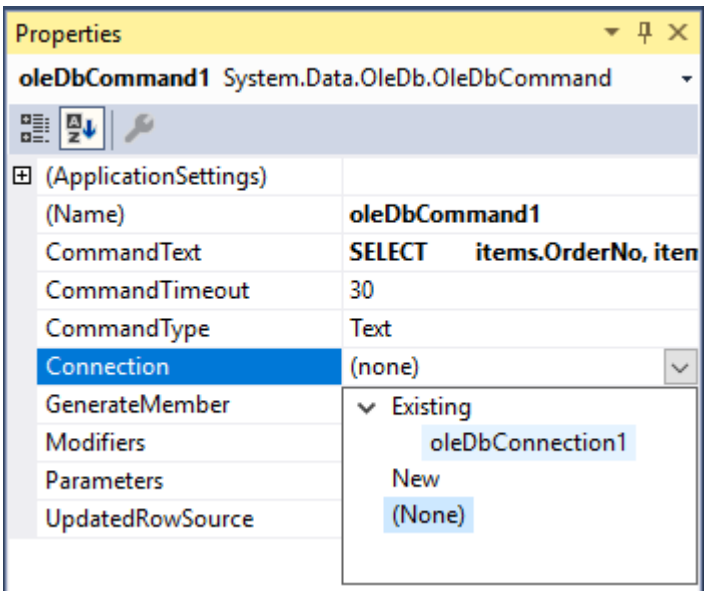
oleDBConnection settings



Let's create new connection:

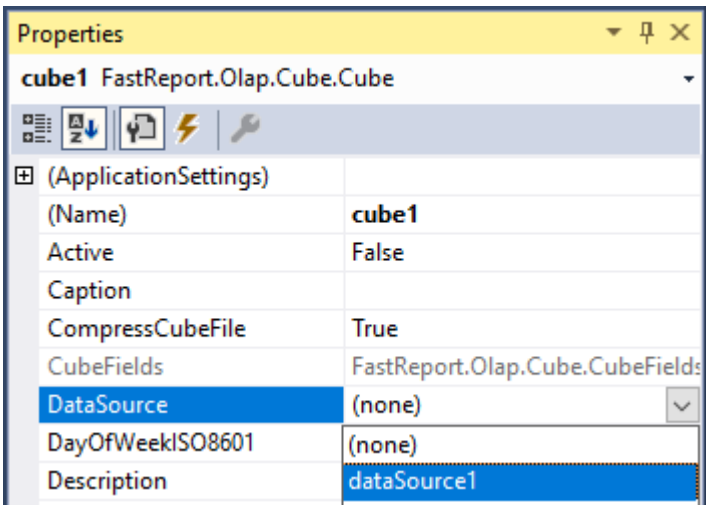


oleDBCommand settings:



It is necessary to create a command – SQL query. You need also to choose the connection to the data.

Cube settings



2) From the application code

```
OleDbCommand command = new OleDbCommand();
command.CommandText = "Select * from Sales";
dbDataSet1.DbCommand = command;
dataSource1.DataSet = dbDataSet1;
cube1.DataSource = dataSource1;
cube1.Open();
```

DTDataSet

DTDataSet - a set of data for the DataSource, obtained from a DataTable object. In turn, the DataTable may be populated with data from a database, text file or the application code.

Properties:

Property	Description
`public DataTable DataTable`	Reference to the Table

Methods:

Method	Description
`public override bool AssignedSource()`	Checks for presence of the linked source (dataTable).
`public override bool Open()`	Opens the DataSet to read operation.

Connection setup procedure to a DataTable using visual components is very simple.

Cube connection to a DataTable setup

1) Configuration of Visual components is similar to configuring a connection to a database, with the only difference that while configuring DataSet of DataSource component you should choose dtDataSet1. Accordingly, oleDBConnection and oleDBCommand are not required.
2) From your application code:

```
DataTable dataTable = new DataTable(); // Create a table and fill it with data
…
cube1.Close(); // Close the cube to unload data from it (if you have previously loaded)
dtDataSet1.DataTable = dataTable; //Assign the created table to the property DataTable
for the dateset
dataSource1.DeleteFields(); // Clear the field in the data source (if previously loaded)
```

```
dataSource1.DataSet = dtDataSet1;
cube1.Open(); // Open the cube to load data into it
```

Filter setup

Filters are used to select data for the calculation in accordance with predetermined criteria.

Methods and properties of a slice field can be used to control filter.

Examples:

```
// Begin batch edit filter
slice1.SliceFields.GetFieldByIndex(0).BeginUpdateField();
// Finish batch edit filter (apply changes)
slice1.SliceFields.GetFieldByIndex(0).EndUpdateField();
// Remove the filter mark for unique value with index 3 of the slice field with index 0
slice1.SliceFields.GetFieldByIndex(0).SetAllowedUniqueValue(3, false);
// Remove the mark from all filter values in the field ' FirstName '
slice1.SliceFields.GetFieldByName("FirstName").SetAllDenied();
// Set filtration mark for value 'Sergey' in the field 'FirstName'
slice1.SliceFields.GetFieldByName("FirstName").SetAllowedUniqueValue(slice1.SliceFields.G
etFieldByName("FirstName").CubeField.Values.GetValueIdAtValue("Sergey"), true);
// Set filtration mark for value with index 12 of the field 'FirstName'
slice1.SliceFields.GetFieldByName("FirstName").SetAllowedUniqueValue(12, true);
// Set filtration mark only for value with index 4 of the slice field with index 0
slice1.SliceFields.GetFieldByIndex(0).UniqueValueSingleIndex = 4;
// Invert filtration for values of slice field with index 0
slice1.SliceFields.GetFieldByIndex(0).InverseFilter();
// Set filtration mark of the slice field with index 0 according with the criteria specified in “range”
slice1.SliceFields.GetFieldByIndex(0).SetRangeFilter(range);
// Set the filter type “radio”
slice1.SliceFields.GetFieldByIndex(0).UVFilterType = UniqueValuesFilterType.Single;
```

Group management

You can use groups to improve the visibility of the data. You can use them to group dimension values.
You can manage groups through the methods and properties of the slice and GroupManager

property of a slice field.
Examples:

```
// Create new group “MyGroup” for the field with index 1

var groupIndex =
slice1.SliceFields.GetFieldByIndex(1).GroupsManager.CreateGroup("MyGroup");

// Add unique value to the index 1 in the groupIndex

slice1.SliceFields.GetFieldByIndex(1).GroupsManager.AddUniqueValueIdToGroup(1,
groupIndex);


// Add all the not grouped values  in a group OtherGroup


slice1.SliceFields.GetFieldByIndex(1).GroupsManager.CreateOtherGroup("OtherGroup");
```

Measure highlights

MeasureField.Highlights property defines the rules of measure highlight. Rules apply to the cells consistently and the last rule wins when more than one rule changes the same property.

All highlight rules has abstract class `CustomHighlight` as the root ancestor.

Properties:

Property	Description
`public HighlightApply ApplyTo`	Defines measure cell types to apply the rule.
`public HighlightScanDirection ScanDirection`	Defines scan direction where the neighbor cells should be considered while aggregates calculation for group.
`public virtual bool IsCustomDrawn`	Defined the need for custom drawing by the highlight class. In the negative case settings are taken from the Style object.
`public virtual string Caption`	Text representation of the rule to show in the rule editor.
`public virtual bool HideValue`	Defines the need to hide cell values.
`public CustomHighlights Owner`	Reference to Highlights list.
`public int Index`	Rule index in the Highlights list.

Methods:

Method	Description
`public virtual void BeginUpdate()`	Begin the group operation for highlight change. Highlight wont react on changes until the EndUpdate() method call. We recommend using this method if you change more than one property.
`public virtual void EndUpdate()`	End the group change operation.
`public virtual void Load(XmlItem item)`	Load the rule from the XML node.
`public virtual void Save(XmlItem item)`	Save the rule to the XML node.
`public virtual void Assign(CustomHighlight source)`	Copy settings from the source.
`public abstract bool`	Checks if highlight can be applied to the measure

`AcceptCell(MeasureCell cell)`	cell.
`public virtual Expression[] GetExpressions()`	Get array of expressions which are used by highlight. Called internally by the FastCube engine

GraphicHighlight abstract class is the actual ancestor for all the highlight classes adds the following properties and methods.

Properties:

Property	Description
`public Style Style`	Defines the cells style.

Methods:

Method	Description
`public virtual void DrawExample(Graphics g, Rectangle r)`	Draw a highlight example to the given Graphic and Rectangl. Method is used by the highlight editor for the highlight example drawing.
`public virtual void DrawValue(Graphics g, Rectangle r, MeasureCell Value, ref bool CanDrawImage, ref bool CanDrawText)`	Draw cell values. Only for Highlights with IsCustomDrawn = true.
`public virtual Style GetStyleFor(MeasureCell value)`	Get drawing style.

4.1 Highlight all cells depending on values

ContinuousHighlight class implements highlight of all cells depending on values functionality. Class allows to setup highlight not only by changing multiple properties but also by loading them from a preset.

Properties:

Property	Description
`public ContinuousHighlightKind Kind`	Highlight kind (Scales, Bar, IconSet).
`public double MinValue`	Minimal value (for scales and bar).
`public double MaxValue`	Maximal value (for scales and bar).
`public double MidValue`	Average value (for three color scale).
`public ContinuousHighlightValueType MinValueType`	Minimal value type.
`public ContinuousHighlightValueType MaxValueType`	Maximal value type.
`public ContinuousHighlightValueType MidValueType`	Average value type.
`public Color MinValueColor`	Minimal value color.
`public Color MaxValueColor`	Maximal value color.
`public Color MidValueColor`	Average value color.
`public Color BarColor`	Bar color.
`public Color FrameColor`	Bar frame color.
`public bool GradientDraw`	Use gradient fill for bar.
`public bool ShowCellValue`	Show cell values (for bar and icon set).
`public string IconSet`	Current IconSet name.
`public ImageList Images`	IconSet ImageList reference.

`public int ImageCount`	Amount of icons for IconSet.

Methods:

Method	Description
`public void DrawIconSet(Graphics g, Rectangle r, string name)`	Draw icon set.
`public int GetImageIndex(int icon)`	Get image index for the given icon index of the current IconSet.
`public void SetImageIndex(int icon, int imageIndex)`	Set image index for the given icon index of the current IconSet.
`public double GetImageValue(int icon)`	Get image value for the given icon index of the current IconSet.
`public void SetImageValue(int icon, double value)`	Set image value for the given icon index of the current IconSet.
`public ContinuousHighlightValueType GetImageValueType(int icon)`	Get image value type for the given icon index of the current IconSet.
`public void SetImageValueType(int icon, ContinuousHighlightValueType value)`	Set image value type for the given icon index of the current IconSet.
`public ContinuousHighlightIconCondition GetImageValueCondition(int icon)`	Get image value condition for the given icon index of the current IconSet.
`public void SetImageValueCondition(int icon, ContinuousHighlightIconCondition value)`	Set image value condition for the given icon index of the current IconSet.
`public string GetImageInfo(int icon)`	Get text representation for the given icon index to show in the editor.
`public void SetImageReverseOrder()`	Reverseicon order иконок of the current IconSet.
`public bool LoadPreset(ContinuousHighlightKind kind, string name)`	Load setting from a preset.

The following presets exist (depends on kind).

ContinuousHighlightKind.TwoColorScale:

- White - Red
- Red - White
- Green - White
- White - Green
- Green - Yellow
- Yellow - Green

ContinuousHighlightKind.ThreeColorScale:

- Green - Yellow - Red
- Red - Yellow - Green
- Green - White - Red
- Red - White - Green
- Blue - White - Red
- Red - White - Blue

ContinuousHighlightKind.BarChart:

- Blue
- Green
- Red

- Orange
- Light Blue
- Purple
- Blue Gradient
- Green Gradient
- Red Gradient
- Orange Gradient
- Light Blue Gradient
- Purple Gradient

ContinuousHighlightKind.IconSet:

- Arrows (Colored)
- Arrows (Gray)
- Flags (Colored)
- 3 Traffic Lights (Unrimmed)
- 3 Traffic Lights (Rimmed)
- 3 Signs
- 3 Symbols (Circled)
- 3 Symbols (Uncircled)
- 3 Stars
- 3 Triangles
- Arrows (Colored)
- 4 Arrows (Gray)
- Red To Black
- 4 Ratings
- 4 Traffic Lights
- Arrows (Colored)
- 5 Arrows (Gray)
- 5 Ratings
- 5 Quarters
- 5 Boxes

Example:

```
// Create new ContinuousHighlight for the first measure
ContinuousHighlight h = new
ContinuousHighlight(slice1.MeasuresContainer.MeasureFields[0].Highlights);
// Load from BarChart "Orange Gradient" preset
h.LoadPreset(ContinuousHighlightKind.BarChart, "Orange Gradient");

// Create new ContinuousHighlight for the second measure
ContinuousHighlight h = new
ContinuousHighlight(slice1.MeasuresContainer.MeasureFields[1].Highlights);
// Load from IconSet "3 Flags (Colored)" preset
h.LoadPreset(ContinuousHighlightKind.IconSet, "3 Flags (Colored)");
```

4.2 Highlight only cells that match condition

RangeHighlight – select cells by condition in Range property.

Property	Description

`public Range Range`	Select condition.

Example:

```
// Create new RangeHighlight with Fuchsia fill color for values > 1000
RangeHighlight h = new
RangeHighlight(slice1.MeasuresContainer.MeasureFields[0].Highlights);
// Begin group update operation
h.BeginUpdate();
// Set fill color
h.Style.FillColor = Color.Fuchsia;
// Compare by cell values
h.Range.CompareObject = RangeCompareObject.Value;
// Condition >
h.Range.ValueCondition = RangeValueCondition.Greater;
// Compare with value 1000
h.Range.LowRange = 1000;
// Finish group update operation
h.EndUpdate();
```

4.3 Highlight top and least cells

TopHighlight – selects top and least cells.

Property	Description
`public TopType TopType`	Top or least cells.
`public int TopCount`	Amount of cells.
`public bool TopPercent`	Flag sets amount as percent.

Example:

```
// Create new TopHighlight with Fuchsia color fill for the top 30% values
TopHighlight h = new TopHighlight(slice1.MeasuresContainer.MeasureFields[0].Highlights);
// Begin group update operation
h.BeginUpdate();
// Set fill color
h.Style.FillColor = Color.Fuchsia;
// Set most cells
h.TopType = TopType.Max;
// Set amount
h.TopCount = 30;
// Set that we are using percents
h.TopPercent = true;
// Finish group update operation
h.EndUpdate();
```

4.4 Highlight cells comparing with average

AverageHighlight – select values with deviation from average.

Property	Description
`public AverageValueCondition Condition`	Compare condition with average.
`public decimal StdDev`	Amount of standard deviations from average.

Example:

```
// Create new AverageHighlight with Fuchsia color fill for values greater average by 0.1
standard deviations
```

```
AverageHighlight h = new
AverageHighlight(slice1.MeasuresContainer.MeasureFields[0].Highlights);
// Begin group update operation
h.BeginUpdate();
// Set cell fill color
h.Style.FillColor = Color.Fuchsia;
// Set compare condition
h.Condition = AverageValueCondition.Above;
// Set standard deviations
h.StdDev = 0.1M;
// Finish group update operation
h.EndUpdate();
```

4.5 Highlight repeatable and unique values

UniqueHighlight – selects repeatable and unique values.

Property	Description
`public UniqueValueCondition Condition`	Compare condition.

Example:

```
// create new UniqueHighlight with Fuchsia color fill for repeatable values
UniqueHighlight h = new
UniqueHighlight(slice1.MeasuresContainer.MeasureFields[0].Highlights);
// Begin group update operation
h.BeginUpdate();
// Set fill color
h.Style.FillColor = Color.Fuchsia;
// Set compare condition
h.Condition = UniqueValueCondition.Repeatable;
// Finish group update operation
h.EndUpdate();
```

4.6 Highlight cells that match expression

ExpressionHighlight – selects values which match Expression.

Property	Description
`public string Expression`	Compare expression.

Example:

```
// create new ExpressionHighlight with Fuchsia color for expression
ExpressionHighlight h = new
ExpressionHighlight(slice1.MeasuresContainer.MeasureFields[0].Highlights);
// Begin group update operation
h.BeginUpdate();
// Set fill color
h.Style.FillColor = Color.Fuchsia;
// Set compare expression
h.Expression = "Substring([Manager], 0, 1)== \"B\"";
// End group update operation
h.EndUpdate();
```