

FastReport Studio 3.0

User manual

Edition 1.01.01

Copyright (c) 1998-2005, Fast Reports, Inc.

Table of Contents

Designer.....	5
Control keys.....	6
Mouse control.....	6
Toolbars.....	7
Designer mode bar.....	7
“Standard” toolbar.....	7
“Text” toolbar.....	8
“Frame” toolbar.....	9
“Align” toolbar.....	9
Designer options.....	10
Report settings.....	11
Page options.....	13
Creating reports.....	15
“Hello, World!” report.....	15
The “Text” object.....	17
HTML-tags in the “Text” object.....	19
Displaying expressions with the help of the “Text” object.....	20
Bands in FastReport.....	21
Databands.....	23
Data Source connection.....	24
“Customer List” report.....	24
Displaying DB fields with the help of the “Text” object	27
Aliases.....	28
Variables.....	29
“Picture” object.....	31
Report with pictures.....	32
Multi-lined text displaying.....	33
Data splitting.....	35
Text wrap of objects.....	38
Displaying data in the form of a table.....	39
Printing labels.....	42
Child-bands.....	44
Shifting objects	45
Report with two data levels (master-detail).....	47
Data linkage.....	48
Headers and footers of a data band.....	50
Report with groups.....	51
Other group features.....	53
Lines numbering.....	55
Aggregate functions.....	57
Page and report totals.....	59
Inserting aggregate function	61
The aggregate function call features.....	62

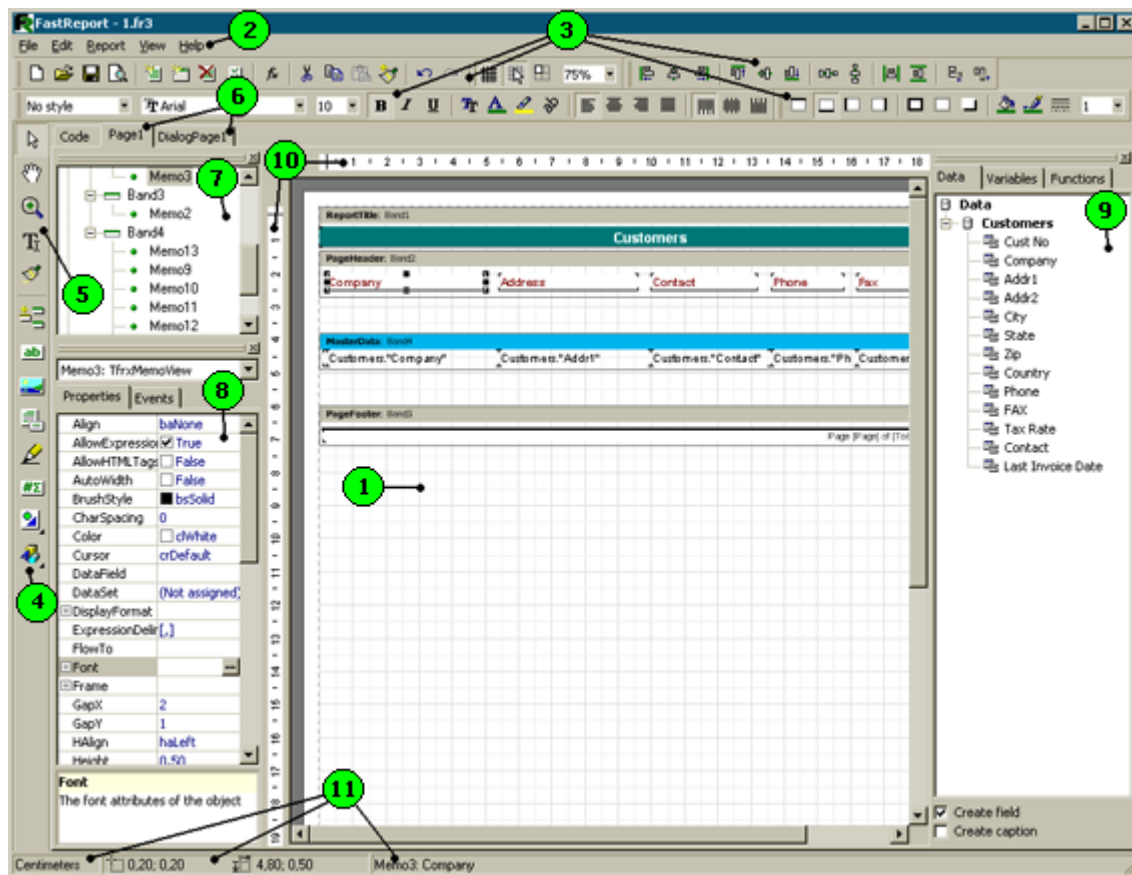
Values formatting.....	62
Inline formatting.....	63
Conditional highlighting.....	65
Show stripes.....	66
Multipage reports.....	67
Nested reports (subreports).....	69
Side-by-side subreports.....	70
Limitations on using subreports	70
PrintOnParent option.....	71
Cross-tab reports.....	73
Construct a cross-report.....	74
Using functions.....	78
Sorting values	79
Table with composite headers.....	79
Adjusting cell width	81
Font colors and highlighting.....	83
Managing a cross-table from the script.....	83
Adjusting rows/columns size.....	88
Filling a table manually.....	89
Diagrams.....	92
Limitation of number of diagram values	95
Some useful settings.....	95
Diagram with specified values.....	96
Script.....	98
Taste of script.....	99
Structure of a script	101
"Hello, World!" script.....	102
Using objects in the script.....	103
Calling the variables from the report's variables list	104
Calling the DB fields.....	105
Using aggregate functions in the script.....	106
Displaying the variable's value in a report.....	106
Events.....	107
Example of using the "OnBeforePrint" event.....	108
Printing the group's sum total in the group's header	110
"OnAfterData" event.....	115
Service objects.....	115
"Report" object.....	116
"Engine" object.....	116
"Outline" object.....	117
Using the "Engine" object.....	118
Anchors.....	121
Using the "Outline" object.....	123
"OnManualBuild" page's event.....	127
Creation of objects in the script.....	132

Dialogue forms.....	134
Controls.....	134
“Hello, World!” report.....	136
Entering parameters and transferring them into a report.....	137
Interaction of controls	137
Data access components.....	139
Components’ description.....	139
TfrxADOTable.....	140
TfrxADOQuery.....	142
TfrxADODataBase.....	143
Report constructing.....	144
Simple report of the “List” type.....	144
Report with parameters’ query.....	145

Designer

FastReport Studio is supplied with the embedded designer, which can be called from “Start” menu (“Start”->“Programs”-> «FastReports» -> «FastReport Studio» -> «FastReport Designer»). The designer provides the user with convenient tools for developing report’s appearance, along with the ability of simultaneous previewing. Designer’s interface meets up-to-date requirements. It contains several toolbars, which can be located wherever you want. The information about bar’s location is stored in a windows registry and is restored each next time you launch the program. All other designer’s settings are stored in the windows registry as well.

Using the designer in application allows a user to set the report’s appearance, as well as to edit a finished report.



In the picture, denoted with numbers:

- 1 – designer’s working space;
- 2 – menu bar;
- 3 – toolbars;
- 4 – object's toolbar;
- 5 – designer mode toolbar;

- 6 – report pages' tabs;
- 7 – “Report tree” window;
- 8 – “Object inspector” window;
- 9 – “Data tree” window. You can drag elements to a report page from this window;
- 10 – rulers. When dragging a ruler to a report page, the guide line (which objects can be adhered to) appears;
- 11 – status line.

Control keys

Keys	Description
Ctrl+O	“File Open...” menu command
Ctrl+S	“File Save” menu command
Ctrl+P	“File Preview” menu command
Ctrl+Z	“Edit Undo” menu command
Ctrl+C	“Edit Copy” menu command
Ctrl+V	“Edit Paste” menu command
Ctrl+X	“Edit Cut” menu command
Ctrl+A	“Edit Select all” menu command
Arrow	Move between objects
Del	Delete of the selected objects
Enter	Call the editor of the selected object
Shift+arrows	Modify sizes of the selected objects
Ctrl+arrows	Move the selected objects
Alt+arrows	The selected object is adhered to the nearest one in the specified direction.

Mouse control






Operation	Description
Left button	Selecting an object; pasting a new object; moving and resizing objects. For the selected objects, you can perform zooming in and out by dragging the red square in the left bottom corner of the selected objects' group.
Right button	Selected object's contextual menu.
Double-click	Object editor call. Double-clicking on the white space of a page calls the “Page Settings” dialogue box.
Mouse wheel	Scrolling a report page.
Shift + left button	Toggle object's selection.
Ctrl + right button	If you hold the left mouse button during moving a mouse, a frame appears. As soon as you release the mouse button, all the objects captured in the frame would be selected. This operation can also be performed by clicking on the blank space on the page, and moving the mouse cursor to the

	required position.
Alt + left button	If the “Text” object is selected, it edits its contents in place.

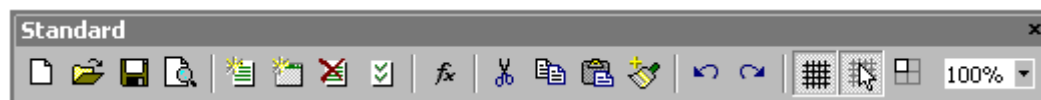
Toolbars





Designer mode bar

















The bar is integrated with the object bar and has the following buttons:

Icon	Name	Description
	Object selecting	A standard mode of operation, in which a cursor allows to select objects, modify their sizes, etc.
	Hand	Clicking on this icon allows dragging a report page.
	Zoom	Clicking on the left button doubles the zoom (adds 100%), while clicking the right one zooms out by 100%. When holding the left mouse button while dragging, the selected area would be zoomed.
	Text editor	Clicking on the “Text” object allows editing its contents right on the report page. If you hold the left mouse button when moving the cursor, the “Text” object appears in the selected place, and then its editor launches.
	Format copying	The button becomes enabled when the “Text” object is selected. When clicking on the “Text” object with the left button, it copies formatting, which has the previously selected “Text” object, into the object.

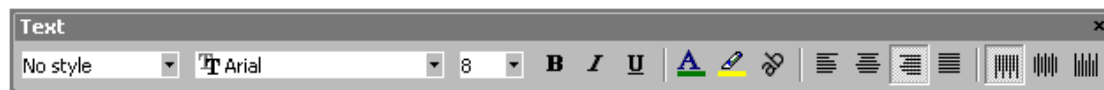
“Standard” toolbar














Icon	Name	Description
	New report	Creates a new blank report.
	Open report	Opens an existing report from the file. Hotkeys combination – “Ctrl+O.”
	Save report	Saves a report into the file. Hotkeys combination – “Ctrl+S.”
	Preview	Performs report constructing and its previewing.

		Hotkeys combination – “Ctrl+P.”
	New page	Adds a new page into the report.
	New dialogue form	Adds a new dialogue form into the report.
	Delete page	Deletes the current page.
	Page properties	Calls a dialogue with page properties.
	Variables	Calls the report variable’s editor.
	Cut	Cuts the selected objects into the clipboard. Hotkeys combination – “Ctrl+X.”
	Copy	Copies the selected objects into the clipboard. Hotkeys combination – “Ctrl+C.”
	Paste	Pastes objects from the clipboard. Hotkeys combination – “Ctrl+V.”
	Formatting pattern	Sets the “Text” object’s formatting pattern. Select the “Text” object and click on this button. All the following pasted “Text” objects will have the same formatting as specified in the pattern. To reset formatting settings, click on blank space of the page and click on this button.
	Cancel	Undo the last operation. Hotkeys combination – “Ctrl+Z.”
	Repeat	Redo the last cancelled operation. Hotkeys combination – “Ctrl+Y.”
	Show grid	Shows the grid on the page. Grid pitch can be set in designer options.
	Grid alignment	During dragging or during modifying object sizes, the coordinates/sizes are modified step-wise, according to grid pitch.
	Locate in grid crosspoints	Modifies sizes/location of the selected objects so that they would be located at grid crosspoints.
	Zoom	Sets the zoom.

“Text” toolbar













Icon	Name	Description
	Style	Allows to select a style. To define the style list, call the “Report Styles...” menu item.
	Font	Allows to select font name from the drop-down list. Stores last five fonts previously used.

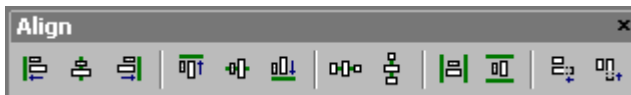
8	Font size	Allows to select font size from the drop-down list. Size can also be entered manually.
B	Bold	Enables/disables font bolding.
I	Italic	Enables/disables italics.
U	Underline	Enables/disables font underlining.
	Font color	Selects font color from the drop-down list.
	Highlighting	Shows the dialogue with highlighting attributes for the selected “Text” object.
ab	Text rotation	Allows to select text rotation.
	Left alignment	Enables text left alignment.
	Center alignment	Enables text center alignment.
	Right alignment	Enables text right alignment..
	Justify by width	Enables text justifying by width.
	Top alignment	Enables text top alignment..
	Height alignment	Enables text height alignment..
	Bottom alignment	Enables text bottom alignment.













“Frame” toolbar



Icon	Description	Description
	Top line	Enables/disables top frame line.
	Bottom line	Enables/disables bottom frame line.
	Left line	Enables/disables left frame line.
	Right line	Enables/disables right frame line.
	All lines	Enables all frame lines.
	No lines	Disables all frame lines.
	Shadow	Enables/disables shadow.
	Background color	Selects background color from the drop-down list.
	Line color	Selects line color from the drop-down list.
	Line style	Selects line style from the drop-down list.
2	Line width	Selects line width from the drop-down list.

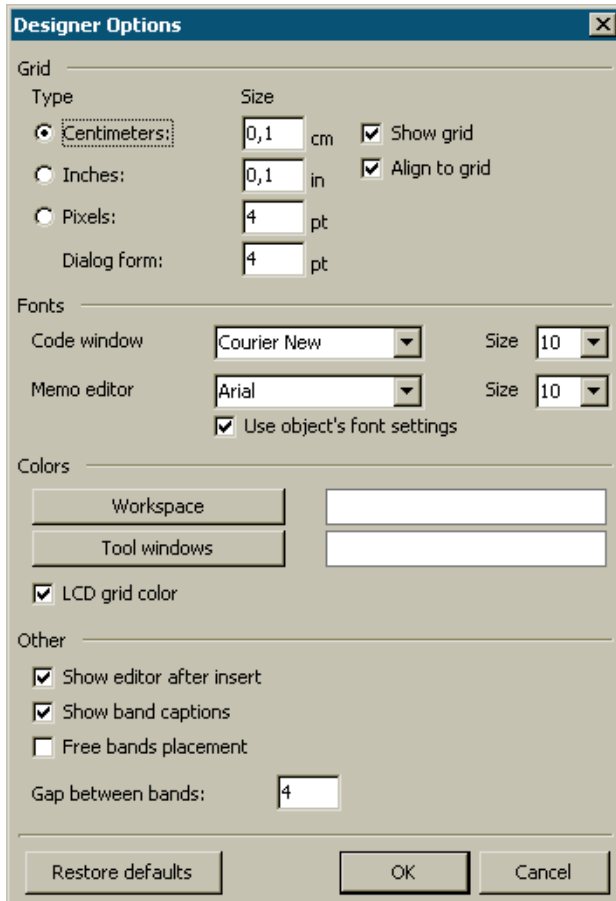
“Align” toolbar



Icon	Description
	Align left edges.
	Centre across.
	Align by right edges.
	Align top edges.
	Justify vertically.
	Align bottom edges.
	Justify by width.
	Justify by height.
	Center across in window.
	Center vertically in window.
	Set the same width as in the first selected object
	Set the same height as in the first selected object

Designer options

Set the designer options via the “View|Options...” menu command.



Designer Options

Grid

Type: ☒ Centimeters: cm ☒ Show grid
☐ Inches: in ☒ Align to grid
☐ Pixels: pt
Dialog form: pt

Fonts

Code window: Size:
Memo editor: Size:
☒ Use object's font settings

Colors

Workspace:
Tool windows:
☒ LCD grid color

Other

☒ Show editor after insert
☒ Show band captions
☐ Free bands placement
Gap between bands:

Restore defaults OK Cancel

Here you can set the necessary units (centimeters, inches, pixels), and specify grid step for each unit. You can switch units in the designer as well, via double-clicking on the left part of the status line where the current units are displayed.

You can also specify whether grid should be displayed, and align to grid. This can be performed via buttons in the “Standard” toolbar of the designer as well.

You can set a font for the code editor window and for the “Text” object editor. If the “Use object's font settings” option is enabled, the font in the text editor window would correspond with the font of the edited object.

If white background of the designer display field and service windows is not convenient for you, you can modify it via the “Workspace” and the “Tool windows” buttons. The “LCD grid color” option increases contrast of the grid lines a little, and it improves their visibility on LCD monitors.

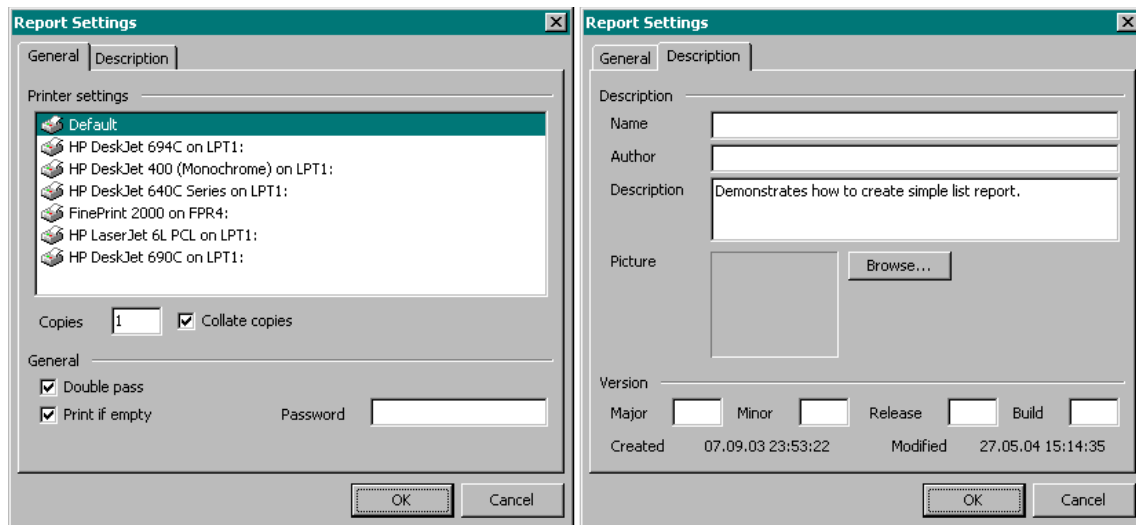
The “Show editor after insert” option controls the process of new objects inserting. If the option is enabled, its editor will be displayed each time an object is inserted. When creating a large number of blank objects, it is recommended to disable the option.

Disabling the “Show band captions” option, you can disable bands’ captions in order to save some place in a page. At that, the band’s captions would be written inside of it.

The “Free band placement” option disables snapping of bands to the page. This option is disabled by default and bands are automatically grouped in page according to their function. A gap between bands can be set in the “Gap between bands” field.

Report settings

A window with report parameters is available from the “Report|Options...” menu. A dialogue has two pages:



You can tie a report with one of the printers installed in the system. This means that report printing will be performed by the selected printer by default. This might be useful in case when there are several different printers in the system; e.g. text documents can be tied with monochrome printer, while documents with graphic - with the color one. There is the “Default printer” item in the list of printers. When this item is selected, the report will not be tied with any printer, and therefore printing will be performed by a printer, which is set as the default one.

You can also set number of report copies to be printed and specify, whether it is necessary to perform collation. The values, which a user sets in this dialogue, would be displayed in the “Print” window.

If the “Double pass” flag is selected, report’s formation will be performed in two steps. During the first pass, a report is formed, and is divided into pages, but the result is not saved anywhere. In the second pass a standard report formation with saving a result in the stream is performed.

Why two passes are necessary? Most often, this option is used in cases when in a report there is mentioning about the total number of pages in it, i.e. the information of the “Page 1 of 15” type. The total number of pages is calculated during the first pass and is available via the “TOTALPAGES” system variable. The most frequent mistake is an attempt to use this variable in a single-pass report; in this case it returns “0.”

Another field of application is performing any calculations in the first pass and displaying of results in the second pass. It might be convenient, for example, in case when it is required to display a sum in the group header, which usually is calculated and displayed in the group footer. Calculations of such kind are connected with usage of the FR embedded language.

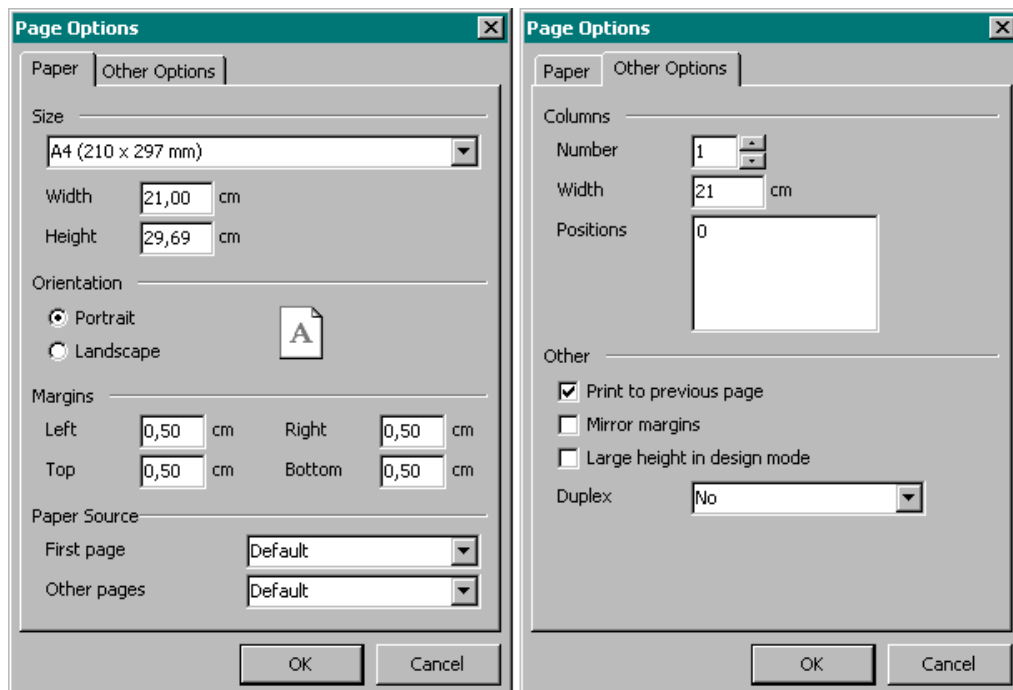
The “Print if empty” flag allows to construct a report, which contains no data lines. If this option is disabled, blank reports would not be constructed.

The “Password” field enables to set a password which will be inquired on opening a report.

Controls in the second page of the dialogue enable to set report’s description.

Page options

Page’s parameters are available via either the “File|Page settings...” menu, or double-clicking on page blank space. The dialogue has two pages:



On the first dialogue page, you can select size and alignment of paper, as well as set margins. In “Paper source” drop-down lists you can select a printer tray for the first page and the rest report pages.

You can point the number of columns for multi-column reports’ printing. The current settings are displayed in the designer.

The “Print to previous page” flag allows to print pages, beginning from blank space of the previous page. This option can be used in case when a report template consists of several pages or when printing batch (composite) reports.

The “Mirror margins” option switches right and left margins of page for even pages during previewing or printing a report.

The “Large height in design mode” option increases page’s height several times more. This feature can be useful if many bands are located in the page. At the same time, the height value of a page will not be changed during report construction.

Creating reports

In this chapter we will observe features of the FastReport Studio main components and objects, which constitute the basis of reports. One will learn about how to build simple reports, containing data from DB tables.

Report objects

A blank report in FastReport Studio is presented as a paper. At any part of the page, a user is able to allocate objects, which can display different information (such as text and/or graphics), as well as to define report's appearance. Let us describe briefly the assignment of the FastReport Studio objects, which are included into the standard package:

- “Band” object allows create a report area, which would have definite behaviour;
- “Text” object displays one or several text lines within the rectangular area;
- “Picture” object displays a graphic file in “BMP,” “JPEG,” “ICO,” “WMF,” or “EMF” format;
- “Line” object displays a horizontal or a vertical line;
- “System text” object displays service information (date, time, page number, etc), as well as aggregate values;
- “Subreport” object allows inserting an additional report inside the basic one;
- the objects of “Draw” category represent different geometrical figures (diagonal line, rectangle, rounded rectangle, ellipse, triangle, and diamond);
- “Chart” object displays data in the form of diagrams of different kinds (circle diagram, histogram, and so on);
- “RichText” object displays a formatted text in Rich Text Format (RTF);
- “CheckBox” object displays a checkbox with either a check or a cross;
- “Barcode” object displays data in the form of barcode (many different types of barcodes are available);
- “OLE” object is able to display any object using OLE technology.

The basic objects you would mostly need to work with are the “Band” and “Text” objects. You will acquaint with their capabilities in detail further in this chapter.

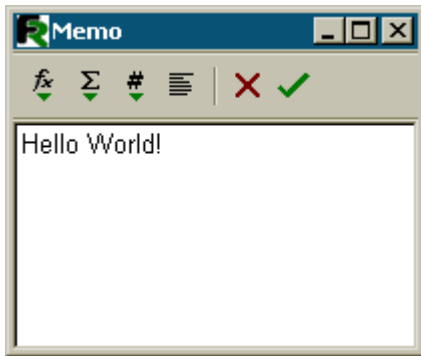
“Hello, World!” report

Thus, we create a new report designer by selecting the “File|New” report. This is all we would need to create our first report.

The report will contain one inscription only (“Hello, World!”). Open the designer by doubleclicking on the “TfrxReport” component (otherwise you can select the “Design Report...” item in the component pop-up menu). After that, click the “Text” button in the “Objects” designer panel. Move the mouse cursor to the necessary place on the page, and click again. The object thus has been inserted.



The text editor window will be displayed right away; if it does not appear (this can be set in the designer settings), then doubleclick the object. Enter the “Hello, World!” text, and then click the “OK” button.

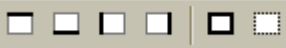



The report is created. To preview it, either select the "File|Preview" menu item, or click the corresponding button in the toolbar. The preview window containing an only report page with the “Hello, World!” inscription will appear. This report can be printed out, saved to a file, or exported to one of the supported formats.

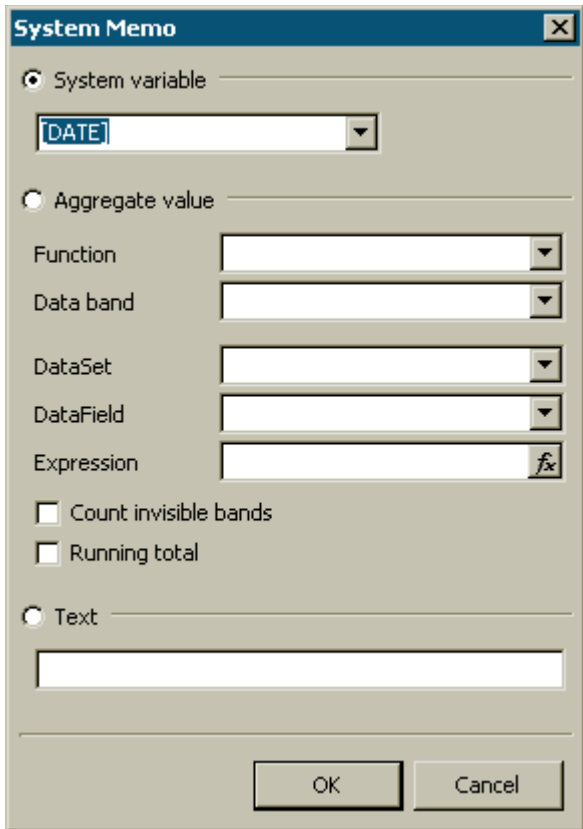
We have done all the work in FastReport Studio designer without writing a code line.

Let us complicate our first report a little. Let the text (“Hello, World!”) be displayed in bold letters with yellow background and a frame. Also let us make the current date to be displayed next to the text.

Close Preview window and select an object with a text with the help of the mouse. Note that some buttons on toolbars become active. Find the “Bold” button in the “Text” toolbar, and click on it. To enable the frame, click on “All frame lines”. If necessary,

some frame lines can be disabled by using  buttons. A user can set line color, its width, and style if necessary. We find the “Fill color”  button and select yellow from the dropped list.

The easiest way to display a date is to use the “System text” object. We add it to the page in exactly the same way as we have done it with the first object. Then we select “System variable” in the editor window and “[DATE]” in the dropped list below.

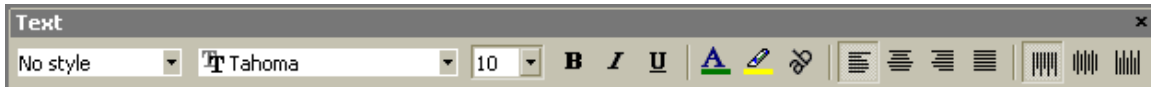


Close the editor by pressing the “OK” button, and start the report to see the result.

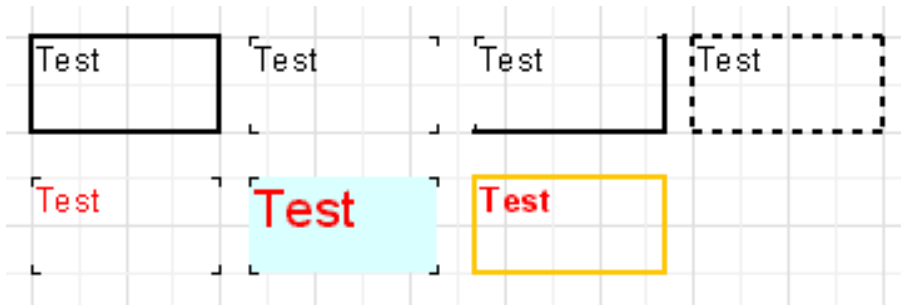


The “Text” object

The “Text” object possesses manifold features. Now we already know that it allows to display a text, a frame, and filling. A text can be displayed using any font of any size and style. All the parameters can be set visually, with the help of the toolbars:



Here are some examples of text design:

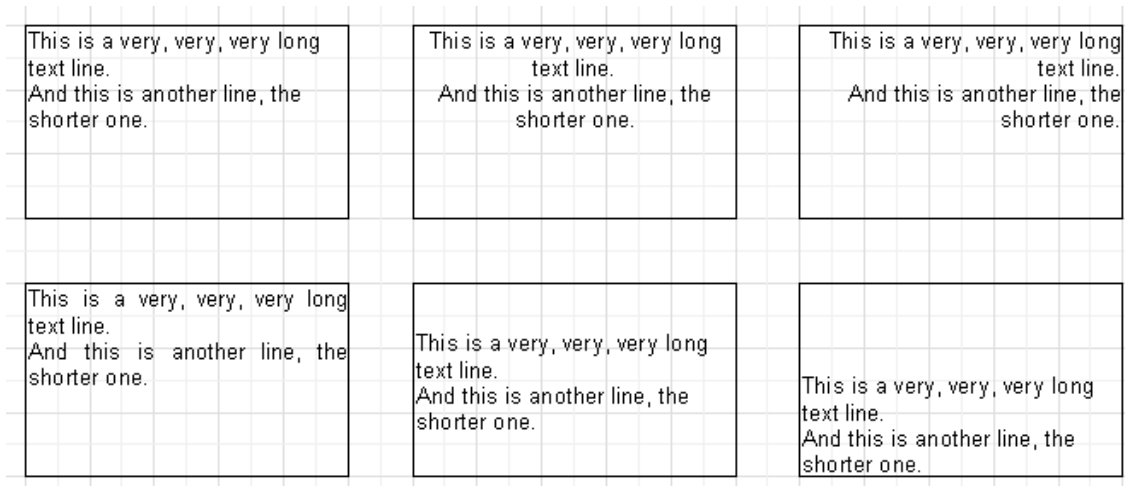



Now let us get acquainted with other features of this basic object. As an example, let us create a new text object and put two lines into it:

*This is a very, very, very long text line.
And this is another line, the shorter one.*

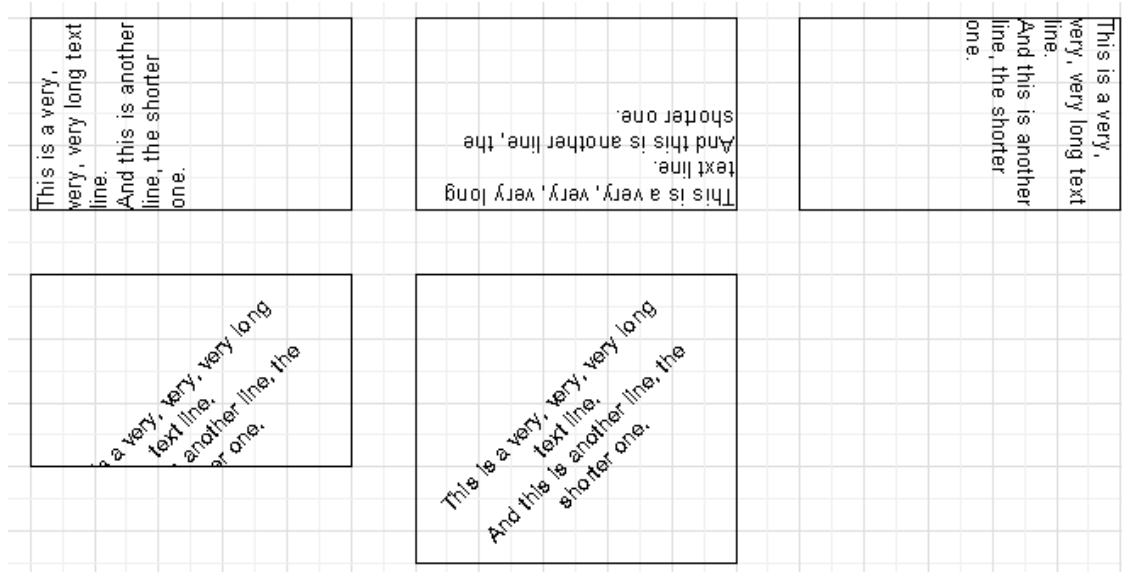
Let us enable the object frame, and then resize the object up to 9x3 cm with the help of the mouse. We see that the object can display not only a single-lined text, but several lines as well. Now let us reduce the object width up to 5cm. It is obvious that long lines did not find room in the object and therefore were wrapped. This happens due to the “WordWrap” object property. If it is disabled (either in inspector or in the object context menu), the long lines will be simply cut off.

Now let us check up, how the text alignment inside the object works. Alignment buttons are located in the “Text” toolbar and allow to set horizontal or vertical text alignment. Pay attention to the “Block Align” button; this button allows to align the paragraph on both object edges. When performing this operation, the “WordWrap” option must be enabled.



A whole text can be rotated at any angle within the limit of 0..360 degrees. The  button in the “Text” toolbar allows to quickly rotate the text at 45, 90, 180 and 270 degrees. If you wish to rotate the text at any other value, use the object inspector. The

“Rotation” property sets the required angle. When rotating a text, setting values other than 90, 180, 270 the text can exceed bounds of the object, as in our case (see the picture below). Let us increase object height a little, so that the text would fit the object.



Let us briefly dwell on some “Text” object properties, which influence its appearance. Most of these properties are available in the object inspector only:

- BrushStyle – type of object filling;
- CharSpacing – space between symbols in pixels;
- GapX, GapY – text indents from object’s left and top boundaries (in pixels);
- LineSpacing – space between lines (in pixels);
- ParagraphGap – the first paragraph line indent (in pixels).

HTML-tags in the “Text” object

Yes, this object does understand some simplest HTML tags. Tags can be located inside the text of an object. Tags are disabled by default; to enable them, either select the “Allow HTML tags” item in the object context menu, or enable the “AllowHTMLTags” property in the object inspector. Here is the list of supported tags:

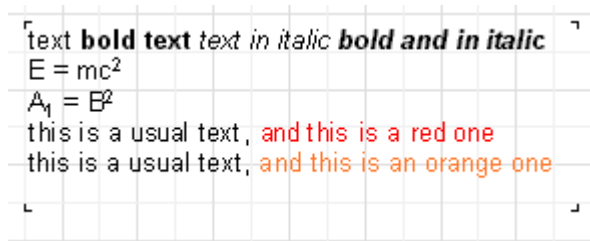
- - bold text
- <i> - text in italic
- <u> - underlined text
- <sub> - subscript
- <sup> - superscript
- - font color

As one may notice, not too many tags are supported, but it is rather enough for the

majority of applications. It is impossible to modify font size and name; otherwise the text-rendering unit in FastReport must be appreciably complicated.

The following examples demonstrate how these tags can be used.

```
text <b>bold text</b> <i>text in italic</i> <b><i>bold and in italic</i></b></i>
E = mc<sup>2</sup>
A<sub>1</sub> = B<sup>2</sup>
this is a usual text, <font color=red>and this is a red one</font>
this is a usual text, <font color="#FF8030">and this is an orange one</font>
```



Displaying expressions with the help of the “Text” object

One of the most important features of this universal object is a possibility to display not only a static text, but expressions as well. At the same time, expressions can be located in the object together with a text. Let us examine a simple example of how it can be performed.

In the previous section, we have already made a report, which printed the “Hello, World” line and displayed a current date. To perform that, we had to allocate two objects in the report. Thus, one of them contained a greeting text, while the other one contained the “DATE” system variable. However, to display both a line and a date, we can use the “Text” object only. To accomplish this, we would need to put a line into the object, and this would look something like follows:

Hello, World! Today is [DATE].

Thus, when running the report, we can get something like follows:

Hello, World! Today is 01.01.2004.

What lead to such result? During FastReport report construction, it met in the text an expression enclosed in square brackets, calculated it and inserted the received value back into the text (having removed brackets, of course). The “Text” object can contain any number of expressions, together with a usual text. Both single variables and expressions can be enclosed in brackets (for example, $[1+2*(3+4)]$). Any constants, variables, functions, and DB fields can be used in expressions. We will observe these features later, further in the chapter.

Thus, FastReport automatically recognizes expressions enclosed in square brackets in the text. But what should be done if our object contains square brackets, and we do not want them to be considered as expressions? For example, if we need to display such text as following:

a[1] := 10

FastReport considers [1] as an expression, and displays the following:

a1 := 10

that is not convenient to us, of course. One of the ways to avoid such situation is to disable the expression. Just disable the “AllowExpressions” property (“Allow Expressions” in the context menu), therefore all the expressions in the text will be ignored. In our example, FastReport would display exactly what we need:

a[1] := 10

Sometimes a text is required to contain both an expression and a text in square brackets, for example:

a[1] := [myVar]

Disabling of an expression allows to display square brackets in the required place, but it also disables handling of expression. In this case, FastReport allows to create another set of symbols, designating the expression. The “ExpressionDelimiters” object property, which is equal to “[,]” is responsible for it. In our case, the user can use angular brackets for the expressions, instead of square ones:

a[1] := <myVar>

At the same time, the “<,>” value must be set in the “ExpressionDelimiters” property. As you can see, the comma divides opening and closing symbols. There is one limitation however: the opening and closing symbols cannot be similar, so “%,%” will not work. One can set several symbols, for example “<%,%>” Thus, our example will look as follows:

a[1] := <%myVar%>

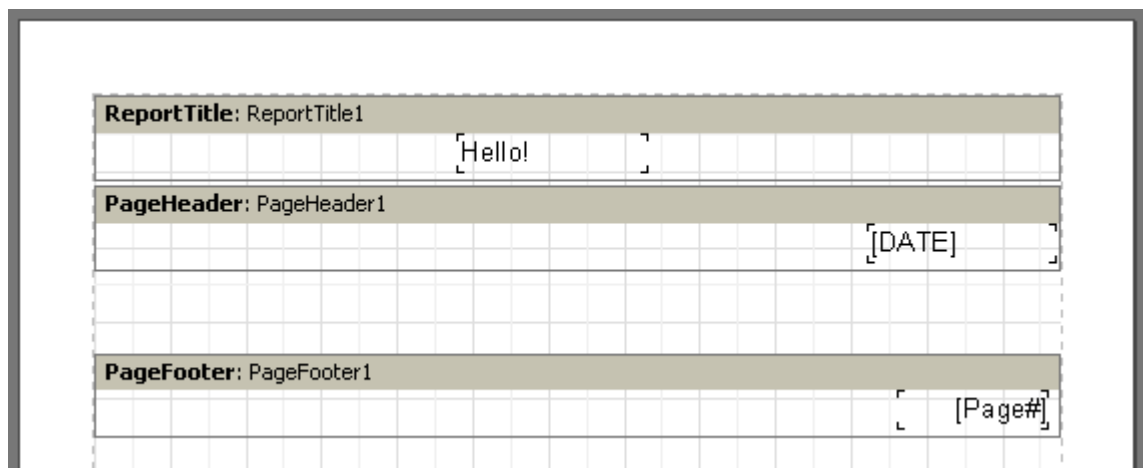
Bands in FastReport

Bands are used for logical object grouping. Thus, when placing an object to a band, such as “Page Header,” we inform FastReport that the given object must be displayed on the top of each page of a finished report. In the same way, the “Page Footer”

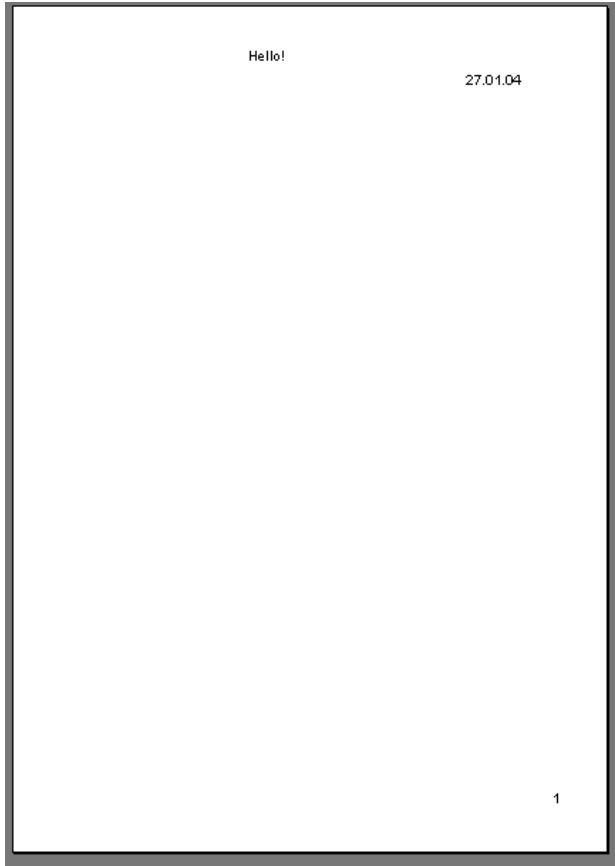
band is displayed at the bottom of each page together with all the objects allocated in it. Let us demonstrate it with an example. Let us create a report, which contains the “Hello!” inscription on the top of the page, a current date to the right of it, and a page number at the foot of the page (to the right).

Open the FastReport designer and click the “New report” button in the toolbar. You will see a report template, which already contains three bands: “Report title,” “Master data,” and “Page footer.” Let us remove the “Master data” band for a while (to do that, click either on any free space inside the band, or on its header, and then remove it by pressing the “delete” key or using the corresponding section in the contextual menu). Now let us add a new band (“Page header”). To perform this, click the “Add band” button and select “Page header” in the dropped list. We see that a new band is added to the page. At the same time, the existing bands were moved down. FastReport designer automatically allocates bands on the page, and, as a result, header-bands are allocated on the top, data-bands are in the middle, and footer-bands are at the bottom.

Now let us allocate objects. Allocate “System text” object in the “Page header” band and select “System variable” in its editor “[DATE]” (you should remember that the date can be displayed with the help of a usual “Text” object by typing “[DATE]” in its editor). We allocate “Text” object, which will contain the “Hello!” text in the “Report title” band. Moreover, as you can see, the required object, which displays page number, is already allocated in the “Pagefooter” band.



If running the report, you would see that the objects in the finished report are allocated in the way we need.



Thus, bands are responsible for object allocation in required places. Depending on band type, we can allocate an object either at the top or at the bottom of the page, on the first page, or on the last one. The basic bands, which we would need in most reports, work in the following way:

- “Page header” band is displayed at the very top of each page;
- “Page footer” band is displayed at the very bottom of each page;
- “Report title” band is displayed at the top of the first page, but below the “Page header” band;
- “Report summary” band is displayed at the very end of a report, at white space.

Databands

Thus, we are about to examine the most interesting thing, a possibility of printing the data from DB tables or queries. What is considered a table in such case? It is a required number of lines (records), each of which has a certain number of columns (fields). To print information of this kind, FastReport uses a special type of bands (databands). These are bands with names of “xxx data level” type. To print a whole table or some of its fields, it is necessary to add such band to the report, connect it to the table, and allocate in it the objects with the fields a user wants to be printed out. When building a FastReport report band, printing operation should be performed as many times, as there

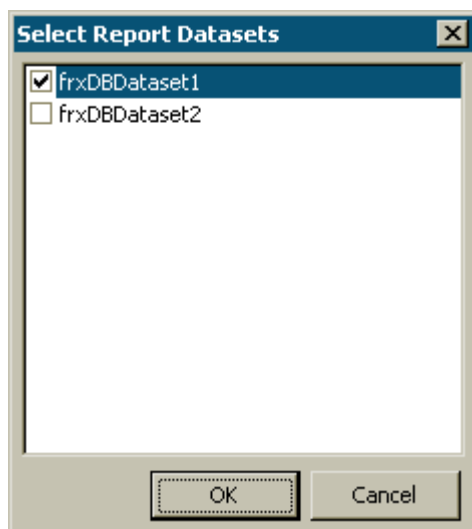
are records in the table. Thus, if there is no free space on the page, new report pages will be formed.

Data Source connection

The “TfrxADOTable”, “TfrxADOQuery” and “TfrxUserDataSet” component-connectors from the FastReport component palette is applied to the band in order to connect a table (or any other data source). This component plays a role of a messenger between the data source and the FastReport core. The component is responsible for records navigation and field reference. This allowed the FastReport core to be independent from any data access library. FastReport can simultaneously work with “ADO” as well as to receive data from a source, not connected with DB, for example, from an array or a file. TfrxADODataBase component is intended for working with data sources, compatible with ADO. The “TfrxUserDataSet” component works with other data sources (arrays, files, etc.).

It is very easy to use the “TfrxADODataBase” component. To connect it with the data source, you should set the “DataBaseName” property (which connects directly to a DataBase).

To make the component (and the data connected to it) available in the report, data sources used in the report must be clearly specified. To do that, select the “Report|Data...” menu item in the “FastReport” designer, and then select the required sources in the opened window.



“Customer List” report

Our second report will be much more complicated than the first one (it will

contain DB table data, a list of clients of a firm). To perform this, let us use the demonstration database DEMO.mdb, which is included in the FastReport Studio distribution kit. Let us create a new report in FastReport. Open the designer, and click the “New report” button, so that FastReport would automatically create a blank pattern with three bands (“Report title,” “Master data,” and “Page footer”). Click ADOTable in FastReport components palette and set its properties:

```
DatabaseName = 'DefaultConnection'  
TableName = 'Customer'
```

Now let us settle down report form creating. Put the “Text” object with the “List of clients” text to the “Report title” band. After that, connect the “Master data” band to our data source. It can be performed in one of the following three ways:

- double-click on the band;
- select the “Edit...” item in the band contextual menu;
- click on the “DataSet” property in the object inspector.

Now we will place four objects (which would display a client’s number, a customer name, phone and fax) on the band. Let us do it in several different ways in order to demonstrate the features of the FastReport designer. Put the first “Text” object on the band and enter “[frxADOTable1.”CustNo”]” to it. It is the most inconvenient way, since the link has to be entered manually, and there is a possibility to enter the text incorrectly. To make inserting of such links into the text easier, we can use the expression designer (its button is located in the toolbar of the “Text” object editor). To insert our field, double-click on the required element in the opened dialogue. By clicking the “OK” button, we close the dialogue and see the field inserted into the text.

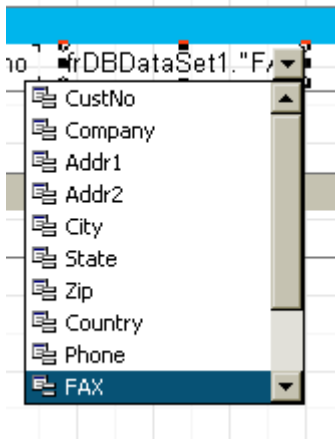
The second way of the DB field inserting into the report is quite similar to the one widely used in the Delphi environment; we will perform it with the help of property setting in the object inspector. Put the second object in the band, without writing anything in the editor. Let us set object properties in the inspector:

```
DataSet = ADOTable1  
DataField = 'Company'
```

Since both of the properties are presented as a list, we only need to select the required values with the help of the mouse.

The third way is to “drag and drop” the required field from the “Data” service window into the report. It is the most simple and obvious way. Take the “Phone” field with the help of the mouse, and then drag it to the band. The only one thing, which should be done in our case, is to disable the “Create header” flag at the bottom of the “Data” window (otherwise we would create a superfluous object, containing the field title, in addition to the required field).

Finally, here is the fourth way. Place a blank “Text” object on the band, and then move the cursor to the object. In the right part of the object you will see the image of the button with the down arrow (as in opening lists). This is the DB fields’ opening list. Click the button and select the ”FAX” field in the list. You can use this feature when the band is connected to data.



Thus, our report is finished:

ReportTitle: ReportTitle1			
[Customer list]			
MasterData: MasterData1			
[frDBDataSet1]	[frDBDataSet1."Company"]	[frDBDataSet1."Pho"]	[frDBDataSet1."FAX"]
PageFooter: PageFooter1			
[Page#]			

Click on the “Preview” button and see, what we have got.

Customer list			
1221	Kauai Dive Shoppe	808-555-0269	808-555-0278
1231	Unisco	809-555-3915	809-555-4958
1351	Sight Diver	357-6-876708	357-6-870943
1354	Cayman Divers World Unlimited	011-5-697044	011-5-697064
1356	Tom Sawyer Diving Centre	504-798-3022	504-798-7772
1380	Blue Jack Aqua Center	401-609-7623	401-609-9403
1384	VIP Divers Club	809-453-5976	809-453-5932
1510	Ocean Paradise	808-555-8231	808-555-8450
1513	Fantastique Aquatica	057-1-773434	057-1-773421
1551	Marmot Divers Club	416-698-0399	426-698-0399
1560	The Depth Charge	800-555-3798	800-555-0353
1563	Blue Sports	610-772-6704	610-772-6898
1624	Makai SCUBA Club	317-649-9098	317-649-6787
1645	Action Club	813-870-0239	813-870-0282
1651	Jamaica SCUBA Centre	011-3-697043	011-3-697043
1680	Island Finders	713-423-5675	713-423-5676

Displaying DB fields with the help of the “Text” object

As you can see, the “Text” object is able to display data from DB, in addition to displaying static text and expressions. Moreover, we can do it in two ways: by either placing a link to the DB field into the object text, or connecting an object to the required field with the help of the “DataSet” and “DataField” properties. The first way is rather good in terms of possibility to display both field contents and any explanatory statement in one and the same object. For example:

Contact person: [ADTable1."Contact_Person"]

As you can see, special syntax is used for links to the DB field: data_set_name. “field_name.” The field name (as well as the set name) can contain spaces. Space between the “point” and “quote” symbols is not permitted.

Not only a link to a field can be placed in the text of the object. We can apply different computing operations to a field as well:

Length (cm): [<ADOTable1."Length_in"> * 2.54]

Pay attention to how square and angle brackets have been used. Remember that square brackets are used by default for marking out the expressions, which are included in the object text. In case of need, square brackets can be substituted for a pair of any other opening/closing sequences (see the “Displaying expression with the help of the

“Text” object” section). Angular brackets are used inside expressions for marking out the FastReport variables and DB fields. To be logical, we should write

Contact person: [*<ADOTable1."Contact_Person">*]

instead of

Contact person: [*ADOTable1."Contact_Person"*]

Nevertheless, both these notations are correct, since FastReport allows absence of angular brackets, in case when an expression contains only one variable or only one DB field. However, if an expression contains several members, the brackets are obligatory:

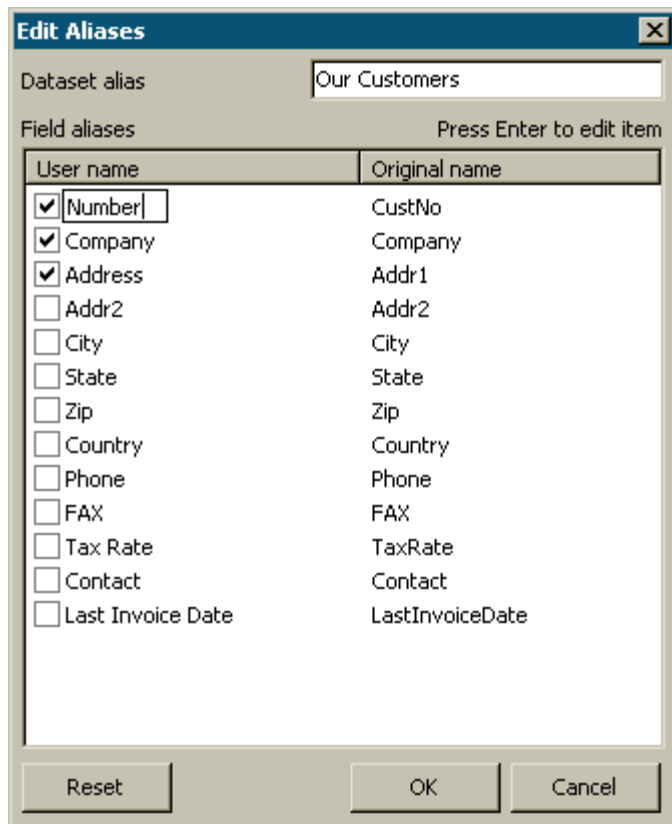
Length (cm): [*<ADOTable1."Length_in"> * 2.54*]

This was made due to the fact that all expressions are processed with the script language, where square brackets are used for marking out either sets or open arrays.

Aliases

In the previous report, we used the data source with the “ADOTable1” name and the following fields: “CustNo,” “Company,” “Phone,” and “FAX.” Accordingly, we had to insert something like “[ADOTable1.”CustNo”]” into the report. Does it seem to be quite clear? No, it does not. One would want to rename the data source, and the field, naming it “Our clients” and “Number” respectively. However, “ADOTable1” is a name of the component, which does not support spaces. And “CustNo” is a name of the field; it cannot be renamed directly (without database restructuring). There is however a way out. The user can use so-called pseudonyms or aliases in such situations. Both the data source and the field possess second names, i.e. aliases, which can easily be modified (the original names, of course, are not modifiable). If a name has an alias, this alias is what is used in FastReport. Otherwise, the original name is used.

It is very easy to rename a data source and its fields in FastReport. To open the alias editor, double-click on the “ADOTable1” component. You can modify the data source name, names of its fields, and select the fields you would need in the report. Let us rename the source and fields (see picture):



Note, that an alias of the source can be modified without using the alias editor. To perform this, modify the “UserName” property of the “ADOTable” component.

Now we need to modify the report, as the names of the fields have been changed. To modify the names of fields in objects, it is easier to use the fourth way, which was described in the "List of clients Report" chapter. Move the mouse cursor to the “Text” object so that the button in the right part of the object would appear, click on the button, and select a necessary field in the list. As you can see, now the data source name and its fields’ names are more than understandable.

The only thing remains to be said is that it is better to perform the operation of assigning an alias in the very beginning, before starting building a report. This can help to avoid subsequent field renaming in the report.

Variables

In addition to usage of aliases, there is one more way, which allows to set more understandable names for DB fields (and not only for them). One can compare a DB field name, as well as any expression, to the variable. To operate with variables in FastReport, select the “Report|Variables...” menu item, and then click “Variables” in the toolbar.

The list of variables in FastReport has a two-level structure. The first level

contains categories, and the second contains the variables themselves. Categorization of the variables is designed for convenience when a list of variables is too long. A list must contain at least one category. That means, that the variables cannot be located at the upper level. Furthermore, categories are needed for logical variables classification only, therefore, they are not included in reports. That is why, when setting a name for a variable, do not forget that it must be unique; it is impossible to create two identical variables in different categories.

Let us illustrate use of variables by the following example. Assume we have two data sources: the first is “ADOTable1” with the “CustNo” and “Name” fields and the second is “ADOTable2” with the “OrderNo” and “Date” fields. We can compare the following list of variables to the fields:

Clients

Client number

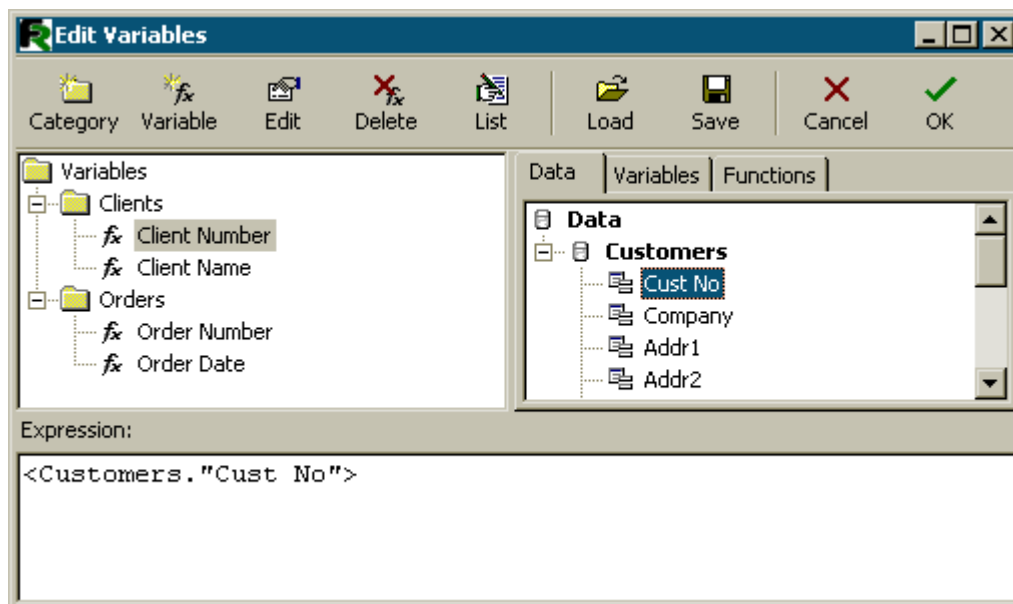
Client name

Orders

Order number

Order date

where “Clients” and “Orders” are two categories. Let us open the variables editor and create a required structure with the help of the “New category,” “New variable,” and “Edit” buttons. To compare the variables to the DB fields, let us select a variable and double-click on the required DB field in the right part of the window. The link to the DB field will be moved to the bottom of the window. An expression at the bottom of the window would be the value of variable. If it were necessary, it can be edited manually. The categories must not be compared to anything.



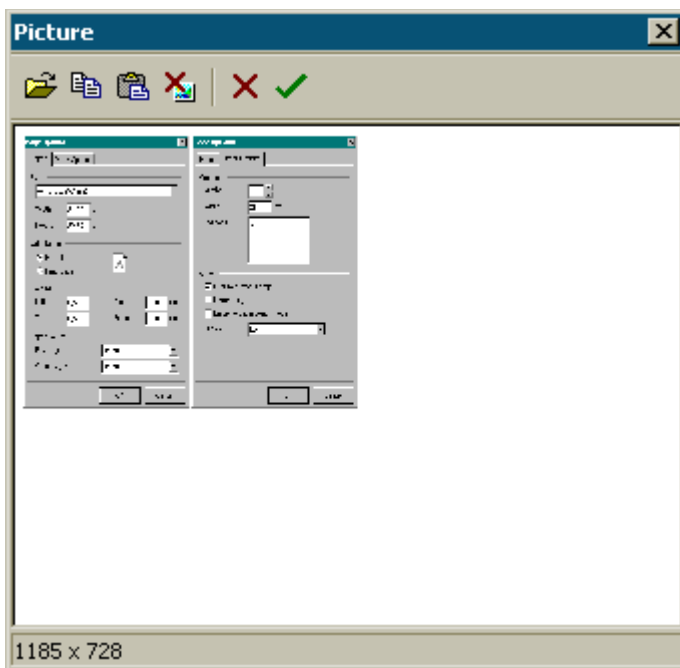
After the list of variables is created, close the variables editor. Now we need to

insert the variables into the report. In contrast to inserting DB fields, there are fewer variants here. We can either insert a variable into the object text manually by typing the “[Client number]” text, or drag a variable from the “Data” service window to the required place of the report. In the second case, it is required to switch to the “Variables” bookmark in this window.

“Picture” object

The next object to be examined is the “Picture” object. It is also often used in reports. With the help of this object, you can insert a trademark of your firm, a photo of your employee or any other graphical information. The object is able to display graphics in “BMP,” “JPEG,” “ICO,” “WMF,” and “EMF” formats.

Let us examine the capabilities of the object. Create a blank report and place the “Picture” object to the report list. You can load a picture from the file or clear an existing picture in the object editor (if it does not open automatically, then doubleclick on the object). Load any desired picture and click “OK.”



There are the following options in the object contextual menu (in brackets are the corresponding names of the properties in the object inspector):

- AutoSize
- Stretch – enabled by default
- Center
- KeepAspectRatio – enabled by default

If the “AutoSize” option is enabled, we can see that the object is being resized, according to the size of the picture it contains. Sometimes such feature can be useful, if pictures of different sizes are to be displayed. This option is disabled by default, due to the fact that it is rather convenient in most cases.

The “Stretch” option is enabled by default. This option stretches the picture inside an object. Modify object size with the help of the mouse and you will see, that the picture size always corresponds to the object’s size. If this option is disabled, the picture will be displayed in its original size. This behaviour differs from the “AutoSize” option because the object dimensions are not adjusted according to the picture size, which means that the object can be larger or smaller than a picture is.

The “Center” option allows aligning a picture inside the object.

The “KeepAspectRatio” option is enabled by default and performs a very useful task: it does not allow the picture ratios to distort when object sizes are modified. This option works only together with the “Stretch” option. When applying any object dimensions, a drawn circle will remain a circle, without turning into an oval. At that, the stretched picture occupies not the whole internal space of an object but only a part of it, necessary for displaying of the picture in correct ratios. If the option is disabled, a picture will be stretched by whole object size, and if object’s size does not correspond to the initial dimensions of the picture, it will be distorted.

Report with pictures

The “Picture” object, as well as many objects in FastReport, can display data from DB. The connection of an object to a required DB field is realized with the help of the “DataSet” and “DataField” properties in the object inspector. In contrast to the “Text” object, this is the only way to connect an object to data.

Let us demonstrate the aforesaid with a report, which would have images of fishes, and their names. To perform this, we will again need the “DEMO” demonstration database, which is included in FastReport Studio distribution kit.

Let us create a blank report in FastReport, and then put the “ADOTable” component on the form and set its properties:

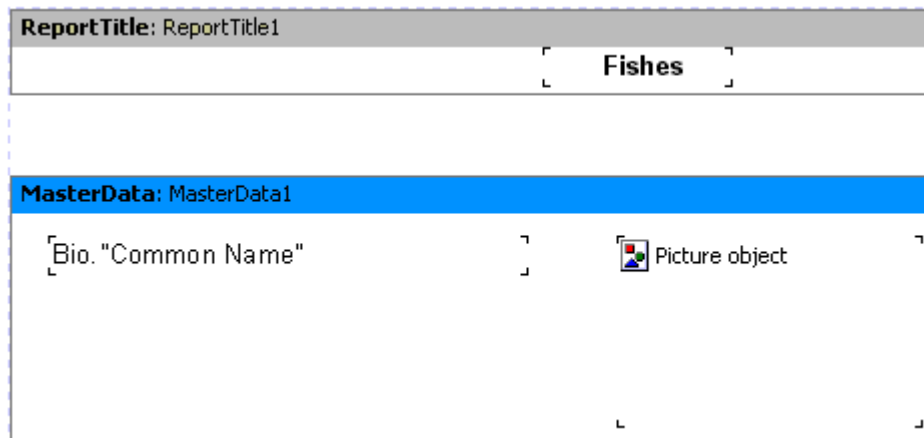
```
DatabaseName = 'DefaultConnection'  
TableName = 'Biolife'
```

Let us begin creating a report form. Put the “Text” object with the “Fish” text on the “ReportTitle” band. Connect the “First level data” band to the data source (double-click on the band and select “Bio” from the list). Let us increase the band height up to 3 cm so that the picture finds room in it. Let us put the “Text” object to the band and

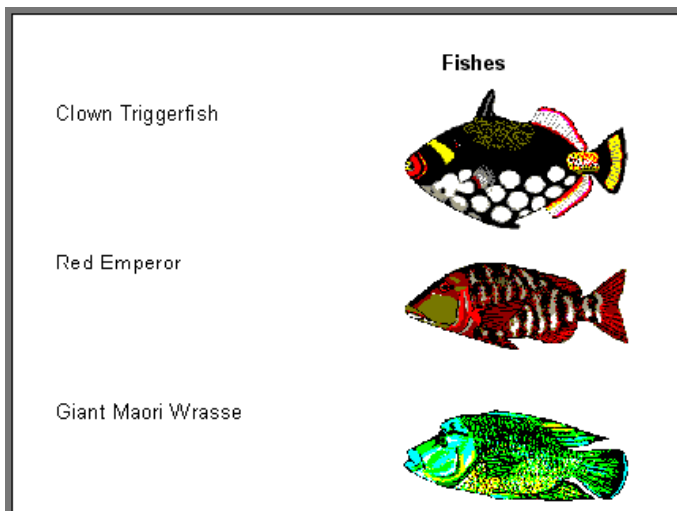
connect it to the "CommonName" field using any of the methods described above. After that, drop the "Picture" object alongside, and connect it to the "Graphic" field. To perform this, set properties in the object inspector:

```
DataSet = Bio  
DataField = 'Graphic'
```

Note, that both of these properties are of the "List" type, and that is why one can select the required values with the help of the mouse. To find room for the picture, stretch the object up to 4x2.5cm.



That is all. The report is finished (see the picture below):



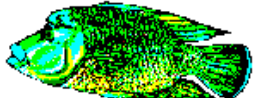


Multi-lined text displaying

Let us revert to the previous example with fishes. In the "Biolife" table, there is a "Notes" field, which contains a detailed description of each fish. Let us update our report

by adding this field into it.

At first sight, everything seems to be easy: add the “Text” object to the band with data, connect it to the “Notes” field and set the object’s size (8x2.5 cm). If starting the report, you will see that we receive not exactly what we expected:

Fishes		
Clown Triggerfish	Also known as the big spotted triggerfish. Inhabits outer reef areas and feeds upon crustaceans and mollusks by crushing them with powerful teeth. They are voracious eaters, and divers report seeing the clown triggerfish devour beds of pearl nvetare	
Red Emperor	Called seaperch in Australia. Inhabits the areas around lagoon coral reefs and sandy bottoms.	
Giant Maori Wrasse	The red emperor is a valuable food fish and considered a great sporting fish that fights with fury when hooked. The flesh of an old fish is just This is the largest of all the wrasse. It is found in dense reef areas, feeding on a wide variety of mollusks, fishes, sea urchins, crustaceans, and other invertebrates. In spite of its immense size, divers find it a very wary fish.	

However, FastReport performed just what he was asked to. The “Notes” field contains a multi-lined text, size of which may vary. At the same time, the “Text” object, which displays the information from this field, has fixed sizes. That is why some lines could not place the object and were cut. What should be done in such situation?

Of course, either sizes of the window can be specified “in reserve,” or font size can be reduced. However, this may lead to the uneconomical usage of space on the page, due to the fact that some fishes have long descriptions, while others have short ones. In FastReport, there are resources, which allow solving this problem.



The matter concerns the band’s ability to automatically adjust its height in order to find room for all included objects. To perform this, we just need to enable the “Stretch” property. However, that is not all, because an object with a longer description should be able to stretch by itself. The “Text” object is able to manage it.

The object can automatically set its height and width in order to find room for the whole text it contains. One can use the “AutoWidth” and “StretchMode” properties to perform this. The “AutoWidth” property selects the object width in a way, which allows all the lines find room without division of words. This mode is convenient when an object has a single text line. The “Stretch” property allows to select the object’s height in a way that the whole text finds room. The object width is not being changed during it. This property performs listing, and you can select one of the modes in the object inspector:

smDontStretch – do not stretch an object, by default;

smActualHeight – stretch an object in order to find room for the whole text;
 smMaxHeight – stretch an object so that its bottom would coincide with the bottom band line (where the object is placed). We will examine this example a little later.

Now we are interested in the “Stretch” property of the “Text” object. Enable it in the object context menu or set the “StretchMode = smActualHeight” property value. Also, enable the “Stretch” band property. Start the report and make sure that everything works in a proper way now.

Fishes		
Clown Triggerfish	<p>Also known as the big spotted triggerfish. Inhabits outer reef areas and feeds upon crustaceans and mollusks by crushing them with powerful teeth. They are voracious eaters, and divers report seeing the clown triggerfish devour beds of pearl oysters.</p> <p>Do not eat this fish. According to an 1878 account, "the poisonous flesh acts primarily upon the nervous tissue of the stomach, occasioning violent spasms of that organ, and shortly afterwards all the muscles of the body. The frame becomes rocked with spasms, the tongue thickened, the eye fixed, the breathing laborious, and the patient expires in a paroxysm of extreme suffering."</p> <p>Not edible.</p>	
Red Emperor	<p>Range is Indo-Pacific and East Africa to Somoa. Called seaperch in Australia. Inhabits the areas around lagoon coral reefs and sandy bottoms.</p> <p>The red emperor is a valuable food fish and considered a great sporting fish that fights with fury when hooked. The flesh of an old fish is just as tender to eat as that of the very young.</p>	

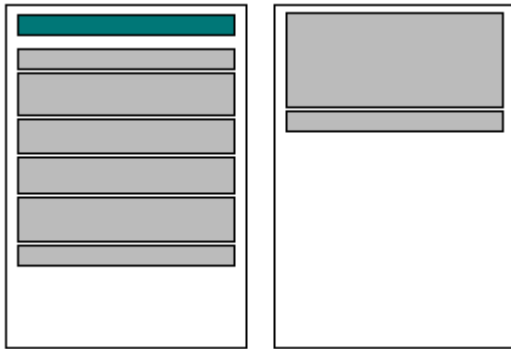
As you can see, when constructing a report, FastReport fills objects with data, stretches them with the “Stretch” option enabled, and then collates band’s height in order to find room for all the objects. If the band “Stretch” option is disabled, the height setting is not performed, and the band is displayed according to height specified in the designer. If we try to disable this option, we would see that the objects with longer texts are still stretched, although bands are not. That leads to text overlaying, since each next band is displayed right after the previous one.

Data splitting

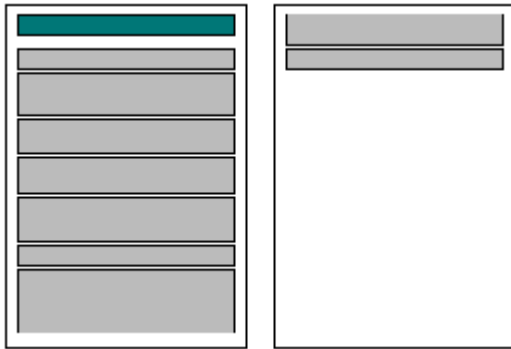
Let us pay attention to a peculiarity of the report with fishes: there is much blank space at bottom of the pages. Why does it happen? When a report is constructed, the

FastReport core fills whitespace of the page with bands. After displaying each band, the current position shifts down. When FastReport finds out that there is not enough space to display the next band (its height is larger than white space left on the page), then a new page is formed and band displaying continues there. This operation continues being performed as long as there are notes in data set.

Our report contains an object with large text, and that is why the band height is rather large. Furthermore, if a large band does not find room on a page, it is transferred to the next one, and much unused space remains at the bottom of the page. This is shown at the following picture:



To use paper more rationally, let us use a FastReport feature, which paragraphs the band contents. All we need is to enable the “AllowSplit” option of the “First level data” band. You see that there becomes less of white space at the bottom of report pages:

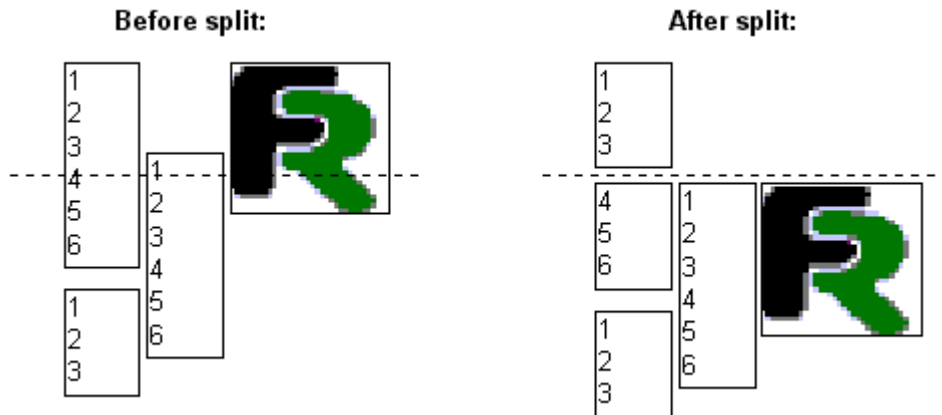


How does the band splitting work? There are some objects in FastReport, which support this feature. They are the “Text,” “Line,” and “RichEdit” objects. They can be “splitted,” while other objects cannot. When FastReport comes across the necessity of splitting accomplishment, it performs it in the following way:

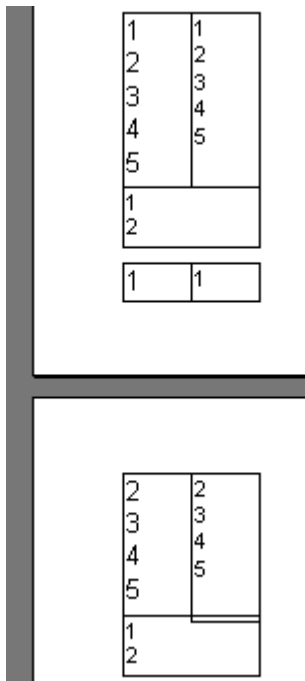
- displays the non-splittable objects, which find room on white space;
- partially displays splittable objects (text objects are displayed in a way that all lines find room in the object);
- forms a new page and continues object displaying;

- if a non-splittable object does not find room on whitespace, it is transferred to the next page; at the same time, all the objects located underneath, are shifted according to transferring;
- the process continues until all the band objects are wholly displayed.

The splitting algorithm will become clearer if to look at the picture:



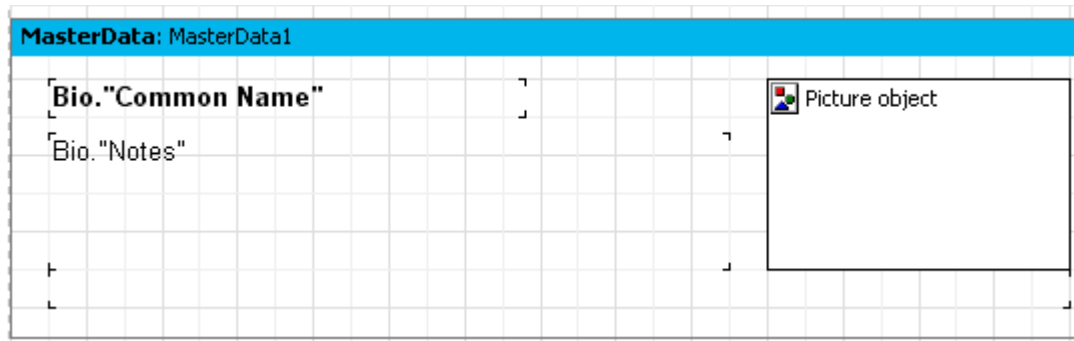
It should be noted, that the splitting algorithm does not provide 100% quality of the received report. That is why you should use this option very carefully in cases when objects on the splitted band are grouped in a complicated way, and, in addition, their font sizes differ. Here is the example of what could be received:



Text wrap of objects

For report designing, in some cases it becomes necessary to create text wrap of objects (often, when using pictures). Let us demonstrate this FastReport feature with the example with fishes.

Let us add one more “Text” object to the report, and then arrange the objects as shown in the following picture:



We will disable stretching for the “Bio.”Notes” object. On the contrary, we will enable this property for the bottom object. To make the text “blend” from the “Bio.”Notes” object to the bottom one, set the “FlowTo” property in the “Bio.”Notes” object. This property is set in the object inspector and is of the “dropping list” type. The bottom object’s name must be selected from this list. The result would look as shown in the following picture:

Clown Triggerfish

Also known as the big spotted triggerfish. Inhabits outer reef areas and feeds upon crustaceans and mollusks by crushing them with powerful teeth. They are voracious eaters, and divers report seeing the clown triggerfish devour beds of pearl oysters.



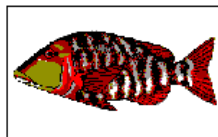
Do not eat this fish. According to an 1878 account, "the poisonous flesh acts primarily upon the nervous tissue of the stomach, occasioning violent spasms of that organ, and shortly afterwards all the muscles of the body. The frame becomes rocked with spasms, the tongue thickened, the eye fixed, the breathing laborious, and the patient expires in a paroxysm of extreme suffering."

Not edible.

Range is Indo-Pacific and East Africa to Samoa.

Red Emperor

Called seaperch in Australia. Inhabits the areas around lagoon coral reefs and sandy bottoms.



The red emperor is a valuable food fish and considered a great sporting fish that fights with fury when hooked. The flesh of an old fish is just as tender to eat as that of the very young.

Range is from the Indo-Pacific to East Africa.

When constructing a report, if a text does not find room in the top object, the part, which does not fit the page, will be transferred to the bottom object. Since the objects are located around the picture, the effect of text wrapping is performed.

Attention: the main object should be inserted to the report before inserting the linked one. Otherwise, text wrapping may function incorrectly! If your report operates incorrectly, select the linked object, and then transfer it to the forefront by the “Edit|Bring to the forefront ” menu command.

Displaying data in the form of a table

Sometimes it is necessary to display a report in the form of a table with a frame. One of the examples of such report could be any price list. To build such report in FastReport, a user just needs to enable framing function for the objects located in the “Data” band. Let us demonstrate several variants of frames with the test report example.

Let us create a blank report in FastReport. Put the “ADOTable” component to the form, and then set its properties:

DatabaseName = 'DefaultConnection'

TableName = 'Biolife'

UserName = 'Bio'

Let us create a report of the following kind:

PageHeader: PageHeader1			
MasterData: MasterData1			
[Bio."Specie"]	[Bio."Common Name"]	[Bio."Length"]	
ReportSummary: ReportSummary1			

Place the objects on the band line-on-line, and minimize band's height.

The first and the simplest type of the table is a table with a full frame. To perform it, one needs to enable all frame lines in every object:

90020	Clown Triggerfish	50
90030	Red Emperor	60
90050	Giant Maori Wrasse	229
90070	Blue Angelfish	30
90080	Lunartail Rockcod	80

The next type of framing would display only horizontal or only vertical lines. Such framing is performed in exactly the same way. Horizontal or vertical frames can be enabled in objects.

90020	Clown Triggerfish	50
90030	Red Emperor	60
90050	Giant Maori Wrasse	229
90070	Blue Angelfish	30
90080	Lunartail Rockcod	80

Finally, to construct only the external framing, the report needs to be slightly modified:

PageHeader: PageHeader1		
MasterData: MasterData1		
[[Bio."Specie"]	[[Bio."Common Name"]]	[[Bio."Length"]]
ReportSummary: ReportSummary1		

As you can see, we have added two “Text” objects and enabled frame lines for the objects along the edges of the data-band. As a result, the report will look as follows:

90020	Clown Triggerfish	80
90030	Red Emperor	60
90050	Giant Maori Wrasse	229
90070	Blue Angelfish	30
90080	Lunartail Rockcod	80
90090	Firefish	38
90100	Omate Butterflyfish	19
90110	Swell Shark	102
90120	Bat Ray	66
90130	California Moray	150
90140	Lingcod	150
90150	Cabezon	99
90160	Atlantic Spadefish	90
90170	Nurse Shark	400
90180	Spotted Eagle Ray	200
90190	Yellowtail Snapper	75
90200	Redband Parrotfish	28
90210	Great Barracuda	150
90220	French Grunt	30
90230	Dog Snapper	90
90240	Nassau Grouper	91
90250	Bluehead Wrasse	15
90260	Yellow Jack	90
90270	Redtail Surperch	40
90280	White Sea Bass	150
90290	Rock Greenling	60
90300	Senoiita	25
90310	Surf Smelt	25

All examples aforesaid contained bands, which possessed fixed sizes. But how is it possible to display a table, in case when the band is stretched? Let us explain that, using the example below. Add a new field (a multi-lined text from “Bio.Notes”) to our report. As you already know, the “Stretch” property must be enabled both for this object and for the band, in which the object is located. In this case, the band height is selected depending on size of the text in the “Text” object. Thus, we would receive a report of the following kind:

90020	Clown Triggerfish	80	<p>Also known as the big spotted triggerfish. Inhabits outer reef areas and feeds upon crustaceans and mollusks by crushing them with powerful teeth. They are voracious eaters, and divers report seeing the clown triggerfish devour beds of pearl oysters.</p> <p>Do not eat this fish. According to an 1878 account, "the poisonous flesh acts primarily upon the nervous tissue of the stomach, occasioning violent spasms of that organ, and shortly afterwards all the muscles of the body. The frame becomes racked with spasms, the tongue thickened, the eye fixed, the breathing laborious, and the patient expires in a paroxysm of extreme suffering."</p> <p>Not edible.</p> <p>Range is Indo-Pacific and East Africa to Samoa.</p>
90030	Red Emperor	60	<p>Called seaperch in Australia. Inhabits the areas around lagoon coral reefs and sandy bottoms.</p>

It is a little bit different from the one we need; one would prefer the frames of the neighboring objects to be able to stretch as well. FastReport allows to solve this problem easily. For constructing such reports, it is enough to enable the “Stretch downwards” property (or StretchMode = smMaxHeight in the object inspector) for all objects, which are to be stretched. Thus, the FastReport core firstly counts the maximum band height,

then it “stretches” objects with the enabled option to the bottom band edge. Due to the fact that object frames stretch together with the object, the report’s appearance changes:

90020	Clown Triggerfish	50	<p>Also known as the big spotted triggerfish. Inhabits outer reef areas and feeds upon crustaceans and mollusks by crushing them with powerful teeth. They are voracious eaters, and divers report seeing the clown triggerfish devour beds of pearl oysters.</p> <p>Do not eat this fish. According to an 1878 account, "the poisonous flesh acts primarily upon the nervous tissue of the stomach, occasioning violent spasms of that organ, and shortly afterwards all the muscles of the body. The frame becomes racked with spasms, the tongue thickened, the eye fixed, the breathing laborious, and the patient expires in a paroxysm of extreme suffering."</p> <p>Not edible.</p> <p>Range is Indo-Pacific and East Africa to Samoa.</p>
90030	Red Emperor	60	<p>Called seaperch in Australia. Inhabits the areas around lagoon coral reefs and sandy bottoms.</p>

Printing labels

In contrast to table reports, data in reports such as “label,” are allocated one under another. Let us examine an example of such report, which displays data about fishes (see the previous example). The report is presented in the form of a label, and has the following structure:

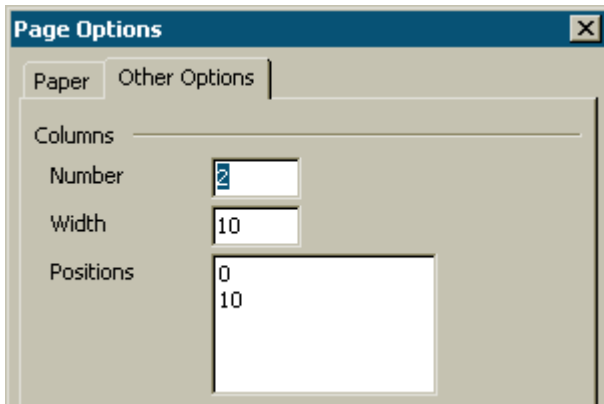
MasterData: Band3			
Number:	[Bio."Species No"]]
Category:	[Bio."Category"]]
Name:	[Bio."Common Name"]]
Length, cm:	[Bio."Length (cm)"]]

If starting execution of this report, we would receive the following:

Number:	90020
Category:	Triggerfish
Name:	Clown Triggerfish
Length, cm:	50
Number:	90030
Category:	Snapper
Name:	Red Emperor
Length, cm:	60

As you can see, there is much whitespace in the right part of the page. To fill the whole page, the number of columns, where the data will be displayed, can be set in report

page settings. To perform this, you should either double-click on the area of white space on the page, or call the “File Page|parameters...” menu item.



In this bookmark, one can set such column parameters, as number of columns, its width, and position. In our case, it would be enough to specify a number = 2, since FastReport adjusts all the rest parameters automatically. The column frame is displayed in the designer as a thin vertical line:

MasterData: Band3	
Number:	[Bio."Species No"]
Category:	[Bio."Category"]
Name:	[Bio."Common Name"]
Length, cm:	[Bio."Length (cm)"]

At that, printing will be performed in the following way. FastReport will display the “First level data” band as long as there is white space on the page. After that, a new column in the very page will be formed (in contrast to simple reports, in which a new page is created in such cases), and band would continue to be displayed on the top. However, now all the objects are shifted to the right, according to column’s width. It will continue until all the columns are displayed. After that, FastReport forms a new page and continues to display data from the first column.

Our report with two columns will look as follows:

Number:	90020	Number:	90140
Category:	Triggerfish	Category:	Cod
Name:	Clown Triggerfish	Name:	Lingcod
Length, cm:	50	Length, cm:	150
Number:	90030	Number:	90150
Category:	Snapper	Category:	Sculpin
Name:	Red Emperor	Name:	Cabezon
Length, cm:	60	Length, cm:	99

The “Columns” property, available in all data-bands, is another way to set number of columns. It allows to set number of columns for a particular band and not for the whole page (as it was in previous example). Thus, the principle of data displaying will be not “from the top to the bottom, then from the left to the right,” but “from left to right, then from top to bottom.”

Let us disable columns in the page (set the columns number = 1) and enter “2” in the “Columns” band property. FastReport displays the column frames as dotted lines. Let us get the required column dimensions by modifying the “ColumnWidth” property:

MasterData: Band3	
Number:	[Bio."Species No"]
Category:	[Bio."Category"]
Name:	[Bio."Common Name"]
Length, cm:	[Bio."Length (cm)"]

The report constructed in such way, would differ from the previous one only by the “from-left-to-right, then from-top-to-bottom” order of data displaying.

Child-bands

Let us examine the case when one of the lines in a report of “label” type, may have a variable size. To simulate the situation using our our example, let us reduce the “Bio.”Common Name”” object width to 2.5 cm, and enable the “Stretch” option for it. Let us also enable stretching in the “First level data” band. Enable all the frame lines in all objects so that the principle of the stretching function would become clear. We will receive a report of the following kind:

Number:	90020
Category:	Triggerfish
Name:	Clown
Length, cm:	Triggerfish
	50
Number:	90030
Category:	Snapper
Name:	Red Emperor
Length, cm:	60

You see, that in the first case the first object contains a longer text, and that is why it was stretched in two lines. Thus, the object (located underneath it and linked to the Bio.”Length (sm)” field) was shifted down. That happens because all the objects have the “Shift” property enabled by default (or ShiftMode = smAlways in the object inspector).

Such objects shift downwards if there is a stretchable object above them (the “Text” object with the “Stretch” property enabled). The height value, by which the object shifts, depends on how the object from above is stretched.

However, it is unacceptable in our case, since we need the object with the “Length, cm.” text to be shifted as well. To perform this, there is a special band type in FastReport, “Child-band.” It is linked to (and is displayed after) the basic band. Let us update our report:

MasterData: MasterData1	
Number:	Bio."Species No"
Category:	Bio."Category"
Name:	Bio."Common"
Child: Child1	
Length, cm:	Bio."Length (cm)"

To link the basic component to the child one, let us set the “Child = Child1” property in the object inspector. Now, each time you print the basic band, the child one would be displayed as well:

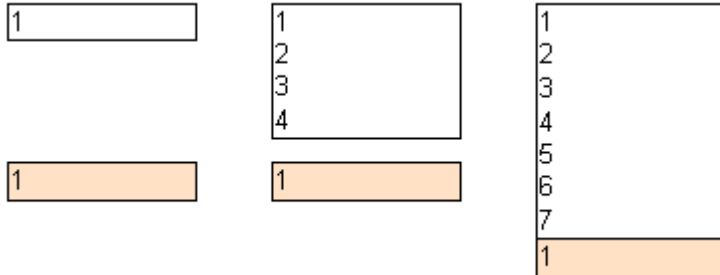
Number:	90020
Category:	Triggerfish
Name:	Clown Triggerfish
Length, cm:	50
Number:	90030
Category:	Snapper
Name:	Red Emperor
Length, cm:	60

As you can see, now the title is typed exactly where it is supposed to be. In order to avoid child-band’s transferring to the next page (which basically means, it will be separated from the basic band), enable the “Keepchild” property for the basic band (“KeepChild” in the object inspector).

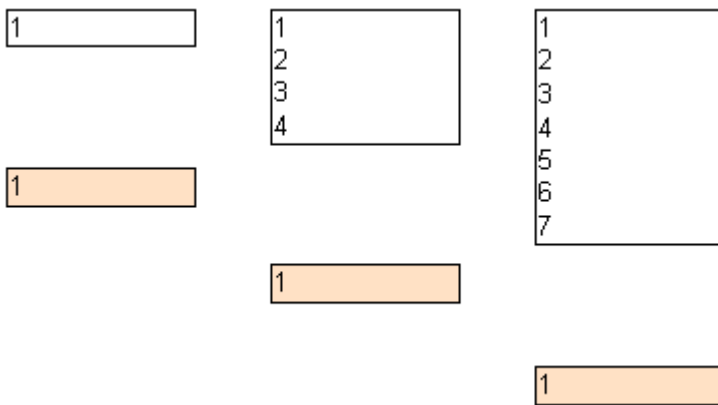
Shifting objects

You have already seen how the “Shift” property works. Let us examine the next mode of shifting, “Shift on overlapping.” In the object inspector, the “ShiftMode=smWhenOverlapped” property value corresponds to this mode. Thus, object shifting will

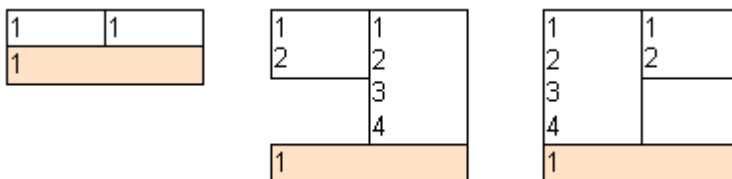
be performed in case, when the object from above overlaps the given object during stretching. Three cases are shown in the picture below. As you see, the bottom object with the enabled “Shift when overlapping” option shifts only in the latter case, i.e. when there is much text in the top object and it overlaps the bottom one.



If the “Shift” option is enabled, the bottom object will be shifted anyway:



In some cases, it allows to realize rather complicated logic of object design, especially if an object overlaps several other reports at the same time. Thus, in the following example both of the upper objects contain stretchable texts, and the bottom one has the enabled “shifting when blocking” option. The bottom object will always be displayed closely to the object, which contains more text, irrespective of text size in the upper objects:



In this example, if the “Shift” option is enabled for this object, the bottom object will shift twice, since it is located underneath two objects and thus an unnecessary gap is formed.

Report with two data levels (master-detail)

So far we examined reports where only one data-band was presented (“First level data”). That enabled typing data from one DB table. FastReport allows to type reports containing up to six data levels (it is also possible to type more levels via the “subreport” object; this feature will be examined later). In real applications, reports with large data applications are rarely typed. As a rule, they are limited to 1-3 levels.

Let us examine the two-leveled report creation process. It will contain data from the DEMO tables: “Customer” и “Orders.” The first table is the list of clients; the second one is the list of orders placed by the clients. The tables contain data of the following type:

Customer:

CustNo	Company
1221	Kauai Dive Shoppe
1231	Unisco
1351	Sight Diver
....	

Orders:

OrderNo	CustNo	SaleDate
1003	1351	12.04.1988
1023	1221	01.07.1988
1052	1351	06.01.1989
1055	1351	04.02.1989
1060	1231	28.02.1989
1123	1221	24.08.1993
....		

As you can see, the second table contains the list of all the orders placed by all companies. To receive the list of orders placed by a particular company, the notes, in which the “CustNo” field contains the number of the chosen company, should be selected in the table. The report constructed on such data will look as follows:

1221	Kauai Dive Shoppe
1023	01.07.1988
1123	24.08.1993

1231	Unisco
1060	28.02.1989
1351	Sight Diver
1003	12.04.1988
1052	06.01.1989
1055	04.02.1989

Let us get down to the report creation. Create a new report in FastReport, put two “ADOTable” to the reporting form. Set the components in the following way:

Table1:

```
DatabaseName = 'DefaultConnection '
TableName = 'Customer'
UserName = 'Customers'
```

Table2:

```
DatabaseName = 'DefaultConnection '
TableName = 'Orders'
UserName = 'Orders'
```

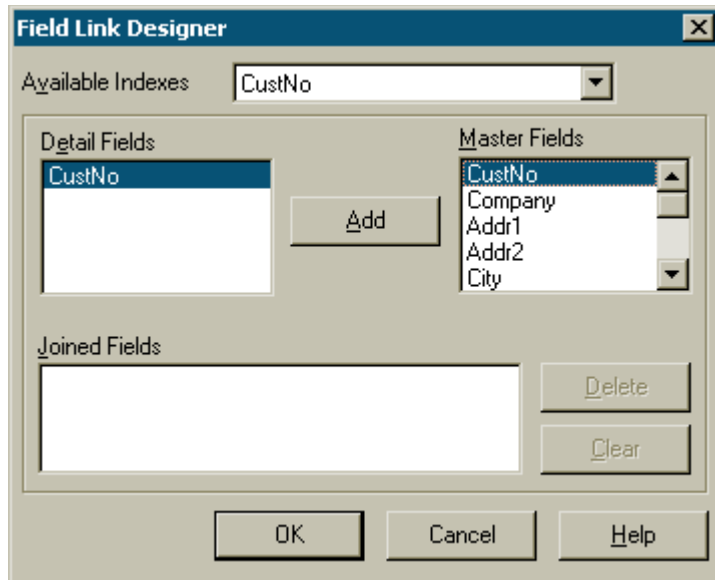
In the report designer, let us connect our data sources in the “Report|Data...” window. Put the “Master data” and “Detail data” bands on the page:

MasterData: MasterData1		Customers	
[Customers."CustNo"]	[Customers."Company"]		
DetailData: DetailData1		Orders	
[Orders."CustNo"]	[Orders."OrderNo"]	[Orders."SaleDate"]	

Note that the “Master Data” band must be allocated above the “Detail Data” band! If allocated under, FastReport will inform you about an error occurrence when the report starts.

Data linkage

If starting the report now, you would see that the list of orders remains the same for every client and contains all notes from the “Orders.db” table. That happens because we did not enable notes filtering in the “Orders” table. Let us revert to our data sources. Set the “*Master = Table1*” property in the “Table2” component. Thus, we have set a “master-detail” connection. After that, a condition for subordinate source notes filtration should be established. To perform this, call the “MasterFields” property editor of the “Table2” component:



We need to link together two “CustNo” fields in both sources. To perform this, select the desired fields, and then click on “Add” button. Fields linkage will shift to the bottom window. After that, close the editor and click “OK.”

When starting a report, FastReport does the following. After allocating the next note from the master table (Customer), it will set the filter on the subordinate table (Orders). Only those notes, which would satisfy the “Orders.CustNo = Customer.CustNo” condition will remain in the table. That means that for each client only his/her orders will be displayed:

1221			Kauai Dive Shoppe
1221	1023	01.07.88	
1221	1076	16.12.94	
1221	1123	24.08.93	
1221	1169	06.07.94	
1221	1176	26.07.94	
1221	1269	16.12.94	
1231			Unisco
1231	1060	28.02.89	
1231	1073	15.04.89	
1231	1102	06.06.92	
1231	1160	01.06.94	

Reports, containing up to 6 data levels can be constructed in the similar way.


```

masterheader
masterdata
  detailheader
  detaildata
  detaildata
  detailfooter
masterfooter
masterdata
  detailheader
  detaildata
  detaildata
  detailfooter
masterfooter

```

Report with groups

We constructed a two-leveled report on the basis of the data from two tables in the example above. FastReport allows constructing analogous reports on the basis of one set of data, formed in a unique way.

To perform this, one needs to create a query using SQL language, which would return data, arranged according to a certain condition, from both of the tables. In our case, a condition is a correspondence of the “CustNo” fields in both of the tables. An SQL-query may look as follows:

```

select * from customer, orders
where orders.CustNo = customer.CustNo
order by customer.CustNo

```

The "order by" line is necessary for sorting the records in the “CustNo” field. The example below shows how the query data would be returned:

CustNo	Company	...	OrderNo	SaleDate
1221	Kauai Dive Shoppe		1023	01.07.1988
1221	Kauai Dive Shoppe		1123	24.08.1993
1231	Unisco		1060	28.02.1989
1351	Sight Diver		1003	12.04.1988
1351	Sight Diver		1052	06.01.1989
1351	Sight Diver		1055	04.02.1989

How can a multi-leveled report be constructed on the basis of this data? In FastReport there is a special band – “Group Header”. A special condition is established for the band (DB field value or an expression); the band is displayed as soon as the field's value is changed. The following example illustrates this.

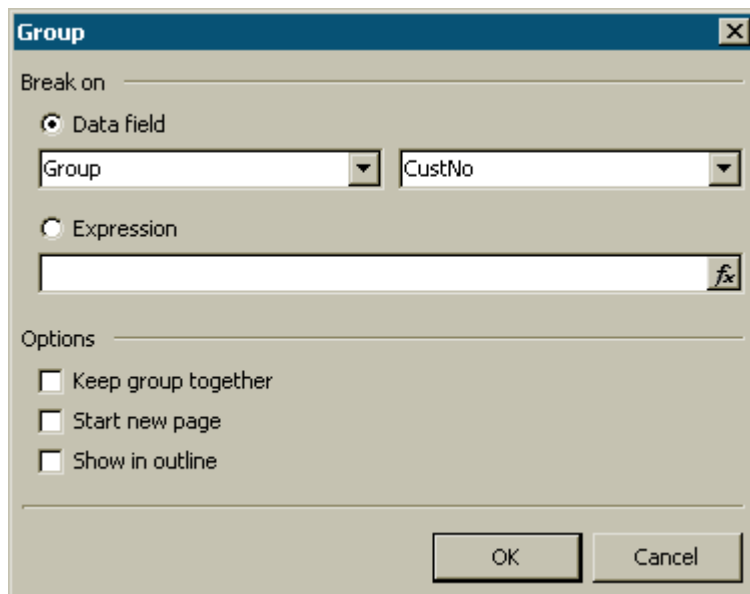
Let us create a new report in FastReport, put the “ADOQuery” on the report form. Let us set it in the following way:

Query1:

```
DatabaseName = 'DefaultConnection '
SQL =
  select * from customer, orders
  where orders.CustNo = customer.CustNo
  order by customer.CustNo
```

UserName = 'Group'

Let us open the designer and connect our data source to the report. After that, add the “Group header” and “Master data” bands to the report. Set a condition (in this case, it is “Group.CustNo” data field) in the “Group header” band editor:



Let us link data-band to the “Group” data source and place the objects in the following way (note, that the group header must be allocated above the data-band):

GroupHeader: GroupHeader1		Group."CustNo"	
[Group."CustNo"]	[Group."Company"]		
MasterData: MasterData1		Group	
[Group."OrderNo"]	[Group."SaleDate"]		

On starting, we would get a report similar to the one shown below:

1221	Kauai Dive Shoppe
1269	16.12.94
1023	01.07.88
1176	26.07.94
1076	16.12.94
1123	24.08.93
1169	06.07.94
1231	Unisco
1173	16.07.94
1178	02.08.94

As you can see, the “Group header” band is displayed only when the field, to which it is connected, changes its value. Otherwise, the data-band connected to the group is displayed. If compare this report to the master-detail report, which we constructed earlier, it seems to be obvious that order numbers are not sorted in ascending order. It can be easily corrected by changing the SQL query text:

```
select * from customer, orders
where orders.CustNo = customer.CustNo
order by customer.CustNo, orders.OrderNo
```

Similarly, the reports with nested group can be constructed. At the same time, the number of enclosures is unlimited in such reports. Thus, the reports with groups have some advantages over the reports of the master-detail type:

- the whole report needs only one table (query);
- the number of the data enclosuring levels is unlimited;
- the additional data sorting feature;
- more rational usage of the DB resources (the query returns only the data, which should be printed, without having to filtrate the data).

The only disadvantage is the necessity of writing queries in SQL language. However, you can use internal visual SQL-generator FastQueryBuilder.

Other group features

Let us pay our attention to how the group is transferred to the next page:

1380	Blue Jack Aqua Center
1006	06.11.94
1079	03.05.89
1106	23.09.92
1153	16.04.94
1253	26.11.94
1384	VIP Divers Club
1007	01.05.88
1027	07.07.88

If looking through the printout of such report, it seems to be not clear, which client the list of orders on the very top of the second page refers to. FastReport allows repeating of group titles displaying (which in our case contains information about the client) on the next page. To perform this, the “Reprint on new page” menu item (or the “ReprintOnNewPage” property in the object inspector) should be enabled in the “Group header” band. Thus, the report will look in the following way:

1380	Blue Jack Aqua Center
1006	06.11.94
1380	Blue Jack Aqua Center
1079	03.05.89
1106	23.09.92
1153	16.04.94
1253	26.11.94
1384	VIP Divers Club
1007	01.05.88

There is another way, which allows to avoid breaking of groups. To perform this, the “Keep together” group header property (or “KeepTogether” in the object inspector) should be enabled. Thus, if the whole group does not find room on the page, it is transferred to a new page. In our example, it will look in the following way:

1356	Tom Sawyer Diving Centre
1005	20.04.88
1059	24.02.89
1072	11.04.89
1080	05.05.89
1105	21.07.92
1180	06.08.94
1266	15.12.94
1280	26.12.94
1305	20.01.95

1380	Blue Jack Aqua Center
1006	06.11.94
1079	03.05.89

Thus, much blank space may appear on several pages, but the group will be displayed as a whole on the page.

In conclusion, the “StartNewPage” group header property allows to display each group on a separate page. This probably would lead to misuse of paper, however it might be useful in some cases.

Lines numbering

Let us use our example in order to show how to number lines in the group. To perform this, let us add the “Text” object with a system variable [Line] to both of our bands (it is easier to perform this with the help of the drag&drop method in the “Variables” bookmark of the “Data Tree” tool window).

GroupHeader: GroupHeader1		
[Line]	[Group."Cust"]	[Group."Company"]
MasterData: MasterData1		
[Line]	[Group."Orde"]	[Group."SaleDate"]

When starting the report, we can see that both the data levels now have their numbers:

1	1221	Kauai Dive Shoppe
1	1023	01.07.88
2	1076	16.12.94
3	1123	24.08.93
4	1169	06.07.94
5	1176	26.07.94
6	1269	16.12.94
2	1231	Unisco
1	1060	28.02.89
2	1073	15.04.89
3	1102	06.06.92
4	1160	01.06.94

In some reports, one might need continuous numeration of the second level data. To perform this, we should use the "Line#" variable instead of "Line" on the data-band. The result will be as follows:

1	1221	Kauai Dive Shoppe
1	1023	01.07.88
2	1076	16.12.94
3	1123	24.08.93
4	1169	06.07.94
5	1176	26.07.94
6	1269	16.12.94
2	1231	Unisco
7	1060	28.02.89
8	1073	15.04.89
9	1102	06.06.92

Aggregate functions

In most cases, group reports should display some resulting information (such as: “total of a group,” “number of group elements,” etc). There are the so-called aggregate functions in FastReport designed for this purpose. With their help, one can count up some function of a defined value according to data span. Below is the list of aggregate functions:

SUM	Returns the total of the expression
MIN	Returns the minimal value of the expression
MAX	Returns the maximal value if the expression
AVG	Returns the average value of the expression
COUNT	Returns the number of lines in the data span

The syntax of all aggregate functions (except COUNT) is the following (let us examine it using the example of the “SUM” function):

SUM(expression, band, flags)
SUM(expression, band)
SUM(expression)

The parameters assignment is the following:

expression – the expression, the value of which is to be handled

band – the name of data band, on which handling of values will be performed

flags – the bit field, which can contain the following values and their combinations

1 – consider the invisible bands

2 – accumulate the value (do not reset the value during next displaying)

As you can see, an expression is the only obligatory parameter; all the rest can be skipped. Nevertheless, it is recommended to always use band parameters, since it would allow to avoid mistakes.

The “COUNT” function has the following syntax:

COUNT(band, flags)
COUNT(band)

The parameters assignment is similar to the one described above.

There is a general rule for all aggregate functions: a function can be counted only for the data-band and displayed only in the band’s footer (the following bands refer to the latter: footer, page footer, group footer, column footer, and report footer).

How do aggregate functions work? We will examine it using our example of report with groups. Let us add new elements to the report:

GroupHeader: GroupHeader1		
[Group."Cust"]	[Group."Company"]	
MasterData: MasterData1		
[Group."Orde"]	[Group."SaleDate"]	[Group."ItemsTotal"]
GroupFooter: GroupFooter1		
[SUM(<Group."ItemsTotal">,MasterData1)]		

The Group."ItemsTotal" field on the data-band will display the current order total. We place the "Text" object, containing the aggregate SUM call, to the group footer. It will display the total of all orders placed by the given client. Starting the report on accomplishment and using a calculator, we can make sure that everything works:

1221	Kauai Dive Shoppe	
1023	01.07.88	\$4 674,00
1076	16.12.94	\$17 781,00
1123	24.08.93	\$13 945,00
1169	06.07.94	\$9 471,95
1176	26.07.94	\$4 178,85
1269	16.12.94	\$1 400,00
		51450,8

So, how do the aggregate functions work? Before constructing a report, FastReport scans the "Text" objects' contents in order to find the aggregate functions. The functions found will be anchored to the corresponding data-bands (in our example, the "SUM" function is anchored to the "MasterData1" band). During construction of a report (when the data-band is displayed) the value of the aggregate functions linked to it is counted up. In our case, the "Group."ItemsTotal"" field's values are accumulated. After outputting a group footer (the one where the accumulated value of the aggregate function is displayed) the function value is reset, and the cycle is repeated for the next groups.

Now we should comment the purpose of the "Flags" parameter in the aggregate functions. In some reports, some of data-bands (or all of them) may be hidden, however, we might anyway need to count a value of the aggregate function considering all data-bands. So, in our example, the "Visible" property of the data-band can be disabled; after that it will stop displaying. To count a total on the hidden data-band, let us add the third parameter to the call of the function:

```
[SUM(<Group."ItemsTotal">,MasterData1,1)]
```

It will give us a report, which would look as follows:

1221	Kauai Dive Shoppe	51450,8
1231	Unisco	85643,6
1351	Sight Diver	261575,8

The “Flags = 2” parameter value allows to avoid reset of the accumulated function value right after it is displayed. This allows to receive the so-called “running total”. Let us update the call of the function:

[SUM(<Group."ItemsTotal">,MasterData1,3)]

The “3” value is a bit combination of “1” and “2,” which means that we need to take into consideration the invisible bands without resetting the total. As a result, we have:

1221	Kauai Dive Shoppe	51450,8
1231	Unisco	137094,4
1351	Sight Diver	398670,2

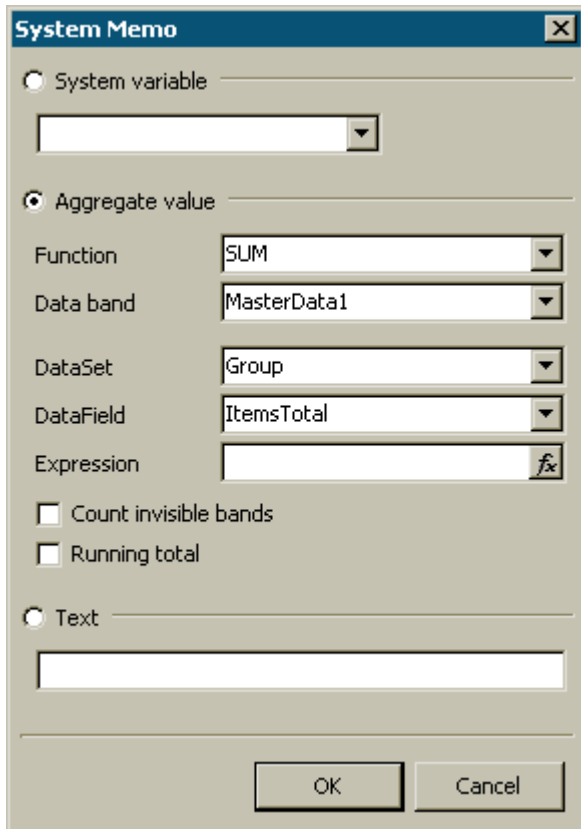
Page and report totals

Quite often, one needs to display total value of a page or a whole report. It can be performed with the help of the aggregate functions as well. Let us examine it with the help of our example.


Inserting aggregate function

So far we inserted the aggregate functions into the “Text” object manually. Let us examine more convenient ways of aggregate functions insertion.

First of all, we can use the “System text” object for the aggregate function value outputting. As a matter of fact, it is the same “Text” object, but one that has a special editor for more convenient insertion of system variables or aggregate functions.



You should step by step select a function type, a data-band (according to which it will be counted), and a DB field or an expression, value of which will be computed. You can otherwise mark the “Count invisible bands” and “Running totals” flags.

The second way is to use the “Text” object and the  button in its editor. At that, the additional window, similar to the examined “System text” object editor, appears. When clicking the “OK” button, the call of the aggregate function is inserted into the object's text.

The aggregate function call features

In this chapter we were discussed the cases when square or angle brackets must be used during insertion of expressions into the “Text” object. Remember that all the expressions, being non-standard in terms of the internal language interpreter (which is used for the expressions’ value calculation), should be enclosed in angle brackets. The DB fields (access to them is performed via the special construction of the “*Table name.*”*Field name*” kind), as well as variables from the variables list (as well as system variables) are included in this group.

It is necessary to use angle brackets when calling the aggregate functions because of their realization. So, the following record form is not correct:

```
[SUM(<Group."ItemsTotal">,MasterData1) * 2]
```

and this is the correct one:

```
[<SUM(<Group."ItemsTotal">,MasterData1)> * 2]
```

Let us also remember that in case, when the only member of the expression is enclosed in square brackets, FastReport allows omitting angle brackets, which means that both the record forms

```
[<SUM(<Group."ItemsTotal">,MasterData1)>]
[SUM(<Group."ItemsTotal">,MasterData1)]
```

are identical.

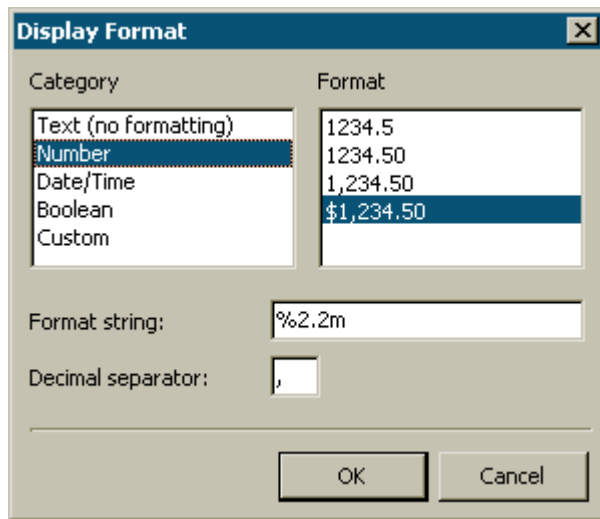
Values formatting

Draw attention to a peculiarity of use of aggregate functions: numerical values they return are not formatted. It becomes evident when referring to the first example with the “SUM” function:

1176	26.07.94	\$4 178,85
1269	16.12.94	\$1 400,00
		51450,8

This happens because, as a rule, the data fields return a formatted value, which is simply displayed by the “Text” object, with no changes applied. To apply external view to the “SUM” function result, let us use the values formatting tools of FastReport.

Let us select the object with the sum and call its contextual menu. The format editor is called either by using the “Formatting...” menu command, or with the help of the “DisplayFormat” property editor in the object inspector.



As you can see, the list of formatting categories and the list of the chosen category formats are placed on the left and on the right respectively. Let us select the “Number” category, and the “\$1,234.50” format. At that, the formatting line corresponding to the selected format and the decimal separator character will be displayed below. The formatting line is nothing but an argument of the “Format” function, with the help of which FastReport accomplishes formatting of numbers. You can modify a formatting line as well as a separator.

After clicking the “OK” button and report constructing, you might see that the sum total in the report is correct:

1176	26.07.94	\$ 4 178,85
1269	16.12.94	\$1 400,00
		<hr/>
		\$51450,80

Inline formatting

The examined way of formatting can be applied to any of the expressions, located in the object. In our case, everything works correctly because there is only one expression in the object. However, if we have two expressions and, in addition, they are of different types?

Let us examine the following case: the total and number of orders displaying in one object. To perform this, the following text must be placed into the object:

Total: [SUM(<Group."ItemsTotal">,MasterData1)]
Number: [COUNT(MasterData1)]

When starting, we can make sure that both of the values are presented in monetary format (which we have set in the previous example), which is rather incorrect:

1269	16.12.94	\$1 400,00
		Total: \$51 450,80
		Number: \$6,00

To receive correct display of values, each of them should be formatted individually. To perform this, there are the so-called format tags. They are added before the closing square bracket of the expression. In our example, let us disable the object formatting (select the “Text (without formatting)” category in the format editor). Now we need to modify the format of the first variable, since the second will definitely be displayed correctly (without formatting, i.e. as the integer, and this would be exactly what we need). To perform this, let us modify the object text in the following way:

Sum: [SUM(<Group."ItemsTotal">,MasterData1) #n%2,2m]
Number: [COUNT(MasterData1)]

And make sure that now the report works correctly:

1269	16.12.94	\$1 400,00
		Total: \$51 450,80
		Number: 6

Now let's draw attention to use of tags. The general syntax is the following:

[expression #tag]

Note that space between the expression and the “#” sign is obligatory! The tag itself might look as follows:

#nFormattingLine– the numerical format

#dFormattingLine– date/time format

#bFalse,True– boolean format

“*FormattingLine*” in every case is an argument for the function, with the help of which formatting is accomplished. Thus, for numerical formatting, such function would be the Delphi's Format function, for date/time it is the FormatDateTime function. Below are several lines' values used in FastReport:

for the numerical formatting:

%g – a number with the minimal signs number after decimal point

%2.2f – a number with the fixed number of signs after decimal point

%2.2n – a number with bits delimiter

%2.2m – a monetary format, accepted in the Windows OS, depending on the regional settings in the control panel.

for the date/time format:

dd.mm.yyyy – date of the 23.12.2003 type

dd mmm yyyy – date of the 23 Nov. 2003 type

dd mmmm yyyy – date of the 23 November 2003 type


hh:mm – time of the 23:12 type

hh:mm:ss – time of the 23:12:00 type


dd mmmm yyyy, hh:mm – time and date of the 23 November 2003, 23:12 type

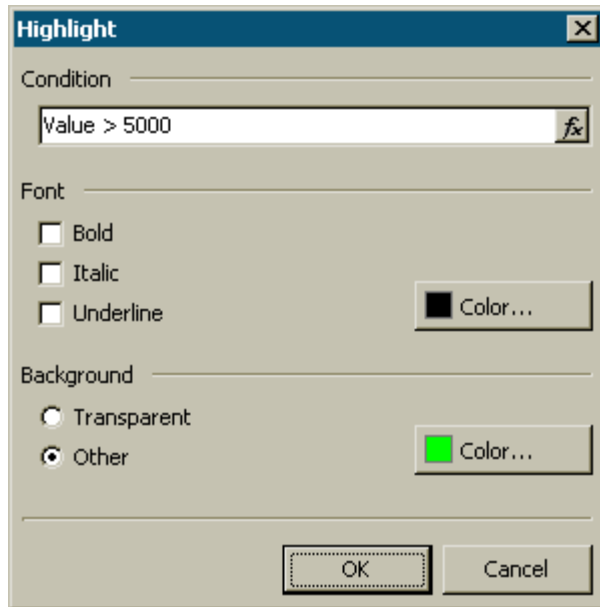
It is acceptable to place comma or dash instead of period in the line for the numerical format. In this case, this symbol will be used as a separator between the integer and the fractional parts of the figure. Usage of other separators is not acceptable.

As for formatting of the “#b” type (boolean), the formatting line is presented as two values separated by comma. The first value corresponds to “False,” the second one corresponds to “True.”

In order to avoid necessity to memorize all these tags and their meanings, there is a convenient resource for formatting insertion in the “Text” object editor. When clicking the  button, the format editor (which we have already examined) is called. After selecting a format, it is inserted to the text. Thus, if the cursor is placed before or after the closing square bracket, the format is inserted correctly.

Conditional highlighting

This feature of the “Text” object allows to color an object according to a specified condition. Any expression can be a condition. Let us exemplify coloring by the example with groups. Let the order totals, which are larger than 5000, be green-colored. To perform this, select an object with the “Group.ItemsTotal” field and click on the “Conditional highlighting”  button in the designer toolbar. In the opened conditional highlight editor, let us enter a condition, after performing which the object will be highlighted, and specify the color attributes (font parameters and background color).



The result will be as follows:

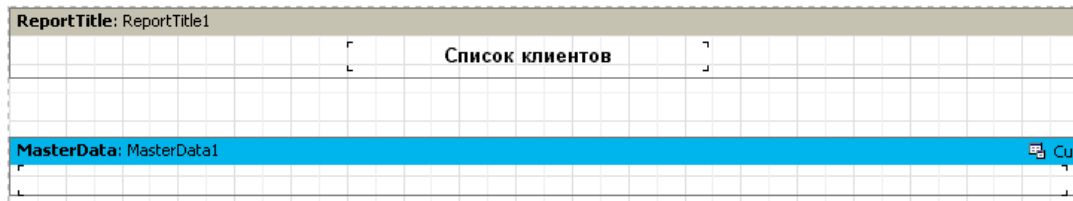
1221	Kauai Dive Shoppe	
1023	01.07.88	\$4 674,00
1076	16.12.94	\$17 781,00
1123	24.08.93	\$13 945,00
1169	06.07.94	\$9 471,95
1176	26.07.94	\$4 178,85
1269	16.12.94	\$1 400,00
		Total: \$51 450,80

Pay attention to the condition we specified (Value > 5000). Value is the DB field value, to which the object is linked. In similar way, the “<Group."ItemsTotal"> > 5000” condition may be set. In general, any expression, which is correct in terms of FastReport, may be specified here.

Show stripes

With the help of the conditional highlighting, it is easy to make a report look more sophisticated by “coloring” every second data line. Let us exemplify it by the report of the “List” type, which we constructed in the previous chapter.

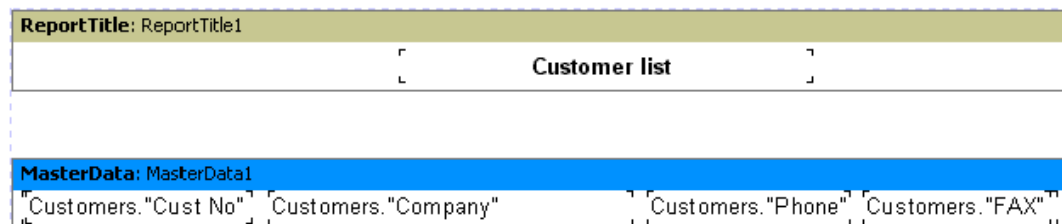
First of all, let us place the “Report title” and “Master data” bands on the list. Put the “Text” object to the data-band and stretch it, so that it would occupy practically all the band space:



This object will perform the role of the wafer, which will modify its color depending on the data line number. After that, we select the object and set the following condition in the allocation editor:

<Line> mod 2 = 1

Let us select gray color for highlighting, but not too saturated (closer to white). Now other objects can be put on the data-band:




As new objects lay on the wafer, it can easily be unnoticed. If starting a report, we can see the following:

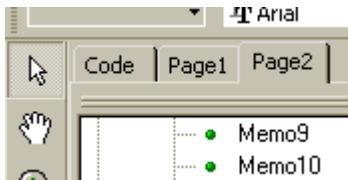
Customer list			
1645	Action Club	813-870-0239	813-870-0282
3158	Action Diver Supply	22-44-500211	22-44-500596
1984	Adventure Undersea	011-34-09054	011-34-09064
3053	American SCUBA Supply	213-654-0092	213-654-0095
6312	Aquatic Drama	613-442-7654	613-442-7678
3984	Blue Glass Happiness	213-555-1984	213-555-1995
1380	Blue Jack Aqua Center	401-609-7623	401-609-9403
1563	Blue Sports	610-772-6704	610-772-6898
2118	Blue Sports Club	612-897-0342	612-897-0348
3054	Catamaran Dive Club	213-223-0941	213-223-2324
1354	Cayman Divers World Unlimited	011-5-697044	011-5-697064
5151	Central Underwater Supplies	27-11-4432458	27-11-4433259
2156	Davy Jones' Locker	803-509-0112	803-509-0553
3055	Diver's Grotto	213-432-0093	213-432-4821


Multipage reports

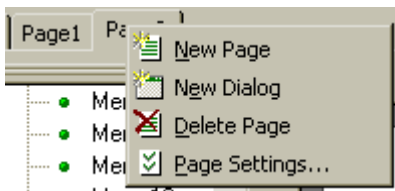
FastReport report can consist of several pages. You can adjust such parameters as size and orientation for each page, as well as to place different objects and bands on it.

When constructing a report, all bands from the first page will be displayed, then the bands from the second one, etc.

When a user creates a new report in the designer, it already contains one page by default. You can add a new page by clicking on the  button in the toolbar or by selecting the “File|New page” menu command. Then you would see that a new bookmark appears in the designer:



One can easily switch between pages by clicking on the required bookmark. Moreover, bookmarks can be dragged (“drag&drop”), thus easily modifying order of pages. An unnecessary page can be deleted with the help of the  button in the toolbar or by selecting the “Edit|Delete page” menu command. One can also call the context menu by right-clicking the bookmark:




Number of pages in a report is unlimited. As a rule, additional pages are used either for previewing title pages, or in more complicated reports, those which contain data from many sources.

Let us examine a simple example of title page creation. Let us use the report with one data level, which we have built before. Add a new page to it, and thus it would be the second page. To move it to the top of the report, seize the page bookmark with the help of the mouse, and then drag it to the place near the first page. At that, the pages order will be changed. Let us switch to a new page and place the “Text” object (with the “Our report” text inside) in the middle of the page. That is all; the report with a title page is finished:

Customers				
Company	Address	Contact	Phone	Fax
Action Club	PO Box 9481-F	Michael Sourling	813-870-0280	813-870-0282
Action Diver Supply	Blue Spar Box 83	Marianna Milos	22-44-800 211	22-44-800 896
Adventure Undersea	PO Box 744	Glenn Gonzalez	011-34-09 054	011-34-09 054
American SCUBA Supply	1750 Atlantic Avenue	Lynn Cinquini	213-854-0302	213-854-0305
Aquatic Drama	621 Everglades Way	Gillian Owen	613-442-7854	613-442-7878
Blue Glass Happiness	8946 W. Shona Lane	Christine Taylor	213-895-1884	213-895-1885
Blue Jack Aqua Center	25-701 Piedington Lane	Ernest Swart	401-899-7823	401-899-4403
Blue Sports	203 12th Ave. Box 748	Thomas Kunic	610-772-8704	610-772-8788
Blue Sports Club	63395 Nix Pines Street	Henry Balthazar	612-887-0342	612-887-0348
Catamenian Dive Club	Box 254 Pleasure Point	Nicola Dupont	213-223-0411	213-223-2324
Cayman Divers World Unlimited	PO Box 541	Jon Bailey	011-8-887 044	011-8-887 054
Central Underwater Supplies	PO Box 787	Maria Eventash	27-11-443 248	27-11-443 3293
Dave Jones' Locker	246 South 19th Place	Tanya Wagner	803-889-0112	803-889-0553
Dive'n'Gratin	24801 Universal Lane	Peter Owen	213-432-0363	213-432-4821
Divers of Blue-green	634 Complex Ave.	Nancy Bean	206-885-7184	206-885-8059
Divers of Corfu, Inc.	Memmoat Place 54	Charles Lopez	30-881-88394	30-881-88343
Divers of Venice	230 Elm Street	Simona Gnan	813-443-2399	813-443-8342
Divers-for-Hire	G.O. P Box 911	Jon Hatter	879-804 878	879-809 348
Flamethike Aquatics	232 585 S 5A-77 A.A.	Susan Wong	087-1-775 434	087-1-775 421
Flamethike's Eye	PO Box 7562	Bethan Lusk	803-885-4880	803-885-4889
Frank's Divers Supply	1488 North 44th St.	Lloyd Faltow	803-885-2778	803-885-2789
George Bean & Co.	873 King Salmon Way	Bill Weyers	803-435-2771	803-435-3003
Gold Coast Supply	223-B Houston Place	Elaine Falls	206-885-2940	206-885-4094
Island Finders	6133 1/3 Stone Avenue	Diamond Ortega	713-423-6775	713-423-8778
Jamaica SCUBA Centre	PO Box 88	Barbara Harvey	011-3-887 043	011-3-887 043
Jamaica Sun, Inc.	PO Box 545	Jonathan Wiles	803-885-2748	803-885-0329
Kauai Dive Shop	4-8788 Sugarloaf Hwy	Erica Norman	803-885-0299	803-885-0276
Kirk Enterprises	42 Aqua Lane	Rudolph Claus	713-895-6437	713-895-1073
Larry's Diving School	3952 NW Buca Street	Isabell's Nisco	803-403-7777	803-403-0059
Makal SCUBA Club	PO Box 8834	Donna Slaus	317-840-0088	317-840-8787
Marine SCUBA Center	PO Box 82483 Zulu 7831	Stephen Bryant	88-33-881 222	88-33-881 046
Marine Divers Club	872 Ocean St.	Joyce Marsh	415-885-0369	415-885-0369
Naptime's Trident Supply	PO Box 120	Louise Francis	778-887-3548	778-887-8543
Northeast SCUBA Limited	PO Box 8934	Angela Jones	778-123-0146	778-123-0165
Ocean Adventures	PO Box 492 Khal	Paul Gill	778-888-8334	778-888-8383
Ocean Paradise	PO Box 8748	Paul Gardner	803-885-8231	803-885-8480
On-TARGET SCUBA	7-737 63 Nankwara Road	Brian Phillips	415-445-0283	415-445-0223
Princess Island SCUBA	PO Box 32 Waiyevo	Anna Marachi	879-311 623	879-311 203
Professional Divers, Ltd.	4704 Waiwala St.	Arney Mathers	206-885-6333	206-885-4094
Safari Under the Sea	PO Box 7488	Anna Beck	803-403-4233	803-403-3002
San Pablo Dive Center	1701-D N Broadway	Patricia O'Brien	823-044-2010	823-044-2060
SCUBA Heaven	PO Box C-8874	Robert Michelland	011-32-09 488	011-32-09 488
Shangri-La Sports Center	PO Box D-9468	Frank Parique	011-32-08 874	011-32-44 838
Sight Diver	1 Naptime Lane	Phyllis Spooner	387-8-878 103	387-8-878 943
The Daph Charge	15243 Underwater Hwy.	Sam Witherspoon	803-885-3768	803-885-0383
The Diving Company	PO Box 8938	Brian Miles	22-44-80 088	22-44-80 878
Tom Sawyer Diving Centre	823-1 Third & Frydsholj	Chris Thomas	804-788-3023	804-788-7772
Tony Tony Tony	PO Box H-4873	Kevin Rider	803-885-0343	803-885-8584
Underwater Fantasy	PO Box 842	Glen Newirth	803-885-2214	803-885-2234

It is necessary to focus attention on one feature of multipage reports. If the “Print to previous page” option is enabled in the second page (use the “PrintToPreviousPage” property in the object inspector), then the second page objects will start printing not from a new list, but on the white space of the previous one. This allows to print the pages’ contents “line-to-line.”

Nested reports (subreports)

Sometimes it is required to display in a particular place additional data, which may represent a separate report with rather complicated structure. One can try to construct such report by using a set of FastReport bands, but it is not always possible. In such cases, the “Subreport” object can be used .

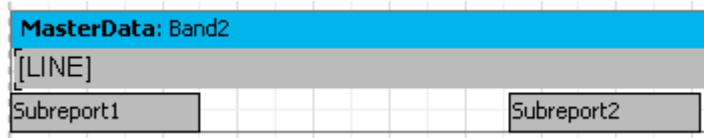
After inserting such object to a report, we can see that FastReport automatically adds a new page, connected to this object. A nested report resembles a multipage one in terms of structure. The only difference is that the nested report is displayed in a specified place of the basic report, and not after it. When creating a report, as soon as the “Subreport” object occurs, the report, allocated in the connected page, will be displayed. After that, basic report creation will continue.

One can also place the “Subreport” object on a nested report page, thus increasing enclosure level. An example of such report can be found in the demo program, the “Subreports” report.

It should be noted that the FastReport's ability to construct subreports enables to increase nesting level of data. Remember that number of enclosure levels in FastReport is limited when you do not use the "Subreport" object (not more than six).

Side-by-side subreports

You can allocate two or more "Subreport" objects side by side on the same band:



This allows to construct reports, which cannot be constructed in a different way (i.e. when the lists of different length are displayed in each nested report):

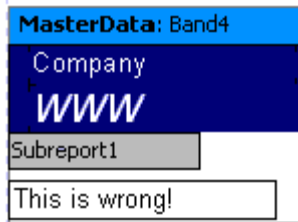


As you can see, FastReport continues to construct the basic report, beginning from the position, where previewing of the longer list is already finished.

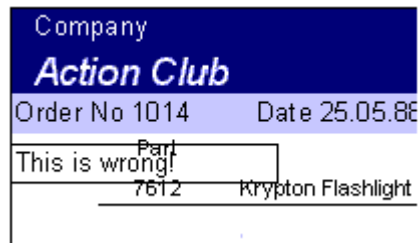
Limitations on using subreports

Since nested reports are created on the basic report page, it is unable to contain the following bands: "ReportTitle/ReportFooter," "PageTitle/PageFooter/PageBackground," and "ColumnTitle/ColumnFooter." It is possible to put these bands on the nested report page, but they however will not be handled. For the same reason, there is no sense in modifying nested report pages options, inasmuch as the options of basic report's page are used during constructing a report.

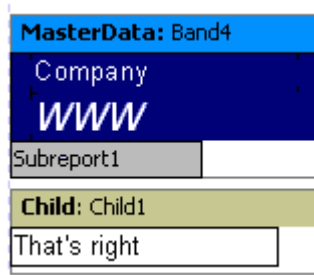
It is impossible to put objects below the "Subreport" object:



When displaying a nested report, the nested report objects will overlay everything placed below, and a user will receive something like that:



To display the objects below the nested report anyway, use the child-band:

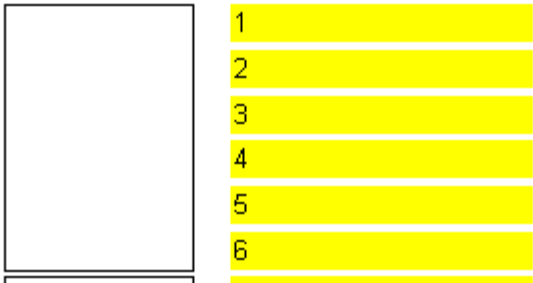


It also concerns cases when it is necessary to display several nested reports one under another.

PrintOnParent option

The "Subreport" object has the "PrintOnParent" property which can be useful in some cases. This property is False by default.

Usual subreport is printed as a set of bands on the basic report page. In this case the parent band (which contains a "subreport" object) do not depends on the subreport bands, i.e. can't stretch. If the "PrintOnParent" property is True (you can set it from the object inspector or in the context menu), subreport's objects are printed physically on the band which contains the "subreport" object. You can make this band stretched and put on it stretched objects:



Cross-tab reports

This kind of report has table structure, which means that it consists of lines and columns. At the same time, it is not known beforehand, how many lines and columns a table would possess. That is why a report grows not only downwards (as the report types examined above) but sideways as well. A typical example of a report of such type is shown below.

Let us examine the elements of the table:

	1	2	3	4
a	a1	a2	a3	a4
b	b1	b2	b3	b4

In the picture, we see a table with two lines and four columns, where “a” and “b” are *line titles*, “1,” “2,” “3,” and “4” are *column titles*, and “a1”..”a4,” “b1”..”b4” are *cells*. To construct a report like this, we need just one set of data (a query or a table), which has three fields and contains the following data:

a	1	a1
a	2	a2
a	3	a3
a	4	a4
b	1	b1
b	2	b2
b	3	b3
b	4	b4

As you can see, the first field contains a line number, the second one have a column number, and the third one contains the cell contents at intersection of the table with the selected number. When constructing a report, FastReport creates a table in memory and fills it with data. Thus, the table expands dynamically, if a line or a column with a specified number does not exist.

Titles can consist of more than one level. Let us examine the following example:

	10		20	
	1	2	1	2
a	a10.1	a10.2	a20.1	a20.2
b	b10.1	b10.2	b20.1	b20.2

In this example, the number, or *index* of the column is composite, i.e. it consists of two values. This report requires the following data:

a	10	1	a10.1
a	10	2	a10.2
a	20	1	a20.1
a	20	2	a20.2
b	10	1	b10.1
b	10	2	b10.2
b	20	1	b20.1
b	20	2	b20.2

In this example, the first field contains the line index, as it was before; the second and the third fields contain column indexes. The last field contains the cell value. Let us examine the following picture in order to make it clear, how FastReport constructs a tables with complex titles:

	10	10	20	20
	1	2	1	2
a	a10.1	a10.2	a20.1	a20.2
b	b10.1	b10.2	b20.1	b20.2

Before handling is accomplished, our table would look like the table shown in the picture. During handling, FastReport unites the title cells with equal values, which are allocated on one level.

The next table element, which is shown in the following picture, displays intermediate totals and totals:

	10			20			Total
	1	2	Total	1	2	Total	
a	a10.1	a10.2	a10.1+a10.2	a20.1	a20.2	a20.1+a20.2	sum(a)
b	b10.1	b10.2	b10.1+b10.2	b20.1	b20.2	b20.1+b20.2	sum(b)
Total	a10.1+b10.1	a10.2+b10.2	a10.1+b10.1+a10.2+b10.2	a20.1+b20.1	a20.2+b20.2	a20.1+b20.1+a20.2+b20.2	sum(a)+sum(b)

This report is constructed using the same data, as were used in the previous one. The columns, highlighted with gray in the picture, are calculated automatically and are not included into the initial data set.

Construct a cross-report

Now let us turn from theory to practice. Let us construct a simple cross-report, which would display employees' salary during four years. To perform this, we need the "crosstest" table, which is available in the FastReport "DEMO" DB. The table contains data of the following kind:

Name	Year	Salary
Ann	1999	3300
Ben	2002	2000
....		


As usually, let us create a new report in FastReport, put the “ADOTable” component on the report form and set it:

Table1:

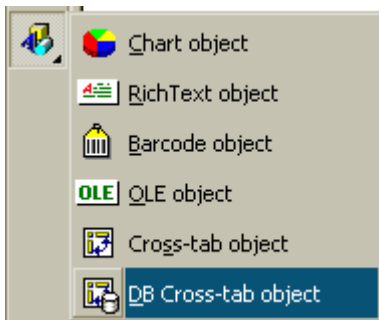
DatabaseName = 'DefaultConnection '

TableName = 'crosstest'

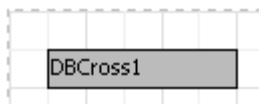
UserName = 'SimpleCross'

For cross-reports construction, one should use the “TfrxCrossObject” component  from the FastReport component palette. Just put it on the form; it is not required to set anything.

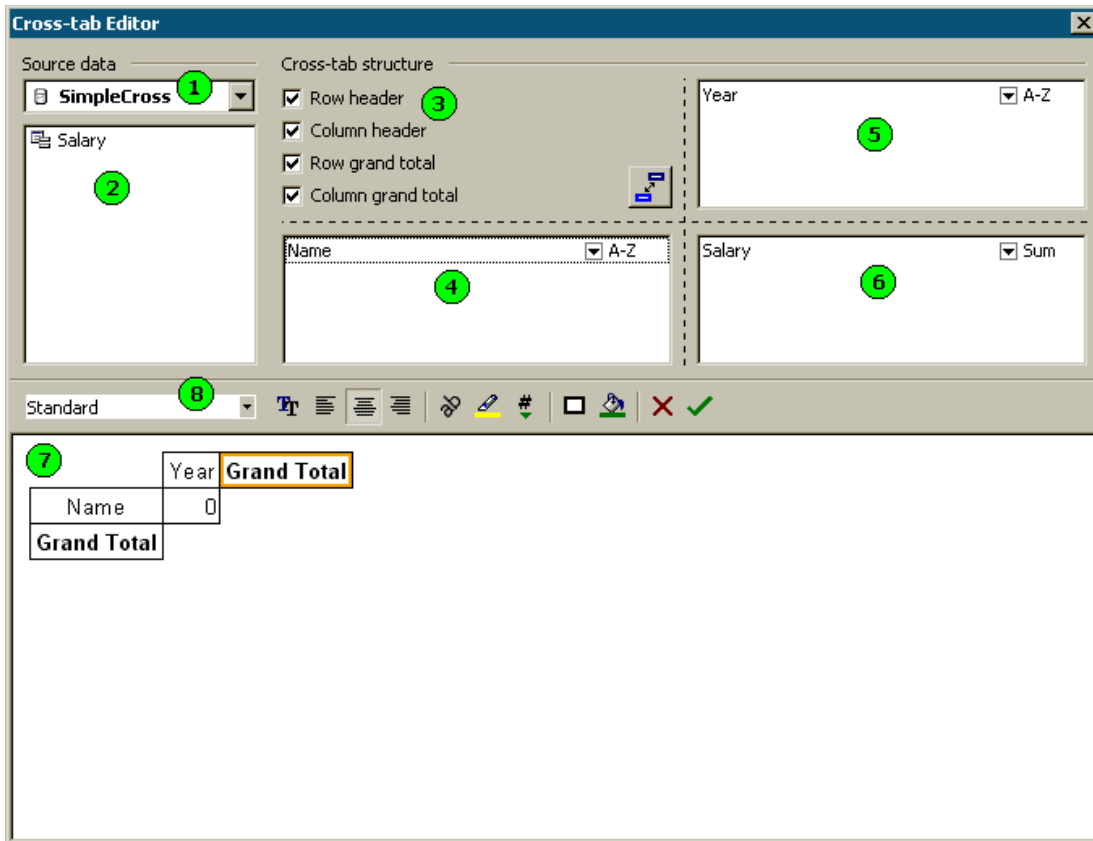
Let us enter the report designer. First of all, connect our data source to the “Report|Data...” menu. Put the “DB cross-tab” object on the report list:



On the designer list the object looks lowly:



All settings are specified with the help of the object editor. Let us call it by double-clicking on the object:



The following items are denoted by figures in the picture:

- 1 – the drop-down list with available data sources;
- 2 – the list of fields in the selected data source. The fields from this list can be dragged to the “4,” “5,” and “6” lists;
- 3 – here one can specify whether it is necessary to display titles and totals;
- 4 – the list of fields, which generate a line title;
- 5 – the list of fields, which generate a column title;
- 6 – the list of fields, which generate a table cell;
- 7 – here the future table structure is previewed. All the elements in the table are clickable;
- 8 – toolbar for modifying table design:



- table style select;



- cell font parameters;



- text alignment;



- text rotation;




- conditional highlighting;



- cell format;




- cell frame and filling.

As you can see, it is possible to operate here only with the help of the mouse. In our case, it is enough to drag fields from the “2” list to the “4,” “5,” and “6” lists, as it is shown in the picture. Let us not do anything yet. Close the editor by clicking the “OK” button . If starting the report now, you would see a table like the one below:

	1999	2000	2001	2002	Grand Total
Ann	3300	2700	3100	1700	10800
Ben	4300	2400		2000	8700
Catherine	6100	3200			9300
Den		3999	8100		12099
Grand Total	13700	12299	11200	3700	40899

Well, it is exactly what we wanted to receive. Let us continue examining the object. Call the object editor once again. The first thing we want to perform is to modify the titles’ colors and to display “Total” instead of “Grand total.” It is very easy to perform when using the bottom editor field (N7 in the picture). Here the cross-table structure is displayed, and it can also be set with the help of the mouse. The active call is displayed with an orange frame:

	Year	Grand Total
Name	0	
Grand Total		

To change the title color into gray, click on the “Year,” “Name,” and “Grand Total” objects one after another, and then select the desired color via the  button in the toolbar. To change the “Grand Total” inscription, double-click on the cell, and then you will see the familiar text editor, where one should type “Total.” After that, our report will look as follows:

	1999	2000	2001	2002	Total
Ann	3300	2700	3100	1700	10800
Ben	4300	2400		2000	8700
Catherine	6100	3200			9300
Den		3999	8100		12099
Total	13700	12299	11200	3700	40899

It remains to set a format, where the currency values are displayed. To perform this, in the cross-object editor, click on the “Total” object and the object, representing a cell (with the “0” text) one after another and select the required format by clicking on the



button in the toolbar. You will get the following result:

	1999	2000	2001	2002	Total
Ann	\$3 300,00	\$2 700,00	\$3 100,00	\$1 700,00	\$10 800,00
Ben	\$4 300,00	\$2 400,00		\$2 000,00	\$8 700,00
Catherine	\$6 100,00	\$3 200,00			\$9 300,00
Den		\$3 999,00	\$8 100,00		\$12 099,00
Total	\$13 700,00	\$12 299,00	\$11 200,00	\$3 700,00	\$40 899,00

Using functions

In our example, we previewed the sum total of each employee’s salary during four years in the “Total” line. One can use the following functions:

SUM – sum of values

MIN – minimal value

MAX – maximal value

AVG – average value

COUNT – number of values

Let us try to use the “MIN” function in our example. To perform this, open the cross-object editor and click on the "Salary" field in the area of the item with down arrow.



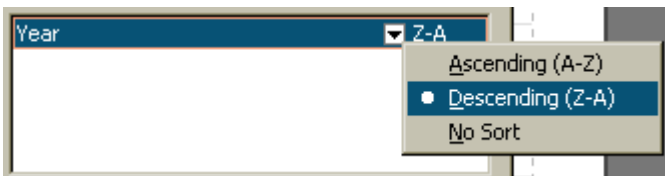
Select the “MIN” function in the menu. Now one can modify a text in the cell of totals from “Total” to “Minimum.” A finished report will look as follows:

	1999	2000	2001	2002	Minimum
Ann	3300	2700	3100	1700	1700
Ben	4300	2400		2000	2000
Catherine	6100	3200			3200
Den		3999	8100		3999
Minimum	3300	2400	3100	1700	1700

Sorting values

Lines and columns values are arranged in ascending order. At that, if values have numerical type, they are sorted by value, and if they have line type, they are sorted alphabetically. We can separately set our own sorting mode for each line and/or column value. The following modes are available: “arrange in ascending order,” “arrange in descending order” and “perform no sorting.” In the latter case, values in lines/columns will be displayed depending on their entries.

Let us modify column sorting in our example. Let years be arranged in decreasing order. To perform this, let us enter the cross-object editor and select the “Year” column element. To modify sorting, click on the area of the item with down arrow:



After that, close the editor and start the report. It will look as follows:

	2002	2001	2000	1999	Total
Ann	1700	3100	2700	3300	10800
Ben	2000		2400	4300	8700
Catherine			3200	6100	9300
Den		8100	3999		12099
Total	3700	11200	12299	13700	40899

Table with composite headers

Our previous example contained one value per line, and column headers. Let us examine in practice the table construction with a complex header, which means that it

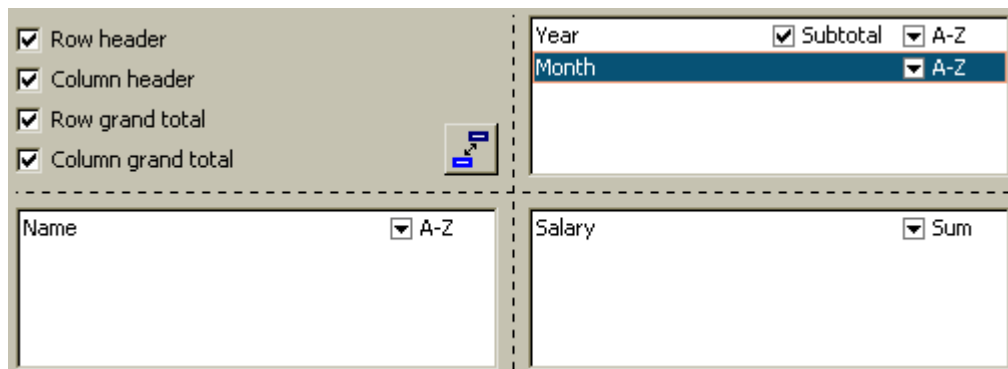
would contain two or more values. To perform this, we also would need the “crosstest” table, which is located in the FastReport Studio “DEMO” DB. The table contains data of the following kind:

Name	Year	Month	Days	Salary
Ann	1999	2	3	1000
Ben	2002	1	5	2000
....				

We have added the “Month” and “Days” fields, which contain month number and the number of working days respectively. One can construct several reports on the basis of this data, for example, salary of all the employees during all years, burst by months.

To set the cross-object, let us start its editor by double-clicking on the object.

What kind of a report we are going to get? It must resemble the report from the previous example, but at the same time it must be burst by months. Consequently, the cross-object must be set in the same way, but with adding the “Month” field into the column header:



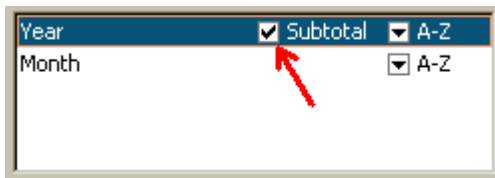
Thus, in the bottom part of the editor the future report structure is displayed:

	Year	Grand Total
	Month	
Name		
Grand Total		

As a result, we would get the following report:

	1999					2000				2001				2002		Grand Total
	2	10	11	12	Total	1	2	3	Total	1	2	3	Total	1	Total	
Ann	1000		1100	1200	3300	1300	1400		2700		1500	1600	3100	1700	1700	10800
Ben		2100	2200		4300		2400		2400				0	2000	2000	8700
Catherine		3000	3100		6100			3200	3200				0		0	9300
Den					0	3999			3999	4000	4100		8100		0	12099
Grand Total	1000	5100	6400	1200	13700	5299	3800	3200	12299	4000	5600	1600	11200	3700	3700	40899

Note, that FastReport automatically added a column of the intermediate totals, which are displayed after each year. This option can be set in the cross-object editor: it is enough to select the “Year” column element and disable the “Subtotal” flag:



In addition, one can note that there is no intermediate total in the bottommost column element (the same is true in cases, when this element is the only one). Actually, in our example, we do not need intermediate totals for each month.

Let us examine another feature, concerning intermediate totals. In our example, it is desirable to display “2000 year total” instead of the “Total” inscription. It is quite easy to be performed. Enter the cross-object editor, select the required object in the bottom part of the editor, and then enter the following text to it:

Total for [Value]

During construction, the “Value” expression will be replaced by the table inscription value, located above:

	1999				
	2	10	11	12	Total for 1999
Ann	1000		1100	1200	3300
Ben		2100	2200		4300
Catherine		3000	3100		6100
Den					0
Grand Total	1000	5100	6400	1200	13700

Adjusting cell width

When looking at the previous picture, it becomes obvious that FastReport automatically adjusts cells width in a way, which allows the longer lines to fit the cells. It is not desirable in some cases however, since the table with very long lines becomes not

good-looking. What can be done in such case? The simplest way is to break lines in the text of object with intermediate totals, i.e. to insert a line into it:

Total
for[Value]

You see that the table looks better now:

	1999				
	2	10	11	12	Total for 1999
Ann	1000		1100	1200	3300
Ben		2100	2200		4300
Catherine		3000	3100		6100
Den					0
Grand Total	1000	5100	6400	1200	13700

However, not always such method can be used. If the lines'/columns' values are rather long, they cannot be corrected by breaking the line manually. That is why the cross-object has the "MinWidth" and "MaxWidth" properties (minimal and maximal cell width respectively). Both these properties are accessible only via the object inspector.

The "MinWidth" value is "0," and the "MaxWidth" value is "200" by default. This is quite enough in most cases. You can set your values, according to any special requirements concerning table design.

Thus, in our example, we can set the following: MinWidth = MaxWidth = 50. This would signify that table cell width must be 50 pixels at any rate. If a cell is smaller, it is "adapted" to the "MinWidth" value, if it is bigger, its width is fixed according to the "MaxWidth" value, and the text in the cell is divided. In our example, it would look as follows:


	1999				
	2	10	11	12	Total for 1999
Ann	1000		1100	1200	3300
Ben		2100	2200		4300
Catherine		3000	3100		6100
Den					0
Grand Total	1000	5100	6400	1200	13700

Font colors and highlighting

Sometimes it is necessary to highlight values and/or change font color. We have examined such task in the example of a report with groups. Then, we used conditional highlighting for the “Text” object, which can be useful for us now as well.

Let us examine process of highlighting, using our example. Assume that we need to change font color for the values, which are more than 3000. To perform this, let us enter the cross-object editor and click on the object, representing the table cell, in the bottom of the editor window:

	Year		Grand Total
	Month	Total for Year	
Name			
Grand Total			

To set highlighting parameters, click on the  button in the toolbar. The already familiar highlighting editor window will open, where one should set the following condition:

Value > 3000

This is all we need. Close the editor with the help of the “OK” button and start our report:

	1999					2000			
	2	10	11	12	Total for 1999	1	2	3	Total for 2000
Ann	1000		1100	1200	3300	1300	1400		2700
Ben		2100	2200		4300		2400		2400
Catherine		3000	3100		6100			3200	3200
Den					0	3999			3999
Grand Total	1000	5100	6400	1200	13700	5299	3800	3200	12299

In exactly the same way, a user is able to highlight total values, columns and lines, if necessary.

Managing a cross-table from the script

If setting table visual resources are not enough, one can use the script for detailed settings for the appearance of the table. The “Cross-table” object has the following events:

Event	Description
OnAfterPrint	Event is called after printing a table.
OnBeforePrint	Event is called before printing a table
OnCalcHeight	Event is called before calculating length of a row in the table. The event handler can return either the required value of height, or “0” when the row needs to be hidden.
OnCalcWidth	Event is called before calculating column’s width in a table. The event handler can return either the required value of width, or “0” when the column needs to be hidden.
OnPrintCell	Event is called before displaying a table’s cell. The event handler can modify the cell’s design or its contents.
OnPrintColumnHeader	Event is called before displaying a title of the table’s columns. The event handler can modify design or contents of the title’s cell.
OnPrintRowHeader	Event is called before displaying a title of the table’s rows. The event handler can modify design or contents of the title’s cell.

It is convenient to use the following methods of the “Cross-table” object in events:

Method	Description
function ColCount: Integer	Returns the number of columns in a table.
function RowCount: Integer	Returns the number of rows in a table.
function IsGrandTotalColumn (Index: Integer): Boolean	Returns “True,” if the column with specified number is the total one.
function IsGrandTotalRow (Index: Integer): Boolean	Returns “True,” if the row with specified number is a total one.
function IsTotalColumn (Index: Integer): Boolean	Returns “True,” if the column with specified number is a column with intermediate totals.
function IsTotalRow (Index: Integer): Boolean	Returns “True,” if the line with specified number is a line with intermediate totals.
procedure AddValue(const Rows, Columns, Cells: array of Variant)	Adds a value to the table.

Let us exemplify, how one can highlight the third column (in our example it is the “November 1999” date). To perform this, select a cross-table and create the OnPrintCell event’s handler:

C++ Script:

```
void Cross1OnPrintCell(
TfrxMemoView Memo,
int   RowIndex,
int    ColumnIndex,
int    CellIndex,
Variant RowValues,
Variant ColumnValues,
Variant Value)
{
    if (ColumnIndex == 2)
    { Memo.Color = clRed; }
}
```

Pascal script:

```
procedure Cross1OnPrintCell(Memo: TfrxMemoView;
    RowIndex, ColumnIndex, CellIndex: Integer;
    RowValues, ColumnValues, Value: Variant);
begin
    if ColumnIndex = 2 then
        Memo.Color := clRed;
end;
```

We will see the following result:

	1999				
	2	10	11	12	Total
Ann	1000		1100	1200	3300
Ben		2100	2200		4300
Catherine		3000	3100		6100
Den					0
Grand Total	1000	5100	6400	1200	13700

To highlight a column title, create an “OnPrintColumnHeader” event’s handler:

C++ Script:

```
void Cross1OnPrintColumnHeader(
TfrxMemoView Memo,
Variant HeaderIndexes,
Variant HeaderValues,
Variant Value)
{
    if ((VarToStr(HeaderValues[0]) == "1999")
```

```

    &&
    (VarToStr(HeaderValues[1]) == "11")
  {
    Memo.Color := clRed;
  }
}

```

Pascal Script:

```

procedure Cross1OnPrintColumnHeader(Memo: TfrxMemoView;
  HeaderIndexes, HeaderValues, Value: Variant);
begin
  if (VarToStr(HeaderValues[0]) = '1999') and
    (VarToStr(HeaderValues[1]) = '11') then
    Memo.Color := clRed;
end;

```

Result would appear as follows:

	1999				
	2	10	11	12	Total
Ann	1000		1100	1200	3300
Ben		2100	2200		4300
Catherine		3000	3100		6100
Den					0
Grand Total	1000	5100	6400	1200	13700

Let us explain how the scripts work. The “OnPrintCell” event handler is called before printing a cell included in the table’s body (when printing cells from the table title, either the “OnPrintColumnHeader,” or the “OnPrintRowHeader” handler is called). At the same time, a link to the “Text” object, which represents a table’s cell (“Memo” parameter), and the cell’s “address” in two variants: the number of row, column and cell (the last is relevant, if your table contains multi-leveled cells) in the “RowIndex,” “ColumnIndex,” and “CellIndex” parameters respectively, are transmitted into the “OnPrintCell” handler. The “RowValues” and the “ColumnValues” parameters are the second variant of the “address.” The “Value” parameter is the cell’s contents.

To specify an “address,” you can use the second variant (RowValues, ColumnValues), since it is more convenient in the given case (as well as the first one (RowIndex, ColumnIndex)). In our case, it was necessary to highlight the third column; therefore, it would be more convenient to analyze the first variant. Since numbering of columns and rows begins with “0,” the “ColumnIndex = 2” checking allows us to define the third column. One could do it in a different way, i.e. by analyzing the required column by its data (we need the 11th month of 1999):

C++ Script:

```

void Cross1OnPrintCell(
    TfrxMemoView Memo,
    int RowIndex,
    int ColumnIndex,
    int CellIndex,
    Variant RowValues,
    Variant ColumnValues,
    Variant Value)
{
    if ((VarToStr(ColumnValues[0]) == "1999") &&
        (VarToStr(ColumnValues[1]) == "11"))
    {
        Memo.Color = clRed;
    }
}

```

Pascal Script:

```

procedure Cross1OnPrintCell(Memo: TfrxMemoView;
    RowIndex, ColumnIndex, CellIndex: Integer;
    RowValues, ColumnValues, Value: Variant);
begin
    if (VarToStr(ColumnValues[0]) = '1999') and
        (VarToStr(ColumnValues[1]) = '11') then
        Memo.Color := clRed;
end;

```

Values, which are transferred in the “RowValues” and the “ColumnValues” parameters, are arrays of the “Variant” type with a zero base. The zero element is a value of the highest level of the table’s title; the first one is a value of the next level, etc. In our case, the “ColumnValues[0]” contains years, and the “ColumnValues[1]” contains months.

Why is “VarToStr” transformation necessary? This guarantees absence of mistakes during type conversion. When operating with the “Variant” type, FastReport attempts to automatically cast the strings to number format, which, in its turn, can lead to an error when attempting to cast the “Total” and “Grand Total” columns’ values.

The “OnPrintColumnHeader” event handler is called during typing column title cells. The set of parameters is similar to the parameters of the “OnPrintCell” handler, although in this case the cell’s “address” (the “HeaderIndexes” and “HeaderValues” parameters) is transferred in a different way. The “HeaderValues” parameter returns the same values, as the “ColumnValues” and “RowValues” parameters in the “OnPrintCell” handler. The “HeaderIndexes” parameter is also an array of values of the “Variant” type, which contains an address of the title’s cell in a different form: the zero element is the serial number of the highest level of the table’s title, the first one is the number of the next level, etc. To make the principle of cells numbering clear, refer to the picture below:

	0					1				2			
	0	1	2	3	4	0	1	2	3	0	1	2	3
0	1000		1100	1200	3300	1300	1400		2700		1500	1600	3100
1		2100	2200		4300		2400		2400				0
2		3000	3100		6100			3200	3200				0
3					0	3999			3999	4000	4100		8100
4	1000	5100	6400	1200	13700	5299	3800	3200	12299	4000	5600	1600	11200

In our case, it is more convenient to analyze the “HeaderValues” value, but one can write the following handler as well:

C++ Script:

```
void Cross1OnPrintColumnHeader(
TfrxMemoView Memo,
Variant HeaderIndexes,
Variant HeaderValues,
Variant Value)
{
    if (HeaderIndexes[0] == 0) && (HeaderIndexes[1] == 2)
        Memo.Color = clRed;
}
```

Pascal Script:

```
procedure Cross1OnPrintColumnHeader(Memo: TfrxMemoView;
HeaderIndexes, HeaderValues, Value: Variant);
begin
    if (HeaderIndexes[0] = 0) and (HeaderIndexes[1] = 2) then
        Memo.Color := clRed;
end;
```

Adjusting rows/columns size

The user can adjust width and height of the table’s rows and columns with the help of the “OnCalcWidth” and “OnCalcHeight:” events’ handlers. Let us show how to increase width of the column, which corresponds to the 11th month of 1999 by the following example. To perform this, let us create the “OnCalcWidth” event’s handler:

C++ Script:

```
void Cross1OnCalcWidth(
int ColumnIndex,
variant ColumnValues,
Extended &Width)
{
    if ((VarToStr(ColumnValues[0]) == "1999") &&
        (VarToStr(ColumnValues[1]) == "11"))
```



```

    {
      Width = 100;
    }
  }
}

```

Pascal Script:

```

procedure Cross1OnCalcWidth(ColumnIndex: Integer;
  ColumnValues: Variant; var Width: Extended);
begin
  if (VarToStr(ColumnValues[0]) = '1999') and
    (VarToStr(ColumnValues[1]) = '11') then
    Width := 100;
end;

```

And the result would look as follows:

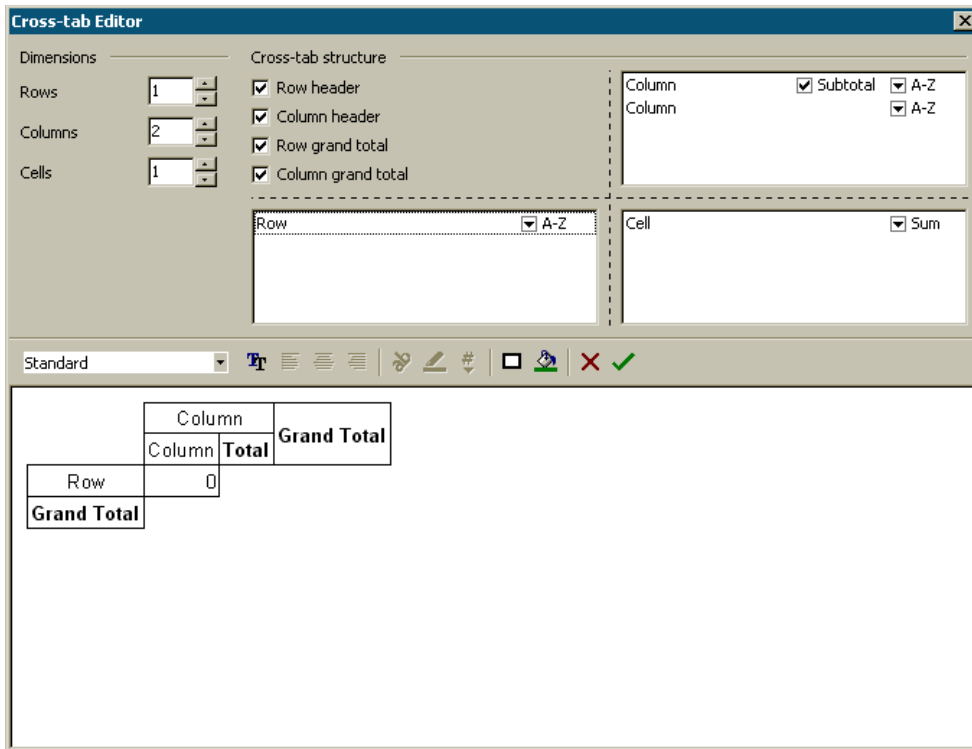
	1999				
	2	10	11	12	Total
Ann	1000		1100	1200	3300
Ben		2100	2200		4300
Catherine		3000	3100		6100
Den					0
Grand Total	1000	5100	6400	1200	13700

In our example, to hide a column, it is enough to return the Width := 0. Note, that the sums are not recalculated at the same time, since the matrix is already full of values by this moment.

Filling a table manually

As you already know, there are two versions of the cross-table: the “DB cross-table” and the “Cross-table.” All this time we’ve been working with the first object attached to the data from the DB table and fills itself automatically, as soon as the report runs. Let us examine the second object, “Cross-table.”

This object is not attached to the data from DB. Therefore, you are to fill the table with data manually. This object possesses a similar editor, but you would have to select the number of dimensions in the table’s titles and in its cells instead of DB fields:



Let us demonstrate operating with the “Cross-table” object with an example. Put an object on the list of the report and set it in a way, as it is shown in the previous picture: the number of levels in the strings’ title is “1,” in the columns’ title – “2,” in the cell – “1.” To fill the table with data, let us use the “OnBeforePrint” object’s event handler:

C++ Script:

```
void Cross1OnBeforePrint(TfrxComponent Sender)
{
    Cross1.AddValue(["Ann"], [2001, 2], [1500]);
    Cross1.AddValue(["Ann"], [2001, 3], [1600]);
    Cross1.AddValue(["Ann"], [2002, 1], [1700]);

    Cross1.AddValue(["Ben"], [2002, 1], [2000]);

    Cross1.AddValue(["Den"], [2001, 1], [4000]);
    Cross1.AddValue(["Den"], [2001, 2], [4100]);
}
```

Pascal Script:

```
procedure Cross1OnBeforePrint(Sender: TfrxComponent);
begin
    with Cross1 do
    begin
        AddValue(['Ann'], [2001, 2], [1500]);
        AddValue(['Ann'], [2001, 3], [1600]);
        AddValue(['Ann'], [2002, 1], [1700]);
```

```

    AddValue(['Ben'], [2002, 1], [2000]);

    AddValue(['Den'], [2001, 1], [4000]);
    AddValue(['Den'], [2001, 2], [4100]);
end;
end;

```


In the handler, it is necessary to add the required data into the table via the “TfrxCrossView.AddValue” method. This method has three parameters; each of them is an array of values of the “Variant” type. The first parameter is the row's value, the second one is the column's value, and the third one contains the cells' values. Note that the number of values in each array should correspond to the object's setting! In our case, the object has one level in the rows' title, two levels in the columns' title, and one level of cells. Therefore, we transfer one value for rows, two values for columns, and one value for cells into the AddValue.

When running the report, we would see the following:

	2001				2002		Grand Total
	1	2	3	Total	1	Total	
Ann		1500	1600	3100	1700	1700	4800
Ben				0	2000	2000	2000
Den	4000	4100		8100		0	8100
Grand Total	4000	5600	1600	11200	3700	3700	14900

One can use the “AddValue” method for the “DB cross-table” object as well. This allows adding the data (which are not in the data source attached to the object) into the cross-table. Otherwise, if there are such data, they are summarized with the data in the table.

Diagrams

FastReport allows to insert diagrams into the report. For this purpose, the “TfrxChartObject”  object from the FastReport component palette is used.

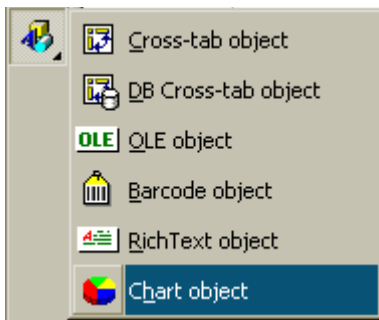
Let us illustrate a simple construction of a diagram using the following example. To perform this, we would need the “country” table from the demo database distribution kit. The table contains data about countries, their area and population:

Name	Area	Population
Argentina	2 777 815	32 300 003
Bolivia	1 098 575	7 300 000
....		

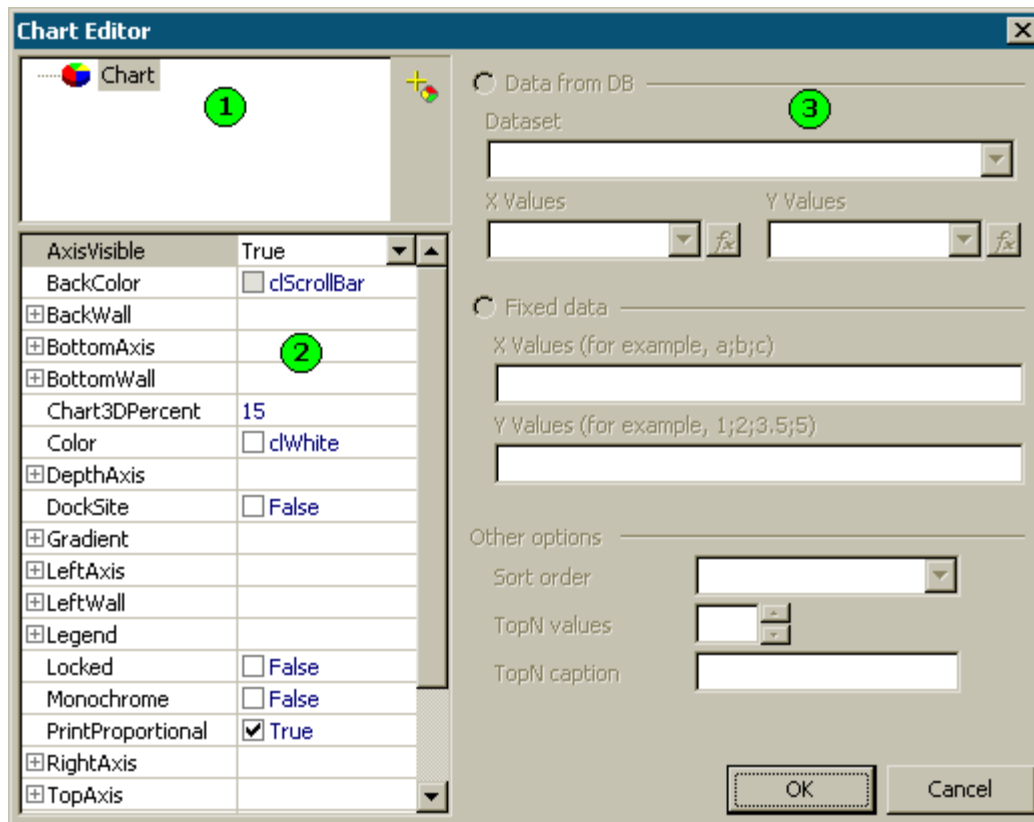
Let us create a new report in FastReport. Put the “ADOTable” component on the report form and then customize it:

```
ADOTable1:  
DatabaseName = ' DefaultConnection '  
TableName = 'country'  
UserName = 'Country'
```

Let us put the “Diagram” object to the report list:




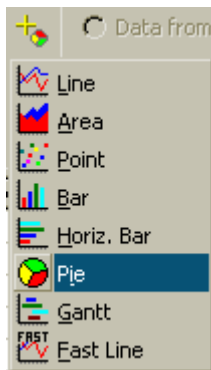
Let us set the object size (18x8 cm). To customize the object, call its editor by double-clicking on it.



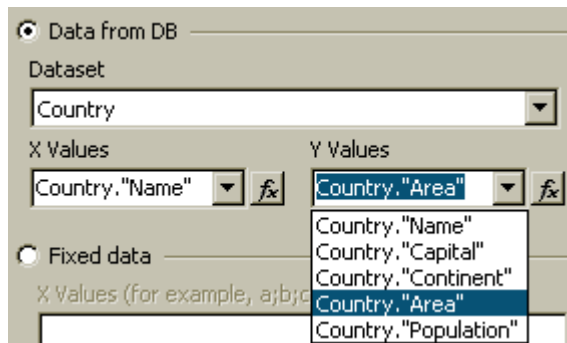
The following items are denoted by figures in the picture:

- 1 – diagram structure. Diagram can contain either one or several series.
- 2 – object inspector, which displays the properties of the element selected in the window. Thus, you can perform tweaking of the diagram properties.
- 3 – toolbar for connection the series to data; it is activated as soon as the series in the window 1 are selected.

During the first activation, the editor window will contain an image shown in the picture. The first thing to be done is to add one or several series (one series in our example). To perform this, click the  button and select the pie diagram in the menu:

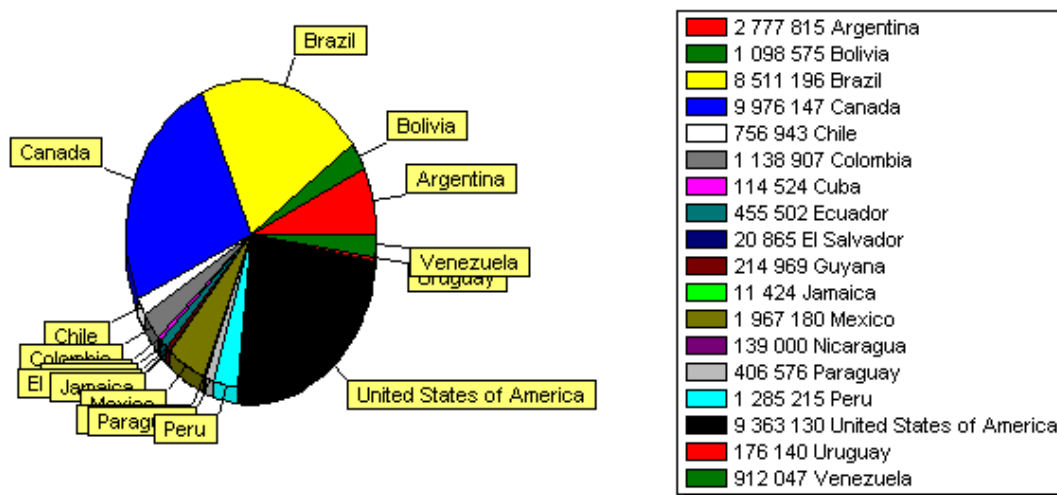


As you can see, there are eight different types of series available. After adding the series, the bar 3 becomes active. Here you should specify, which data should be used for plotting. First of all, let us select the data set in the “Data set” pulldown. Fill the “X values” and “Y values” fields in the following way (they can also be selected from the pulldowns):

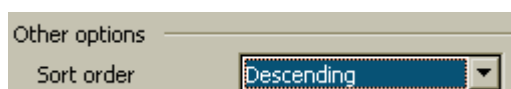


It is necessary to note here, that X-axis values can be of any type (string, for example), since they are informational. However, the Y-axis values must rigorously be of figure type. In our case (with circle diagram) the X-axis values are used for comments of inscriptions displaying, while only the Y-axis values are used for the diagram construction.

Let us close the setting (click “OK” to close the editor), and then start report construction:



What can be improved in this report? First of all, it would be nice to sort values in descending order. Again, we enter the diagram editor and select the series in the upper part of the window. Now we select the required sorting mode:



If starting the report now, we would see that the data in the explaining table is sorted.

Limitation of number of diagram values

Our diagram looks rather overloaded, since there are too many small values in the diagram, which are invisible anyway. FastReport allows limiting the values number in a diagram by a predefined value. Thus, all the values, which do not belong the limit set, would be displayed as a single value, representing the sum of values, which did not fit the diagram.

In our example, the diagram has 18 values, and only 8 of them can be displayed. Let us enter the editor and set limiting:

Other options

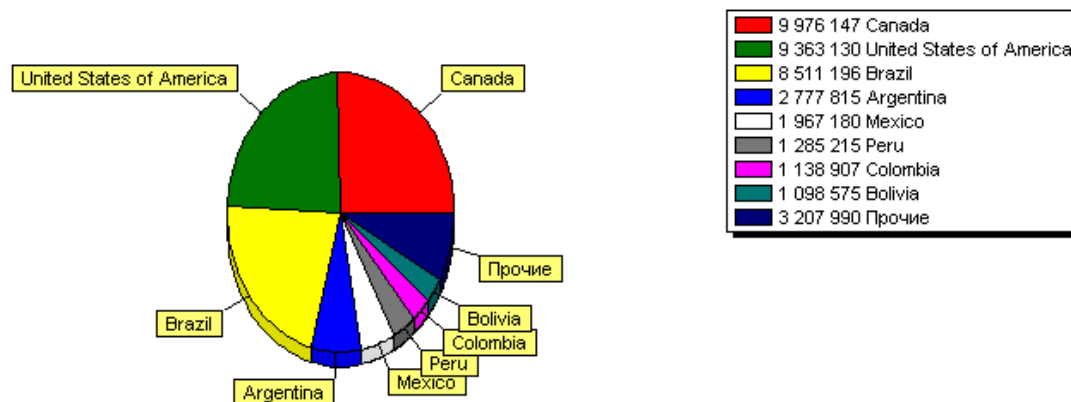
Sort order: Descending

TopN values: 8

TopN caption: Прочие

The limiting will work if the “TopN” is nonzero. The name in the “TopN title,” which will be displayed opposite to the sum value, should be specified. Sorting mode is not significant; values will be sorted by default.

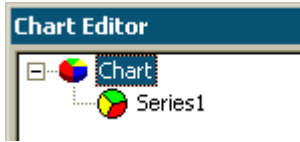
As a result, the report will look as follows:



Some useful settings

Let us examine several sets, which can be useful for setting diagram appearance. These settings can be specified in the object inspector only.

The following basic properties are available when selecting a diagram in the top of the list:




- Gradient – settings for gradient background filling. Enable the “Gradient.Visible” property for gradient displaying.
- Legend – settings for explanatory table appearance. The table can be disabled with the help of the “Legend.Visible” property. The table position is set with the help of the “Legend.Alignment” property.

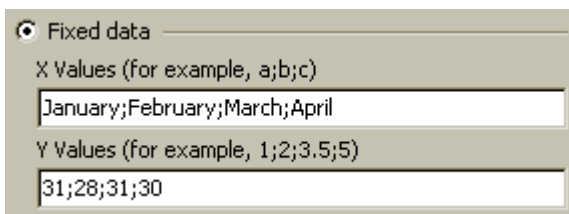
The following properties are available when selecting a series:

- ColorEachPoint – color each value with different colors.
- ExplodeBiggest – select the largest value (only for the series of the “circle diagram” type).
- Marks – settings for the explanatory hints appearance.
- ValueFormat – the line for formatting values.

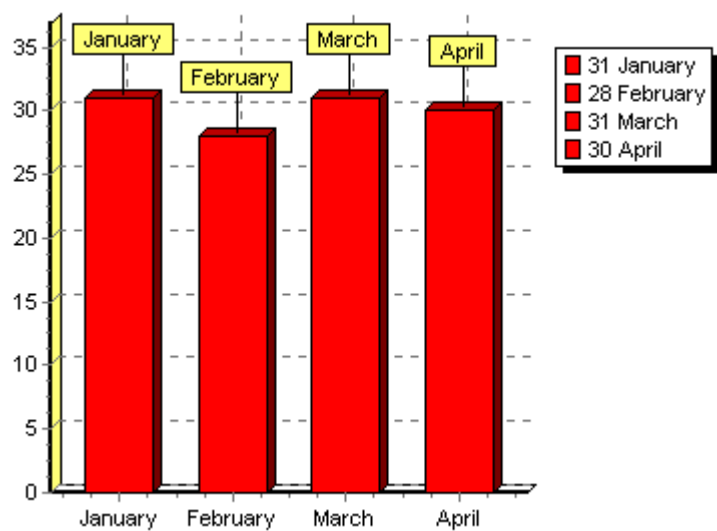
Diagram with specified values

In the previous example, we constructed a diagram on the basis of the DB table data. There is another way of constructing a diagram: to enter the necessary data manually. This way is convenient when constructing small diagrams.

Let us demonstrate how it works with a simple example. Put a diagram to the report list and enter its editor. Add the series of the “column diagram” type  and set its properties:



The result can also be viewed in the designer, without starting the report:



Script

Script is a program written in a higher-level language, which is a part of a report. As a report runs, the script runs as well. A script is able to perform data handling, which cannot be performed via regular means of the FastReport core, for example, to hide useless data according to any predefined condition. The script is also used for controlling properties of dialog forms, which are the components of the report.

The script should be written using one of the languages, which are the components of the script engine (FastScript). Currently, the following languages are supported:

- BasicScript
- C++Script
- JScript
- PascalScript

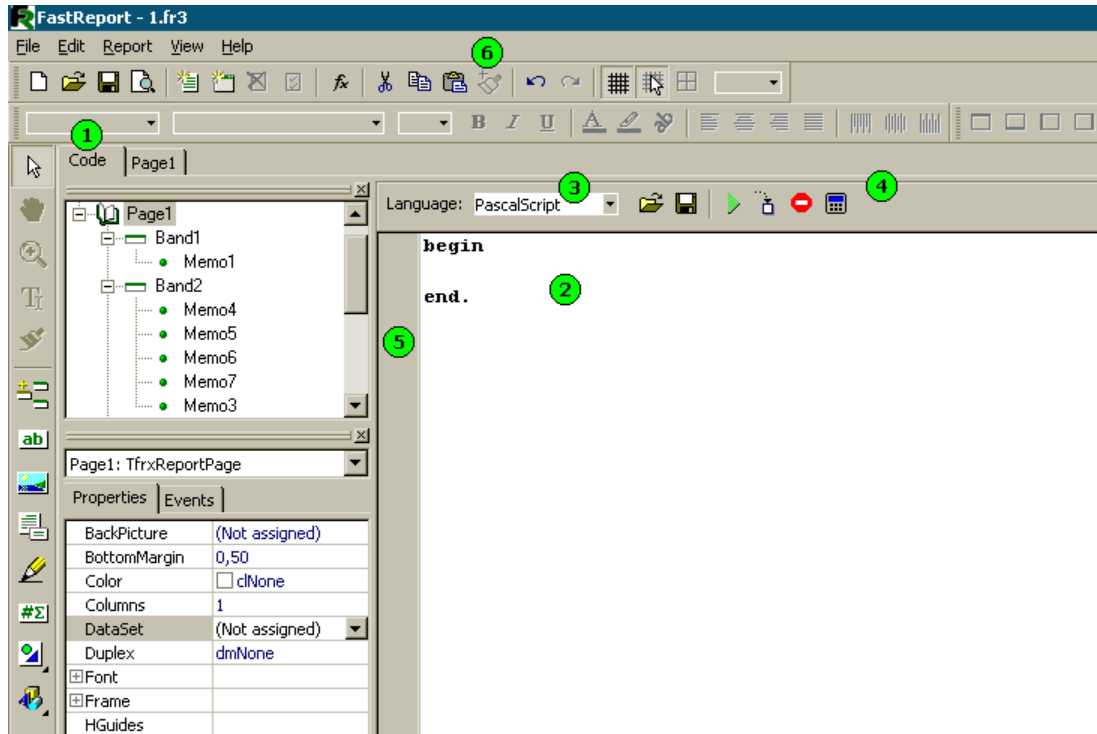
The following FastScript features available in the script engine:

- standard language set: variables, constants, procedures, functions (with nesting capability) with variables, constants, default parameters, all standard operators (including case, try, finally, except, with), types (integral, fractional, logical, character, line, multidimensional arrays, set, variant), classes (with methods, events, properties, indexes, and default properties);
- declarations of the following types absent: records, classes in the script; no records, no pointers, no sets (however, the 'IN' - "a in ['a'..'c','d']" operator usage is possible), no shortstring type, no unconditional jump (GOTO);
- types' compatibility checking;
- ability to access any report's object.

You can create scripts in the FastReport designer, which contains the scripts' editor with syntax's highlighting. Also there is an embedded debugger, which possesses the following functions: "Step," "Breakpoint," "Run to cursor," and "Evaluate."

Taste of script

Tools for working with the script are located in the “Code” tab of the FastReport editor. When switching to this tab, the designer appears as follows:



In the picture above, the figures denote:

- 1 – “Code” tab;
- 2 – script’s editor window;
- 3 – a dropdown for selecting a language, in which the script is to be written;
- 4 – debugger’s toolbar:



- run report in debugging mode;



- perform the regular code line (Step into);



- interrupt script’s work;



- preview expressions’ evaluation (Evaluate).

- 5 – bookmarks and breakpoints are displayed in this field; in addition, the lines, possessing the executable code are highlighted there;
- 6 – you can also use the buttons in the basic toolbar:



- cut text;



- copy text;



- paste text;



- undo the previous action.

Below there is the list of the keys, which can be used in the script editor.

Key	Meaning
Cursor arrows	Move the cursor
PageUp, PageDown	Go to the previous/next page
Ctrl+PageUp	Go to the beginning of the text
Ctrl+PageDown	Go to the end of the text
Home	Go to the beginning of the line
End	Go to the end of the line
Enter	Go to the next line
Delete	Delete the symbol at cursor's position; delete the selected text
Backspace	Delete the symbol to the left from the cursor
Ctrl+Y	Delete the current line
Ctrl+Z	Undo last action (up to 32 events)
Shift+Cursor arrows	Select a text block
Ctrl+A	Select the whole text
Ctrl+U	Shift the selected block by 2 symbols to the left
Ctrl+I	Shift the selected block by 2 symbols to the right
Ctrl+C, Ctrl+Insert	Copy the selected block to the clipboard
Ctrl+V, Shift+Insert	Paste the text from the clipboard
Ctrl+X, Shift+Delete	Cut the selected block to the clipboard
Ctrl+Shift+<number>	Set a bookmark with the 0..9 number on the current line
Ctrl+<number>	Jump to the set bookmark
Ctrl+F	Search a line
Ctrl+R	Replace a line
F3	Repeated search/replacement from the cursor's position
F4 or F5	Set the breakpoint and script's running (Run to cursor)
Ctrl+F2	Reset the program
Ctrl+F7	Preview variables' values (Evaluate)
F9	Run the script (Run)
F7 or F8	Execute code line (Step into)

Structure of a script

Script's structure depends on the language you use; however there are some common elements. They are the script's title, body, and the main procedure, which will be executed when the report runs. Below there are examples of the scripts for all four supported languages:

BasicScript's structure:

```
#language BasicScript // optionally

// the "imports" chapter should be located before any other chapter
imports "unit1.vb", "unit2.vb"

dim i, j = 0 // the "variables" chapter can be placed anywhere

function p1() // functions
{ //
}

// main procedure.
for i = 0 to 10
    p1()
next
```

C++Script's structure:

```
#language C++Script // optional

// the "include" chapter should be placed before any other chapter
#include "unit1.cpp", "unit2.cpp"

int i, j = 0; // the "variables" chapter can be placed anywhere

#DEFINE pi = 3.14159 // "constants" chapter

void p1() // functions
{ // no nested procedures
}

{ // main procedure.
}
```

JScript's structure:

```
#language JScript // optionally

// the "import" chapter should be before any other chapter
import "unit1.js", "unit2.js"

var i, j = 0; // the "variables" chapter can be located
anywhere

function p1() // functions
```

```
{                                //  
}  
                                // main procedure.  
p1();  
for (i = 0; i < 10; i++) j++;
```

PascalScript's structure:

```
#language PascalScript // optional  
program MyProgram;      // optional  
  
// the "uses" chapter should be located before any other chapter  
uses 'unit1.pas', 'unit2.pas';  
  
var                      // the "variables" chapter can be placed anywhere  
  i, j: Integer;  
  
const                    // "constants" chapter  
  pi = 3.14159;  
  
procedure p1;            // procedures and functions  
var  
  i: Integer;  
  
  procedure p2;          // nested procedure  
  begin  
    end;  
  
begin  
end;  
  
begin                    // main procedure.  
end.
```

More detailed description of the FastScript script engine can be found in its documentation. The author did not duplicate the following moments in the manual:

- syntactic diagrams of all the supported languages;
- supported data types;
- operations with classes, properties, methods, and events;
- nested functions;
- enumerations and sets.

Later, we will examine examples of scripts written in "C++Script" and "PascalScript" languages.

"Hello, World!" script

We have already examined an example of the "Hello, World!" report; now let us view, how to create a simplest script, which would display a window with a greeting.

Let us create a blank project in Delphi. Put the “TfrxReport” component to the form. Enter the designer and click on the “New report” button for FastReport to automatically create a blank template. Switch to the “Code” bookmark and write the following script:

C++ Script:

```
{  
    ShowMessage("Hello, World!");  
}
```

Pascal Script:

```
begin  
    ShowMessage('Hello, World!');  
end.
```

After that, run the report. As we expected, FastReport displays a little window with a greeting:



Let us explain some details. We created a script consisting of a single “begin..end” block. Thus, our script has a very simple structure; it consists of a main procedure only (see the “Structure of a script” chapter). The main procedure is executed as soon as the report runs. In our case, it displays a greeting window; the procedure ends right after the window is closed. After the main procedure finished, report building starts.

Using objects in the script

One can address any report’s object from the script. So, if there are, say, the “Page1” page and the “Memo1” object, one can use them in the script, calling them by names, for example:

C++ Script:

```
Memo1.Color = clRed
```

Pascal Script:

```
Memo1.Color := clRed
```

The list of the report's objects available from the script is displayed in the "Report tree" service window. What objects' properties are available in the script? The answer is simple: those ones, which are visible in the objects' inspector. At the same time, at the bottom of the inspector, there is a hint concerning the selected property. Both windows (report's tree and inspector) are available during working with the script. To get a detailed help about objects' properties and methods, use the FastReport help file, which is included in distribution kit.

Let us demonstrate the aforesaid with a simple example. Put the "Text" object with the "MyTextObject" name and the "Test" text into the report's page. Then write in the script:

C++ Script:

```
{  
    MyTextObject.Color = clRed  
}
```

Pascal Script:

```
begin  
    MyTextObject.Color := clRed  
end.
```

Run the report and see that our object's color became red.

Calling the variables from the report's variables list

One can call any variable, which is specified in the list of the report's variables ("Report|Variables..." menu item), from the script. Variable's name thus should be enclosed in angle brackets:

C++ Script:

```
if (<my variable>==10)  
{  
    ...  
}
```

Pascal Script:

```
if <my variable> = 10 then ...
```

An alternative way is to use the "Get" function:

C++ Script:

```
if (Get("my variable") == 10)
```

```
{  
  ...  
}
```

Pascal Script:

```
if Get('my variable') = 10 then ...
```

Modification of such variable's value is available only via the “Set” procedure:

C++ Script:

```
Set("my variable", 10);
```

Pascal Script:

```
Set('my variable', 10);
```

One should address the system variables, such as “Page#,” in exactly the same way:

C++ Script:

```
if (<Page#> == 1)  
{  
  ...  
}
```

Pascal Script:

```
if <Page#> = 1 then ...
```

Calling the DB fields

Just as in case with variables, one should use angle brackets for calling the DB fields:

C++ Script:

```
if (<Table1."Field1"> == Null)  
{  
  ...  
}
```

Pascal Script:

```
if <Table1."Field1"> = Null then...
```

And just as well, one can use the “Get” function (as a matter of fact, this function is always used in implicit way for calculating expressions, enclosed in angle brackets).

Using aggregate functions in the script

Unlike other functions, one should call the aggregate functions (“SUM,” “MIN,” “MAX,” “AVG,” and “COUNT”) either using angle brackets, or via the “Get” function (see the “Features of calling the aggregate function” section):

C++ Script:

```
if (<Sum(<Table1."Field1">, MasterData1)> > 10) { ... }
```

Pascal Script:

```
if <Sum(<Table1."Field1">, MasterData1)> > 10 then...
```

Another feature of an aggregate function is that it should be used inside the “Text” object; one can call it in the script afterwards. If the aggregate function is used in the script only (without using it in the “Text” object), an error message will appear. That happens due to the fact that an aggregate function must be connected with a definite band, and only then it would work correctly.

Displaying the variable’s value in a report

To display the contents of any script variable in a report, one should describe this variable and bind a value to it. Here is a simple example of the script:

C++ Script:

```
char MyVariable()  
{  
    MyVariable = "Hello!";  
}
```

Pascal Script:

```
var  
    MyVariable: String;  
begin  
    MyVariable := 'Hello!';  
end.
```

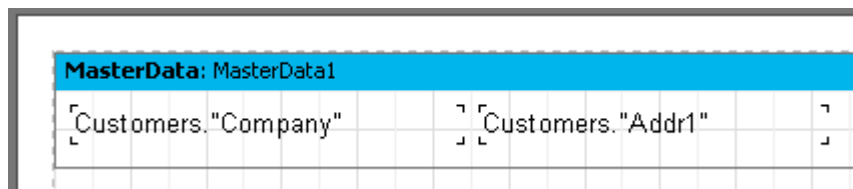
The variable’s value can be displayed in the “Text” object, for example, by placing the *[MyVariable]* line into it.

A variable's name should be unique, which means that it should not coincide with the names of the report's objects, standard functions, and constants. If there is an error in the script, a message will be displayed, and report construction process will be stopped.

Events

So far we have examined scripts with only one main procedure, which is performed when a report starts running. In the main procedure, one can perform any initial settings, as well as to initialize variables. However, this is not enough for total control over the process of report's forming. To control a report as much as possible, every report's object has several events, to which a handler (i.e. a procedure from the script) may be assigned. For example, in the handler, connected to the data-band, one can perform records' filtering, which means that the band will be hidden or displayed according to any specified conditions.

Let us demonstrate the process of creation of a report and of events, which are generated during it, with the example of a simple report, which contains one page, one "Master data" band, and two "Text" objects on the band:



MasterData: MasterData1	
Customers. "Company"	Customers. "Addr1"

As it was said, the main script's procedure is called in the very beginning of a report. After that, the essential process of report construction starts. In the beginning of the report, the "OnStartReport" event of the "Report" object is called. Before the page is being formed, the "OnBeforePrint" page event is called. This event is called once for each page of the report's template (it should not be confused with the pages of a finished report!). In our case, the event is called once, as the report's pattern consists of one page, notwithstanding the number of pages in the finished report.

Then typing of data-bands begins. It is performed in the following way:

1. the "OnBeforePrint" band's event is called;
2. the "OnBeforePrint" events of all the objects, belonging to the band, are called;
3. all the objects are filled with data (in our case with values of the "Company" and "Addr1" DB fields); after that, the "OnAfterData" events of all the objects are called;
4. such actions as positioning of objects on the band (if there are stretchable objects among them), calculating of the band's height, and stretching (if it is stretchable) are performed;
5. the "OnAfterCalcHeight" band's event is called;
6. a new page is formed, if the band does not find room in white space of the page;
7. the band and all of its objects are displayed on the finished report's page;

8. the “OnAfterPrint” event of all the band's objects is called;
9. the “OnAfterPrint” event of the band itself is called.

Bands are printed as long as there are data in the source connected to the band. After that, in our case, forming of a report stops; the “OnAfterPrint” report's page events and, finally, the “OnStopReport” event of the "Report" object are called.

Thus, via using events of different objects, one can manage practically each moment of report's forming process. A key to correct use of events is complete understanding of the bands' typing process, stated in nine latter sections. Thus, most of the actions can be performed via using the “OnBeforePrint” band's event only; any modifications made to an object are displayed simultaneously. However, in this event it is impossible to analyze, in which page the band will be printed, if it is stretchable, since calculation of band's height will be performed in the step 4. This can be performed either via the “OnAfterCalcHeight” event in the step 5, or the “OnAfterPrint” event in the step 8, but in the latter case a band will be already printed and that is why any operations with objects will change nothing. In a word, you should clearly see, in what period of time each of the events should be called and use those events, which correspond to the set task.

Example of using the “OnBeforePrint” event

Let us show the aforesaid in practice. Let us create a report, which represents the list of clients. This report will include only those companies, which names begin with the letter “A.”

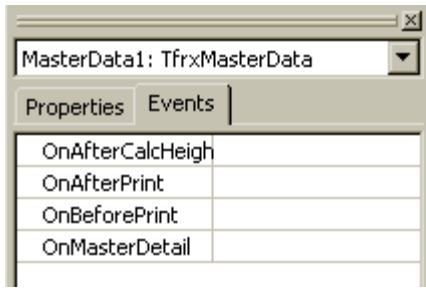
Let us create a new report in FastReport, put the “ADOTable” component to the report form and set it:

```
ADOTable1:  
DatabaseName = ' DefaultConnection '  
TableName = 'customer'  
UserName = 'Customers'
```

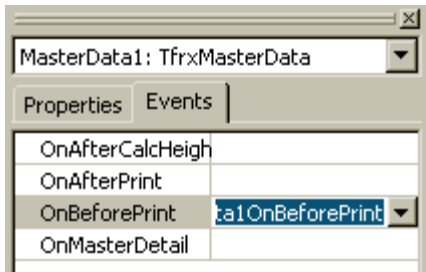
Enter the report's editor and create a report of the following type:

MasterData: MasterData1		
[Customers."Company"]	[Customers."Contact"]	[Customers."Phone"]

Let us extract a data-band and switch to the “Events” bookmark in the objects' inspector:



To create the “OnBeforePrint” event’s handler (this is exactly what would be most appropriate to us), double-click on the blank field in front of the event’s name:



At the same time, a blank handler is being added to the script’s text, and the designer switches to the “Code” bookmark:

```
procedure MasterData1OnBeforePrint(Sender: TfrxComponent);  
begin  
  |  
end;  
  
begin  
  
end.
```

As you can see, everything operates in a similar way as in Delphi environment. The only thing we should do after that is to write the following code in the handler’s body:

C++ Script:

```
if (Copy(<Customers."Company">, 1, 1) == "A")  
{  
  MasterData1.Visible = true  
}  
else  
{  
  MasterData1.Visible = false;  
}
```

Pascal Script:

```

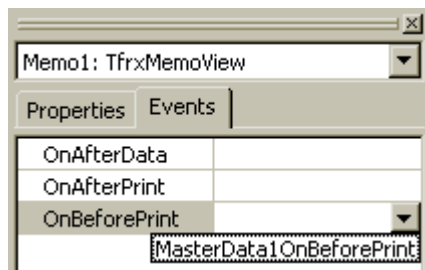
if Copy(<Customers."Company">, 1, 1) = 'A' then
  MasterData1.Visible := True else
  MasterData1.Visible := False;

```

Run the report and make sure, that the script works correctly:

Action Club	Michael Spurling	813-870-0239
Action Diver Supply	Marianne Miles	22-44-500211
Adventure Undersea	Gloria Gonzales	011-34-09054
American SCUBA Supply	Lynn Cinciripini	213-654-0092
Aquatic Drama	Gillian Owen	613-442-7654

Let us explain several details. You can assign one handler to several events of different objects at once; in this case the “Sender” parameter defines the object, which has initiated the event. To assign a name of the already existing handler to the event, one can either enter it manually in the objects’ inspector, or select it in the pulldown:



The link to the handler can be easily deleted. To do that, select a required property and click the “Delete” key.

Printing the group’s sum total in the group’s header

This quite often-used method requires use of scripts because total value in an ordinary report becomes available only after all group's records are handled. To display a sum in the group's header (before the group is handled), the following algorithm is used:

- the two-pass option of the report is turned on ("Report|Options..." menu item);
- in the first pass, the sum of each group is calculated and saved in an array;
- in the second pass, the values are extracted from the array and typed in the group's header.

Let us show, two ways of how this task may be accomplished. First of all, let us create a new report in FastReport, put the “ADOQuery” component to the report form. Set it in the following way:

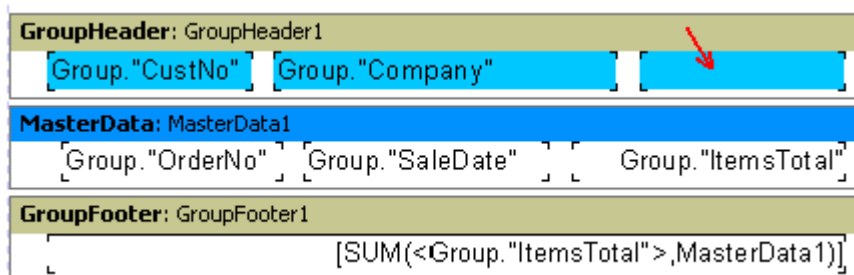
```

ADOQuery1:
DatabaseName = 'DefaultConnection'
SQL =
select * from customer, orders
where orders.CustNo = customer.CustNo
order by customer.CustNo, orders.OrderNo

UserName = 'Group'

```

Enter the designer and connect our data source to the report. Enable the double pass in report's settings (the "Report|Options..." menu item). Add two bands to the report: "Group header" and "Master data." In the "Group header" band's editor, specify the condition ("Group.CustNo" data field). Connect the data-band to the "Group" data source, and then arrange objects in the following way:



For entering sum value, we use the selected object in the picture (in our example its name is "Memo8").

The first way.

We use the "TStringList" class as an array for sums' storage. Store values in the form of lines. At the same time, the first line in the list corresponds to the value of the first group, etc. The integer-valued variable (which we will augment after printing the next group) is used for calculating the group's number.

Thus, our script will look as follows:

C++ Script:

```

TStringList List;
int i;

void frReport1OnStartReport(TfrxComponent Sender)
{
    List = TStringList.Create();
}

```

```

void frReport1OnStopReport(TfrxComponent Sender)
{
    List.Free();
}

void Page1OnBeforePrint(TfrxComponent Sender)
{
    i = 0;
}

void GroupHeader1OnBeforePrint(TfrxComponent Sender)
{
    if (Engine.FinalPass)
    {
        Memo8.Text = "Sum: " + List[i];
    }
}

void GroupFooter1OnBeforePrint(TfrxComponent Sender)
{
    List.Add(FloatToStr(<SUM(<Group."ItemsTotal">,MasterData1)>));
    i++;
}

{

}

```

Pascal Script:

```

var
    List: TStringList;
    i: Integer;

procedure frReport1OnStartReport(Sender: TfrxComponent);
begin
    List := TStringList.Create;
end;

procedure frReport1OnStopReport(Sender: TfrxComponent);
begin
    List.Free;
end;

procedure Page1OnBeforePrint(Sender: TfrxComponent);
begin
    i := 0;
end;

procedure GroupHeader1OnBeforePrint(Sender: TfrxComponent);
begin
    if Engine.FinalPass then
        Memo8.Text := 'Sum: ' + List[i];
end;

procedure GroupFooter1OnBeforePrint(Sender: TfrxComponent);

```

```

begin
    List.Add(FloatToStr(<SUM(<Group."ItemsTotal">,MasterData1)>));
    Inc(i);
end;

begin

end.

```

Looking at the names of the procedures, you can easily find out the events we have used. They are: “Report.OnStartReport,” “Report.OnStopReport,” “Page1.OnBeforePrint,” “GroupHeader1.OnBeforePrint,” and “GroupFooter1.OnBeforePrint.” As for the first two events, they are called, as it was said, in the beginning and in the end of the report respectively. To create handlers for these events, one should select the “Report” object in the “Report tree” window; its properties will appear in the objects’ inspector. After that, we would act in a standard way: switch to the inspector’s “Events” bookmark and create handlers.

Why didn't we use the main procedure for creation of the “List” list and performed it in the “OnStartReport” event? That is because the created object should be cleared after a report is finished. That is why it is rather logical to create objects in the “OnStartReport” event and clear them via the “OnStopReport.” In other cases (when memory does not need to be emptied) one can use the main procedure for initialization of variables.

Everything concerning creation and clearing of the “List” object seems to be quite obvious. Now let us examine the work of the script. In the beginning of the page, the counter of the current group (the “i” variable) is reset to “0” and increments after printing each group (in the “GroupFooter1.OnBeforePrint” event). The calculated sum's value is added to the list in this very event. The “GroupHeader1.OnBeforePrint” event does not trigger during the first pass (the “Engine.FinalPass” verification). During the second pass (when the “List” list is filled with values), the value, which corresponds to the current group is retrieved into this event, and it is recorded to the “Memo8” object's text, which displays the sum total in the group title. In a finished report, it looks in the following way:

1221	Kauai Dive Shoppe	Sum: 51450,8
1023	01.07.88	4 674,00
1076	16.12.94	17 781,00
1123	24.08.93	13 945,00
1169	06.07.94	9 471,95
1176	26.07.94	4 178,85
1269	16.12.94	1 400,00
		51 450,80

As we can see, the algorithm is rather simple. Nevertheless, it can be simplified.

The second way.

We use the list of report's variables as an array for sums' storage. As we remember, reference to such objects is performed via the “Get” and “Set” functions. That saves us from difficulty to create superfluous objects and to free the storage. Our script will look as follows:

C++ Script:

```
void GroupHeader1OnBeforePrint(TfrxComponent Sender)
{
    if (Engine.FinalPass)
    {
        Memo8.Text = "Sum:" + Get(<Group."CustNo">);
    }
}

void GroupFooter1OnBeforePrint(TfrxComponent Sender)
{
    Set(<Group."CustNo">,
        FloatToStr(<SUM(<Group."ItemsTotal">,MasterData1)>));
}

{
}
}
```

Pascal Script:

```
procedure GroupHeader1OnBeforePrint(Sender: TfrxComponent);
begin
    if Engine.FinalPass then
        Memo8.Text := 'Sum: ' + Get(<Group."CustNo">);
end;

procedure GroupFooter1OnBeforePrint(Sender: TfrxComponent);
begin
    Set(<Group."CustNo">,
        FloatToStr(<SUM(<Group."ItemsTotal">,MasterData1)>));
end;

begin
end.
```

As you can see, the script was rather simplified. A code in the “GroupFooter1.OnBeforePrint” handler sets a variable's value with a name similar to the client's number (one can use any identifier, which unambiguously identifies the client, for example, his name <Group."Company">). If there is no such variable, it would be created; if there is, its value would be changed. In the “GroupHeader1.OnBeforePrint” handler, a variable's value with the number of the current group is computed.

“OnAfterData” event

This event is generated after the report's object is filled with the data, to which it is connected. It is convenient to use this event for analyzing either a DB field value, or an expression contained in the object. The fact is that this value is placed to the “Value” service variable, the value of which is available in this event only. So, having two "Text" objects with the [Table1."Field1"] and [<Table2."Field1"> + 10] contents, it is convenient to analyze the value of these expressions referring to the “Value” variable:

C++ Script:

```
if (Value > 3000)
{
    Memo1.Color = clRed;
}
```

Pascal Script:

```
if Value > 3000 then
    Memo1.Color := clRed
```

instead of writing something like that:

C++ Script:

```
if (<Table1."Field1"> > 3000)
{
    Memo1.Color = clRed;
}
```

Pascal Script:

```
if <Table1."Field1"> > 3000 then
    Memo1.Color := clRed
```

Moreover, using of “Value” instead of an expression provides you with a possibility to write one multipurpose handler of the “OnAfterData” event, and to connect it to several objects.

One more thing is to be noted. If there are several expressions in an object (for example, [expr1] [expr2]) a value of the last expression is transferred to the “Value” variable.

Service objects

In addition to the objects included in the report (pages, bands, "Text" and other objects), some service objects are available in the script, which may be of some use when

managing report's construction. The “Engine” object, which we used in the previous chapter, refers to this kind of objects. The list of service objects is given below:

- Report - the "Report" object;
- Engine - the link to the report's slider;
- Outline - the link to the "Report tree" control element in a preview window.

Let us examine each of the objects.

“Report” object

It represents a link to the current report. The property of this object can be seen when selecting the "Report" element in the "Report tree" window.

Methods:

Method	Description
function Calc(const Expr: String): Variant	Returns the “Expr” expression's value, for example, Report.Calc('1+2') returns “3.” Any expression, which is correct in terms of FastReport's, can be transferred as an expression.
function GetDataSet(const Alias: String): TfrxDataSet	Returns a data set with a soecified name. The data set should be included into the list of the report's data ("Report Data..." dialogue).

“Engine” object

This is the most useful and interesting object, which represents a link to the slider (FastReport’s core, which manages report construction). By using the slider’s properties and methods one can construct really exotic report types. Let us examine methods and properties of this object.

Property	Type	Description
CurColumn	Integer	The number of the current column in a multi-columned report. A value can be bound to this property.
CurX	Extended	The current shift of the coordinates on the X-axis. A value can be bound to this property.
CurY	Extended	The current shift of the coordinates on the Y-axis. A value can be bound to this property.
DoublePass	Boolean	Equal to “True,” if the report is a two-pass one. Analogous to Report.EngineOptions.DoublePass.
FinalPass	Boolean	Equal to “True,” if the last pass of the two-pass report is performed.

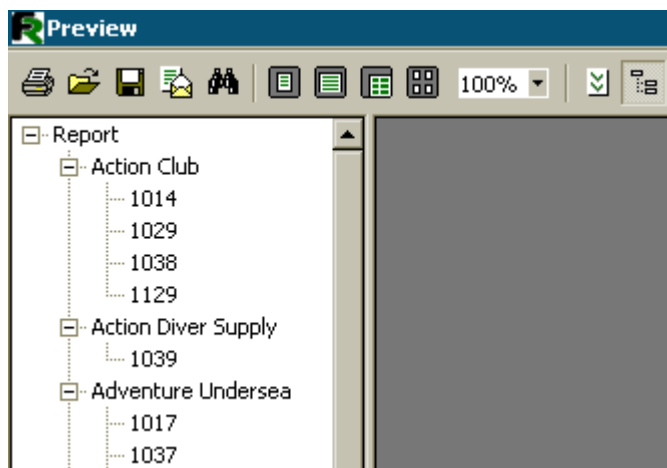
PageHeight	Extended	Printable region's height, in pixels.
PageWidth	Extended	Printable region's width, in pixels.
StartDate	TDateTime	Time of report running. A counterpart of the <Date> system variable.
StartTime	TDateTime	Time of report running. A counterpart of the <Time> system variable.
TotalPages	Integer	A number of pages in a report. A counterpart of the <TotalPages> system variable. The report should be a two-pass one, so that this variable can be used.


Methods:

Method	Description
procedure AddAnchor (const Text: String)	Adds "anchor" to the list of anchors. See more below.
procedure NewColumn	Creates a new column in a multicolumn report. After the last column, page break is automatically inserted.
procedure NewPage	Creates a new page (page break).
procedure ShowBand(Band: TfrxBand)	Displays a band with a specified name. After displaying a band, the "CurY" position is automatically shifted.
function FreeSpace: Extended	Returns height value of white space left on a page, in pixels.
function GetAnchorPage(const Text: String): Integer	Returns the number of the page, in which the specified anchor is placed.

"Outline" object

This object represents the "Report tree" control element in a preview window.



This element displays a treelike structure of a finished report. When clicking on any tree node, there is a jump to the page connected to this node. To display the tree, you should either enable it by clicking the  button in the toolbar of the preview window, or specify it with the help of the “Report.PreviewOptions.OutlineVisible=True” property. The control element's width in pixels can be specified there as well: Report.PreviewOptions.OutlineWidth.

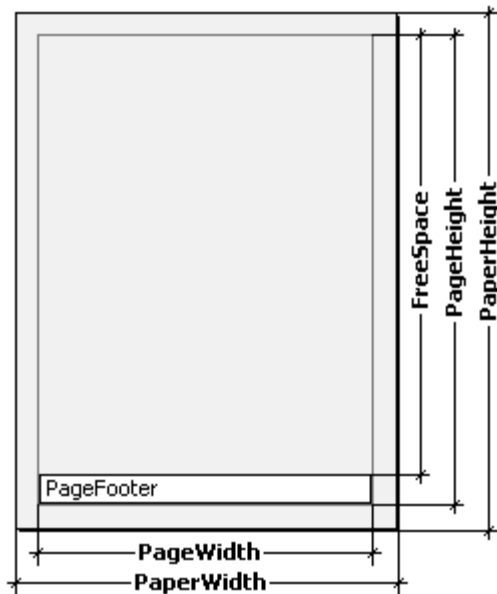
Let us examine this object's methods.

Method	Description
procedure AddItem(const Text: String)	Adds an element with the “Text” name to the current tree position. The current report’s page and the current position on the page are associated with the element.
procedure LevelRoot	Shifts the current position in the tree to the root level.
procedure LevelUp	Shifts the current position in the tree on one level up.

Using the “Engine” object

We have already mentioned, that the “Engine” object represents the report's slider, which manages report's construction. By using the slider's properties and methods, one can manage the process of arrangement bands on a page. First of all let us attend to theory.

The picture below displays the report's page and properties' names, which return different dimensions.



The page has the “PaperWidth” and “PaperHeight” physical dimensions. These dimensions correspond to page's properties of the same name that are visible in the objects' inspector when selecting a page. So, size of an A4-format page would be 210X297mm.

The “PageWidth” and “PageHeight” parameters define the dimensions of a printable region, which is almost always less than physical dimensions of a page. The size of printable region is defined by the page's fields, which depend on such report page properties as “LeftMargin,” “TopMargin,” “RightMargin,” “BottomMargin.” The printable region's size in pixels is returned by the “Engine.PageWidth” and “Engine.PageHeight” properties.

Finally, the “FreeSpace” parameter defines the height of free space on a page. If there is a "Page Footer" band on the page, its height is taken into account when calculating FreeSpace. This parameter is returned in pixels by the “Engine.FreeSpace function.” Note that after displaying the next band, free space reduces on a page, and this is what is considered during calculating FreeSpace.

How do ready report's pages form? The FastReport core produces bands on the page as long as there is enough free space. When there is no free space, the "Page Footer" band is printed (if available) and a new blank page is formed. As it was already said, after displaying the next band, the height of free space descends. Moreover, displaying of a next band begins from the current position, which is defined by coordinates on X-axis and Y-axis. This position returns in the “Engine.CurX” and “Engine.CurY” properties respectively. After printing the next band, the CurY position automatically increases by height value of the printed band. After a new page is formed, the “CurY” position is equal to “0.” The “CurX” position is modified when printing multi-channel reports.

The “Engine.CurX” and “Engine.CurY” properties are available not only for reading, but also for recording. That means that bands can be shifted manually by using one of appropriate events. For example, when you have a report looking as shown at the picture,

MasterData: MasterData1		
Customers."Company"	Customers."Contact"	Customers."Phone"

it can be printed in the following way:

Action Club	Michael Spurling	813-870-0239
Action Diver Supply	Marianne Miles	22-44-500211
Adventure Undersea	Gloria Gonzales	011-34-09054
American SCUBA Supply	Lynn Cinciripini	213-654-0092
Aquatic Drama	Gillian Owen	613-442-7654
Blue Glass Happiness	Christine Taylor	213-555-1984

This is a result of the script's work, dedicated to the “OnBeforePrint” band's event:

C++ Script:

```
void MasterData1OnBeforePrint(TfrxComponent Sender)
{
    Engine.CurX = Engine.CurX + 5;
}
```

Pascal Script:

```
procedure MasterData1OnBeforePrint(Sender: TfrxComponent);
begin
    Engine.CurX := Engine.CurX + 5;
end;
```

Manipulation with the “CurY” property allows, for example, printing bands in splice:

Action Club	Michael Spurling	813-870-0239
Action Diver Supply	Marianne Miles	22-44-500211
Adventure Undersea	Gloria Gonzales	011-34-09054
American SCUBA Supply	Lynn Cinciripini	213-654-0092
Aquatic Drama	Gillian Owen	613-442-7654
Blue Glass Happiness	Christine Taylor	213-555-1984
Blue Jack Aqua Center	Ernest Barratt	401-609-7623
Blue Sports Club	Theresa Kuncic	819-227-8304

The corresponding script:

C++ Script:


```
void MasterData1OnBeforePrint(TfrxComponent Sender)
{
    Engine.CurY = Engine.CurY - 15;
}
```

Pascal Script:

```
procedure MasterData1OnBeforePrint(Sender: TfrxComponent);
begin
    Engine.CurY := Engine.CurY - 15;
end;
```

The “Engine.NewPage” method allows page breaks in any required place of a report. At the same time, printing continues from a new page. Thus, in our example one can insert a break after printing the second record:

C++ Script:

```
void MasterData1OnAfterPrint(TfrxComponent Sender)
{
    if (<Line> == 2)
    {
        Engine.NewPage();
    }
}
```

Pascal Script:

```
procedure MasterData1OnAfterPrint(Sender: TfrxComponent);
begin
    if <Line> = 2 then
        Engine.NewPage;
end;
```

Note, that now we perform it in the “OnAfterPrint” event (that is to say, after the band is already printed). We want to draw your attention to the fact that the “Line” service variable returns the sequence number of the record.

The “Engine.NewColumn” method breaks a column in multi-columned reports. As soon as there is no column left, this method creates a new page

Anchors

Anchor is one of the elements of the hyperlink system, which allows to jump to any element, connected to the finished report’s object by clicking on it (in the preview window).

Anchor is a special tip, which is set via the “Engine.AddAnchor” method. Anchor has a name, which corresponds with the page number position on the page. To jump to an

anchor with a specified name, put the following line into the URL property of any report's object:

#AnchorName
or
#[AnchorName]

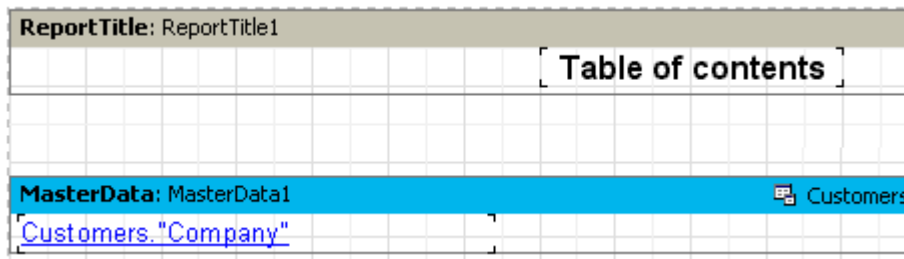
In the latter case, FastReport will expand the square brackets of the expression.

Clicking on this object executes jump to the part of the report, where the anchor was added.

It is convenient to use anchors when constructing the "Contents" chapter with links to corresponding chapters. Let us illustrate this by the following example. To perform this, we need the "Customer" table, which is already familiar to us.

Our report will be a double-page one (which presupposes two pages in designer mode). We will place the "Contents" chapter on the first page, and the list of clients on the second one. Clicking on the content line executes jumping to a corresponding report's element.

The first page:



Let us place the following line to the URL property of the "Text" object, which belongs to the data-band

#[Customers."Company"]

and set the font's properties: blue color and underlining to simulate the hyperlink's appearance.

The second page:

ReportTitle: ReportTitle2			
Customers			
PageHeader: PageHeader1			
Company	Address	Contact	Phone
MasterData: MasterData2			
Customers."Company"	Customers."Addr1"	Customers."Contact"	Customers."Ph

To add an anchor, let us type “MasterData2.OnBeforePrint” in the band’s script:

C++ Script:

```
void MasterData2OnBeforePrint(TfrxComponent Sender)
{
    Engine.AddAnchor(<Customers."Company">);
}
```

Pascal Script:

```
procedure MasterData2OnBeforePrint(Sender: TfrxComponent);
begin
    Engine.AddAnchor(<Customers."Company">);
end;
```

That is all we needed. When starting a report, let us make sure that our “hyperlinks” work.

The last thing to be mentioned is the “Engine.GetAnchorPage” function. This function returns the number of the page, where the corresponding anchor was added. This function is useful when creating the “Contents” chapter as well. A report must be a two-pass one; otherwise the function cannot be used.

Using the “Outline” object

The “Outline” object, as it was already said, represents a report’s tree, which can be displayed in a preview window. Clicking on a tree’s element executes jumping to the report’s page, which is connected to the tree’s element. It is not necessary to use the script for operating with the “Outline,” since some bands have a mechanism, which enables to automatically form a tree. Let us examine two examples of how the “Outline” can be used with the help of bands and the script.

Almost all bands have the “OutlineText” property, into which a line-expression can be put, and this in turn helps to automatically create a tree. The expression will be calculated when forming a report, and its value will be added to the tree when printing the

band. Thus, elements' hierarchy in the tree is similar to the bands' hierarchy in a report. That means, that in the tree there will be main and subordinate elements, corresponding to main and subordinate bands in a report (a report with two levels of data or with groups can exemplify the point). Let us illustrate process of operating with a tree by the example of the report with groups, which we examined in the previous chapter.

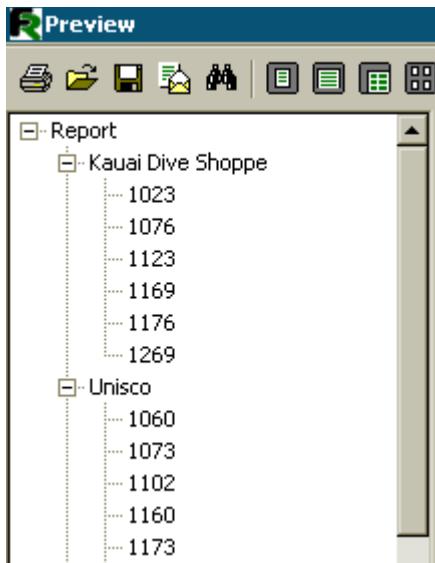
GroupHeader: GroupHeader1		Group."CustNo"
Group."CustNo"	Group."Company"	
MasterData: MasterData1		Group
Group."OrderNo"	Group."SaleDate"	

Specify a value for the "GroupHeader1.OutlineText" band's property as "<Group."Company">." To make the tree be displayed automatically as soon as the preview window opens, one should set the "Report.PreviewOptions.OutlineVisible = True" property. When starting the report, you would see the following:

Report Tree	ID	Company	Date
Kauai Dive Shoppe	1651	Jamaica SCUBA Centre	25.05.88
Unisco	1015		25.05.88
Sight Diver	1028		07.07.88
Cayman Divers World Unlimited	1128		08.10.93
Tom Sawyer Diving Centre	1215		16.11.94
Blue Jack Aqua Center	1315		26.01.95
VIP Divers Club	1680	Island Finders	
Ocean Paradise	1016		02.06.88
Fantastique Aquatica	1034		13.08.88
Marmot Divers Club	1084		11.05.89
The Depth Charge	1093		01.06.89
Blue Sports			
Makai SCUBA Club			
Action Club			
Jamaica SCUBA Centre			

Clicking on any element of the tree executes jumping to the corresponding report's page, and, as a result, the selected element occurs on the top of the window.

Let us add the second level to the report's tree. To perform this, it is enough to set the "MasterData.OutlineText" band's property as "<Group."OrderNo">." Thus, the tree will look as follows:



As you might notice, the navigation even in the orders' numbers is possible, and hierarchy of the tree's elements resembles the report's hierarchy.

Now let us show, how to form/compose an analogous tree via the script without using the "OutlineText" property. In our report, clear the "OutlineText" properties of both bands and create two event's handlers: "GroupHeader1.OnBeforePrint" and "MasterData1.OnBeforePrint":

C++ Script:

```
void GroupHeader1OnBeforePrint(TfrxComponent Sender)
{
    Outline.LevelRoot;
    Outline.AddItem(<Group."Company">);
}

void MasterData1OnBeforePrint(TfrxComponent Sender)
{
    Outline.AddItem(<Group."OrderNo">);
    Outline.LevelUp;
}

{
}
```

Pascal Script:

```
procedure GroupHeader1OnBeforePrint(Sender: TfrxComponent);
begin
    Outline.LevelRoot;
    Outline.AddItem(<Group."Company">);
end;
```

```
procedure MasterData1OnBeforePrint(Sender: TfrxComponent);  
begin  
    Outline.AddItem(<Group."OrderNo">);  
    Outline.LevelUp;  
end;  
  
begin  
  
end.
```

When starting a report, make sure, that it works in the same way as the previous report, where the tree was formed automatically. Let us examine, how a tree is formed.

The “Outline.AddItem” method adds a child block to the current tree block, and then makes the child block a current one. Thus, if “AddItem” were called several times in a row, we would receive a “ladder” as shown below:

```
Item1  
  Item2  
    Item3  
    ...
```

The “LevelUp” and “LevelRoot” Outline methods are used for controlling the current element. The first one moves the cursor to the element, which is located on a higher level. Thus, the script

```
Outline.AddItem('Item1');  
Outline.AddItem('Item2');  
Outline.AddItem('Item3');  
Outline.LevelUp;  
Outline.AddItem('Item4');
```

constructs a tree like this

```
Item1  
  Item2  
    Item3  
    Item4
```

That means, that “Item4” will be a child element in relation to the “Item2” element. The “LevelRoot” method shifts the current element to the root of the tree. For example, the script

```
Outline.AddItem('Item1');  
Outline.AddItem('Item2');  
Outline.AddItem('Item3');  
Outline.LevelRoot;  
Outline.AddItem('Item4');
```

constructs a tree, like the one below

Item1
Item2
Item3
Item4

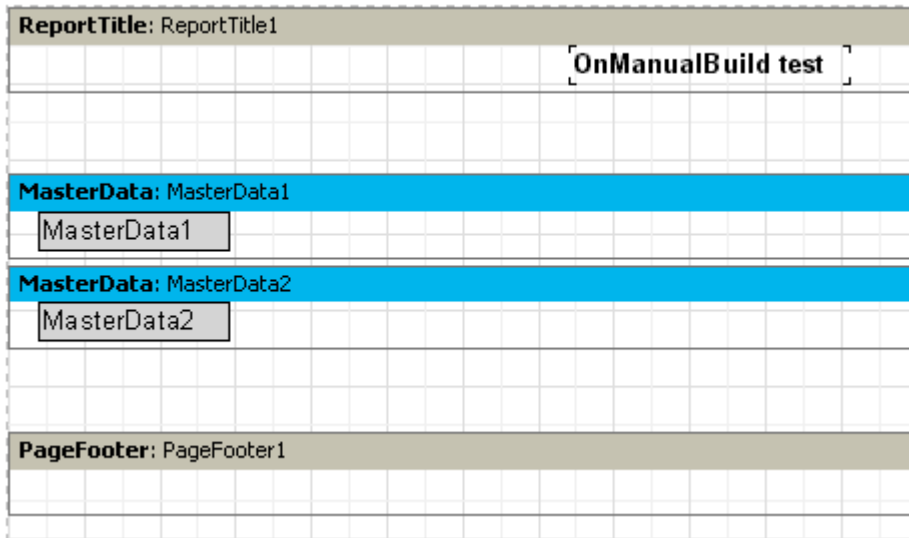
Thanks to these explanations, it becomes clear, how our report works. Every time when printing a group title, the root of the tree becomes the current element, where a company's name is added. After that, the list of orders is typed, and each order is added as a child element of the company. To make the numbers of orders located on one level, and not displayed as a "ladder", the transition to the upper level via the "Outline.LevelUp" method is performed in the script.

"OnManualBuild" page's event

The FastReport core is usually responsible for report construction. It displays report's bands in a definite order, as many times, as there are data, thus forming a finished report. Sometimes it is necessary to display a report in a non-standard form, which the FastReport core is unable to create. In this case, one can use a possibility to construct a report manually via the "OnManualBuild" event, contained in the report's page. If the handler of this event were defined, then when forming a page the FastReport core would transfer control to it. At the same time, the report's core automatically displays the bands located in the page, such as "Report title," "Page title," "Column title," "Report footer," "Page footer," "Column footer," and "Background." The core also handles the process of forming of new pages and columns. The task of the "OnManualBuild" event's handler is to display data bands and their titles and footers in a definite order.

That is to say, "OnManualBuild" handler's essence is to give a command for displaying definite bands to the FastReport's core. The core will do the rest itself: it will form a new page, as soon as there is no free space on the current one; accomplish the scripts attached to events; etc.

Let us demonstrate a handler with a simple example. In the report, there are two master data bands, which are not connected to data:



The handler will display these bands in alternate order (six times each one). After six bands are created, a small gap will be inserted.

C++ Script:

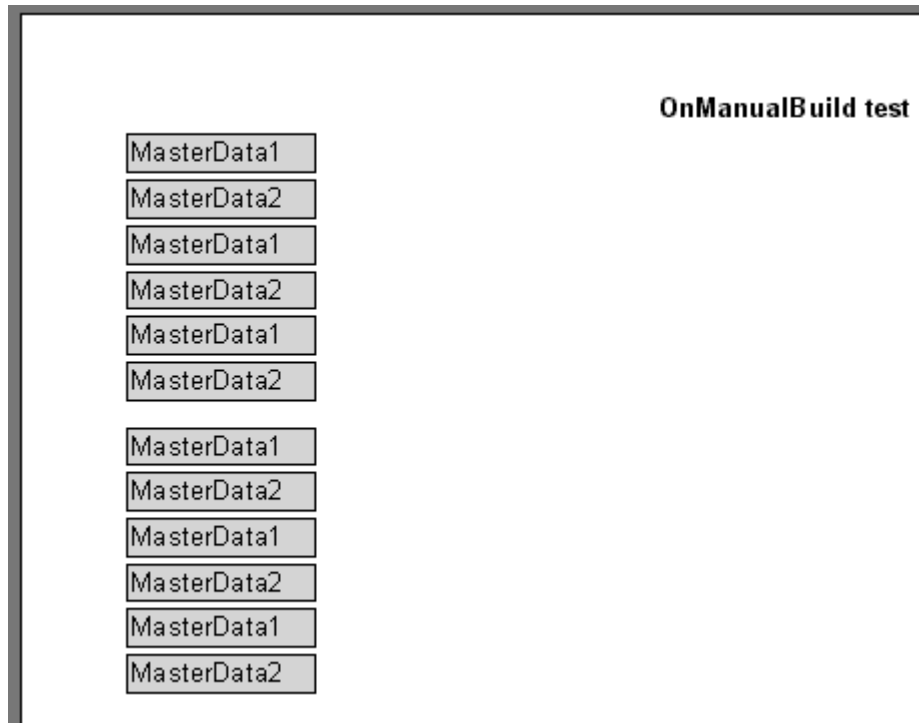
```
void Page1OnManualBuild(TfrxComponent Sender)
int i
{
    for (i = 1; i <= 6; i++)
    {
        // выводим бэнды друг за другом
        Engine.ShowBand(MasterData1);
        Engine.ShowBand(MasterData2);
        // делаем небольшой промежуток
        if (i == 3)
        {
            Engine.CurY = Engine.CurY + 10;
        }
    }
}
```

Pascal Script:

```
procedure Page1OnManualBuild(Sender: TfrxComponent);
var
    i: Integer;
begin
    for i := 1 to 6 do
    begin
        { displaying bands one by one }
        Engine.ShowBand(MasterData1);
        Engine.ShowBand(MasterData2);
        { insert a small gap }
        if i = 3 then
            Engine.CurY := Engine.CurY + 10;
    end;
end;
```



```
end;
end;
```



The following example displays two bands' groups next to each other.

C++ Script:

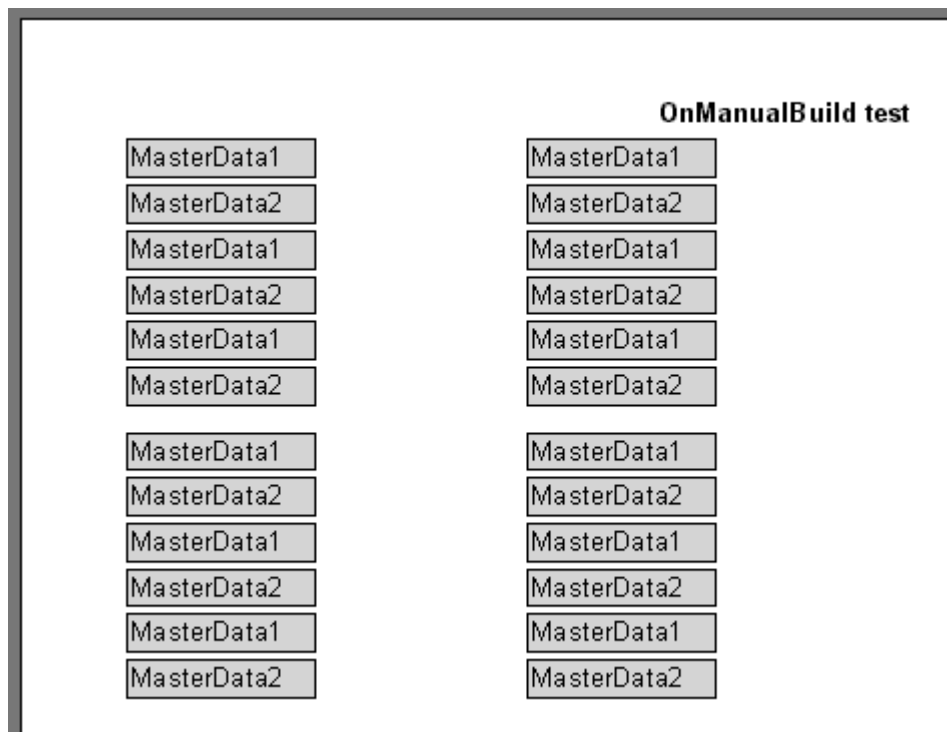
```
void Page1OnManualBuild(TfrxComponent Sender)
{
    int i, j;
    Extended SaveY;
    {
        SaveY = Engine.CurY;
        for (j = 1; j<= 2; j++)
        {
            for (i = 1; i<= 6; i++)
            {
                Engine.ShowBand(MasterData1);
                Engine.ShowBand(MasterData2);
                if (i = 3)
                {
                    Engine.CurY = Engine.CurY + 10;
                }
            }
        }
        Engine.CurY = SaveY;
        Engine.CurX = Engine.CurX + 200;
    }
}
```

Pascal Script:

```

procedure Page1OnManualBuild(Sender: TfrxComponent);
var
    i, j: Integer;
    SaveY: Extended;
begin
    SaveY := Engine.CurY;
    for j := 1 to 2 do
    begin
        for i := 1 to 6 do
        begin
            Engine.ShowBand(MasterData1);
            Engine.ShowBand(MasterData2);
            if i = 3 then
                Engine.CurY := Engine.CurY + 10;
        end;
        Engine.CurY := SaveY;
        Engine.CurX := Engine.CurX + 200;
    end;
end;

```



As you can see, in these examples we controlled typing of data-bands only. All the rest bands (for example, in our case, it was “Report title”) were printed automatically.

Finally, we will demonstrate, how to construct a report of the “List of clients” type (we have constructed it several times in this book) via the “OnManualBuild” event. In our example, let us connect the data-band to the data source.

ReportTitle: Band1		
Customers		
PageHeader: Band2		
Company	Contact	Phone
MasterData: MasterData1		
Customers."Company"	Customers."Contact"	Customers."Ph"
PageFooter: Band3		

Event's script is the following:

C++Script:

```
void Page1OnManualBuild(TfrxComponent Sender)
{
    TfrxDataSet DataSet;
    DataSet = MasterData1.DataSet;
    DataSet.First;
    while (DataSet.Eof != true)
    {
        Engine.ShowBand(MasterData1);
        DataSet.Next;
    }
}
```

Pascal Script:

```
procedure Page1OnManualBuild(Sender: TfrxComponent);
var
    DataSet: TfrxDataSet;
begin
    DataSet := MasterData1.DataSet;
    DataSet.First;
    while not DataSet.Eof do
    begin
        Engine.ShowBand(MasterData1);
        DataSet.Next;
    end;
end;
```

When starting a report, make sure that the result of the script's work does not differ from a standard report at all. Refer to the process of getting a link to the Dataset; in our example we connected a band to the data source, that is why the

```
DataSet := MasterData1.DataSet;
```

line returns a link to the data source. If a band is not connected to the source, the link to the required source can be got in the following way:

```
DataSet := Report.GetDataSet('Customers');
```

Of course, the source, we are interested in, must be added to the report in the “Report|Data...” dialogue

Creation of objects in the script

One can add new objects into a report by using the script. Let us demonstrate with a simple example, how it is performed. Create a blank report, and then write in the main script’s procedure:

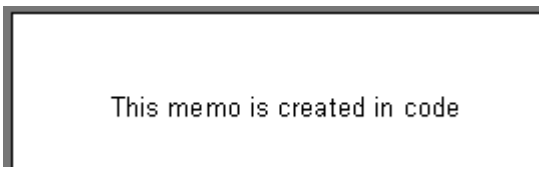
C++ Script:

```
TfrxReportTitle Band;
TfrxMemoView Memo;
{
    Band = TfrxReportTitle.Create(Pagel);
    Band.Height = 20;
    Memo = TfrxMemoView.Create(Band);
    Memo.SetBounds(10, 0, 100, 20);
    Memo.Text = "This memo is created in code";
}
```

Pascal Script:

```
var
    Band: TfrxReportTitle;
    Memo: TfrxMemoView;
begin
    Band := TfrxReportTitle.Create(Pagel);
    Band.Height := 20;
    Memo := TfrxMemoView.Create(Band);
    Memo.SetBounds(10, 0, 100, 20);
    Memo.Text := 'This memo is created in code';
end.
```


Start a report:

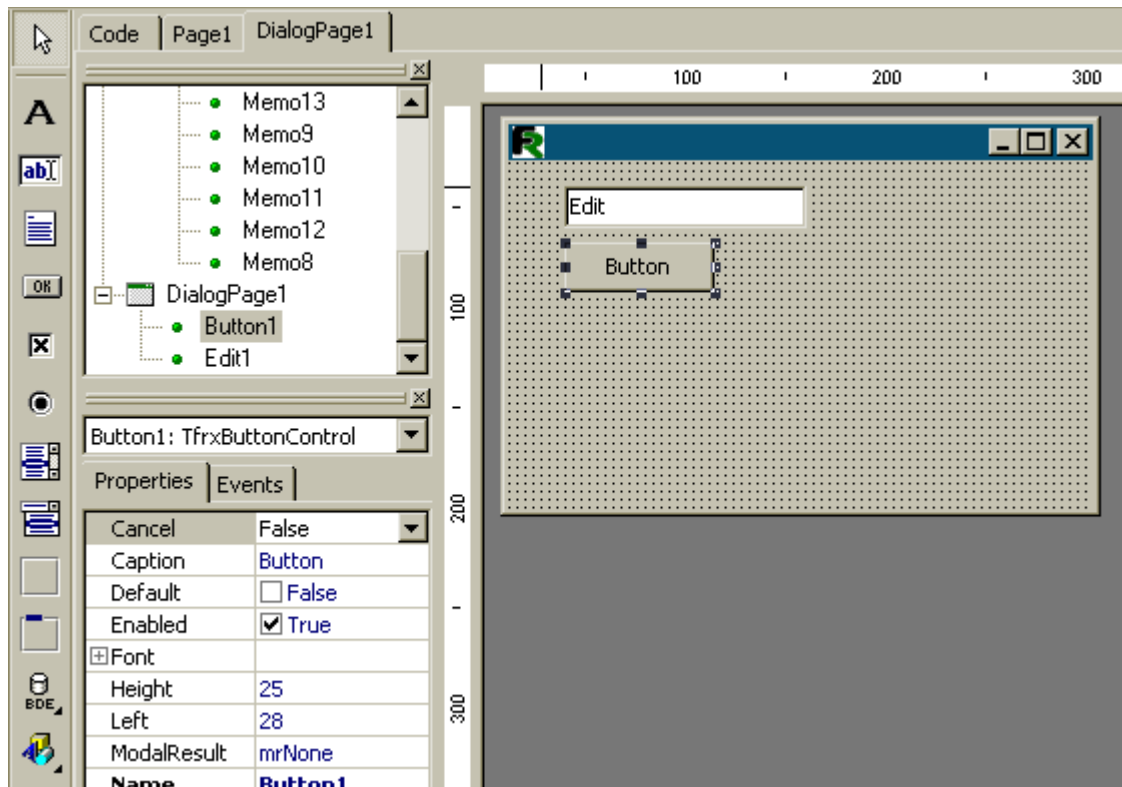


This memo is created in code

Note, that we never destroy the created objects in these examples. It is not required, since objects are automatically destroyed after the report is completed.

Dialogue forms



In addition to usual report pages, you can use several dialogue forms in a report. For dialogue form creation, the same designer as for report pages is used. The  button in the designer toolbar is used for creating a new form; it adds a new page into a report. When switching to the page with the dialogue form, the designer workspace changes, thus becoming a form where objects (i.e. controls) can be placed:


















It resembles the MS Visual Studio environment, doesn't it?

Controls

Following controls are available for use:

Element	Name	Description
	TfrxLabelControl	This control is used for displaying explicative inscription on the dialogue form.
	TfrxEditControl	This control is used for entering a text line with the help of the keyboard.

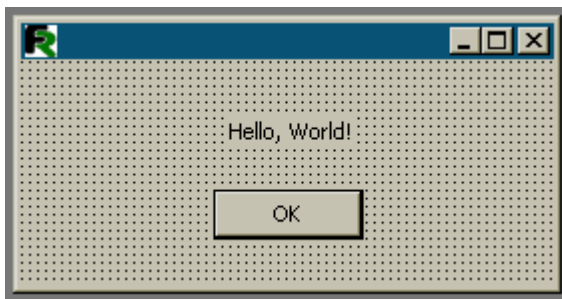
	TfrxMemoControl	This control is used for entering several text lines with the help of the keyboard.
	TfrxButtonControl	The control represents a button.
	TfrxCheckBoxControl	The control represents a flag, which can perform two statuses: enabled and disabled. Near the flag, the explicative inscription is displayed.
	TfrxRadioButtonControl	The control represents a switch key counterpart with radio button. This is the reason why it cannot be used alone.
	TfrxListBoxControl	The control represents the list of lines with a possibility to select one of them.
	TfrxComboBoxControl	The control represents the drop-out list of lines with a possibility to select one of them.
	TfrxDateEditControl	The control represents a field with a drop-out calendar for date entering.
	TfrxGroupBoxControl	The control represents a bar with explicative inscription which is used for uniting several controls.
	TfrxPanelControl	The control represents a bar, which is designed for uniting several controls.
	TfrxBitBtnControl	The control represents a button with picture.
	TfrxSpeedButtonControl	The control represents a button with picture.
	TfrxMaskEditControl	The control represents a text box for entering information set in a template.
	TfrxCheckListBoxControl	The control represents a list of lines with flags.
	TfrxBevelControl	The control is used for the dialogue form design.

	TfrxImageControl	The control represents a picture in “BMP,” “ICO,” “WMF,” or “EMF” format.
---	------------------	---

As you can see, all elements are similar to those used in any IDE. In the FastReport component help, you can obtain the help about realized properties, events and methods of each element.

“Hello, World!” report

This time, we will create a report displaying a greeting window before constructing with the help of a dialogue form. Create a new report in FastReport and add a dialogue form into the report. Put the “TfrxLabelControl” and “TfrxButtonControl” objects on the form:



Set objects' properties:

TfrxLabelControl:
Caption = 'Hello, World!'

TfrxButtonControl:
Caption = 'OK'
Default = True
ModalResult = mrOk

Set the “BorderStyle = bsDialog” property in the form. As we can see, both the controls and the form have the same set of properties as those in the corresponding Delphi controls.

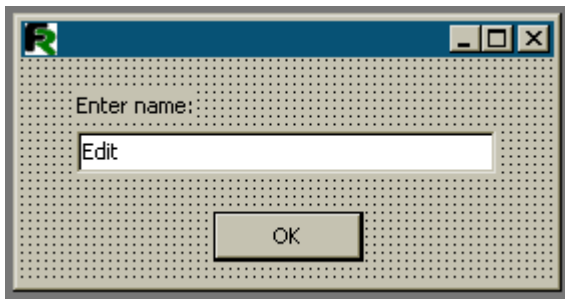
As soon as setting of the dialogue form is finished, let us return to the report page and locate the “Text” object with any text in it there. Run the report and you will see the form:



When clicking on the “OK” button, a report will be constructed and displayed. If closing a form via the “X” button, the report will not be constructed. This is the mechanism of FastReport working: if there are dialogue forms in a report, it is constructed only when each form is closed with the “OK” button, i.e. it returns `ModalResult = mrOk`. That is why the “ModalResult” property of the button is set equal to “mrOk.”

Entering parameters and transferring them into a report

Let us make this example more complicated in order to show how to transfer the values entered in the dialogue form into a report. To perform this, modify the form in the following way:



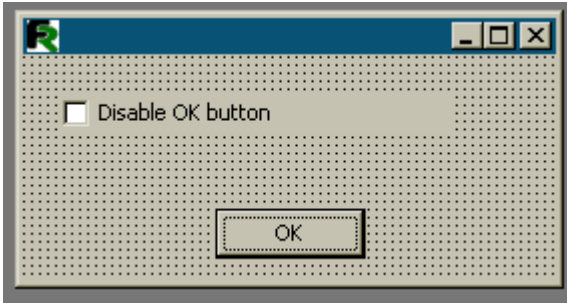
Let us place the “Text” object containing the following text in a page:

You entered:
[Edit1.Text]

Run the report and make sure that the parameter you entered is successfully displayed in the report. In the same way, you can address other objects of the dialogue form. Since each object has a name, which is unique within the whole report, it can be used at any place of the report.

Interaction of controls

Via using script, you can easily incarnate the logic of the dialogue form's work, for example, its controls' interaction. Let us illustrate this by a simple example. Modify the form in the following way:



Double click on the “CheckBox” object, so that the “OnClick” event handle would be created, and then write the following script:

C++ Script:

```
void CheckBox1OnClick(TfrxComponent Sender)
{
    Button1.Enabled = ! CheckBox1.Checked;
}
```

Pascal Script:

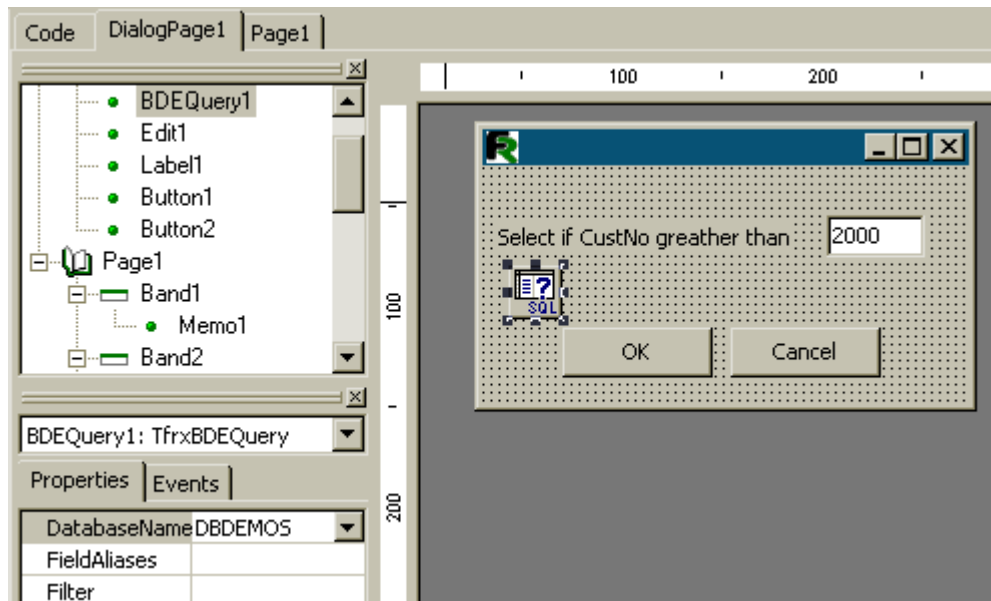
```
procedure CheckBox1OnClick(Sender: TfrxComponent);
begin
    Button1.Enabled := not CheckBox1.Checked;
end;
```

As you can see, the code does not differ much from the one, which we used to see in Delphi. When running the report, you would see that the button responds on the flag condition's modification.

Data access components

Most of reports, as a rule, are based on data from DB. For accessing such data, FastReport Studio offers effective mechanisms, which are used in reports. The matter concerns the “ADOTable” and “ADOQuery” components, which can act as data sources for the report.

In addition to possibility to access data defined in the project, FastReport allows to create new components in run-time.



Components' description

Let us examine usage of components for data access via ADO. They are connected when using in the project the “TfrxADOComponents” component from the FastReport palette. At the same time, the following objects appear in the designer object bar: “TfrxADOTable,” “TfrxADOQuery,” and “TfrxADODataBase.”

Icon	Name	Description
	TfrxBDETable	The control is used for access to DB table.
	TfrxBDEQuery	The control is used for performing SQL-query.
	TfrxBDEDataBase	The control is used for connecting to DB.

Let us examine each component.

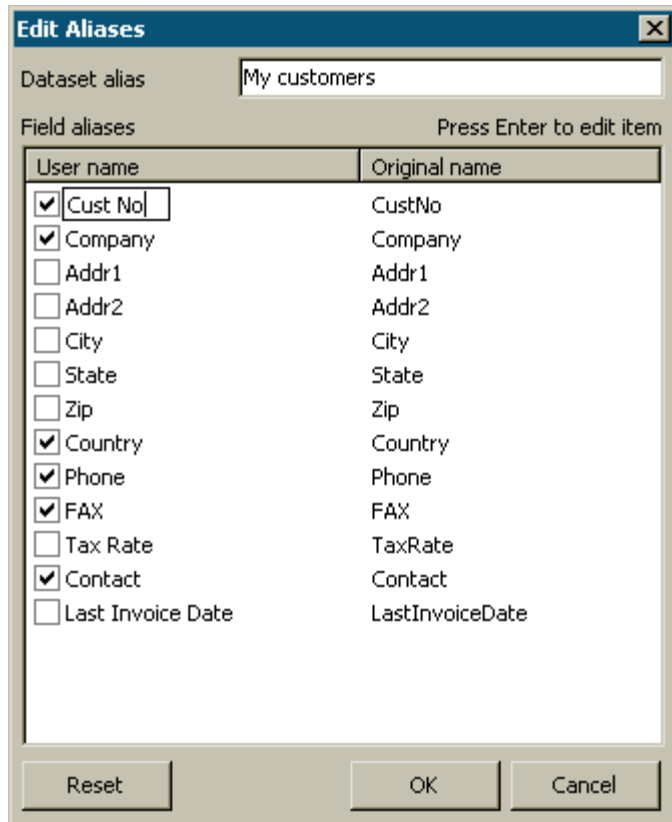
TfrxADOTable

The component is used for organization of DB table access. The component has the following properties:

Property	Description
Active	Defines whether a table is active.
DatabaseName	DB name.
FieldAliases	Enables to set fields aliases.
Filter	Expression for records' filtering.
Filtered	Defines whether it is necessary to use filter.
IndexName	Secondary index name.
MasterFields	Fields connected with master dataset.
Master	Master dataset.
SessionName	BDE session name.
TableName	DB table name.

To connect a component to the DB table, it is enough to fill the “DatabaseName” and “TableName” properties. Table opening is performed either via the “Active: = True” setting, or with the help of the “Open” method.

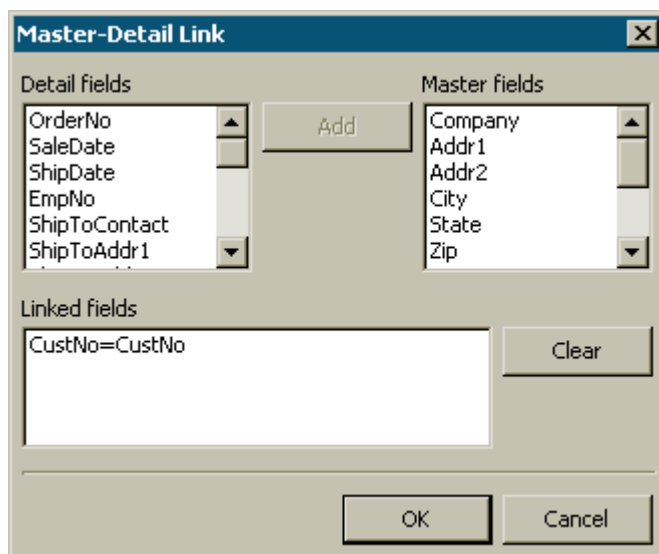
The “FieldAliases” property editor allows to select fields, which will be available upon addressing the table, and to set aliases for the whole table and for each field.



The **Edit Aliases** dialog box is used to manage field aliases for a dataset. It features a text box for the **Dataset alias** (set to "My customers") and a section for **Field aliases** with the instruction "Press Enter to edit item". The field aliases are listed in a table with columns for **User name** and **Original name**. Each user name has a checkbox next to it. At the bottom are **Reset**, **OK**, and **Cancel** buttons.

User name	Original name
<input checked="" type="checkbox"/> Cust No	CustNo
<input checked="" type="checkbox"/> Company	Company
<input type="checkbox"/> Addr1	Addr1
<input type="checkbox"/> Addr2	Addr2
<input type="checkbox"/> City	City
<input type="checkbox"/> State	State
<input type="checkbox"/> Zip	Zip
<input checked="" type="checkbox"/> Country	Country
<input checked="" type="checkbox"/> Phone	Phone
<input checked="" type="checkbox"/> FAX	FAX
<input type="checkbox"/> Tax Rate	TaxRate
<input checked="" type="checkbox"/> Contact	Contact
<input type="checkbox"/> Last Invoice Date	LastInvoiceDate

The “MasterFields” property editor is used for creation of master-detail connections between two tables. To connect two tables with the master-detail relation, a user should specify a general table in the “Master” property and call the “MasterFields” property editor for the subordinate table. If the table has secondary indexes, which are necessary to be used, set the “IndexName” property beforehand.



The **Master-Detail Link** dialog box is used to establish a master-detail relationship between two tables. It contains two list boxes: **Detail fields** (containing OrderNo, SaleDate, ShipDate, EmpNo, ShipToContact, ShipToAddr1) and **Master fields** (containing Company, Addr1, Addr2, City, State, Zip). An **Add** button is located between them. Below these is a **Linked fields** text box containing "CustNo=CustNo" and a **Clear** button. At the bottom are **OK** and **Cancel** buttons.

Here you can visually bind the “master” and the “detail” fields of data sets. When sets’ connection is of “Master-Detail” type, then when moving within the master set, the contents of the detail set is filtered in a way that it contains only records concerned the current record of the master set.

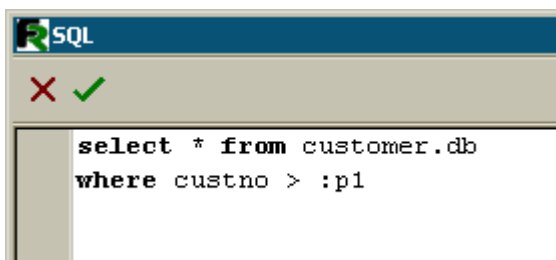
To connect sets’ fields, select a field from the list on the left (detail set), then a field from the list on the right (master set), and click on the “Add” button. Thus, the fields’ bond would be transferred to the bottom list. To clear the bottom list, use the “Clear” button. The bound fields must be of an equal type and be the key ones.

TfrxADOQuery

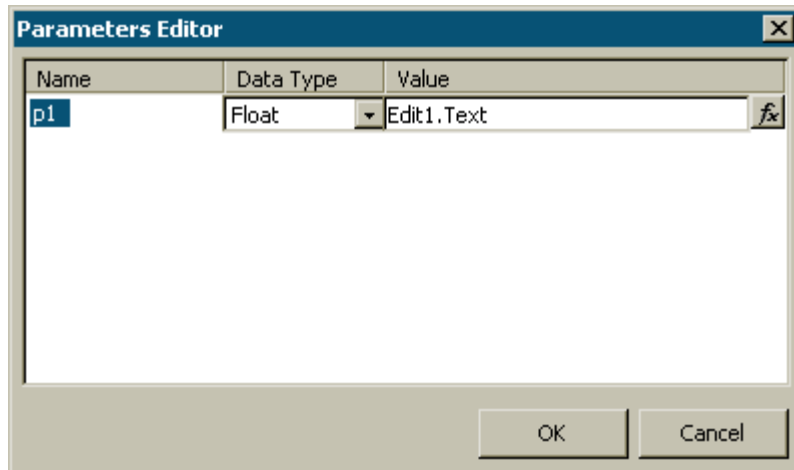
The component is used for performing SQL-queries to DB. The component has the following properties:

Property	Description
Active	Defines whether a query is active.
DatabaseName	DB alias name.
FieldAliases	Allows to set user’s field aliases.
Filter	Expression for records’ filtering.
Filtered	Defines whether it is necessary to use the filter.
Master	Data master-set.
Params	The list of query parameters.
SQL	Query text.

The “Active,” “DatabaseName,” “FieldAliases,” “Filter,” “Filtered,” and “Master” properties are similar to the properties of the “TfrxADOTable” component described above. The “SQL” property has its own editor for filling the SQL-query.



The “Params” property also has its editor. It becomes available as soon as a query text contains parameters.



A parameter can be of two types: either one assigned from the master-source or one having a concrete value (either an absolute symbol or a link to the variable or object's property, as it is shown in the picture, can act as a value).

In case when a parameter is taken from the data master-set, it is necessary to adjust the "TfrxADOQuery.Master" property. The data set must contain a field with the name coinciding with the name of the parameter. At the same time, it is necessary to specify neither a parameter type, nor its value.

TfrxADODataBase

This component is used for performing connection to database. The component has the following properties:

Property	Description
AliasName	Alias name. The connection to DB will be performed on the base of its properties.
Connected	If "True," it activates the connection.
DatabaseName	A name, which will be added to the list of aliases.
DriverName	Driver name, which provides connection to DB.
LoginPrompt	Defines whether it is necessary to request a password upon connection to DB.
Params	Connection parameters.

The component performs connection to the database (as a rule, it is used for connection to back-end). Settings for connection are taken either from the corresponding alias (the "AliasName" property) or entered manually (to perform this, it is necessary to specify the driver's name: "DriverName"). The component must have the "DatabaseName" property filled, since this value would be in the list of aliases.

To set connection parameters, it is necessary to call the “Params” property’s editor.

The LoginPrompt property defines whether it is necessary to request a password when connecting to DB. If “LoginPrompt” = “False,” a user name and a password must be specified in connection parameters, for example:


```
SERVER NAME=Path_to_file_*.gdb  
USER NAME=SYSDBA  
PASSWORD=masterkey
```

Report constructing

Let us examine construction of a simple report containing data access components. We will use demo tables as data for the example.

Simple report of the “List” type

This report will contain data from one DB table. To construct a report, perform the following steps.

Click on the “New report” button  in the designer toolbar. A report page with the “Report title,” the “First level data” and the “Page footer” bands will be created.


Add a dialogue form into the report. This form will be used for placing the DB table access component.

Put the “TfrxADOTable” component on the form, and then set its properties:
DatabaseName = 'DefaultConnection'
TableName = 'Customer'

Go to the page with a report form. To connect the “First level data” band to the table, double-click on it, and then select the required table in the opened window.

Drag the required fields from the “Data tree” window to the report page. After that, the report will look roughly like this:

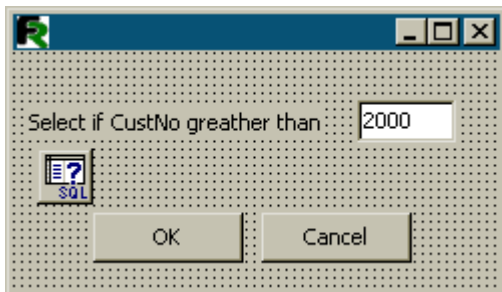
ReportTitle: ReportTitle1		
MasterData: MasterData1		
[BDETable1."Company"]	[BDETable1."Contact"]	[BDETable1."Phone"]
PageFooter: PageFooter1		

To preview the report, click on the “Preview” button  in the toolbar.

Report with parameters’ query

Let us examine construction of a more complicated report, in which parameters would be requested in the dialogue window before the report begins to be constructed. To do this, perform the following actions.

Add a dialogue form into the report. Put the “Query,” “Label,” “Edit,” and “Button” components on the report form:



Set components’ properties:

ADOQuery1:

DatabaseName = 'DefaultConnection '

SQL = 'Select * from Customer where CustNo > :p1'

Label1:

Caption = 'Select if CustNo greather than'

Edit1:

Text = '2000'

Button1:

Caption = 'OK'

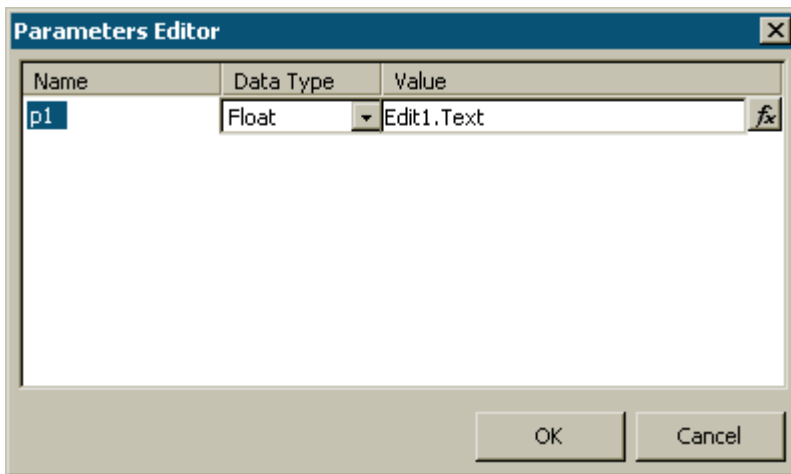
ModalResult = mrOk

Button2:

Caption = 'Cancel'

ModalResult = mrCancel

Open the “Params” property’s editor of the “Query” component, and then set the parameter:



After that, go to page with the report form and create a report in the way similar to that demonstrated in the previous example:

MasterData: Band4	
BDEQuery1."Cust No"	BDEQuery1."Company"

Upon constructing a report, the dialogue, in which a user will be offered to enter a customer number, will be displayed. After entering a requested value and clicking on the “OK” button, the report constructing is finished. The customers with numbers larger than the entered one will be outtyped.

