

FastReport VCL User Manual

Version 2026.2

© 1998-2026 Fast Reports Inc.

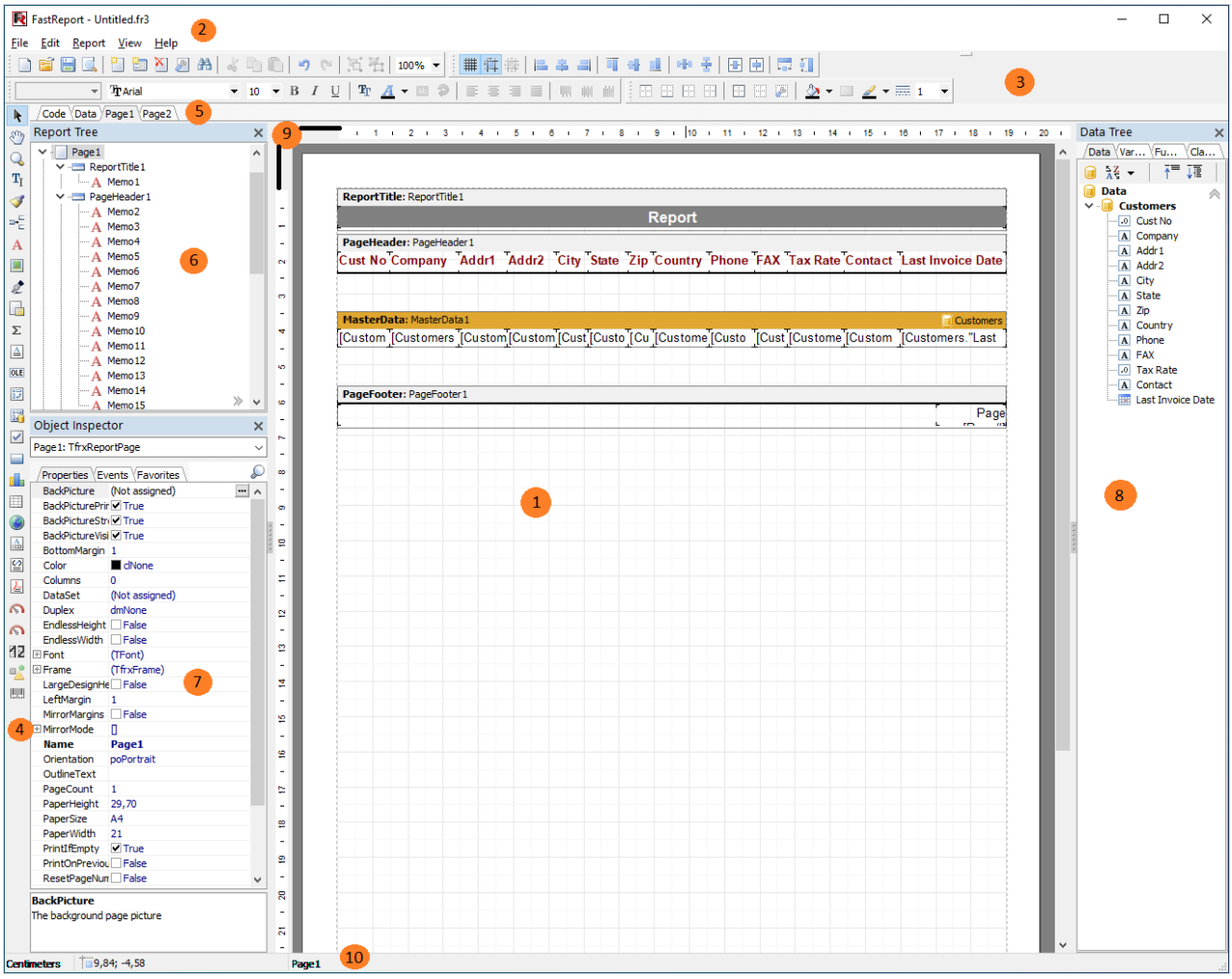
Designer

The report component is supplied with an embedded visual report designer, which can be opened at design-time by double-clicking on the `TfrxReport` component. The designer provides the user with all the tools necessary for designing and previewing reports.

The designer's interface meets current requirements. It contains several toolbars, which can be docked wherever wanted. These toolbar locations are restored every time the designer is opened. Toolbar locations, together with other designer settings, are stored in the registry or, should one be assigned, in an ini-file.

To give the end user of your project the ability to design reports, you should either place a `TfrxDesigner` component from the FastReport component palette onto a Delphi form or add the `frxDesgn` unit to the unit's "Uses" list. Using the designer at run-time allows the user to change the report's appearance, as well as to edit the finished report.

Note: you should also place any other Tfrx components that will be used on the Delphi form.



Key to report designer features:

1 – report design workspace

2 – menu bar

3 – toolbars

4 – object toolbar

5 – report page tabs

6 – "Report tree" pane

7 – "Object inspector" pane

8 – "Data tree" pane : elements can be dragged from this pane onto a report page

9 – rulers : a ruler can be dragged onto a report page to create a blue guideline on the page (objects snap to nearby guidelines)

10 – status bar

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, Tab	move between objects
Del	delete selected object(s)
Enter	open editor for selected object
Shift+arrows	modify sizes of selected object(s)
Ctrl+arrows	move selected object(s)
Alt+arrows	attach selected object to adjacent object in specified direction

Mouse control

Operation	Description
Left button	select object; paste new object; move or resize objects for selected objects, zoom in and out by dragging red square in bottom left corner of selected objects' group
Right button	selected object's context menu
Double-click	open editor for object; double-clicking on white space opens the "Page Settings" dialogue
Mouse wheel	scroll report page
Shift + left button	toggle object selection
Ctrl + left button	create frame by moving mouse; release button to select all objects captured in the frame; can also click on blank space, and move mouse as required
Alt + left button	edit contents in place, if a "Text" object is selected

Toolbars

Designer mode bar

The Mode toolbar is integrated with the Object toolbar at the left hand edge of the designer window and has these buttons:

Icon	Name	Description
	Object selection	standard mode of operation : mouse cursor selects object(s), modifies their sizes, etc.
	Hand	allows dragging of whole report page
	Zoom	allows left-clicking to zoom in (adds 25% if already 25-75%, else adds 100%) or right-clicking to zoom out (subtracts 25% if already 50-100%, else subtracts 100%); holding left mouse button while dragging zooms in to selected area
	Text editor	allows in-place editing of "Text" object; holding left mouse button and moving the cursor creates and sizes a new "Text" object and opens its editor
	Format copying	allows format copying from one "Text" object to others: select a "Text" object, click Format mode button, select target "Text" objects in turn; exit format copying mode by clicking any other mode button

Standard toolbar



Icon	Name	Description
	New Report	creates new blank report
	Open Report	opens existing report from file, hotkey combination – "Ctrl+O"
	Save Report	saves report to file, hotkey combination – "Ctrl+S"
	Preview	previews report, hotkey combination – "Ctrl+P"
	New Report Page	adds new page to report
	New Dialog Page	adds new dialogue form to report
	Delete Page	deletes current page
	Page Settings	opens page properties dialogue
	Cut	cuts selected object(s) onto clipboard, hotkey combination – "Ctrl+X"

Icon	Name	Description
	Copy	copies selected object(s) onto clipboard, hotkey combination – "Ctrl+C"
	Paste	pastes object(s) from clipboard, hotkey combination – "Ctrl+V"
	Undo	undo last operation, hotkey combination – "Ctrl+Z"
	Redo	redo last cancelled (undone) operation, hotkey combination – "Ctrl+Y"
	Group	groups selected objects
	Ungroup	ungroups selected objects
75%	Zoom	sets zoom factor

Text toolbar



Icon	Name	Description
No style	Style	shows style of selected "Text" object; select from drop-down list to change style of selected object; define styles in "Report | Styles" menu
Arial	Font Name	shows font of selected "Text" object; select from drop-down list to change font of selected object; shows last five fonts used at head of list
10	Font Size	shows font size of selected "Text" object; select from drop-down list to change font size of selected object; can also type digits directly into edit box
	Bold (toggle)	toggles font Bold style of "Text" object's content on/off
	Italic (toggle)	toggles font Italic style of "Text" object's content on/off
	Underline (toggle)	toggles font Underline style of "Text" object's content on/off
	Font Settings	opens Font settings dialogue
	Font Color	drops down font color selector
	Highlight	opens highlight dialogue; can set a condition for highlighting
	Text Rotation	changes text rotation from drop-down list

Icon	Name	Description
	Align Left	left aligns text within "Text" object's frame
	Align Center	centre aligns text within "Text" object's frame
	Align Right	right aligns text within "Text" object's frame
	Justify	justifies text within "Text" object's frame
	Align Top	top aligns (vertical) text within "Text" object's frame
	Align Middle	middle aligns (vertical) text within "Text" object's frame
	Align Bottom	bottom aligns (vertical) text within "Text" object's frame

Frame toolbar



Icon	Name	Description
	Top Line (toggle)	toggles top frame line on/off
	Bottom Line (toggle)	toggles bottom frame line on/off
	Left Line (toggle)	toggles left frame line on/off
	Right Line (toggle)	toggles right frame line on/off
	All Frame Lines	sets all four frame lines on
	No Frame	sets all four frame lines off
	Frame editor	invokes the frame editor dialogue
	Background Color	sets background color from the drop-down list.
	Fill editor	invokes the fill editor dialogue
	Frame Color	drops down frame line color selector
	Frame Style	drops down frame line style selector
1	Frame Width	sets frame line width from drop-down list; can also type digits (0.1 to 10) directly into edit box

Align toolbar

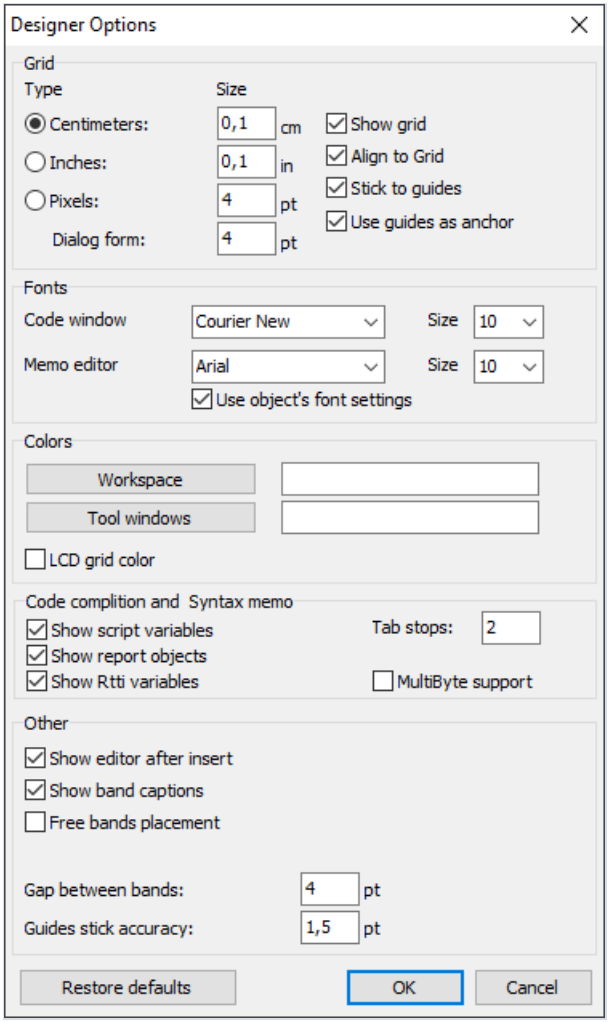


Note that some buttons will be active when more than one object is selected.

Icon	Description
	show Grid (toggles on/off)
	align to Grid (toggles on/off)
	fit to Grid
	align left edges (to first selected object)
	align horizontal centres (to first selected object)
	align right edges (to first selected object)
	align top edges (to first selected object)
	align vertical centres (to first selected object)
	align bottom edges (to first selected object)
	space equally in horizontal direction
	space equally in vertical direction
	individually centre each object horizontally in band
	individually centre each object vertically in band
	set equal widths (same as first selected object)
	set equal heights (same as first selected object)

Designer options

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



The "Grid" group: here you can set the preferred units (centimetres, inches, pixels) and grid spacing.

> You can also cycle through the units from within the designer by double-clicking on the left part of the status bar where the current units are displayed.

You can set grid visibility and alignment to grid. This can also be done via buttons in the “Standard” toolbar from within the designer.

The "Fonts" group: 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 matches the font of the object being edited.

The "Colors" group: the default white background of the designer workspace and tool windows can be modified via the “Workspace” and the “Tool windows” buttons.

The “LCD grid color” option increases contrast of the grid lines a little and improves their visibility on LCD monitors.

The “Show editor after insert” option controls what happens when new objects are inserted. 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 temporarily disable the option.

Disabling the “Show band captions” option hides band captions, so saving some space on a design page. When disabled, the band captions are shown inside the band workspace.

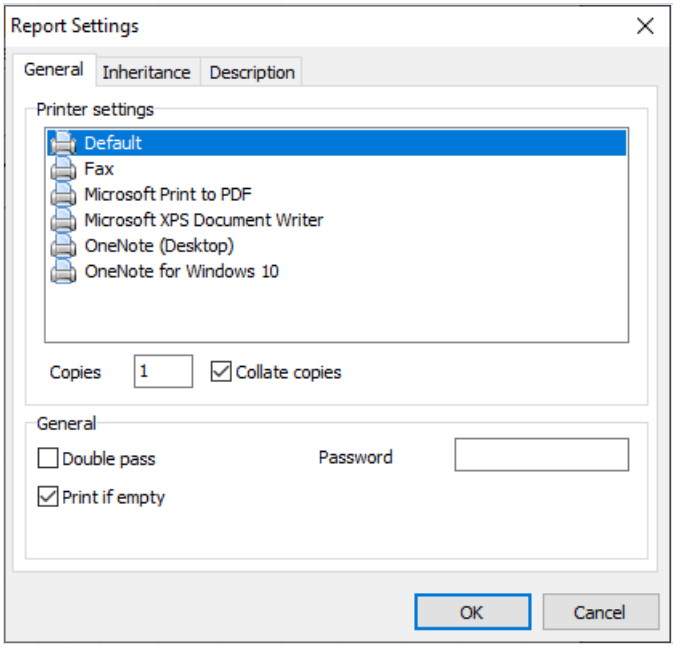
The “Show drop-down list of fields” option controls whether the drop-down list is accessible when pointing with the mouse to “Text” objects which are connected to data fields. This may be necessary if there are many narrow “Text” objects in a band

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

Report settings

The Report Settings dialogue is available from the "Report>Options..." menu. The dialogue has three pages.

On the first page you can see the general settings for the report:



You can tie a report to one of the printers installed in the system. This means that the selected printer will become the default when printing that report. This might be useful in cases where there are several different printers in the system; e.g. text documents can be tied to monochrome printers, while documents with graphics tied to color capable ones. "Default" is listed in Printer settings - when this is selected, the report will not be tied to any particular printer but will be printed on the system's default printer.

You can also set the number of report copies to be printed and whether to collate the output. The value set in this dialogue will be shown in the "Print" dialogue when printing the report.

If the "Double pass" flag is checked report generation will be performed in two steps. During the first pass a draft report is created and divided into pages, but is not available for preview. In the second pass the draft report is converted to a standard report which is then saved in the preview stream.

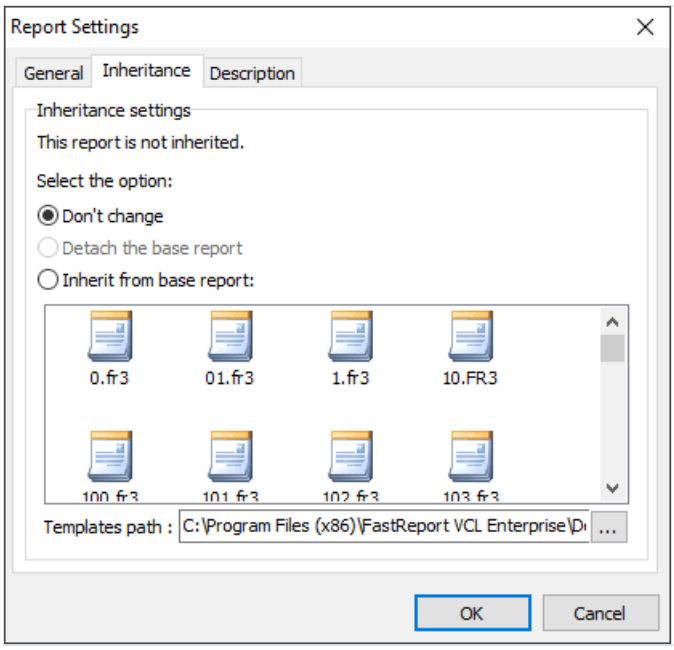
Why are two passes necessary? Mostly this option is used in cases where a report needs access to the total number of pages in the report, i.e. for use as "Page 1 of 15". The total number of pages is calculated during the first pass and is made available via the `TOTALPAGES` system variable. The most frequent mistake is to attempt to use this variable in a single-pass report, when the variable will return 0.

Another use for two passes is to perform some calculations in the first pass and display the results in the second pass. For example, when a sum is to be displayed in a group header instead of in the usual group footer. Calculations of this type are accomplished by writing report script code in the "OnBeforePrint" event of an object.

The "Print if empty" flag allows creation of a report containing no data lines. If this option is disabled blank reports will not be created.

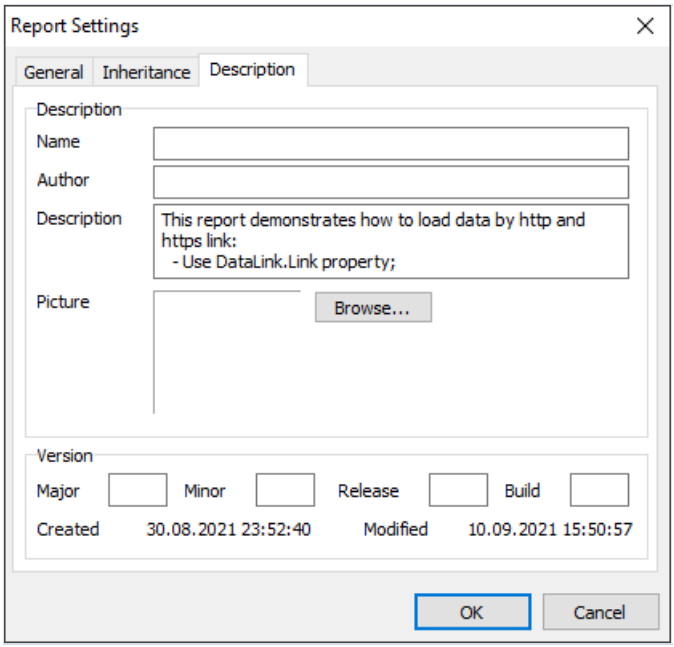
The "Password" field enables the setting of a password, which must be entered before a report can be opened.

On the second page you can set up the report inheritance options:



You will learn about inheritance later in the “Report inheritance” chapter. In this dialogue you can see the base report's name (if the report is inherited), detach the base report (in which case your report will be standalone and non-inherited) or inherit the report from one of the available base reports.

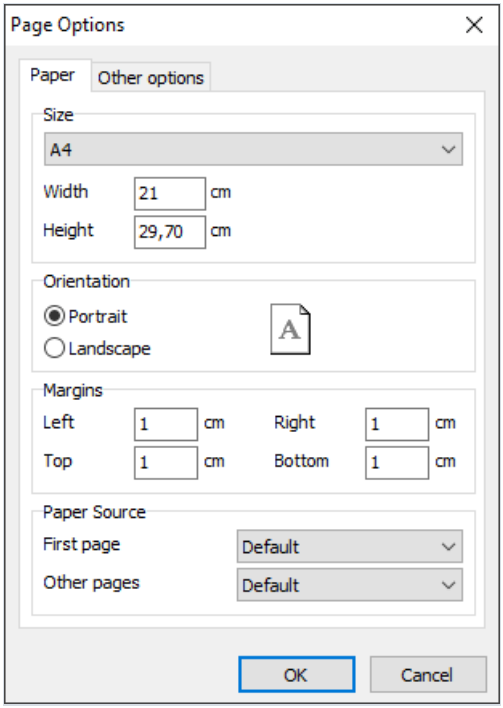
The third page of the dialogue allows you to set descriptive properties for the report:



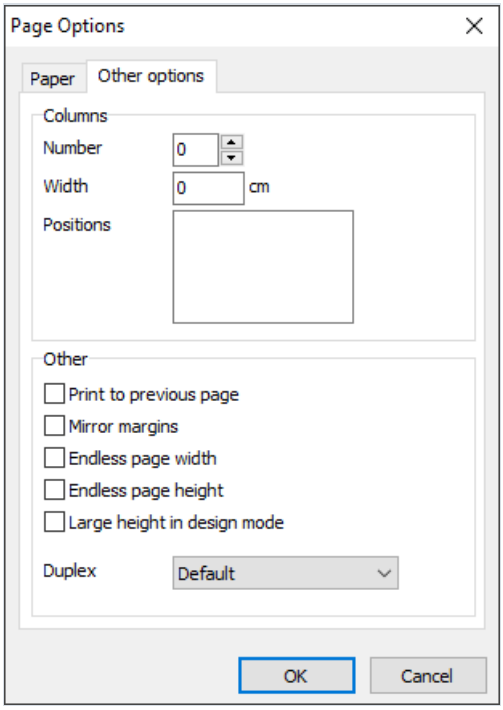
All fields on this page are for information purposes only and are not printable.

Page options

The “Page Options” are available via either the “File>Page settings...” menu or by double-clicking on blank space on the page. The dialogue has two pages:



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



On the second dialogue page, you can set the number of columns for multi-column reports. The current settings are also displayed in the designer.

The “Print to previous page” flag allows you to print pages without starting a new page, instead beginning in any blank space on the previous page. This option can be useful when a report template consists of several pages or when printing batch (composite) reports.

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

The “Endless page width & height” options increase page sizes depending on the number of data records on the page (when the report is run). When set the report will appear on one big page in the preview window instead of on several standard size pages.

The “Large height in design mode” option increases the page height in the designer. This feature can be useful if there are many bands on a page: it must be used when working with overlayed bands. It only effects the page height in design mode.

Creating reports

Report objects

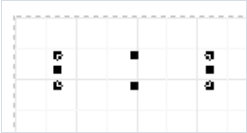
The FastReport Designer represents reports as a collection of schematic pages. Objects are placed anywhere on the report pages and are used to define the report's appearance and to display various information, such as text and graphics. FastReport objects which are included in the standard package are:

Icon	Name	Description
	“Band” object	an area on a design page which behaves according to its type (e.g. Header band, Data band)
	“Text” object	displays one or more lines of text within a rectangular area
	“Picture” object	displays a graphic file in `BMP`, `JPEG`, `ICO`, `WMF` or `EMF` format
	“Line” object	displays a horizontal or vertical line
	“System text” object	displays either system information (date, time, page number, etc) or aggregate values
	“Subreport” object	allows insertion of another report design page within the host page
	“Draw” category objects	displays various geometrical shapes (diagonal line, rectangle, rounded rectangle, ellipse, triangle and diamond)
	“Chart” object	displays data in various chart formats (pie chart, histogram, etc.)
	“RichText” object	displays text in Rich Text Format (`RTF`)
	“CheckBox” object	displays a checkbox with either a tick or a cross
	“Barcode” object	displays data as one of several barcode types
	“OLE” object	displays any object using `OLE` technology.

The basic objects most commonly used are the “Band” and “Text” objects. You will learn about their capabilities in detail later in this chapter.

“Hello, World!” report example

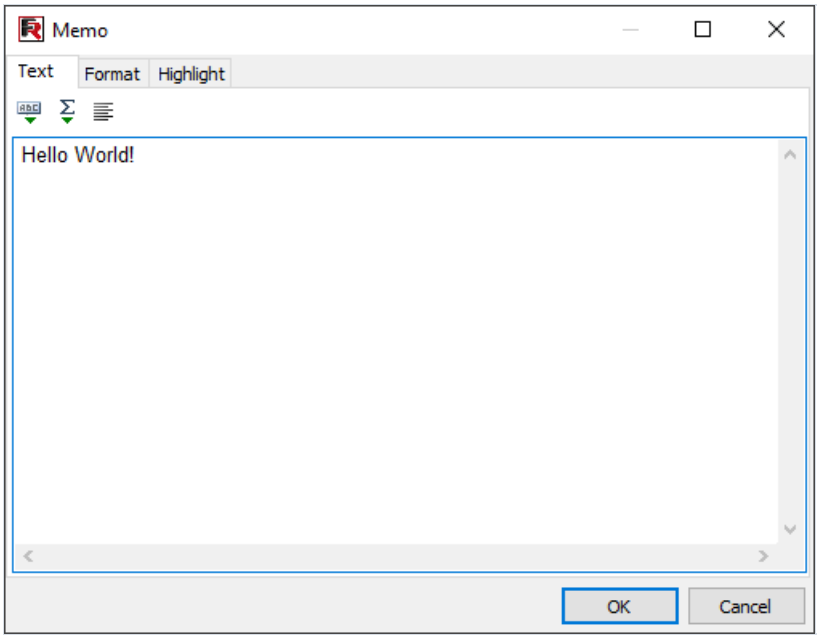
The example report will contain just one piece of information : “Hello, World!”. Open the report designer and click on the “Text” button in the Object toolbar. Move the mouse cursor over the page and click again. The object is inserted at the mouse position.



The text editor window will be opened right away; if it does not appear automatically then double-click the object.

> Text editor opening can be configured in the designer settings - see later on.

Type in "Hello, World!" and then click the OK button.



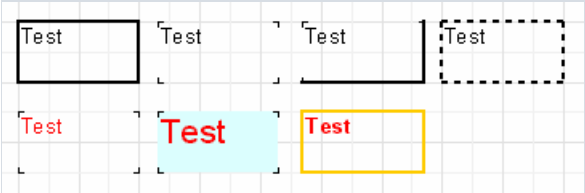
The report design is finished. To preview the report either select the “File>Preview” menu item or click the corresponding button in the toolbar.

The preview window containing a report page with the “Hello, World!” text will appear. This report can be printed out, saved to a file (*.fp3), or exported to one of the supported export formats.

The “Text” object

The “Text” object has many features. It can display text in a frame and be filled with a color. The text can be displayed using any font of any size and style. All the properties can be set visually with the help of the toolbars.

Here are some examples of text design:



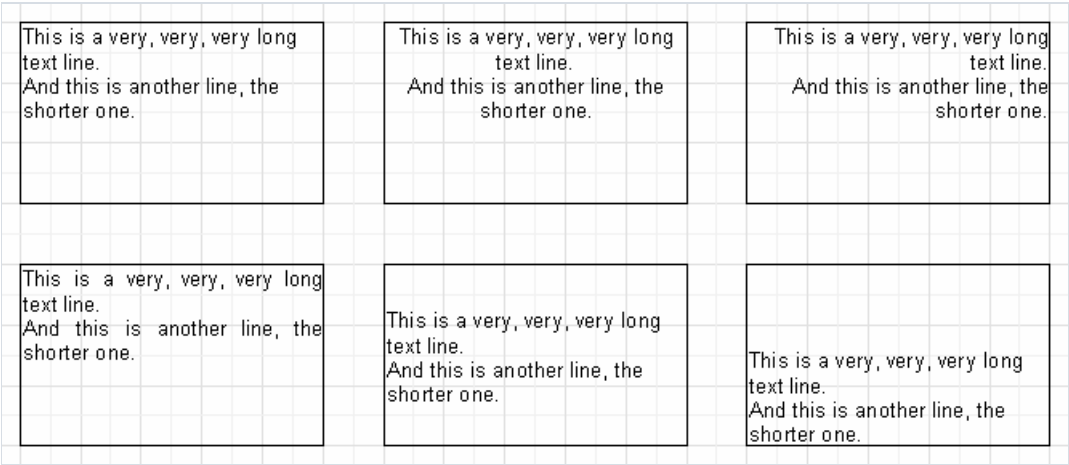
Now let’s look at other features of this basic object. As an example, let's create a new "Text" object to display two lines of text:

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

Enable the object frame from the toolbar and re-size the object up to 9x3 cm using the mouse. We see that the object can display not only a single line but also several lines of text.

Now reduce the object width to 5cm. It is obvious that long lines did not fit across the object and were therefore wrapped. This is controlled by the `WordWrap` object property. If it is disabled (either in the object inspector or via the object context menu) any long lines will be simply cut short.

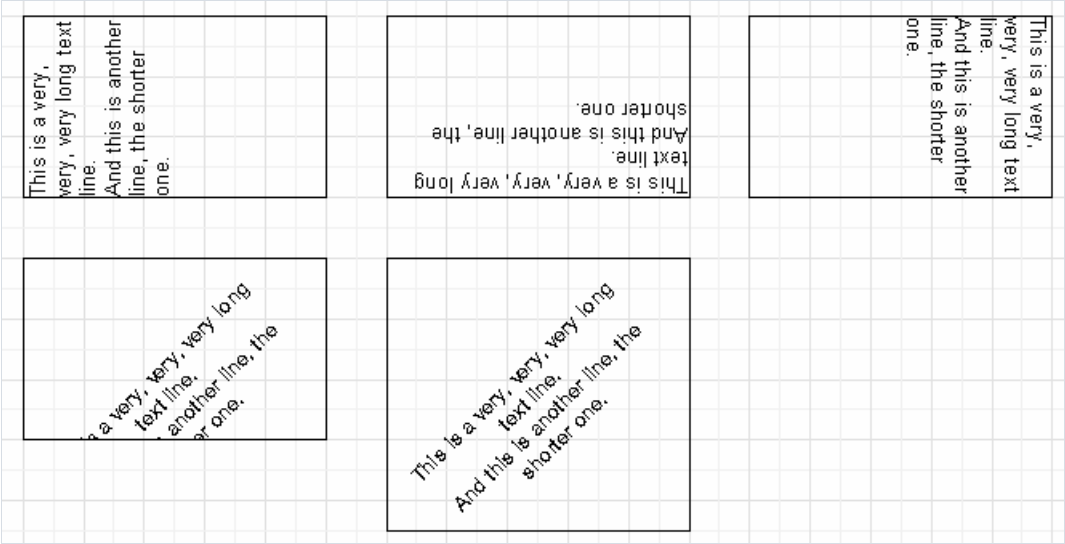
Now let’s see how text alignment inside the object works. Alignment buttons are located in the "Text" toolbar and control horizontal or vertical text alignment. Note the “Justify” button which lets you align paragraphs to both object edges - to do this `WordWrap` must be enabled.



All the text in the object can be rotated to any angle in the range 0..360°. The button in the “Text” toolbar allows you to quickly rotate the text to pre-sets of 0, 45, 90, 180 or 270°.

> For any other value set the required angle in the `Rotation` property of the object inspector.

When rotating text to angles other than 90, 180 or 270° the text may be cut off by the frame of the object, as shown below. To cure this increase the object height a little so that all the text fits within the object.



Let's briefly look at some other “Text” object properties which influence its appearance. Most of these properties are available only in the object inspector:

- `BrushStyle` : type of object filling
- `CharSpacing` : spacing between characters, in pixels
- `GapX` , `GapY` : text indents from left and top edges, in pixels
- `LineSpacing` : spacing between lines, in pixels
- `ParagraphGap` : first line indent for the paragraph, in pixels

HTML-tags in the "Text" object

The “Text” object does understand some simple HTML tags. Tags can be located within the text of the object. Tags are disabled by default, but to enable them either select “Allow HTML tags” in the object context menu or enable the “AllowHTMLTags” property in the object inspector.

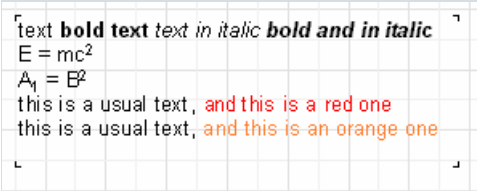
Here is the list of supported tags:

Tag	Meaning
`<b>`	bold text
`<i>`	italic text
`<u>`	underlined text
`<sub>`	subscript
`<sup>`	superscript
`<font color>`	font color
`<nowrap>`	text which is not split when `WordWrap` is enabled, the whole text is shifted to the next line

Please note that only a few tags are supported, but this should be enough for the majority of applications. It is not possible to modify the font size or name by means of HTML tags.

The following examples show how these tags can be used.

```
text <b>bold text</b> <i>text in italic</i> <b><i>bold and in italic</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 basic object is its ability to display not only a static text but expressions as well. Expressions can be located within the object mixed in with normal text. Let's see a simple example of how this works. Type the following into the object:

```
Hello, World! Today is [DATE].
```

When the report is run we can get something like this:

```
Hello, World! Today is 01.01.2020.
```

How does this happen? When FastReport creates the report and encounters an expression enclosed in square brackets the report engine calculates the expression's value and inserts this value into the text in place of the expression.

“Text” objects can contain any number of expressions mixed in with the normal text. Complex expressions can contain brackets (for example [1+2*(3+4)]). Constants, variables, functions and DB fields can all be used in expressions. We will learn more about these later in the chapter.

FastReport automatically recognizes expressions enclosed in square brackets in the text. But what happens if our normal text contains square brackets which we do not want to be considered as expressions? For example, if we need to display the following:

```
a[1] := 10
```

FastReport would consider [1] as an expression and display the text as:

```
a1 := 10
```

which is not what we want, of course. One way to avoid this happening is to disable expression recognition. Disable the `AllowExpressions` property (or “Allow Expressions” in the context menu) and all expressions in the text will be ignored. In our example, FastReport would then display exactly what we need:

```
a[1] := 10
```

But sometimes text is required to contain both an expression and normal text with square brackets, for example:

```
a[1] := [myVar]
```

Disabling `AllowExpressions` lets us display square brackets in the required places, but it also disables handling of expression. In this situation FastReport allows you to use an alternative set of symbols to designate expressions. The `ExpressionDelimiters` property, “[,]” by default, is responsible for this. In our example we can use angular brackets for the expressions instead of square ones:

```
a[1] := <myVar>
```

The “<,>” value must be set in the `ExpressionDelimiters` property, where the comma is required to separate the opening and closing symbols. Another requirement is that the opening and closing symbols cannot be identical, so “%,%” will not work. Complex symbols can be used, for example “<%,%>”. So our example could look like this:

```
a[1] := <%myVar%>
```

Bands in FastReport

Bands are used for placing the objects they contain at particular locations on the output page. When placing an object in the "PageHeader" band we tell the report engine that the given object must be displayed at the top of each page in the finished report. Similarly, objects in the "PageFooter" band are displayed at the bottom of each page.

Let's demonstrate this with an example. We'll create a report containing "Hello!" at the top of the page, the current date to the right of it and the page number at the foot of the page on the right hand side.

Open the FastReport designer and click the "New report" button in the toolbar. You will see a report template which already contains three bands: "ReportTitle", "MasterData", and "PageFooter".

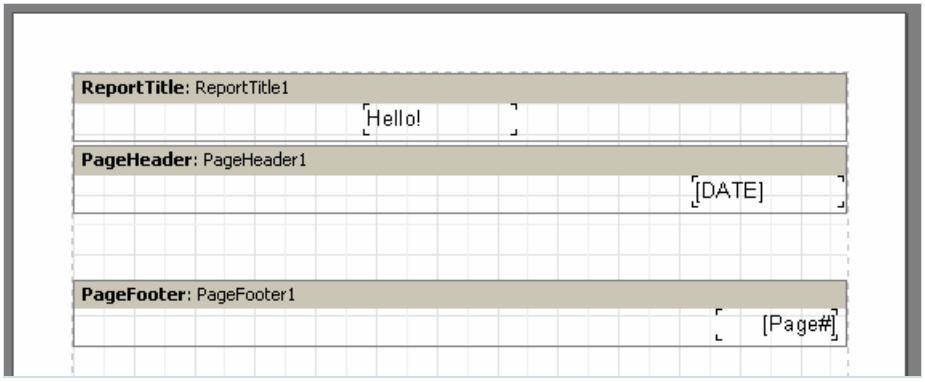
Let's remove the "MasterData" band for a while (click either on any free space inside the band or on its header and remove it with the "Delete" key or "Delete" in the context menu).

Now let's add a new band ("PageHeader"). Click the "Add band" button on the object toolbar and select "PageHeader" from the drop-down list. We see that a new band is added to the page. At the same time the existing bands are moved down.

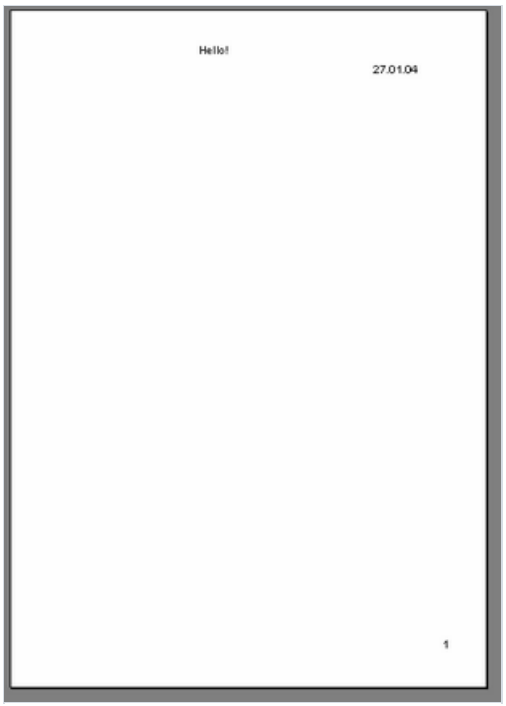
> FastReport designer automatically positions bands on the page - header bands at the top, data bands in the middle and footer bands at the bottom.

Now let's add some objects. Add a "System text" object to the "PageHeader" band and in its editor select `[DATE]` from the System variable drop-down list (you should remember that the current date can also be displayed in a "Text" object by typing `[DATE]` in its editor).

Next add a "Text" object containing "Hello!" to the "ReportTitle" band. A "Text" object displaying the page number has already been automatically added to the "PageFooter" band.



When running the report you will see that the objects in the finished report are printed in the proper positions on the page.



So, bands are responsible for positioning objects on the page. Depending on the type of band, we can print objects at the top or the bottom of a page and on the first or the last page. The basic bands needed in most reports work as follows:

- "PageHeader" band : displayed at the very top of each page
- "PageFooter" band : displayed at the very bottom of each page
- "ReportTitle" band : displayed at the top of the first page - this can be before or after the "PageHeader" band, depending on the page's `TitleBeforeHeader` property (found in the object inspector after clicking on any free space on the page)
- "ReportSummary" band : displayed in the free space at the very end of the report

Data bands

Now we will learn how to print data from DB tables or from queries.

What is considered to be a 'table' or a 'query'? They consist of data organized into lines (records or rows) which contain one or more columns (fields).

To print this sort of data FastReport uses a special type of band, the various bands which are named "...Data". To print a whole table or just some of its rows and fields, you must add one or more of these bands to the report, connect them to the table and place field objects within them.

When FastReport builds the report, the bands will be printed on the output page, once for each record in the table. If there is no free space left on the output page for a band, a new output page will be created by the report engine before continuing.

TfrxDBDataSet component

The `TfrxDBDataSet` connector component  from the Delphi FastReport component palette is used to connect a table (or any other data source) to FastReport. This component acts as the messenger between the data source and the FastReport core.

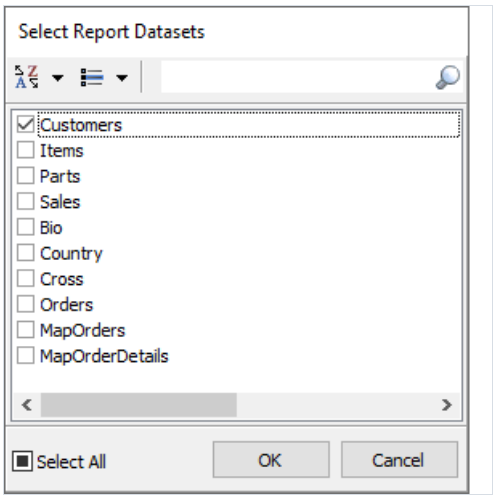
The component is responsible for record navigation and field reference. This allows the FastReport core to be independent of any data access library. FastReport can simultaneously work with "BDE", "IB_Objects" (which have a non-standard implementation, incompatible with `TDataSet`) and other libraries, as well as with data from sources not connected to a DB, for example arrays or files.

- The `TfrxDBDataSet` component is intended for working with data sources compatible with TDataSet (such as BDE, ADO, IBX and the majority of other libraries).
- The `TfrxIBODataSet` component is intended for working with IB_Objects.
- The `TfrxUserDataSet` component  works with other data sources (arrays, files, etc.).

It is very easy to use the `TfrxDBDataSet` component. Connect it to the data source by setting the `DataSet` property (for direct connection to a table or a query) or the `DataSource` property (for connection via a `TDataSource` component).

> The two methods are equivalent and mutually exclusive, though the first does allow data management without requiring a `TDataSource` component.

To make the Delphi component (and the data connected to it) available to the report, any dataset used in the report must be enabled. Do this through the "Report>Data..." menu item in the FastReport designer, selecting the required datasets in the opened dialogue.



“Customer List” report

Our second report will be much more complicated than the first one (it will contain a DB table of company clients). To produce this report let’s use the demonstration database `DBDEMOS` that is included in the Delphi distribution kit.

Create a new project in Delphi, place a `TTable` component on the form and set some of its properties:

```
DatabaseName = 'DBDEMOS'
TableName = 'Customer.db'
```

To make the table’s data available for use in FastReport add a `TfrxDBDataSet` component to the form and then set one property:

```
DataSet = Table1
```

Finally, add a `TfrxReport` (the basic component of FastReport) to the form, double-click it to open the FastReport designer and click the “New report” button there to automatically create a basic report design with three bands (“Report title,” “Master data” and “Page footer”).

To make our table useable we must enable it in FastReport. Do this by clicking the “Report>Data...” menu item, checking `frxDBDataset1` (it is the only dataset listed at the moment) and clicking OK. After the dialogue has closed, the `DBDataset` and the table fields to which it is connected become visible in the “Data tree” pane in the IDE.

Now let’s design the report. First, add a “Text” object containing “List of clients” to the "ReportTitle" band.

Next, connect the "MasterData" band to our dataset. This can done in any of three ways:

- double-click on the band
- select “Edit...” in the band’s contextual menu
- or click on the `DataSet` property of the band in the object inspector

Now place four “Text” objects (which will display the client number, customer name, phone and fax fields from the dataset) on the band. Let’s do this in several different ways to demonstrate some features of the FastReport designer.

The first way is to place a “Text” object on the band and type

```
[frxDBDataSet1."CustNo"]
```

into it. This is the least convenient way, since the field link has to be entered manually, with the possibility of incorrectly typing the text.

Somewhat easier is to use the expression designer - double-click the “Text” object and click the leftmost button on the “Text” tab of the editor window which opens. To insert our field double-click the field name in the expression designer. Click the OK button to close the dialogue and see the field link inserted into the “Text” object.

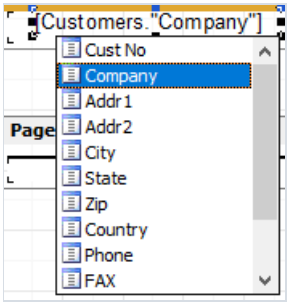
The second way of adding the DB field to the “Text” object is by setting two properties in the object inspector. Place a second “Text” object on the band, without writing anything in the editor window. Set the object’s properties using the object inspector:

```
DataSet = frxDBDataSet1
DataField = 'Company'
```

Both of these properties are presented as a list, so just select the required value from the drop-down using the mouse.

The third way is to "drag and drop" the required field from the "Data tree" pane into the report. This is the simplest and easiest way, but before doing it uncheck the "Create header" checkbox at the bottom of the "Data tree" pane, otherwise a second "Text" object, containing the field's name, is created in addition to the field link object itself. In this way select the "Phone" field and drag it onto the band.

The fourth way requires the designer option "Show drop-down fields list" flag to be set (via main menu : "View>Options") and the band to be already connected to the dataset. Place a blank "Text" object on the band and hover the cursor over the object – a drop-down button will appear at the right hand end. Click this button to open the list of DB fields and select the "FAX" field.



Our report design is complete.



Click on the "Preview" button to see the result.



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 have seen, “Text” objects can display data from a DB as well as static text and expressions. We have also seen that it can be done in two ways: either by placing a link to the DB field into the object text, or by connecting the object to the required field through its `DataSet` and `DataField` properties.

The first way is suitable for displaying both field contents and qualifying text in the same object. For example:

```
Contact person: [frxDBDataSet1."Contact_Person"]
```

Special syntax is required for links to the DB field: `[datasetname.“fieldname”]`. The field name (as well as the dataset name) can contain spaces but there must not be any space between the “point” and “quote” symbols.

Furthermore, we can also apply computing operations to a field in the expression, as shown here:

```
Length (cm): [<frxDBDataSet1."Length_in"> * 2.54]
```

Note how square and angle brackets have to be used. Remember that square brackets are used by default for delimiting expressions in the object’s text.

> Where required, the square brackets can be replaced by any other opening/closing character pairs (see the section “Displaying expression with the help of the “Text” object”).

Angle brackets are used inside expressions for delineating the FastReport variables or DB fields. Logically we should write:

```
Contact person: [<frxDBDataSet1."Contact_Person">]
```

instead of

```
Contact person: [frxDBDataSet1."Contact_Person"]
```

However, both these notations are correct, as FastReport does not require angle brackets where the expression contains only a variable or only a DB field. In all other cases the angle brackets are required, as in:

```
Length (cm): [<frxDBDataSet1."Length_in"> * 2.54]
```

Aliases

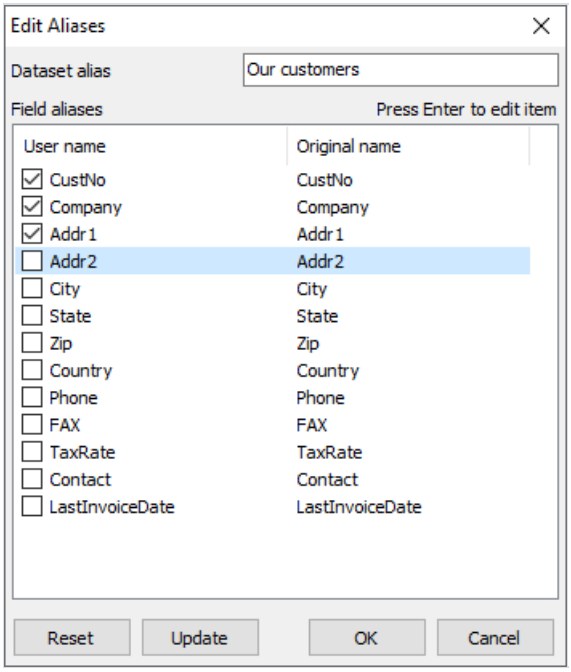
In the previous report the data source was named `frxDBDataSet1`, the fields were named "CustNo", "Company", "Phone", and "FAX" and we referred to them in the report using something like "[frxDBDataSet1."CustNo"]".

Is this easy to understand? Not really. It would be clearer if the data source and the field were named "Our clients" and "Number" respectively. There is a problem, `frxDBDataSet1` is the name of the component, within which spaces are not supported and "CustNo" is the name of the field in the database, which cannot be changed without database restructure.

However there is a way around this. We can use pseudonyms or aliases for these items. In FastReport both the dataset and its fields possess a second name property known as `UserName`, i.e. an alias, which can easily be changed. Whenever a component has been given an alias (i.e. `UserName` changed from its default), this alias must be used in FastReport - the component's `Name` property will not be recognized.

It is very easy to set aliases for a data source and its fields in FastReport. It is done in the Delphi environment. To open the alias editor, double-click on the `frxDBDataSet1` component or use its context menu.

You can modify the dataset and field aliases in the editor, and also specify which fields are needed in the report. Let's rename the dataset and fields as below:



The alias for the dataset can also be modified without using the alias editor, changing the `UserName` property of the `frxDBDataSet1` component.

Having done this we now need to modify the report, as the names of the dataset and fields have been changed. To modify the field names in report objects, it is easiest to use the fourth method described in the "Customer List" report" section.

Move the mouse cursor over the "Text" object so that the button in the right hand end of the object appears, click on the button and select the field from the list. Following this the origin of the data in the dataset and its fields is more apparent.

> It is better to assign aliases at the very beginning of report design to avoid having to later rename all the fields within a report.

Variables

As well as aliases, there is another way to help the report designer set more understandable names for DB fields and other information elements. A DB field name or any expression can be associated with a variable. To create and work with variables in FastReport, select the “Report/Variables...” menu item.

The list of variables in FastReport has a two-level structure. The first level simply contains the category and the second level contains the variable itself. This categorization of variables is convenient when the list of variables is very long. A variable list must contain at least one category, which means that variables cannot be located at the top level. Furthermore, category names cannot be included in reports, so each name within the list must be unique.

Let's illustrate the use of variables with the following example. Assume we have two data sources: the first is `frxDBDataSet1` with "CustNo" and "Name" fields and the second is `frxDBDataSet2` with "OrderNo" and "Date" fields. We can associate the following list of variables with the fields:

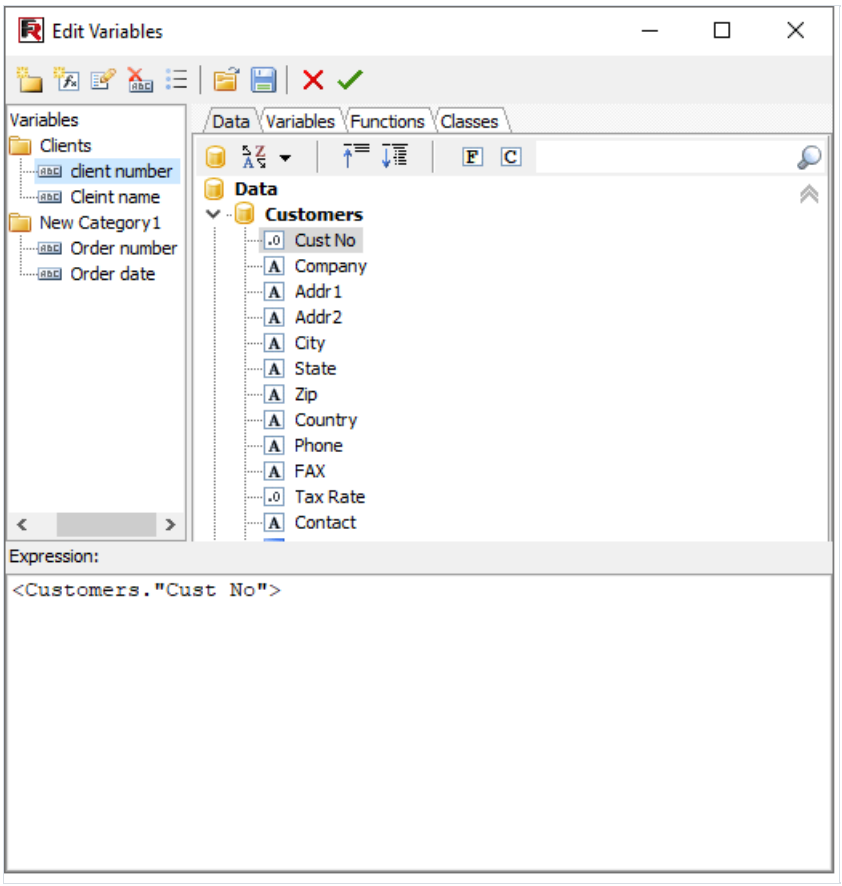
```
Clients
  Client number
  Client name
Orders
  Order number
  Order date
```

where “Clients” and “Orders” are two categories.

Open the variables editor and create this structure by using of the “New category”, “New variable”, and “Edit” buttons. To link the variables to the DB fields select a variable and double-click on the required DB field in the right hand pane. The link to the DB field will be shown in the bottom pane.

The variable is now associated with this expression so the value of the variable becomes the value of the expression. If necessary, the expression can be edited or modified manually and any FR functions or other variables can be used within it.

> Remember that categories must not be associated with any expression.



After the list of variables has been created, close the variables editor. Now we can insert the variables into the report.

In contrast to DB fields, there are fewer insertion methods. We can either insert a variable into the object text manually by typing "[Client number]", or we can drag a variable from the “Data tree” pane onto the report page, in which case switch to the “Variables” tab in this pane.

Let’s review what we have learnt so far:

- A report design is composed of design pages;
- Pages may contain FR objects, either placed on the page or within a band;
- Bands are placeholders on the design page and control where the objects they contain appear on the output page(s) according to the band type;
- “Text” objects contain the text we want to output in a given position, they can be multi-lined and may contain static text, data fields, variables, expressions or a combination of all of these;
- Data type bands (Master, Detail, Subdetail etc.), when connected to a `TfrxDbDataset`, control the number of times these bands appear (rows) and, together with the report engine, determine the number of finished pages output by the report.

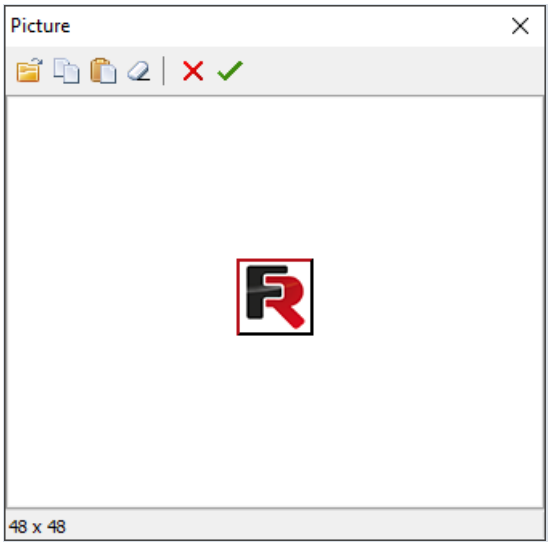
Note: even though these data bands may have types like Master, Detail etc., this is only a place relationship of the bands' hierarchical position on the output pages(s). The actual data relationships are dependent on the table/query relationships within the connected frxdbdatasets. Each data band level requires a separate `TfrxDbDataset` or equivalent association.

“Picture” object

The next object to look at is the “Picture” object. It is also frequently used in reports. Using this object you can insert logos, photos (e.g. of an employee) or any other pictorial information. The object can display graphics in `BMP`, `JPEG`, `ICO`, `WMF` or `EMF` format.

Let’s look at the capabilities of this object. Create a blank report and place a “Picture” object on the report page. If the picture editor does not open automatically then double-click on the "Picture" object.

You can load any picture from a graphic file or clear the existing picture and close the editor by clicking on the green Tick.



Some of the object's properties are available on the context menu as well as in the object inspector:

- `AutoSize`
- `Stretch`: enabled by default
- `Center`
- `KeepAspectRatio`: enabled by default

When `AutoSize` is enabled the object is re-sized up to the size of the picture it contains. Sometimes this can be useful if database pictures of differing size are being displayed. `AutoSize` is disabled by default.

`Stretch` is enabled by default. This property stretches the picture within the object. Change the object’s size using the mouse and you will see that the whole picture is always displayed within the object’s frame. If `Stretch` is disabled the picture (or only part of it) is displayed at its original size. This behaviour differs from `AutoSize` in that you have control of the object's frame size, which can be larger or smaller than the picture it contains. With `AutoSize` enabled you have no control over the object's frame size.

The `Center` property aligns the picture within the object.

`KeepAspectRatio` is enabled by default: it keeps the picture from distorting when the object’s size is changed. This property is only effective when `Stretch` is also enabled. With `KeepAspectRatio` enabled a drawn circle remains a circle without turning into an oval, whatever the “Picture” object's size. The stretched picture occupies only that part of the internal space of the object needed to display it with the correct aspect ratio. When disabled, the picture will be stretched to fill the object's frame and will be distorted if the frame’s aspect ratio is not the same as the picture's aspect ratio.

Another useful property available only in the object inspector is `FileLink`. Entering a filename, such as "c:\picture.bmp", or a variable which contains a filename, such as "[picture_file]", will load the picture from the named file when you run the report.

Report with pictures

The “Picture” object, like many objects in FastReport, can display data from a DB. This object can be connected to a DB field by setting the `DataSet` and `DataField` properties in the object inspector. In contrast to the “Text” object, this is the only way to connect this object to its data.

Let's demonstrate this with a report giving the names and images of some fish. We will again need the `DBDEMOS` database that is included in the Delphi distribution kit.

Create a blank project in Delphi, place a `TTable` component on the form and set some properties:

```
DatabaseName = 'DBDEMOS'
TableName = 'Biolife.db'
```

To work with this table in FastReport add a “TfrxDBDataSet” component and set these properties:

```
DataSet = Table1
UserName = 'Bio'
```

Finally, add a `TfrxReport` component to the form. Open the report designer and click the “New report” button to create a basic design in FastReport. Now enable the use of the `frxDBDataset` and its connected table in the report - from the menu select “Report>Data”, select the "Bio" dataset and click OK.

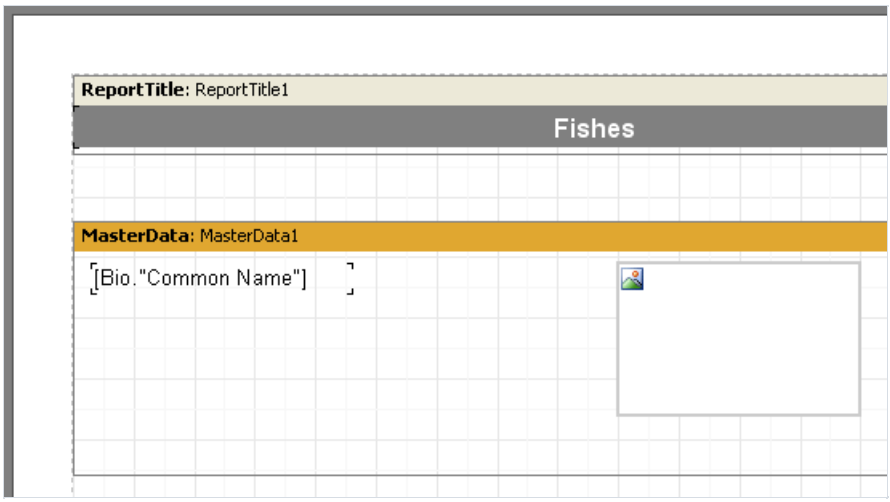
Now we’ll add objects to the report page.

- Place a “Text” object containing "Fish" in the “ReportTitle” band.
- Connect the “Master data” band to the data source (double-click on the band and select “Bio” from the list). Increase the band’s height to 5cm by dragging the bottom of the band down or by using the object inspector.
- Place a “Text” object in the band and connect it to the “CommonName” field using any of the methods previously described.
- After that, drop the “Picture” object alongside, and connect it to the “Graphic” field by setting its properties in the object inspector:

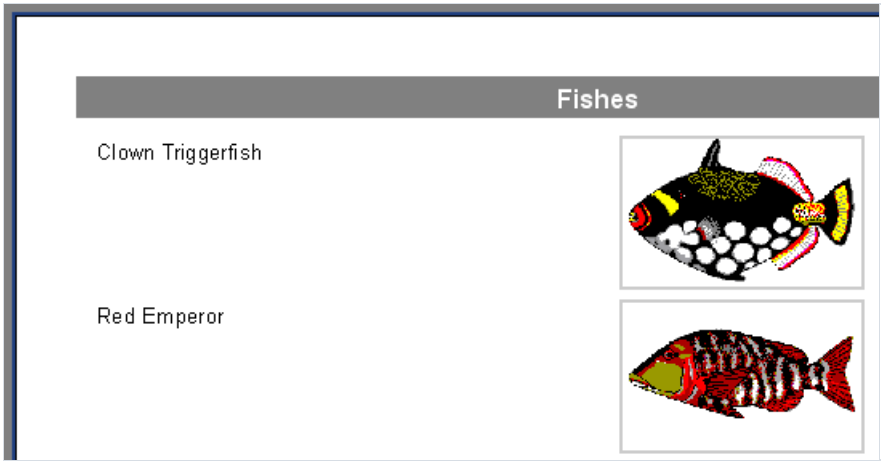
```
DataSet = Bio
DataField = 'Graphic'
```

> Note that both of these properties are of the “List” type, and can be set to the required values using the mouse.

To make room for the picture, stretch the object to 4 x 2.5cm.



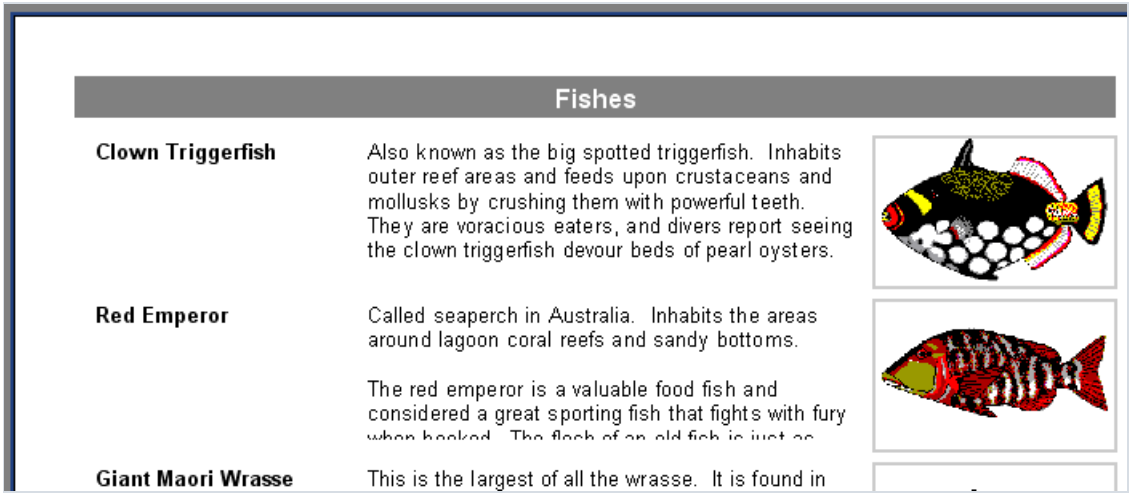
The report design is now finished and it produces the report shown here:



Multi-lined text display

We'll improve the previous example The "Biolife" table has a "Notes" field, which contains a detailed description of each fish. Update our report by adding this field to it.

At first glance, this would seem to be easy - add a "Text" object to the data band between the existing objects, connect it to the "Notes" field and set the object's size to 8 x 2.5 cm. However, the report preview is not exactly what we want:



FastReport performed exactly what it was instructed to do. The "Notes" field contains multi-lined text of varying length but the "Text" object displaying the information from this field has a fixed size. This is why some lines appear to be cut off. What can be done about this?

Of course, either the size of the object could be increased or its font size could be reduced. However, this may lead to wastage of space on the output page as some fish have long descriptions, while others have short ones. FastReport has some properties which allow us to resolve this problem.

These properties allow a band or an object to automatically adjust its height to create the necessary space for a given record (row). To achieve this, we just need to enable the `Stretch` property of both the band and the "Text" object. However, that is not quite all, because a "Text" object with longer text should be able to stretch by itself we'll need to set some of its other properties too.

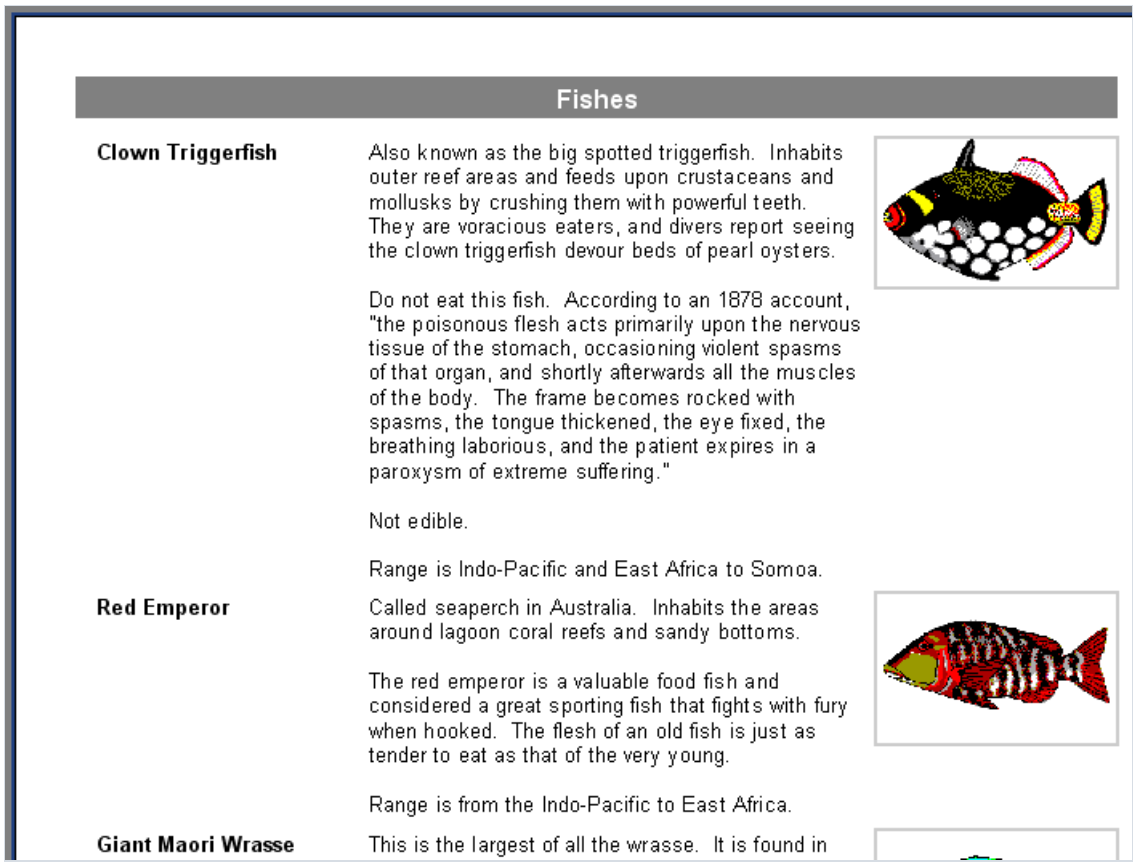
The "Text" object can automatically set its height and width to make space for its contents. `AutoWidth` and `StretchMode` can be used for this. `AutoWidth` allows the "Text" object to vary its width so that all the lines make space without splitting any words. This mode is useful when an object has a single text line and when growth to the right will not effect other objects. The `Stretch` property allows the object's height to grow to accommodate the text without changing the object's width. `Stretch` has several modes which can be selected in the object inspector:

`smDontStretch` – don't stretch the object (the default)

`smActualHeight` – stretch the object so it makes room for the whole text

`smMaxHeight` – stretch the object so that its bottom reaches the bottom of the band in which it is placed - we'll look at this mode later on

Here we are interested in the `Stretch` property of the "Text" object. Enable it using the object's context menu or by setting `StretchMode` to smActualHeight. Also enable the band's `Stretch` property. Preview the report and make sure that everything is now as expected.



As you can see, when constructing the report FastReport fills objects with data and stretches them if `Stretch` is enabled. It then computes the band's height so each object has enough room.

If the band's `Stretch` property is disabled, this height adjustment is not performed and the band is displayed at the height set in the design. In this case we would see that the objects with longer text would still be stretched but the band would not, which leads to overlaying of text, since the following band is displayed immediately after the previous one.

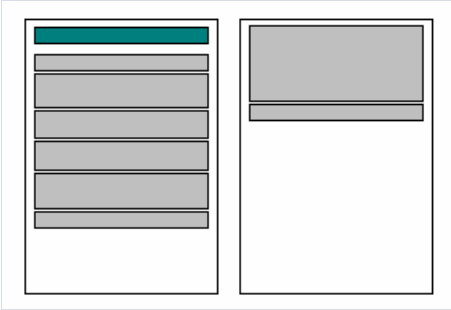
Data splitting

Let's look at a peculiarity of this report: there is a lot of blank space at bottom of the pages. Why is this?

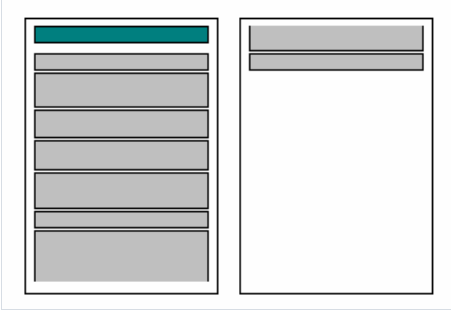
When a report is created the FastReport engine fills the white space of the page with the bands. After displaying each band the current position shifts down.

When FastReport finds that there is not enough white space left to display the next band (its height is larger than the white space left on the page) FastReport creates a new page and carries on with band display from there. This sequence continues for as many times as there are records in the dataset.

Our report contains an object with large text, which is why the band height is quite large. Furthermore, if a large band does not have enough room on a page it is shifted to the next one leaving a lot of unused space remaining at the bottom of the page, as shown here:



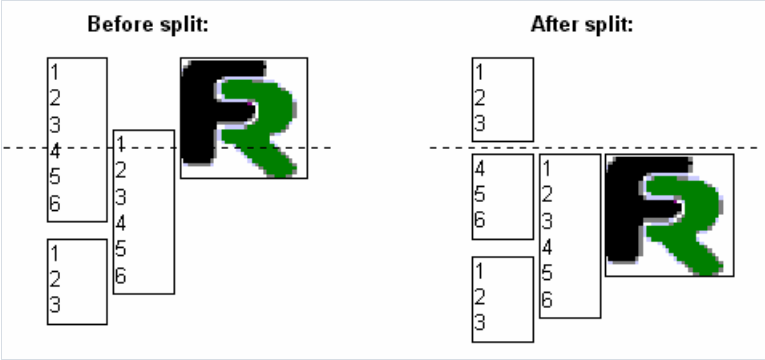
To limit paper wastage, let's use a FastReport feature that makes paragraphs from a band's contents. All we need do is enable the `AllowSplit` property of the "First level data" band. You will see that there is now less white space at the bottom of the report pages:



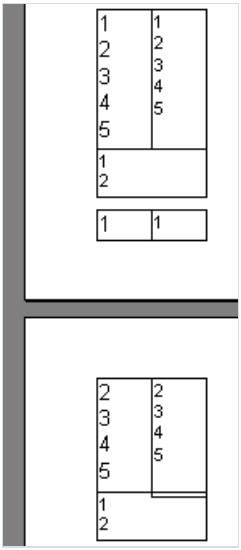
How does this band splitting work? There are some objects in FastReport which support this feature. They are the "Text", "Line" and "RichEdit" objects. They can be "split", while other objects cannot. When FastReport needs to split a band, it does it in the following way:

- displays the non-splittable objects which have room in the white space
- partially displays splittable objects ("Text" objects are displayed in a way that all lines have space in the object)
- creates a new page and continues with object display in the band
- if a non-splittable object does not have room in the white space, it is shifted to the next page; at the same time all the objects located beneath it are shifted as required
- the process continues until all of the band objects are fully displayed.

The splitting algorithm will become clearer by looking at this example:



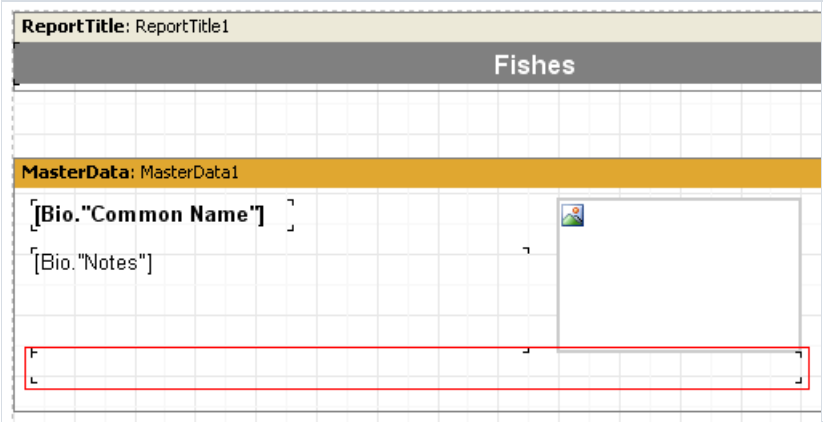
It should be noted, that the splitting algorithm is not perfect and the final output may not be quite as expected. You should use this option very carefully in cases where objects on the split band are grouped in a complicated way, and/or their font sizes differ. Here is an example of what could be generated:



Text wrap of objects

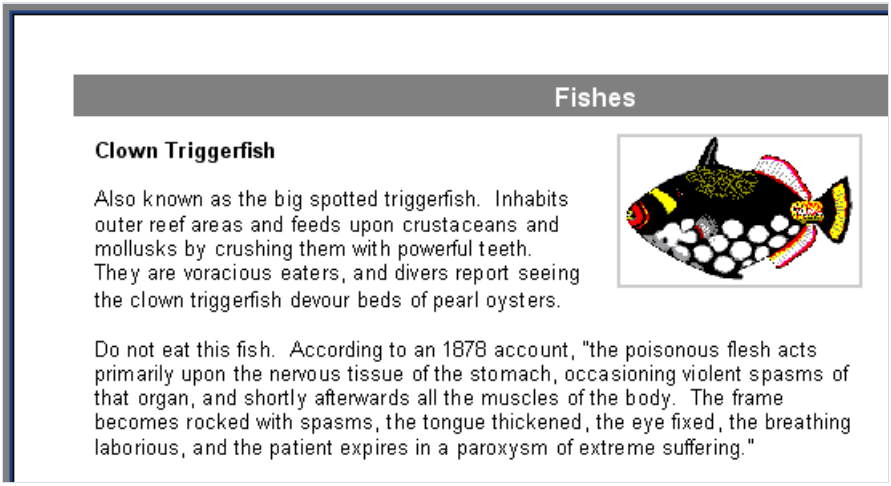
In some report designs text may be required to wrap around other objects (often when using pictures). Let's demonstrate how FastReport can do this in our current example.

Add one more “Text” object to the data band below the Bio.“Notes” object, as shown here:



We will disable stretching for the Bio.“Notes” object and enable it for the bottom object. To make the text “flow” from the Bio.“Notes” object into the bottom one, set the `FlowTo` property of the Bio.“Notes” object in the object inspector - it is a drop-down list. Select the bottom object’s name from this list.

The resulting report will look like this:



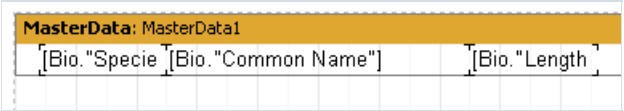
When the report is generated, if the text does not fit into the top object the excess part is shifted to the bottom object. With the arrangement of the two objects around the picture the effect of text wrapping is achieved.

> The main object must be inserted in the report before inserting the linked one, otherwise text flow may not function correctly! If this occurs, select the linked object and bring it to the front using the “Edit>Bring to front ” menu item.

Displaying data in the form of a table

Sometimes it is necessary to display data in the form of a framed table. An example of this type of report might be a price list. To create this type of report in FastReport just requires the enabling of frames for the objects located in the data band. Let's demonstrate several variants of frames with an example.

Create a report similar to this:



Place the "Text" objects side by side on the band and minimize the band's height.

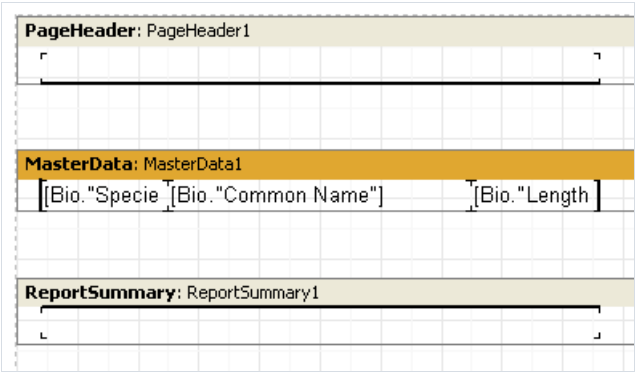
The first and the simplest type of table has full cell framing. To achieve this enable all frame lines (found in Frame.Typ property) for every object:

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 draws only horizontal or only vertical cell lines, again through the Frame.Typ property:

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

Finally, to draw only an external frame to the table the report needs a modification:



You can see that we have added two "Text" objects, one in the pageheader and one in the pagefooter bands. The appropriate frame lines for the objects along the edges of the data band have been enabled, resulting in the report looking like this:

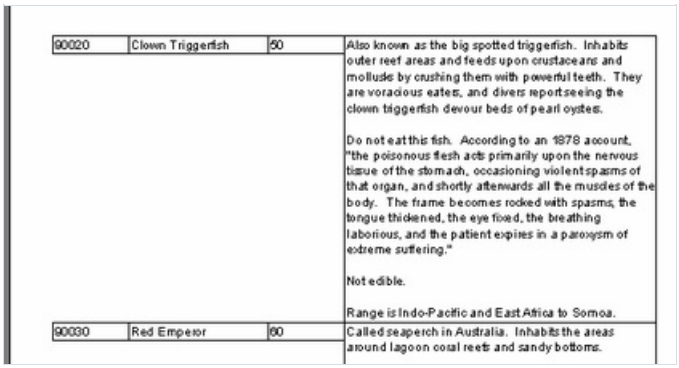
90020	Clown Triggerfish	50
90030	Red Emperor	60
90050	Giant Maori Wrasse	229
90070	Blue Angelfish	30
90080	Lunartail Rockcod	80
90090	Firefish	38

All these examples contained bands which had fixed sizes. How is it possible to display a table when the band is

stretched? Let's explain how, by means of an example.

Add a new field (multi-lined text from Bio."Notes") to our report. As you have already learnt, the `Stretch` property must be enabled both for this object and for the band in which the object is located so that the band height is altered, dependent on the size of the text in the "Text" object.

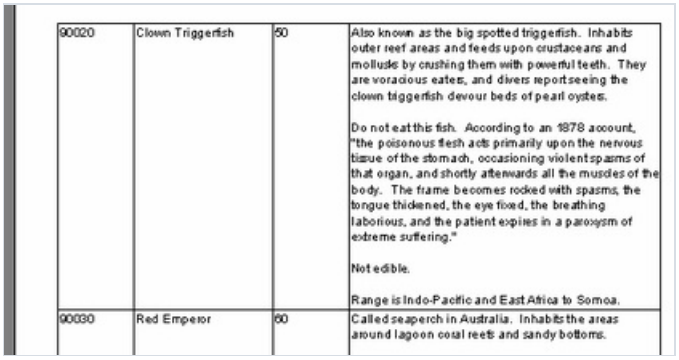
The generated report looks like this:



90020	Clown Triggerfish	50	Also known as the big spotted triggerfish. Inhabits outer reef areas and feeds upon crustaceans and molluscs 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 Somoa.
90030	Red Emperor	60	Called seaperch in Australia. Inhabits the areas around lagoon coral reefs and sandy bottoms.

This is a bit different from what we need - it looks nicer if the frames of the neighboring objects stretch as well. FastReport can solve this problem easily.

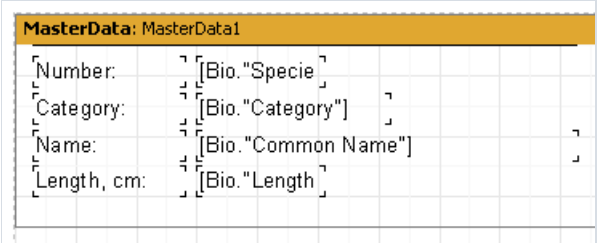
Set the `StretchMode` property to `smMaxHeight` in the object inspector for all objects which are to be stretched and the FastReport core will first calculate the maximum band height and then "stretch" objects with stretch enabled to the bottom edge of the band. Because object frames stretch together with the object the report's appearance changes:



90020	Clown Triggerfish	50	Also known as the big spotted triggerfish. Inhabits outer reef areas and feeds upon crustaceans and molluscs 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 Somoa.
90030	Red Emperor	60	Called seaperch in Australia. Inhabits the areas around lagoon coral reefs and sandy bottoms.

Printing labels

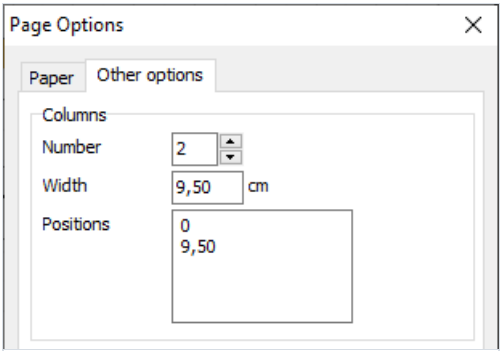
In contrast to table-type reports, other reports (such as label printing reports) may have the data arranged one field under another. Let's look at an example of this kind of report which displays data about fish, as in the previous example. The report data is presented as a label and has the following structure:



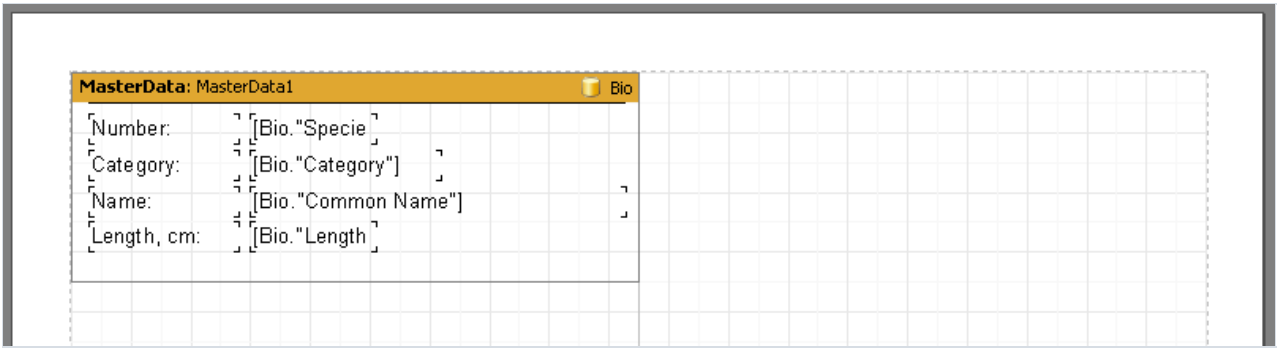
When previewed we would see the following output:



Notice there is a lot of blank space on the right side of the page. To make use of the whole page the number of columns in which data is displayed can be set in the report page settings. To do this either double-click on the white space on the design page or use “File>Page Settings...” on the menu.



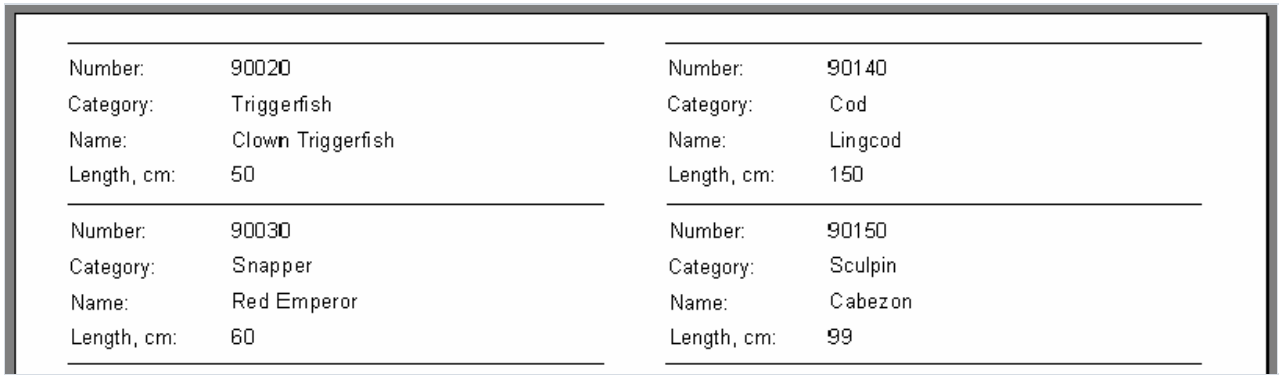
The column count, width and positions are set on the "Other options" tab in the dialogue. In our example only the Number needs to be set at 2, as FastReport adjusts the other options automatically. The column frame is displayed in the designer as a thin vertical line:



The report is now created in the following way.

- FastReport repeats the “First level data” band as long as there is white space at the bottom of the page.
- After that, a new column on the same page is created and data bands continue repeating from the top of the new column : this contrasts with simple reports, where a new page is created instead of a new column.
- When the second column has been filled then a third column is created, and so on - our example only has two columns though.
- When all of the columns have been filled then FastReport creates a new page and continues displaying data starting in the first column again.

Our two column report should look like this:

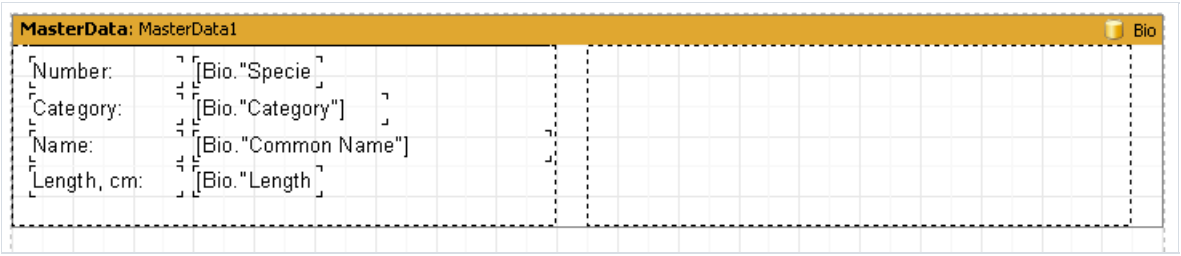


The `Columns` property, available in the object inspector for all data bands, is another way to set the number of columns. If this property is changed from zero however, the number of columns is set only for the selected databand and not for the whole page (as in the previous example). The effect is to display data firstly from 'left-to-right then top-to-bottom', in contrast to the 'top-to-bottom then left-to-the right' output shown above.

Disable the columns in the Page Options dialogue (set Column Number to 1) and enter 2 in the databand `Columns` property.

> Note that the `ColumnWidth` property must also be changed from the default zero to prevent column 2 overlaying column 1; optionally the `ColumnGap` property can also be changed.

FastReport then shows the column frames as dotted lines :

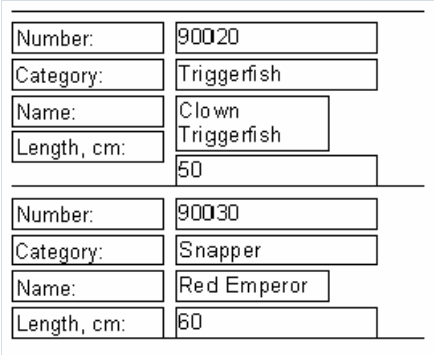


This design displays data in 'left-to-right then top-to-bottom' order.

Child-bands

There can be a problem when one field in a label-type report has content of variable length. To simulate this in our example, let's reduce the width of the Bio."Common Name" object to 2.5 cm and enable the `Stretch` property for this object and also for the "First level data" band. Enable all the frame lines for all the objects so that the effects of the stretching function are clear to see.

The design now outputs a report like this:

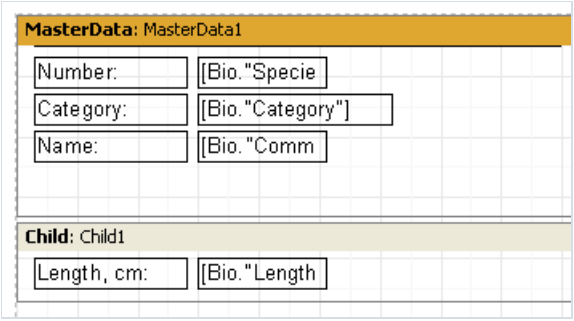


Here the first Bio."Common Name" field object contains a lot of text and is stretched into two lines. This causes the Bio."Length (cm)" field object, located beneath it, to be shifted downwards.

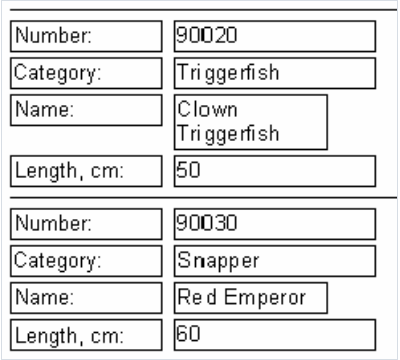
This happens because all the objects have their `ShiftMode` property set to smAlways by default, meaning they shift downwards if there is a stretchable object (a "Text" object with the `Stretch` property enabled) above them. The distance shifted depends on how much the object above is stretched.

But this is not want we want to happen on our label - we want the "Length, cm." object also to be shifted by the same amount. This can be achieved by using a special FastReport band type called the "Child" band.

A "Child" band is linked to (and displayed after) its parent band. Add a "Child" band to the design and drag the two "Text" objects into it, as shown here:



Link the MasterData band to the Child band, by setting its `Child` property to "Child1" in the object inspector. Now, each time the MasterData band prints, the Child band is printed immediately after it:

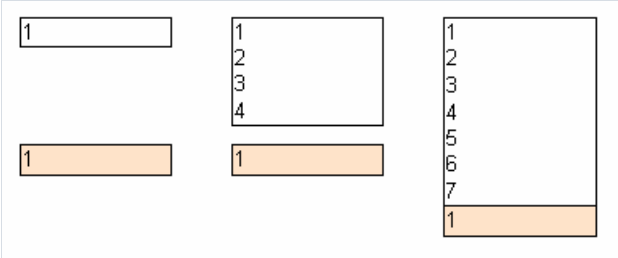


The "Length, cm:" title now lines up exactly with its value field “50”. To prevent a child band from being moved to the next page if there is insufficient white space on the page (and becoming so-called 'orphaned' from its parent band), enable the `KeepChild` property of the parent band in the object inspector.

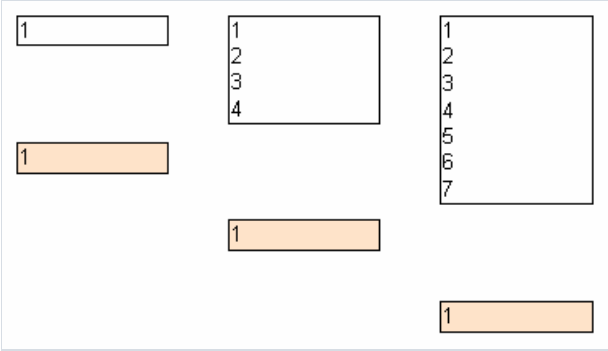
Shifting objects

You have already seen how smAlways works for the `ShiftMode` property. Let's look at the next mode of shifting, smWhenOverlapped. In this mode object shifting occurs when the object above stretches and overlaps the object below. Here are two secenarios:

1. the three top objects have `Stretch` enabled and the three bottom objects have `ShiftMode` set to smAlways. The bottom objects only shift far enough to allow space for the stretched object above it:

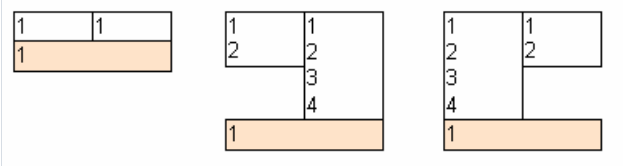


2. the three top objects have `Stretch` enabled and the three bottom objects have `ShiftMode` set to smWhenOverlapped. The bottom objects shift to allow space for the stretched object and also to maintain the designed separation of the two objects:



This allows the creation of complicated reports, in particular where an object could overlap several other objects above it at the same time.

In the example below all the upper objects contain stretchable text and all the lower ones are in smWhenOverlapped mode. The lower objects will always be displayed close to the object above, irrespective of text length in the objects above:



Report with two data levels (master-detail)

So far our example reports have used only one data band (”First level data” or ”MasterData”) to control data output. This was adequate for the output of data from one DB table. FastReport also allows reports to be designed having up to six data levels, all on the one design page. Furthermore, an unlimited number of data levels in reports can be achieved by the use of the "Subreport" object - this object is covered later on.

> Most reports in general need only one, two or three data levels, larger numbers of data levels are rare.

Let's look at how to design a two data level report. The report will output data from the demo tables: "Customer" and "Orders". The first table is a list of customers and the second one is a list of orders placed by those customers. The tables contain data in the following fields:

```
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 the customers. To view the orders from the "Orders" table which are placed by one customer in the "Customers" table, the two tables are linked on the ”CustNo” field, which is common to both tables. The report output from this data should appear 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's design the report. Create a new project in Delphi, place two `TTable`, one `TDataSource`, two `TfrxDBDataSet` and one `TfrxReport` components on the form. Set the component properties as shown here:

```
Table1:
DatabaseName = 'DBDEMOS'
TableName = 'Customer.db'

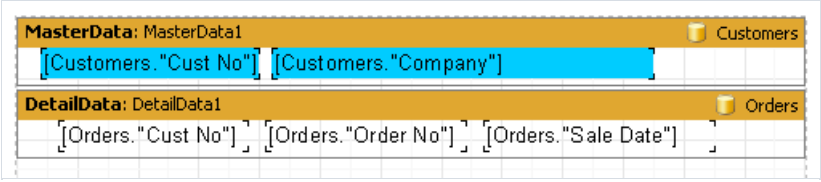
Table2:
DatabaseName = 'DBDEMOS'
TableName = 'Orders.db'

DataSource1:
DataSet = Table1

frxDBDataSet1:
DataSet = Table1
UserName = 'Customers'

frxDBDataSet2:
DataSet = Table2
UserName = 'Orders'
```

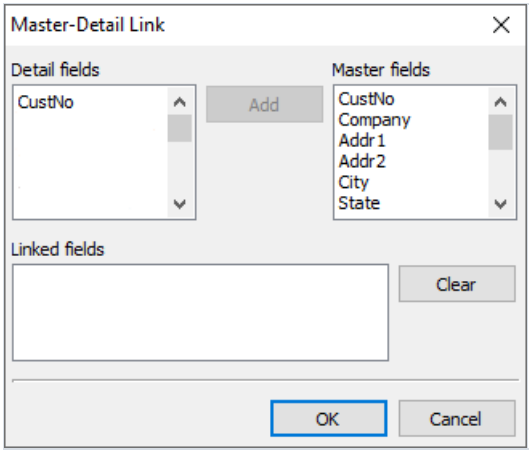
In the report designer, enable the data sources in the “Report>Data...” dialogue. Now add a “MasterData” and a “DetailData” band to the page:



> Note that the “MasterData” band must be placed above the “DetailData” band! Drag it there if necessary. If the Master is placed under the Detail, FastReport will generate an error message when you preview the report.

If you previewed the report now, you would see that the list of orders remains the same for every customer and contains all the records from the "Orders" table. This would be because we have not set the `MasterSource` property of the "Orders" table. Set `MasterSource = DataSource1` for the "Table2" component on the Delphi form.

Now we have set a 'master-detail' relationship. After that, we select the fields to link on. Set the `MasterFields` property of the "Table2" component:



We need to link together the “CustNo” fields in the two sources. To do this, select the desired fields and click the “Add” button. The Field link will appear in the bottom pane. Finish by closing the editor using OK.

When creating the preview FastReport does the following:

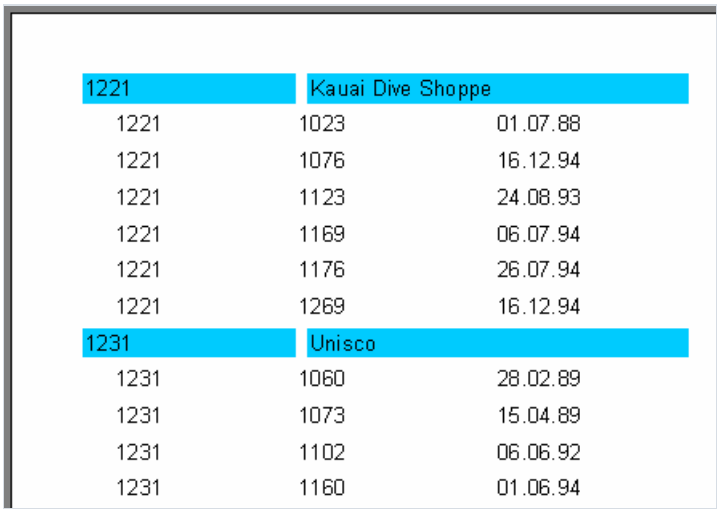
- After a record is output from the master table ("Customer"), it sets the filter on the detail table ("Orders").

- Only those records which meet the `Orders.CustNo = Customer.CustNo` condition will remain in the table.

This means that for each customer only those orders which belong to the current customer will be displayed in the detail band.

This is an important concept to grasp. Even though data bands may be of master or of detail type, they only control the positioning of the data on the output page (order and number of times displayed). The data displayed by the objects in the bands is dependent on which fields the objects are linked to and on the external linking of the two tables.

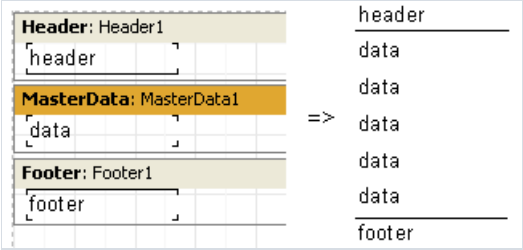
This is the final output:



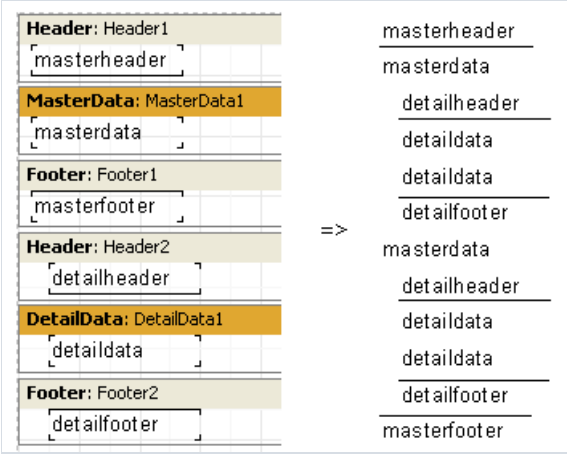
Reports containing up to 6 data levels can be constructed in a similar manner.

Headers and footers of a data band

Each data band may have a header and a footer. Headers are output first, then all the records in the data band, and finally the footers. Here is an example of how the headers and footers work in a simple report:



Let's look at a more complex example using two data levels - master and detail:

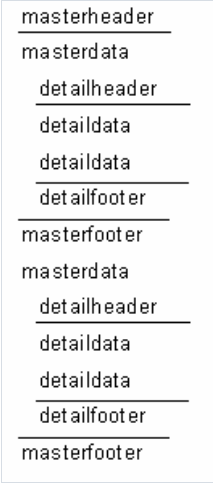


As you can see, the header is output before all data band records. So the master data header is output once at the beginning of the report and a detail data header is output before each group of detail bands belonging to the current master record band.

The detail footer is output after the group of detail bands belonging to the master record band and the master footer is not output until after all the master data band records have been output.

Using the `FooterAfterEach` property of the data band, we can override this behavior. Setting this property to True (you may also use the context menu for the data band - "Footer After Each Row") causes footer output after each data row.

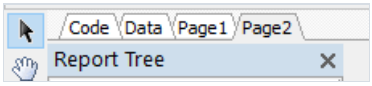
This may be useful in the design of some master-detail reports. If `FooterAfterEach` for the master data band is set to True the report will look like this:


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

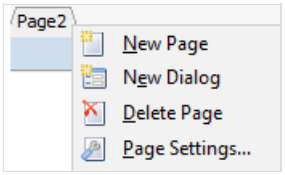
Multi-page reports

A FastReport report can consist of several design pages. Multi-page designs allow for the adjustment of properties such as size and orientation of each page, as well as allowing variation in the placement of objects and bands on the pages. When this type of report is output all bands from the first design page are be displayed, then bands from the second page, and so on.

When a new report is created in the designer it 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 page tab appears in the designer:



You can easily switch between pages by clicking on the page tabs. Page tabs can be dragged ("drag&drop") to easily change their printing order. An unnecessary page can be deleted using the button in the toolbar or by selecting the "Edit>Delete page" menu command. You can also call the context menu by right-clicking on the page tab:

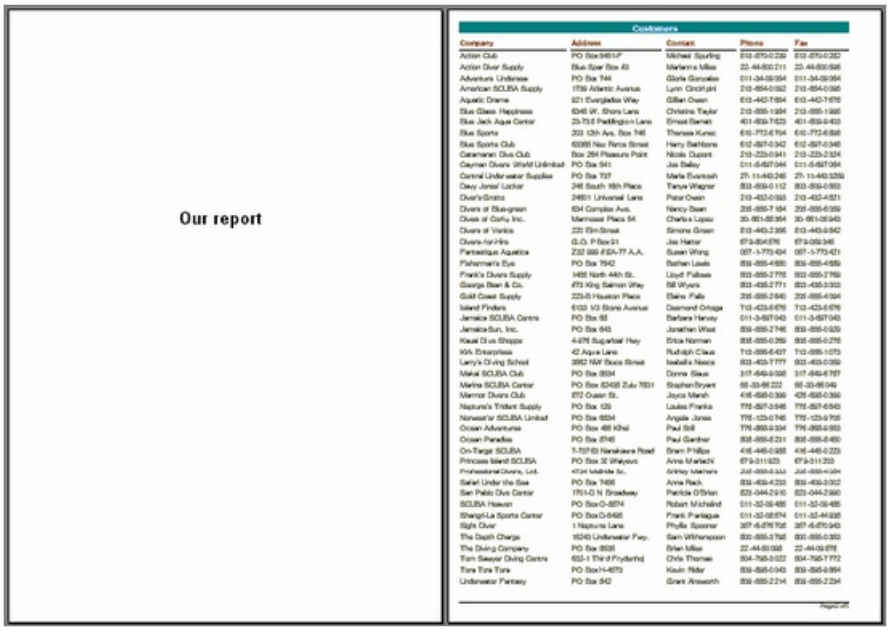


The number of design pages in a report is unlimited. As a rule additional pages are used either for title pages or, in more complicated reports, for data coming from many data sources.

Here is a simple example of creating a title page. Let's use our previous report having one data level.

- Add a new page to it, it will be added as a second page.
- Move it to the front of the report by grabbing the page2 tab with the mouse and dragging it over the first page tab, page1. This changes the page order.
- Select the new page and place a "Text" object containing "Our report" in the middle of the page.

That is all that is needed. The report with a title page is complete:



Pay attention to one feature of multi-page reports, however. If the `PrintOnPreviousPage` property is enabled in the object inspector for the second output page, then the second output page's objects will start printing on the white space of the first output page, and not on the new output page.

RowCount and PageCount properties

Sometimes the need arises to show static data several times over, for example when printing “Blank” business cards or post cards. For this purpose data bands have the `RowCount` property, and the report page has the `PageCount` property.

These two properties control the degree of band/page repetition in the report, without being influenced by the report data.

Groups and aggregates

Report with groups

In the previous example we constructed a two-level report based on the data from two tables. Another report which looks the same can be constructed in FastReport, this time based on a dataset obtained from a joined query.

To do this, an SQL query is needed which returns data from both tables and sorted in a particular way. In our example, the tables will be joined on the “CustNo” fields present in both tables. The query might be:

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

The "order by" line is required to sort the records on the “CustNo” field. This query returns a dataset like:

```
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
```

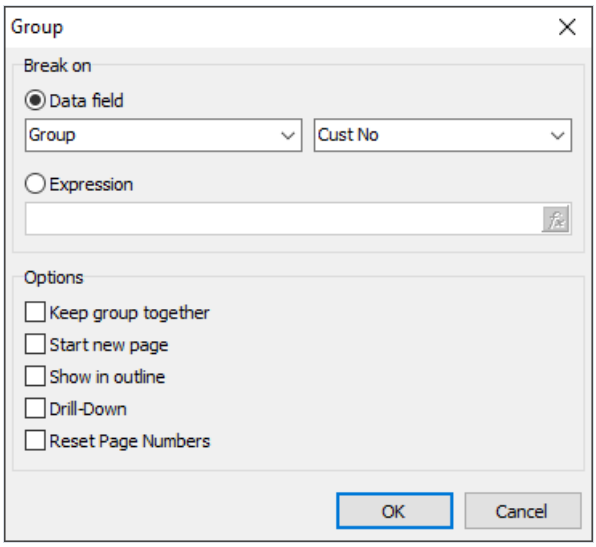
How can a multi-level report be designed using this data? In FastReport there is a special band, the “Group Header”. A condition (a DB field value or an expression) is specifed for the band; the band is output every time the condition's value changes. The following example illustrates this.

Let's create a new project in Delphi and place `TQuery`, `TfrxReport` and `TfrxDBDataSet` components on the form. Set their properties as shown here:

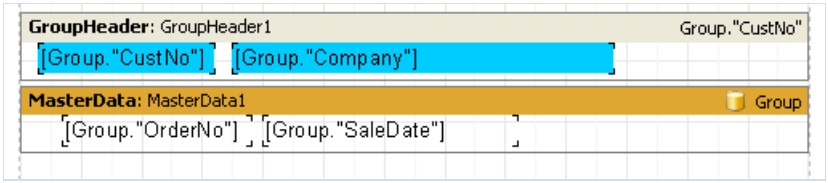
```
Query1:
DatabaseName = 'DBDEMOS'
SQL =
  select * from customer, orders
  where orders.CustNo = customer.CustNo
  order by customer.CustNo

frxDBDataSet1:
DataSet = Query1
UserName = 'Group'
```

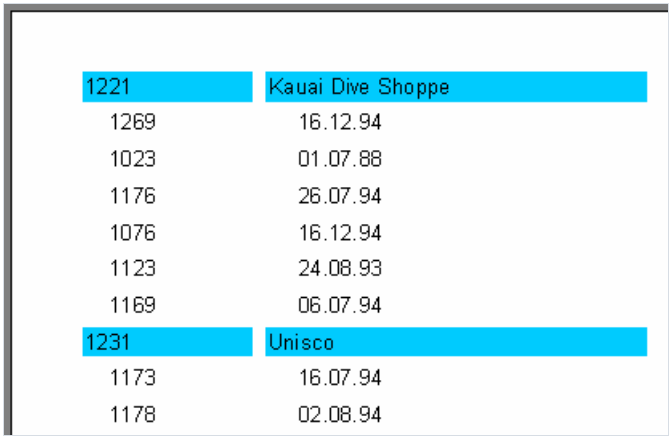
Open the report designer and create a new report. Connect our data source to the report. After that, add a “Group header” band to the report. Set a condition (in this case the "Group.CustNo" data field) in the “Group header” band editor:



Also link the data band to the "Group" data source and arrange some objects as shown (note that the group header must be placed above the data band):



On previewing the report, we obtain output similar to this:



As you can see, the "Group header" band is output only when the field to which it is linked changes in value. Otherwise the data band connected to the "Group" dataset is displayed.

If we compare this report to the master-detail report constructed earlier, it is obvious that order numbers are not sorted at all. This can be easily corrected by changing the SQL query's 'order by' clause:

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

Reports having nested groups can be designed in a similar way. The depth of nesting of groups is unlimited. Reports which use groups have some advantages over reports of the master-detail type:

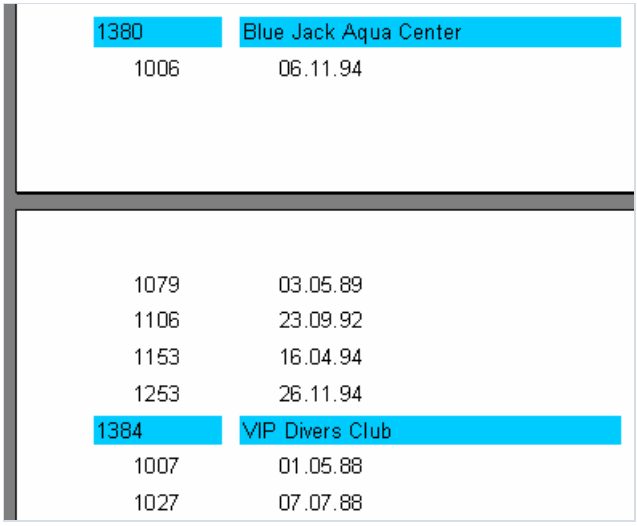
- the whole report needs only one dataset (query)
- the number of data grouping levels is unlimited

- data sorting becomes possible
- more optimal usage of the DB resources (the query returns only one dataset for output, with filtering done by the query).

The only disadvantage is the need to write queries in the SQL language. However, a basic knowledge of SQL is virtually obligatory for any programmer working with databases.

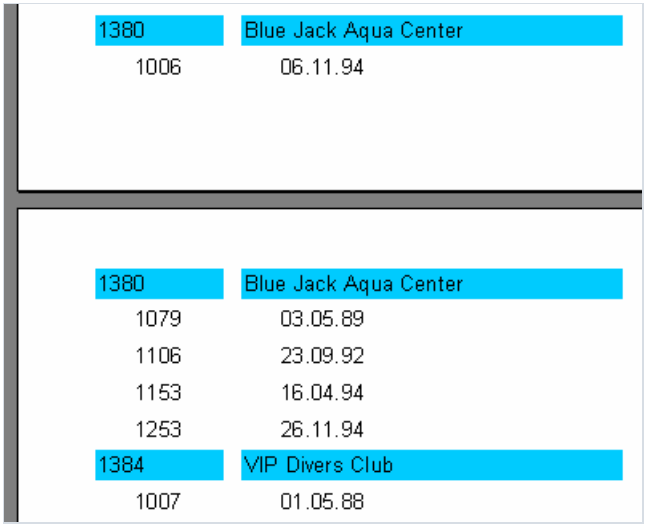
Other group features

Let’s look at how the group is output when the report calls for a new page:



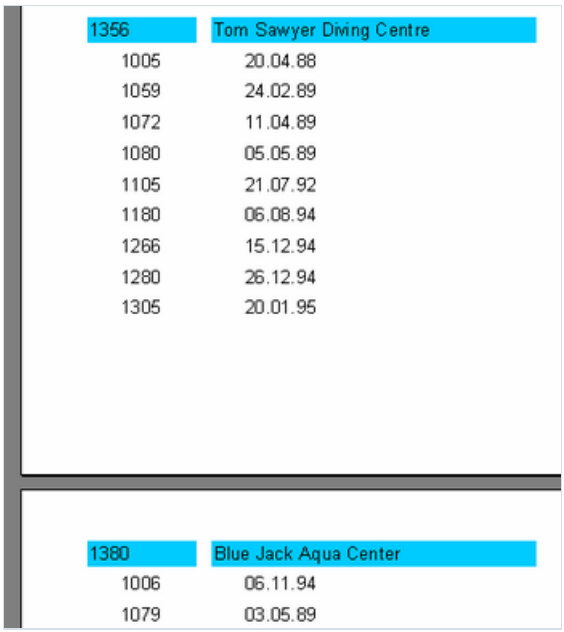
Looking at the report, it is unclear to which client the list of orders at the top of the second page refers. FastReport allows the group header to be output on the new page (which in our case identifies the client).

To do this, enable the `ReprintOnNewPage` property for the “Group header” band using the object inspector or the context menu . This will alter the report as shown:



There is another way to avoid breaking groups at page boundaries. This is to enable the `KeepTogether` property for the group header in the object inspector or context menu. Then, if the whole group doesn't fit into the free space on the output page, it is moved as a whole to a new page.

In our example, appearing like this:



A lot of blank space may be left on some pages but, if possible, the group is displayed complete on one page.

The `StartNewPage` group header property allows the output of groups on separate pages. It possibly will lead to a wastage of paper but might be useful in some situations.

Reset page numbers

The “Group header” band has a `ResetPageNumbers` property which allows us to reset page numbers when printing a group. What is it for?

Here's an example : you have created a report which puts the customer name in a group header and customer orders in the data band. Now you need to print the report and send it to all your customers, each customer getting only the pages of the report which refer to them.

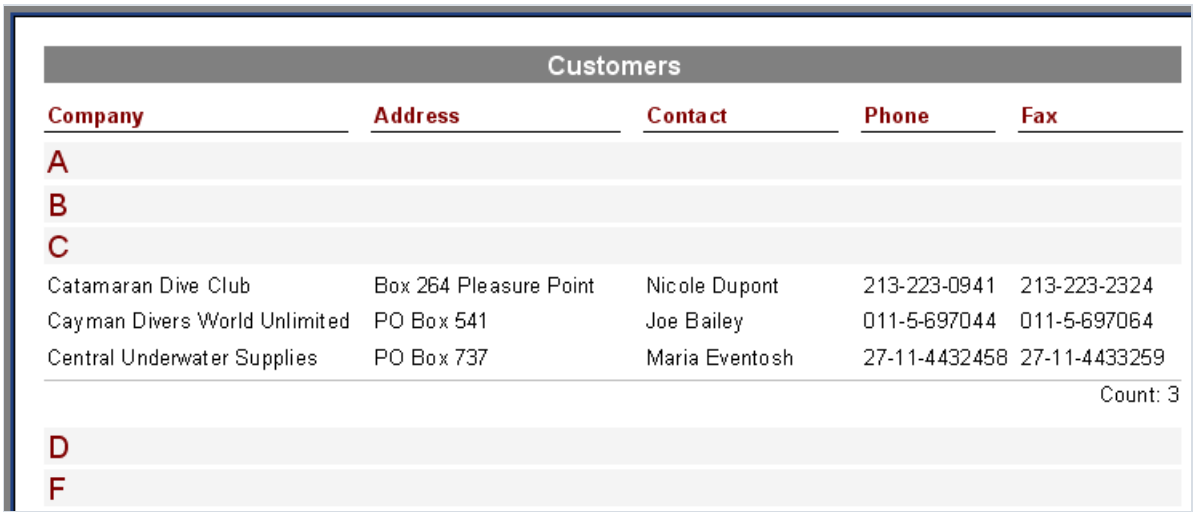
Unfortunately, the page numbering in the report is continuous, so a customer getting the pages numbered 50 to 52 will ask "where are the first 49 pages?". To avoid this situation you have to number to each customer's pages with its own sequence. Inside the report each group will have pages numbered from 1.

> Please note: if you set `ResetPageNumbers` to True, you should also set `StartNewPage` to True, so that each group will start on a new page. To print the page number or total pages, you should use the `[Page]` and `[TotalPages]` system variables and not `[Page#]` and `[TotalPages#]`.

Drill-down groups

The group header has a property called `DrillDown`. If you set it to True, the group becomes interactive. This means you can click on the group header in the preview window and the group will expand (display all records in the group) or collapse (display only the header and, if `ShowFooterIfDrillDown` is True, the footer).

Here is an example of such a group with one expanded header:

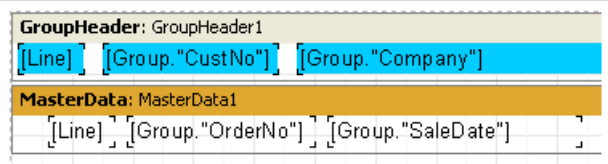


You can control whether all groups are collapsed or expanded when the report first runs. By default a group is collapsed but you can set `ExpandDrillDown` to True if you want it expanded.

> You can also use the preview's context menu to expand or collapse all groups at once.

Line numbering

Let's use our example to show how to number the lines in a group. To do this, we add a "Text" object containing a system variable `[Line]` to both of our bands (this is most easily done by dragging & dropping from the "Variables" tab of the "Data Tree" pane).

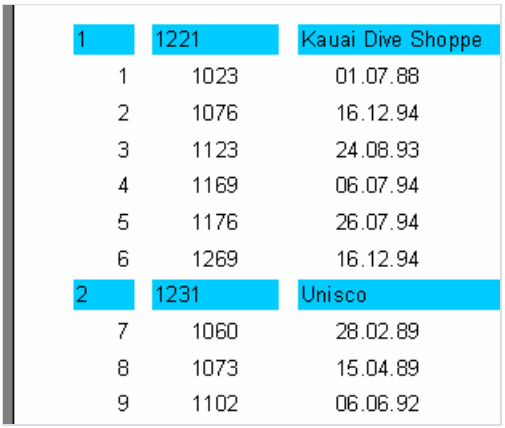


When previewing the report, we can see that both the data levels now have their own line numbers:



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

To continuously number the second level data lines, use the `[Line#]` variable instead of `[Line]` in the "Text" object on the data band. The result will then look like:



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 need to display some summary information (such as: "total of a group", "number of group elements" etc). FastReport provides aggregate functions for calculating aggregate values over some data span. The aggregate functions are:

Function	Description
`SUM`	returns the total of an expression
`MIN`	returns the minimal value of an expression
`MAX`	returns the maximal value of an expression
`AVG`	returns the average value of an expression
`COUNT`	returns the number of lines (rows) in the data span

The syntax of all aggregate functions (except `COUNT`) is similar to that of the `SUM` function:

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

The parameters are:

`expression` – the expression to be calculated

`band` – the name of the data band within which the calculation is performed

`flags` – a bit field, with values

1 : include invisible bands in calculation
2 : accumulate the aggregate as a running total (do not reset the aggregate when the current data span resets)
3 : (both of the two previous options)

An expression is the only mandatory parameter, the other two are optional. Nevertheless, to avoid making mistakes it is recommended that band parameters are always given.

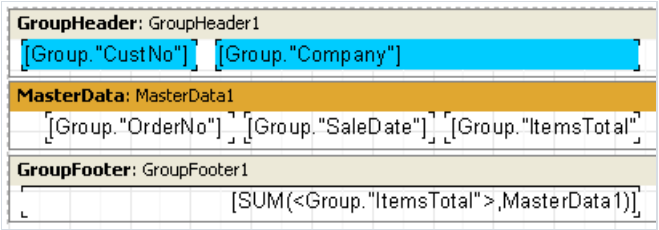
The `COUNT` aggregate function has the following syntax:

```
COUNT(band, flags)
COUNT(band)
```

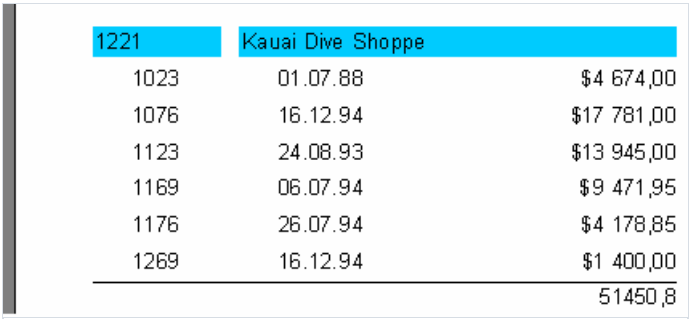
where the parameters have the same meaning as above.

There is a general rule for all aggregate functions: an aggregate can only be calculated over a data band and can only be used in that band's footer, which can be one of : footer, page footer, group footer, column footer or report footer (summary band).

How do aggregate functions work? We will look at this using our group report example. Let's add some new elements to the report:



The Group."ItemsTotal" field in the data band displays the current order total. Place a "Text" object in the group footer containing the aggregate `SUM` shown above. It will display the total of all orders placed by the given customer. Using a calculator, we can check that the result is correct:



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

Aggregate functions work like this:

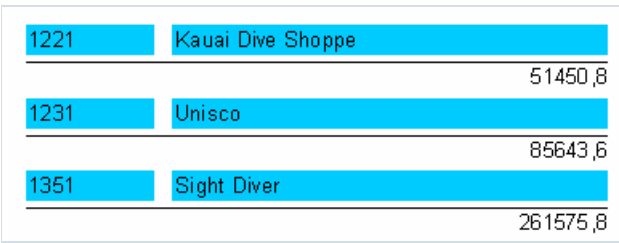
- before outputting a report, FastReport scans the "Text" object contents to find any aggregate functions.
- The aggregates found are linked to the data bands in their parameters (in our example `SUM` is linked to the "MasterData1" band).
- During report output (when the data band is displayed) the values of the aggregates linked to it are calculated. In our case the Group."ItemsTotal" field values are accumulated.
- Once the group footer displaying the aggregate has been output the aggregate value is reset to zero, and the cycle is repeated for the next group, and so on.

What is the purpose of the optional `Flags` parameter in aggregate functions? Reports may have some, or all, of the data bands hidden. We may, however, need to calculate aggregates over all the data bands, whether visible or not.

In our example, set the `Visible` property of the data band to false, so preventing its display. To still have this hidden data band included in the calculations, we have to set the third, optional parameter in the function call to the figure 1, i.e.:

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

This produces a report looking like tis:

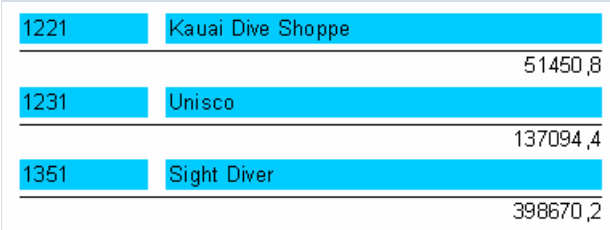


1221	Kauai Dive Shoppe	
		51450,8
1231	Unisco	
		85643,6
1351	Sight Diver	
		261575,8

When the `Flags` parameter value is set to 2, the aggregate value is not reset immediately after it is displayed: the aggregate becomes a "running" calculation for each successive output. Let's modify the function call as shown here:

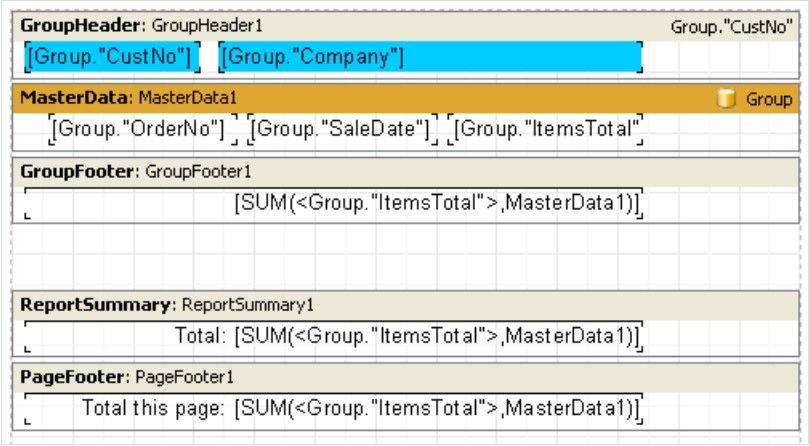
```
[SUM(<Group."ItemsTotal">,MasterData1,3)]
```

The value “3” is a bit combination of “1” and “2”, meaning that we need to include the invisible bands, without resetting the total after each group. As a result, we have:

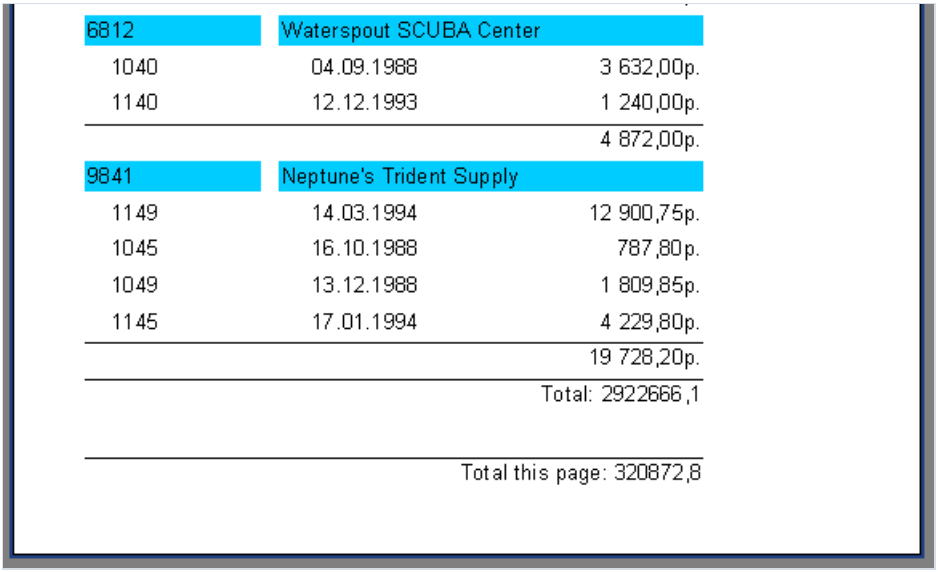


Page and report totals

Quite often, we need to display summary totals for a page or for a whole report. We can use the aggregate functions in this situation as well. We'll show this by making some changes to our example:



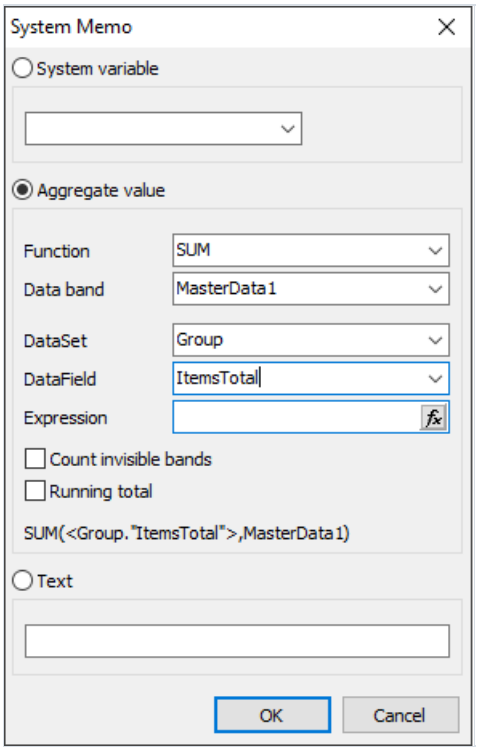
As you can see, we have added a "Report Summary" band containing a "Text" object with the aggregate `SUM` to both the "Report Summary" band and the "Page Footer" band. That is all that is needed:



Inserting aggregate function

So far we have manually inserted the aggregate functions into “Text” objects. Now we will look at other ways to insert aggregate functions.

Firstly, we can use the “System text” object to output an aggregate. In fact this object is similar to a “Text” object with its own special editor for more easily specifying system variables or aggregate functions:



Select a function type, then a data band (over which the aggregate is to be calculated) and finally a DB field or an expression whose value is to be calculated. You can also set the “Count invisible bands” and “Running totals” flags, if required.

The second method is to use a “Text” object and click the Σ button in its editor : this opens a dialogue similar to the “System text” object editor. When the OK button is clicked a call to the aggregate function is inserted into the object's text.

Formatting, highlighting

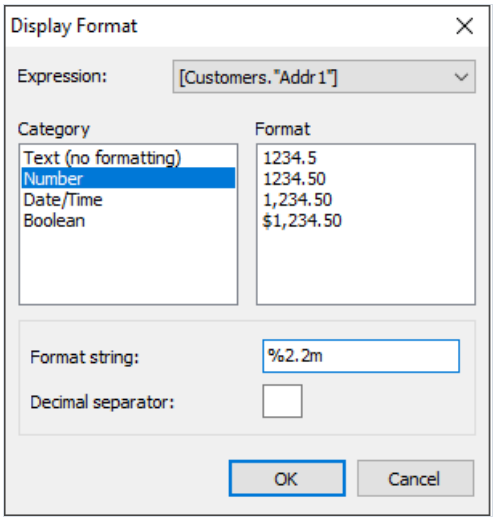
Formatting of values

A feature of aggregate functions is that the returned numerical values are not formatted, as shown by the first example, which uses `SUM`:

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

Data fields usually return a formatted value, which is simply displayed by the "Text" object without any change. To apply formatting to the `SUM` result, let's use the value formatting tools in FastReport.

Select the object containing the `SUM` and open the format editor either from 'Display Format...' in its context menu or through the "DisplayFormat" property in the object inspector.



This editor lists the format categories on the left, the corresponding formats on the right and the format string and decimal separator for the selected category and format below.

We'll select the "Number" category and "$1,234.50" format.

> The format string is an argument for the Delphi `Format` function, which FastReport uses to implement number formatting. The format string and decimal separator can be changed. If the decimal separator is left blank then the current regional setup value is used.

After clicking OK and previewing the report you will see that the Sum in the report is now formatted correctly:

1176	26.07.94	$4 178,85
1269	16.12.94	$1 400,00
		$51450,80

Note the combobox at the top of the dialogue form. If we have more than one expression in an object, we may set different formatting for each expression.

Inline formatting

Inline formatting allows you to set different formatting for each expression contained in the object.

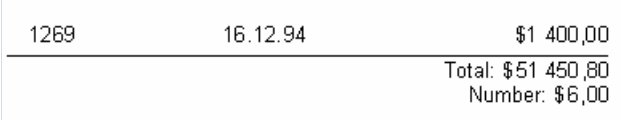
> It was used in previous versions of FastReport. Now it is obsolete (use the formatting dialogue to set different formatting for each expression).

Using the example, re-size the footer and its object and change the object text to this:

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

The total and the number of orders will be displayed in the object.

In the report preview both of these values are shown in monetary format, which we had previously set. This is incorrect:

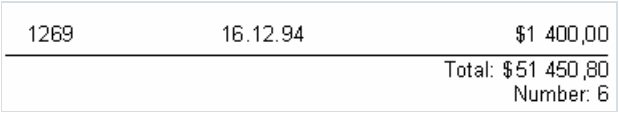


To display each value in its correct format they need to be formatted individually. To do this we use format tags, which are placed just before the closing square bracket of the expression.

In our example, disable formatting for the object (select "Text (no formatting)" category in the format editor). Now we need to specify the format for just the first expression, as the second one will be displayed correctly by default (i.e. as an integer). Change the object text as follows:

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

Preview the report to make sure that the object is displayed correctly:



The general syntax of format tags is:

`[expression #formattag]`

> Note that the space character between the expression and the "#" symbol is mandatory!

The format tag itself might look like:

`#nformat_string` – numerical format

`#dformat_string` – date/time format

`#bFalse,True` – boolean format

Format_string in each case is the argument to the function used for formatting. So, for numerical formatting the Delphi Format function is used, and for date/time the `FormatDateTime` function. The syntax for these functions can be found in the Delphi help system. Below are several values used in FastReport:

for numerical formatting:

`%g` – number with the minimal places after the decimal point

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

`%2.2n` – as previous, but with thousands separator

`%2.2m` – monetary format, accepted by the Windows OS, dependent on the regional settings in the control panel

for date/time formatting:

`dd.mm.yyyy` – date as '23.12.2003'

`dd mmm yyyy` – date as '23 Nov 2003'

`dd mmmm yyyy` – date as '23 November 2003'

`hh:mm` – time as '23:12'

`hh:mm:ss` – time as '23:12:00'

`dd mmmm yyyy, hh:mm` – date and time as '23 November 2003, 23:12'

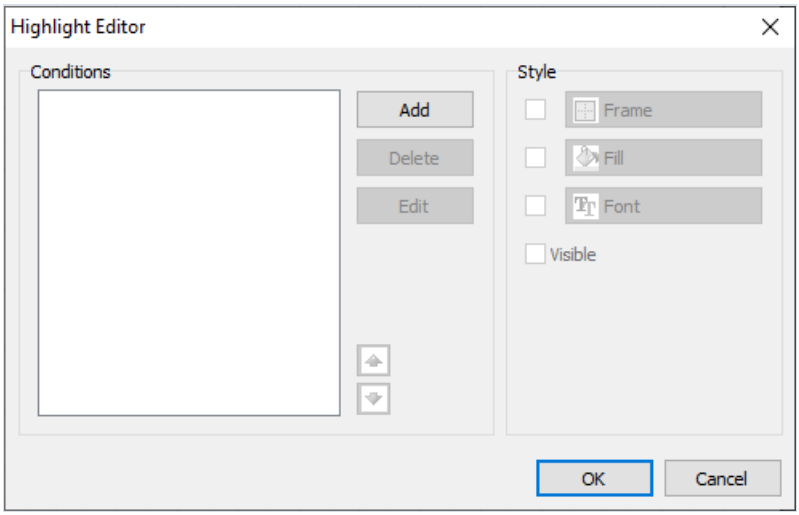
A comma or a dash can be used instead of the dot in the format_string for numerical formatting. This symbol is used as the separator between the integer and the fractional parts of the value. Any other character is not allowed.

For formatting with the `#b` type (boolean), the format_string is entered as two values separated by a comma. The first value corresponds to “False” and the second to “True”.

Conditional highlighting

There is a possibility to change the "Text" object's appearance depending on the given conditions. For example, an object can be highlighted with red color if it has a negative value.

This feature is called "conditional highlighting". To set up it, select the "Text" object and click the button on the "Text" toolbar. You will see the following dialog window:



It is possible to define one or several conditions and set up the style for every condition. Style can contain one or several settings:

- frame;
- fill;
- font;
- object's visibility.

You can indicate, which settings need to be changed when the condition is met. For this, check the needed setting using the checkbox.

In order to create a new condition, click the "Add" button. You will see an expression editor. Here, it is possible to write any expression which returns a boolean result. In many cases you will use the `Value` variable, which contains the currently printing value.

Let us look at the following example: we have a "Text" object, in which we print the amount of products in stock:

```
[Products."UnitsInStock"]
```

We want to paint the object red, if the amount of products = 0. For this, we create the following condition:

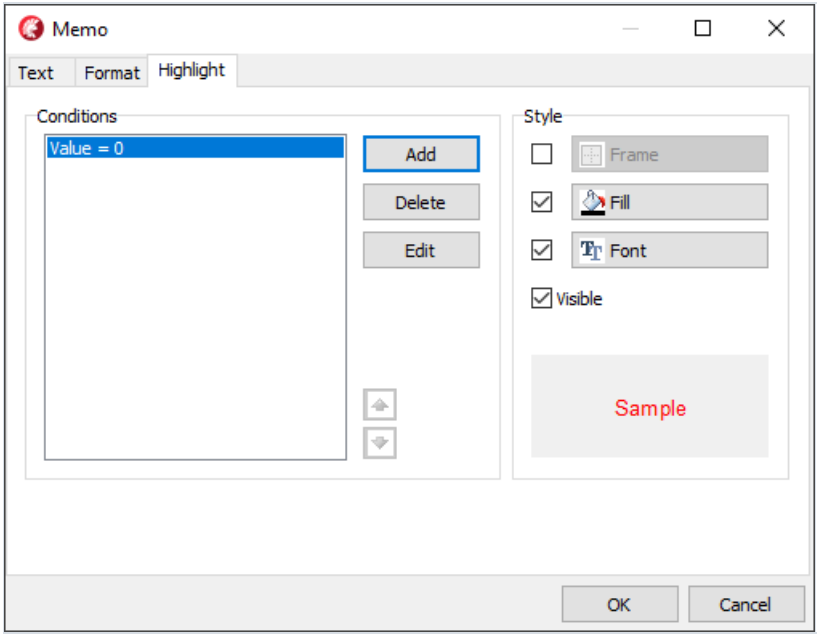
```
Value = 0
```

> Attention: if you have selected C++Script as a script language (see more details in the "Script" chapter), you should write the condition using C++Script:
>
> Value == 0

In the given case, we used the `Value` variable, which has got a printed value. If there are several expressions in an object, then this variable will have the value of the last expression. Instead of `Value`, you can use a data column:

```
<Products."UnitsInStock"> = 0
```

Configure the style for the given condition in such a way that only fill can be used, and choose the red color:



When printing an object which has a zero value, it will be red.

Let us make our example more complex, we will add another condition. If the units in stock is less than 10, it must be printed yellow. To do this, open the condition editor and click the "Add" button. The second condition will be like this:

```
Value < 10
```

In case where several conditions have been indicated, FastReport checks all the conditions, starting from the first one. If a certain condition is met, FastReport applies its style settings to the object, and the process stops.

It is important to put the conditions in a correct order. The order which we have seen in this example is correct:

1. `Value = 0`
2. `Value < 10`

If we swap conditions, then the highlighting will work wrongly.

1. `Value < 10`
2. `Value = 0`

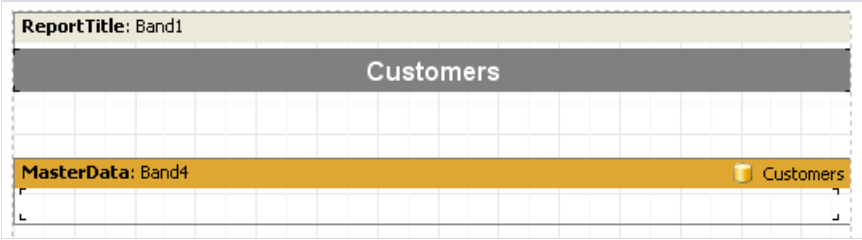
In the given case, the `Value = 0` will not be executed, because when the value is zero, then the first condition will be met.

> In order to change the order of the conditions, use the and buttons.

Coloring alternate data rows

Using conditional highlighting, it is easy to create reports having a "banded" look, where data lines are alternately colored. To save some effort, let's use the “Customer List” example that we designed previously.

Remove all the “Text” objects from the “MasterData” band. Place a “Text” object on the data band and stretch it to cover almost all of the band space:

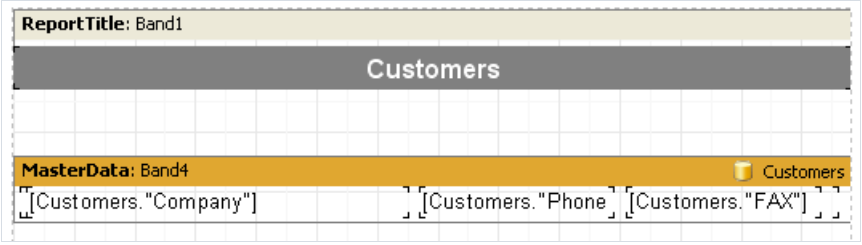


This object will change its color dependent on the data line number. Select the object and set the following conditional expression in the Highlight tab of the object editor:

```
<Line> mod 2 = 1
```

> Note: if you have selected C++Script as the script language (see more details in the "Script" chapter), you should write the condition using C++Script as:
>
> ```
> <Line> % 2 == 1
> ```

Select a gray as the color for highlighting, not too saturated a color, but closer to white. Now the other objects can be added to the data-band on top of the first empty “Text” object:



On preview the report produces this output:



Customers		
Action Club	813-870-0239	813-870-0282
Action Diver Supply	22-44-500211	22-44-500596
Adventure Undersea	011-34-09054	011-34-09064
American SCUBA Supply	213-654-0092	213-654-0095
Aquatic Drama	613-442-7654	613-442-7678
Blue Glass Happiness	213-555-1984	213-555-1995
Blue Jack Aqua Center	401-609-7623	401-609-9403
Blue Sports	610-772-6704	610-772-6898

Nested reports (subreports)

Sometimes very complex report are needed which contain blocks of additional data inserted at particular points in the design. Although many of these reports can be designed using an arrangement of FastReport bands, sometimes it just becomes too complicated. In these circumstances it is necessary to use the “Subreport” object .

When a “Subreport” object is inserted into a design FastReport automatically adds a new page, which is connected to the “Subreport”. Such a nested report resembles a multi-page report in terms of design structure. The only difference is that the nested report is displayed in a specific location on the basic design page, and not after it.

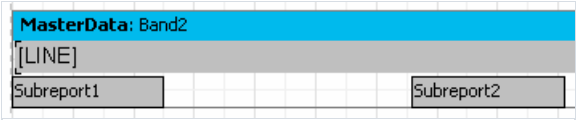
When this report is output, as soon as the “Subreport” object is encountered the report engine outputs the connected subreport page in its entirety. After that output continues with the rest of the basic design page.

Further “Subreport” objects can be inserted into a subreport design page, so increasing the depth of nesting. An example of a nested report can be found in the demo program - the “Subreports” report.

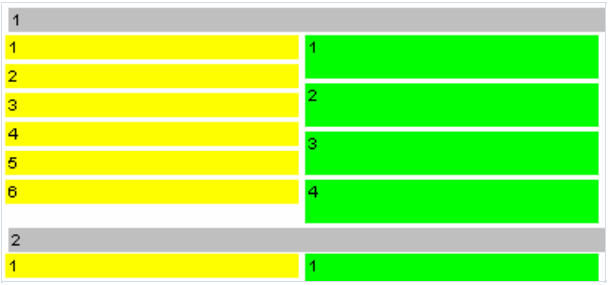
> It is worth noting that FastReport’s ability to use subreports enables deep nesting of data. Remember that the number of data levels in FastReport is limited to only six when Detail data bands are used instead of the “Subreport” object.

Side-by-side subreports

Two or more "Subreport" objects can be placed side-by-side on the same data band:



This design allows reports where the data output of each "Subreport" has varying lengths of rows/records, or varying heights or degree of stretching :



As illustrated, FastReport continues to output the basic design page only after the longest Subreport has finished. The Vertical Alignment property can also be used to adjust "Text" object alignment within each subreport.

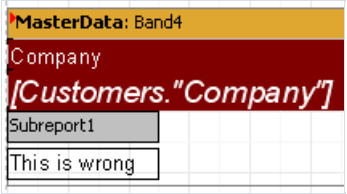
Limitations on use of subreports

Since subreports are placed on the basic design page, they cannot contain any of the following bands:

- ReportTitle
- ReportFooter
- PageTitle
- PageFooter
- PageBackground
- ColumnTitle
- ColumnFooter.

> If any of these bands are placed on a nested report page they will not be recognised. For the same reason there is no point in changing any nested report page options, as the options of the basic report page override those of any nested pages.

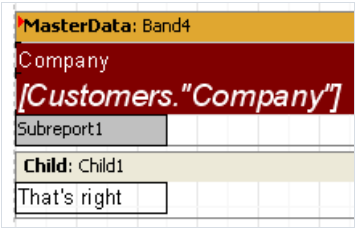
Do not place objects below the "Subreport" object:



If this is done then the objects created in the subreport will overlay everything placed below the subreport object on the main design page, and the output will be something like this:



To display objects below or after a nested report, use a child band:



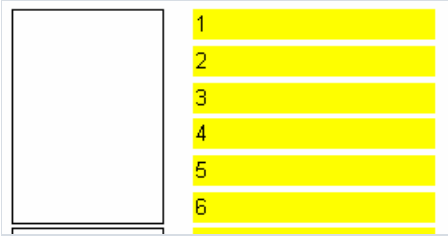
This method is also used when several Subreports must be placed one below the other. Use a child band for each Subreport and chain them together, setting the `Child` property of Child1 to Child2, and so on.

PrintOnParent option

The "Subreport" object has a `PrintOnParent` property which can sometimes be useful. This property is False by default.

Usually a subreport is output as a set of bands on the basic report page. When this is so, the height of the parent band containing the "Subreport" object is not controlled by the bands in the subreport, i.e. it cannot be stretched.

If the subreport `PrintOnParent` property is set to True, either in the object inspector or the context menu, the objects in the subreport are physically printed on the band which contains the "Subreport" object. This band can be made to stretch and can have stretched objects placed on it:



Script

A script is a program, written in a high-level language, which is part of a report. As the report runs, the script runs as well. A script can handle data in ways that are not possible just using the normal operations of the FastReport core; for example, a script can hide redundant data depending on a predefined condition. A script can also be used for controlling the properties of dialogue forms which are part of a report.

A script is written in one of the languages supported by the script engine (FastScript). Currently, these are:

- PascalScript
- C++Script
- BasicScript
- JScript

The following features are supported by the FastScript engine:

- standard language set : variables, constants, procedures, functions (which may be nested and having variables, constants, default parameters), all standard operators (including case, try, finally, except, with), types (integral, fractional, logical, character, line, multi-dimensional arrays, variant), classes (with methods, events, properties, indexes and default properties)
- type compatibility checking
- access to any of the report's objects

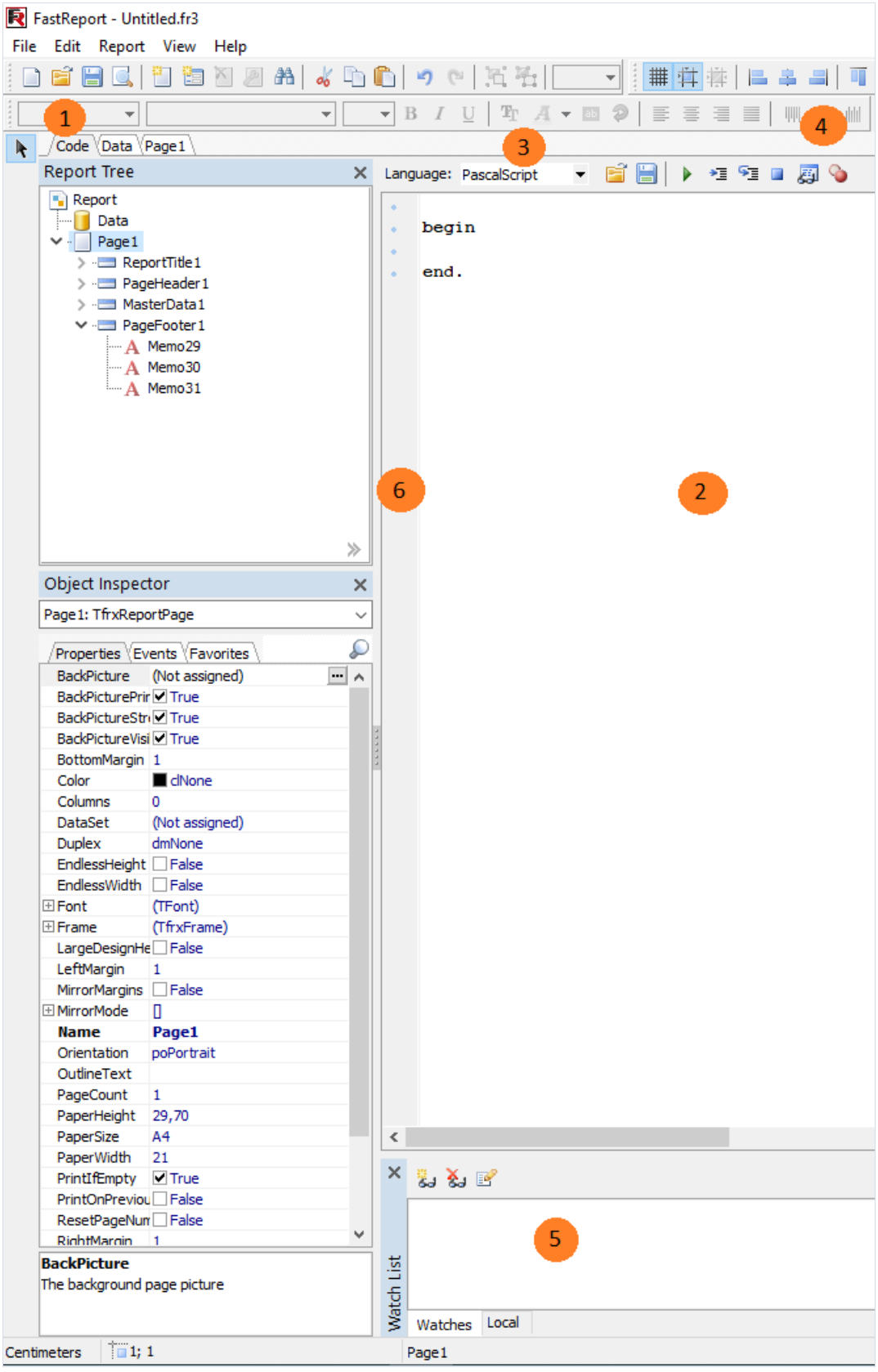
FastScript, however does not support the following:

- declarations of these types : records, classes
- pointers, sets (but the 'IN' operator can be used in expressions such as "a in ['a'..'c','d']")
- shortstring type
- unconditional jumps (GOTO)

Scripts can be created in the FastReport designer, which contains a script editor with syntax highlighting. There is also an embedded debugger which has the following functions: "Step", "Breakpoint", "Run to cursor" and "Evaluate".

A Taste of Script

The tools for working with scripts are located in the “Code” tab of the FastReport designer. When switching to this tab, the IDE looks like this:



Key to the labelling above:

1 – “Code” tab

2 – script editor pane

3 – drop-down list for selecting the language in which the script is written

4 – debugger toolbar

 - run report in debugging mode (F9)

 - run to cursor (F4)

 - execute the regular code line (Step into, F7)

 - interrupt running script (Ctrl+F2)

 - preview value of expression (Evaluate, Ctrl+F7)

 - toggle breakpoint (F5)

5 - “Watches” pane

6 – bookmarks and breakpoints are displayed in this column; also, line of executing code highlighted here

List of the shortcut keys which can be used in the script editor:

Key	Meaning
Cursor arrows	move cursor
PageUp, PageDown	go to previous/next page
Ctrl+PageUp	go to beginning of the text
Ctrl+PageDown	go to end of the text
Home	go to beginning of the line
End	go to end of the line
Enter	go to next line
Delete	delete symbol at cursor position; delete selected text
Backspace	delete symbol to the left of the cursor
Ctrl+Y	delete current line
Ctrl+Z	undo last action (up to 32 events)
Shift+Cursor arrows	select a text block
Ctrl+A	select whole text
Ctrl+U	shift selected block by 2 symbols to the left

Key	Meaning
Ctrl+I	shift selected block by 2 symbols to the right
Ctrl+C, Ctrl+Insert	copy selected block to the clipboard
Ctrl+V, Shift+Insert	paste text from the clipboard
Ctrl+X, Shift+Delete	cut selected block to the clipboard
Ctrl+Shift+<number>	set bookmark with <number> 0..9 on the current line
Ctrl+<number>	jump to bookmark <number>
Ctrl+F	search a line
Ctrl+R	replace a line
F3	repeated search/replacement from the cursor position
F4	set breakpoint for script to run to (Run to cursor)
Ctrl+F2	reset the script
Ctrl+F7	preview value of variable (Evaluate)
F9	run the script (Run)
F7 or F8	execute code line (Step into)
Ctrl + Space	show list of methods and properties of object
Ctrl+Shift+Delete	delete word to right of cursor
Ctrl+Shift+Backspace	delete word to left of cursor

Structure of a script

The structure of a script depends on the language used; however there are some elements common to each language. These are the script's title and body, and the main procedure which will be executed when the report runs. Below are examples of scripts in all four of the supported languages:

PascalScript 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.
```

C++Script 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 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++;
```

BasicScript 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
```

A more detailed description of the FastScript engine can be found in its documentation. This information is not repeated here in this user manual:

- syntactic charts for each of the supported languages
- supported data types
- operations with classes, properties, methods and events
- nested functions
- enumerations and sets

Later we will look at examples of scripts written in the “PascalScript” and "C++Script" language. When a new report is created "PascalScript" language is selected by default.

"Hello, World!" script

We have already seen an example of a “Hello, World!” report; now let's see how to create a simple script which displays a window with the same greeting.

Open the designer and click on the “New report” button so that FastReport automatically creates a basic template. Switch to the “Code” tab and write the following script:

PascalScript:

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

C++ Script:

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

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



Let's explain some details. We created a script consisting of a single `begin..end` block. So our script has a very simple structure; it consists only of a main procedure (see the “Structure of a script” section above).

The main procedure is executed as soon as the report runs. In this case it displayed a greeting dialogue; the procedure ends right after the dialogue is closed. After the main procedure has finished running, the normal report construction begins.

Using objects in the script

Any report object can be accessed from a script. So, if there are for example the “Page1” page and a “Memo1” object, they can be used in the script, calling them by their names:

PascalScript:

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

C++Script:

```
Memo1.Color = clRed
```

The list of report objects accessible from the script is shown in the “Report tree” pane. What object properties are available to a script? The answer is simple: all those that are visible in the object inspector.

The object inspector also shows hints for each property at the bottom. Both panes (report tree and object inspector) are available while working with a script. To get detailed help on object properties and methods use the FastReport help file which is included in the distribution kit.

Here's a simple example. Place a “Text” object named “MyTextObject” and containing “Test” onto the report design page. Then write this script:

PascalScript:

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

C++Script:

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

Run the report and see that the object’s color is red.

Referencing the variables from the report's variables list

Any variable that is defined in the list of the report variables ("Report>Variables..." menu item) can be referenced in a script. The variable's name should be enclosed in angle brackets:

PascalScript:

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

C++ Script:

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

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

PascalScript:

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

C++ Script:

```
if (Get("my variable") == 10) { ... }
```

A variable's value is changed only via the `Set` procedure:

PascalScript:

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

C++ Script:

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

It is worth noting that to assign a string value to the variable you must add quotes around the value:

PascalScript:

```
Set('my variable', '''' + 'String' + '''');
```

C++ Script:

```
Set("my variable", "\"String\"");
```

System variables, such as `Page#`, should be referenced in exactly the same way:

PascalScript:

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

C++ Script:

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

Referencing the DB fields

Just as with variables, angle brackets should be used when referencing DB fields in a report:

PascalScript:

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

C++ Script:

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

Alternatively the `Get` function can be used for accessing DB fields (in fact, this function is used implicitly by FastReport when calculating expressions enclosed in angle brackets).

Using aggregate functions in the script

An idiosyncrasy of aggregate functions is that they must be used inside "Text" objects; once used in this manner they can then be used in the script itself.

If an aggregate function only appears in a script (without appearing in a "Text" object) an error message is generated. This happens because an aggregate function must be connected to a specific band; once so connected it will work correctly.

Displaying a variable’s value in a report

Variables can be declared and used locally within a script. Once declared a script variable can have a value assigned to it. Here is a simple example of a script variable in use:

PascalScript:

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

C++ Script:

```
string MyVariable;
{
  MyVariable = "Hello!";
}
```

The variable’s value can be displayed in a “Text” object, for example, by typing "[MyVariable]" into the object.

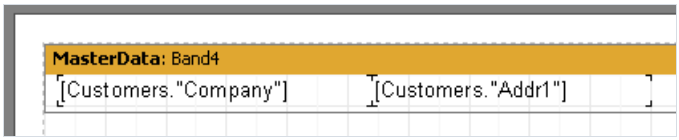
A variable’s name must be unique. This means the name must not duplicate the name of any other report object, standard function or constant. If there is an error in a script, a message will be displayed when the report is run and report construction will be stopped.

Events

So far we have looked at scripts with only a main procedure, which is executed when a report starts running. In the main procedure initial settings can be made and variables initialized. However this is not enough for total control over the process of report generation.

To enable as much control as possible over report generation every object has several events to which handlers (i.e. procedures in the script) can be assigned. For example, connecting a handler to the data band enables records to be filtered, such that the band can be hidden or revealed according to specific conditions being met.

Let's demonstrate the process of report creation and the events triggered by means of a simple report containing one page and having one “MasterData” band, with two “Text” objects on the band:



As described above, the script's main procedure is called at the very start of running the report. After that the essentials of report construction begin.

Firstly the “OnStartReport” event of the "Report" object is called. Then, before an output page is created, the “OnBeforePrint” page event is called. This event is called once for each design page in the report template (design pages should not be confused with the output pages of a report!). In our example the event is called once, as the report design consists of only one design page.

Then the events of the data bands are called in the following order:

1. the band's “OnBeforePrint” event is called
2. the “OnBeforePrint” event of each object contained in the band is called
3. each object is filled with data (in our example with values of the “Company” and “Addr1" DB fields)
4. the “OnAfterData” event of each object is called
5. actions such as positioning objects on the band (if there are stretchable objects among them), calculating band height and stretching it (if it is stretchable) are performed
6. the band's “OnAfterCalcHeight” event is called
7. a new output page is created if the band hasn't enough room in the page's white space
8. the band and all of its objects are displayed on the output page
9. the “OnAfterPrint” event of each band object is called
10. the “OnAfterPrint” event of the band itself is called

Bands continue to be printed as long as the source connected to the band has data. After that report printing stops, the report page's “OnAfterPrint” event is called and finally the “Report” object's “OnStopReport” event.

So by using the events of different objects practically every step of the report creation process can be managed. The key to using events is a thorough understanding of the band output process, which is discussed in the next nine sections.

Most of the actions can be performed using the band's “OnBeforePrint” event only; any modifications made to an

object are displayed immediately. However, if the band is stretchable, it is impossible to say in this event on which page the band will be printed, since calculation of the band's height is performed in step 5.

It can be done, however, either in the “OnAfterCalcHeight” event in step 6 or in the “OnAfterPrint” event in step 9. Note that in the last event the band will already have been output so modification to objects will not have any visible effect.

It is essential to clearly understand “where and when” the bands are output and to understand the timing (calling order) of each of their events. Likewise for each of the objects contained in the bands.

Example of using the “OnBeforePrint” event

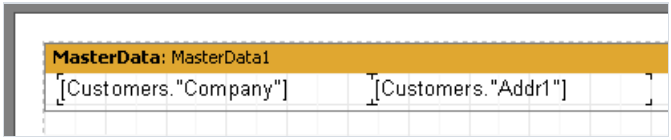
To demonstrate this event create a report representing a list of clients. This report will include only those companies whose name begins with the letter “A”.

Let's create a new project in Delphi, place `TTable` , `TfrxDBDataSet` and `TfrxReport` components on the form and set these properties:

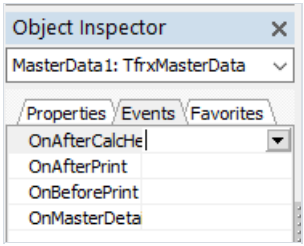
```
Table1:
DatabaseName = 'DBDEMOS'
TableName = 'customer.db'

frxDBDataSet1:
DataSet = Table1
UserName = 'Customers'
```

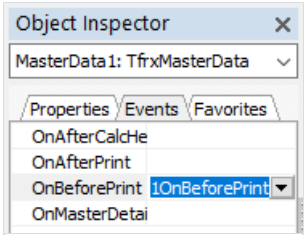
Open the report designer and create a report like this:



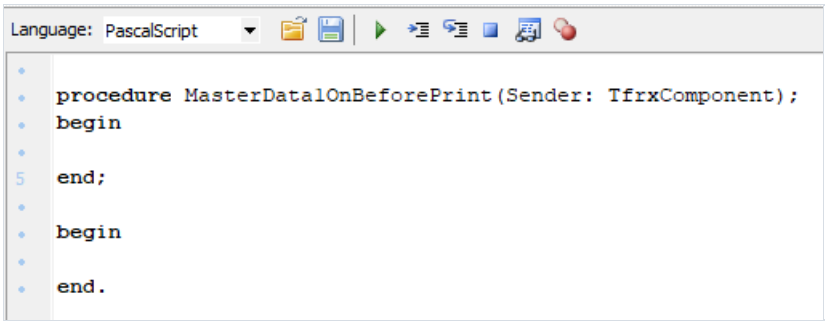
Select the data band and switch to the “Events” tab in the object inspector:



To create an “OnBeforePrint” event handler (which is the most appropriate for us) double-click on the blank field to the right of the event’s name:



This adds a blank handler to the script and the designer switches to the “Code” tab.



All that is needed now is to type the following code in the handler's body:

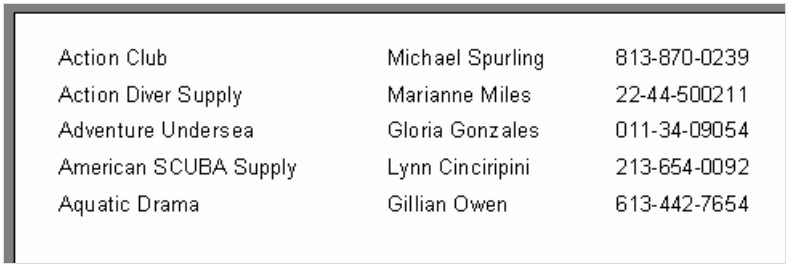
PascalScript:

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

C++Script:

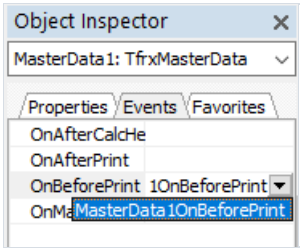
```
if (Copy(<Customers."Company">, 1, 1) == "A")
  MasterData1.Visible = true;
else
  MasterData1.Visible = false;
```

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's explain several things. One handler can be assigned to the events of more than one object - the `Sender` parameter shows which object has initiated the event. To assign an existing handler to an event, either type it directly into the object inspector, or select it from the drop-down list:



A link to a handler can easily be deleted - select the assigned handler in the object inspector and press the "Delete" key.

Printing a group sum in the group header

This method is used quite often and requires the use of a script because the value of a group sum is only known after all the records in the group have been processed. To display the sum in the group header (i.e. before the group is output to the report) the following algorithm is used:

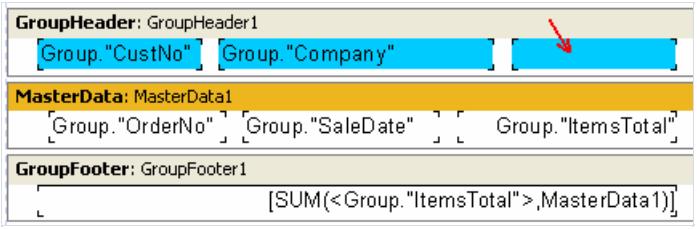
- turn on the two-pass report option (“Report > Options...” menu item)
- in the first pass, calculate the sums for each group and save them in an array
- in the second pass, extract the values from the array and display them in the group headers

Let's show two methods for performing this task. First create a new project in Delphi, place `TQuery`, `TfrxReport` and `TfrxDBDataSet` components on the form and set them up as follows:

```
Query1:
DatabaseName = 'DBDEMOS'
SQL =
  select * from customer, orders
  where orders.CustNo = customer.CustNo
  order by customer.CustNo, orders.OrderNo

frxDBDataSet1:
DataSet = Query1
UserName = 'Group'
```

Open the designer and connect the data source to the report. Enable double-pass in the report's settings (“Report > Options...” menu item). Add two bands to the report: “GroupHeader” and “MasterData”. In the “GroupHeader” band's editor, enter the Group.“CustNo” data field. Connect the data band to the “Group” data source and then arrange some objects in the following way:



To display the sum we use the arrowed object in the design (in our example it is named “Memo8”).

The first method.

We will use the `TStringList` class as an array for storing the sums - we will be storing the numeric values as strings. The first item in the StringList will correspond to the sum of the first group, as so on. An integer variable (which we will increment after printing each group) is used to calculate the group's index number.

So our script will look like this:

PascalScript:

```
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
  if not Engine.FinalPass then
    List.Add(FloatToStr(SUM(<Group."ItemsTotal">,MasterData1)));
  Inc(i);
end;

begin
end.
```

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++;
}

{
}
```

The procedure names in the script show which events we have used. They are: “Report.OnStartReport”, “Report.OnStopReport”, “Page1.OnBeforePrint”, “GroupHeader1.OnBeforePrint” and “GroupFooter1.OnBeforePrint”. The first two events are called at the beginning and the end of the report respectively.

> To create handlers for these two events select the “Report” object in the object inspector's drop-down list and its properties will appear in the object inspector. Then switch to the object inspector's “Events” tab and create the handlers.

Why didn't we create the “List” variable in the script's main procedure ? We created it in the “OnStartReport” event because dynamically created variables should be destroyed after the report has been finished. It is logical to create dynamic variables in the “OnStartReport” event and destroy them in the “OnStopReport” event. In other cases (when memory does not need to be freed on completion of the script) one can use the script's main procedure for initialization of variables.

The creation and destruction of the “List” variable is straight forward. Now let's see how the script works.

- At the start of the page the counter for the current group (the variable “i”) is reset to zero and it is incremented after each group has been printed (in the “GroupFooter1.OnBeforePrint” event).
- The calculated sum is added to “List” in this event before the counter is incremented.
- The “GroupHeader1.OnBeforePrint” event does nothing during the first pass (If “Engine.FinalPass” condition) but during the second pass (when “List” has been filled with values) the sum corresponding to the current group is retrieved from “List” and is output to the “Memo8” object to display the sum in the group header.

In the finished report, it appears as follows:

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

This algorithm is quite straight forward. However, it can be simplified.

The second method.

We will use the collection of report variables as an array for storing the group sums. Remember that report variables are accessed via the `Get` and `Set` functions. Using these functions also saves us from having to explicitly create and destroy these variables. Our script will look as follows:

PascalScript:

```
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.
```

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)));
}

{
}
```

As you can see, this script is somewhat simpler. Code in the “GroupFooter1.OnBeforePrint” handler sets the value of a variable having a name derived from the client number (or any other identifier which unambiguously identifies the client could be used, for example <Group."Company">).

If there isn't a variable with that name already existing then the script automatically creates it; otherwise if it does exist then its value is updated. In the “GroupHeader1.OnBeforePrint” handler the value of the appropriate variable

is retrieved.

“OnAfterData” event

This event is triggered after a report object has been filled with the data from the source to which it is connected. Use this event for accessing either a DB field value or an expression contained in the object. This value is placed in the `Value` system variable which is available only in this event.

So if two “Text” objects contain the expressions `[Table1.“Field1”]` and `[<Table2.“Field1”> + 10]` the values of these expressions can be used by referring to the `Value` variable for the objects:

PascalScript:

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

C++ Script:

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

which is simpler than writing something like this:

PascalScript:

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

C++ Script:

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

Using `Value` instead of an expression enables you to write one multi-purpose handler for the “OnAfterData” event which can be connected to several objects.

> Please note something else - if an object contains several expressions (for example `[expr1] [expr2]`) it is the value of the last expression that is transferred to the `Value` variable.

The “OnAfterData” event is ideal for calculating the height and width of objects such as “Text”. That is, if the exact height of a stretched “Text” object containing an expression is needed in a script you can use this code in the “OnAfterData” event:

PascalScript:

```
var
  MemoHeight: Extended;
begin
  MemoHeight := TfrxMemoView(Sender).CalcHeight;
end;
```

C++ Script:

```
float MemoHeight;
  MemoHeight = TfrxMemoView(Sender).CalcHeight;
```

If this code were used in the "OnBeforePrint" event the result will be the height of the object containing the expression before the expression is evaluated, and not its actual value on printing.

Service objects

There are some other system objects which can be used in scripts, in addition to the normal report objects like pages, bands, "Text" and other objects. They may be useful in the management of report construction. The `Engine` object, used in the previous chapter, is one such object and all of them are listed here:

- `Report` : the Report object
- `Engine` : the link to the report "Engine"
- `Outline` : the link to the "Report tree" element in a previewed report

Let's look at each of these objects.

“Report” object

This object represents a link to the current report. The properties 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 value of `Expr`, which is an expression; for example, `Report.Calc('1+2')` returns “3”. Any valid FastReport expression can be passed in the parameter
`function GetDataSet(const Alias: String): TfrxDataSet`	returns the dataset having the specified name. The dataset must appear in the report's dataset list (“Report>Data...” dialogue)

“Engine” object

This is the most useful and interesting object, it represents a link to the engine (FastReport’s core, which manages report construction). By using the engine’s properties and methods really exotic report design layouts can be created

The methods and properties of this object.

Property	Type	Description
`CurColumn`	Integer	the index of the current column in a multi-columned report; a value can be assigned to this property.
`CurX`	Extended	the current print position on the X-axis; a value can be assigned to this property.
`CurY`	Extended	the current print position on the Y-axis; a value can be assigned to this property.
`DoublePass`	Boolean	equals “True” if the report is a two-pass one (analogous to Report.EngineOptions.DoublePass).
`FinalPass`	Boolean	equals “True” when in last pass of a two-pass report
`PageHeight`	Extended	printable region’s height, in pixels
`PageWidth`	Extended	printable region’s width, in pixels
`StartDate`	TDateTime	starting time of report; the same as the `<Date>` system variable
`StartTime`	TDateTime	starting time of report; the same as the `<Time>` system variable
`TotalPages`	Integer	the number of pages in a report (the same as the `<TotalPages>` system variable). The report should be a two-pass one, if this variable is used
`SecondScriptcall`	Boolean	flag returning 'repeat-call' status of an event (in some cases an event can be called repeatedly during grouping). If True then the script has already been called

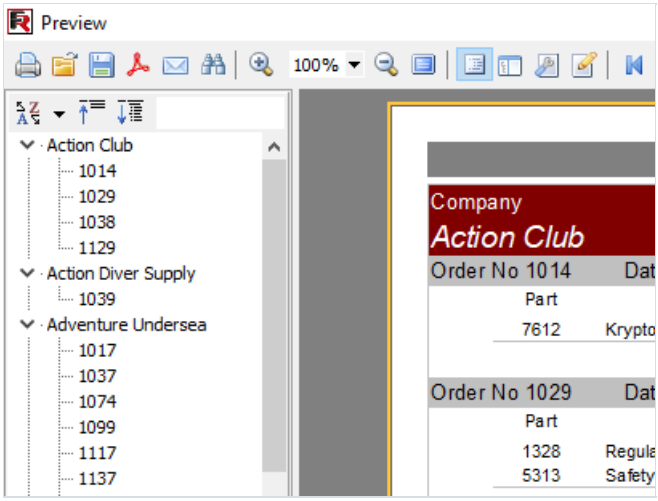
Methods:

Method	Description
`procedure AddAnchor(const Text: String)`	adds “Text” to the list of anchors (see more below)
`procedure NewColumn`	creates a new column in a multi-column report (after the last column a 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 the band the `CurY` position is automatically incremented)

Method	Description
`function FreeSpace: Extended`	returns height of the white space left on page, in pixels.
`function GetAnchorPage(const Text: String): Integer`	returns the page number where the specified anchor has been placed

“Outline” object

This object represents the "Report tree" control element in a previewed report.



The Outline displays a tree-like structure of the finished report. When any tree node is clicked the preview pane jumps to the page displaying this node. To display the Outline it should be enabled either by clicking the  button in the toolbar of the preview window or by setting the `Report.PreviewOptions.OutlineVisible` property to True. The Outline's width in pixels can be set there as well : `Report.PreviewOptions.OutlineWidth`.

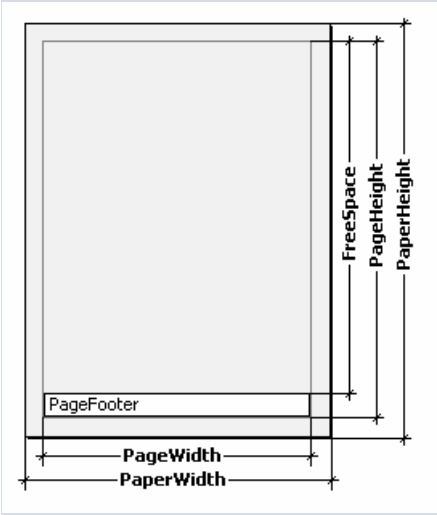
The `Outline` methods are:

Method	Description
`procedure AddItem(const Text: String)`	adds an element having “Text” name at the current tree position. The current report page and position on the page are linked to the element
`procedure LevelRoot`	moves the current position in the tree to the root level
`procedure LevelUp`	moves the current position in the tree up one level

Using the “Engine” object

We have already said that the “Engine” object represents the report's engine, which manages report construction. The process of arranging band(s) on a page can be managed by using the engine's properties and methods. First some theory.

The diagram below shows various dimensions of the report page.



The physical dimensions of the page are the `PaperWidth` and `PaperHeight` properties, visible in the object inspector when the page is selected. So the size of an A4 page is 210 x 297mm.

`PageWidth` and `PageHeight` are the dimensions of the printable region, which is usually less than the physical dimensions of the page. The size of the printable region is dependent on the report page properties `LeftMargin`, `TopMargin`, `RightMargin` and `BottomMargin`. The printable region’s size in pixels is returned by the `Engine.PageWidth` and `Engine.PageHeight` functions.

Finally, `FreeSpace` is the height of the free space on a page. If there is a "Page Footer" band on the page, its height is taken into account when calculating the `FreeSpace`. This height is returned in pixels by the `Engine.FreeSpace` function.

> Note that after displaying the next band the free space is reduced on the page, this is taken into account when calculating the `FreeSpace`.

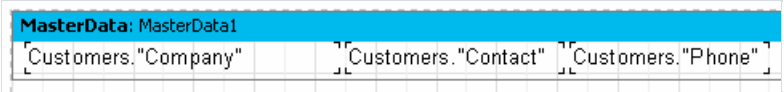
How are report pages constructed? The FastReport core displays bands on a page as long as there is enough free space. When there is no more free space left the “Page Footer" band is printed (if required) and a new blank page is created.

As already said, after displaying the next band the height of free space is reduced. Moreover, display of the next band begins from the current position, which is defined by coordinates on the X-axis and the Y-axis.

> The current position is returned by `Engine.CurX` and `Engine.CurY` respectively.

After printing the next band, `CurY` automatically increases by the height of the printed band. After a new page is created `CurY` equals 0. `CurX` is changed when printing multi-column reports.

`Engine.CurX` and `Engine.CurY` are available not only for reading but also for writing. This means that bands can be shifted by incrementing or decrementing these values. For example, in a report resembling this:



it can be printed in the following way:



This is achieved by writing a handler for the band's "OnBeforePrint" event:

PascalScript:

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

C++ Script:

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

Changing `CurY` can make bands overlap, for example:



Achieved by this script:

PascalScript:

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

C++ Script:

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

The `Engine.NewPage` method inserts a page break at any required point in a report, following which printing continues from the top of the new output page. In our example a break can be inserted after printing the second record:

PascalScript:

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

C++ Script:

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

Note that we used the "OnAfterPrint" event (that is to say, after the band has been printed). Also note that the `Line` system variable returns the sequential number of the record.

The `Engine.NewColumn` method inserts a column break in multi-columned reports. If there are no more free columns left on the page then a new page is created.

Anchors

An Anchor is one of the elements in the hyperlink system which enables jumping to any element connected to the finished report's object by clicking on it (in the preview window).

Anchor can be set via the `Engine.AddAnchor` method. Anchor has a name and a position within a report page. To jump to an anchor with a specified name, type the following line into the URL property of any report object:

```
#AnchorName
```

or

```
#[AnchorName]
```

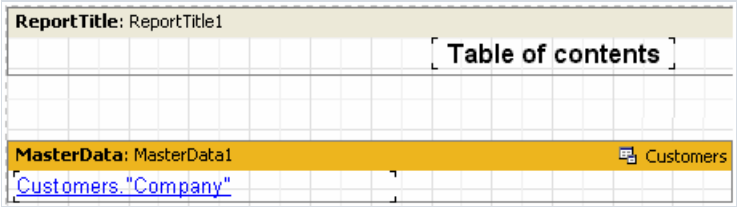
In the second case FastReport substitutes a value for the expression.

Clicking on the object executes a jump to that part of the report where the anchor was added.

Use anchors when constructing a "Contents" table, for example with links to corresponding chapters. Let's illustrate this in the following example. First we need the familiar "Customer" table.

Our report will be a multi-page one (with two design pages). We will place the "Contents" table on the first page and the list of clients on the second page. Clicking on any Content line executes a jump to the corresponding report element.

The first design page:

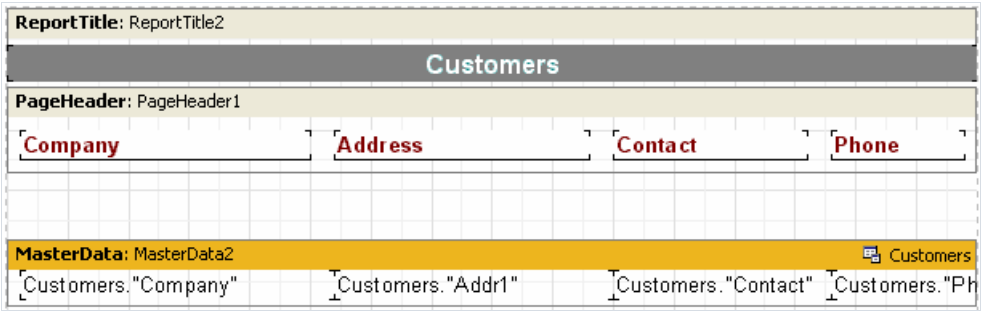


Place the following text in the URL property of the "Text" object contained in the data-band

```
#[Customers."Company"]
```

and set the font properties to blue and underlined, to simulate the look of a hyperlink.

The second design page:



To add an anchor, create a "MasterData2.OnBeforePrint" event handler in the script:

PascalScript:

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

C++ Script:

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

That is all that is needed. Preview the report and make sure that the “hyperlinks” work.

The last thing to be mentioned is the `Engine.GetAnchorPage` function. This function returns the number of the page on which the corresponding anchor was added and is useful when creating the “Contents” table. The report must be a two-pass one, otherwise this function cannot be used.

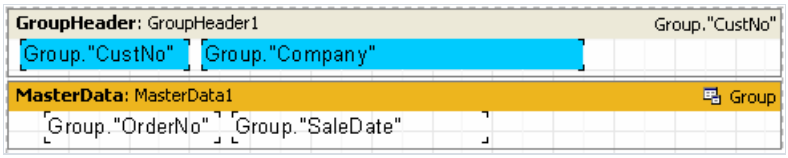
Using the “Outline” object

The "Outline" object, as previously stated, represents a report tree which can be displayed in a preview window. Clicking on an element in the tree executes a jump to the report’s output page that contains the corresponding element.

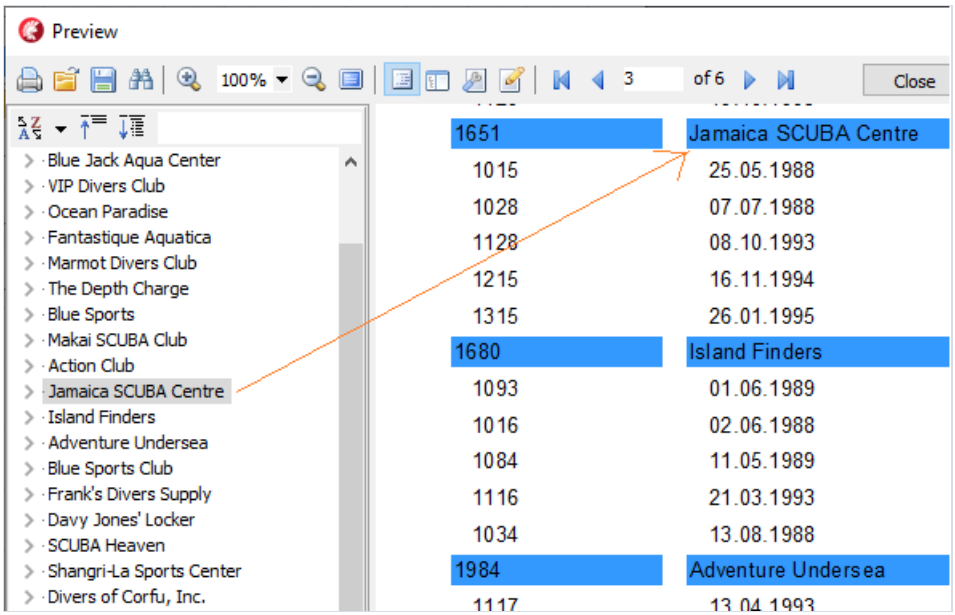
It is not necessary to use a script to work with an Outline as some bands can automatically create a tree. Let's look at two examples showing how an Outline can be used with the help of bands and a script.

Almost all bands have the `OutlineText` property to contain a text expression which automatically creates the tree. The expression is evaluated when creating a report and its value is added to the tree when the band is printed. So the hierarchy of elements in the tree is similar to the hierarchy of bands in the report, meaning that the tree will have main and subordinate elements which correspond to the main and subordinate bands in the report (for example where the report has two levels of data or has groups).

We will use our previous example to show the operation of a tree in a report having groups.

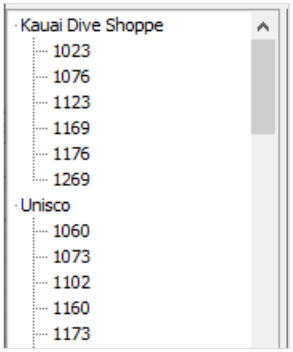


Set the value of the `GroupHeader1.OutlineText` band property to <Group."Company">. To make the tree visible as soon as the preview window opens set the `PreviewOptions.OutlineVisible` property of the “Report” object to True. On previewing the report you will now see the following:



Clicking on any element in the tree executes a jump to the corresponding page in the report with the selected element at the top of the window.

Let's add the second level to the report tree. Just set the `OutlineText` property of the “MasterData” band to <Group.“OrderNo”> and the tree will change to this:



It is evident that navigation right down to order numbers is possible, and that the hierarchy of the elements in the tree resembles that in the report.

Now we will create a similar tree, but using a script instead of the `OutlineText` property. In the report clear the `OutlineText` properties of both of the bands and create two event handlers: "GroupHeader1.OnBeforePrint" and "MasterData1.OnBeforePrint":

PascalScript:

```
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.
```

C++ Script:

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

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

{
}
```

Preview the report to make sure that it works in the same way as previously, where the tree was created automatically. Let's see how the tree is created by the script.

The `Outline.AddItem` method adds a child node to the current tree node and then makes the child node the current one. So if `AddItem` were called several times in a row it would create the "ladder" shown here:

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

The `LevelUp` and `LevelRoot` Outline methods are used to control which element is the current one. The first method moves the cursor to the element located one level up. So this script:

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

constructs a tree like this:

```
Item1
  Item2
    Item3
    Item4
```

This shows that "Item4" is a child element of the "Item2" element. The `LevelRoot` method on the other hand moves the cursor up 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 this tree:

```
Item1
  Item2
    Item3
Item4
```

Knowing this it is clear how the report works.

- Before every group title (company name) is output the root of the tree is made the current element.
- After that, the list of orders is output, each order being added as a child element of the company.
- To ensure that all the orders are located on one level and not displayed as a "ladder", the `Outline.LevelUp` method is called after each order addition to shift the cursor back to the company level.

“OnManualBuild” page event

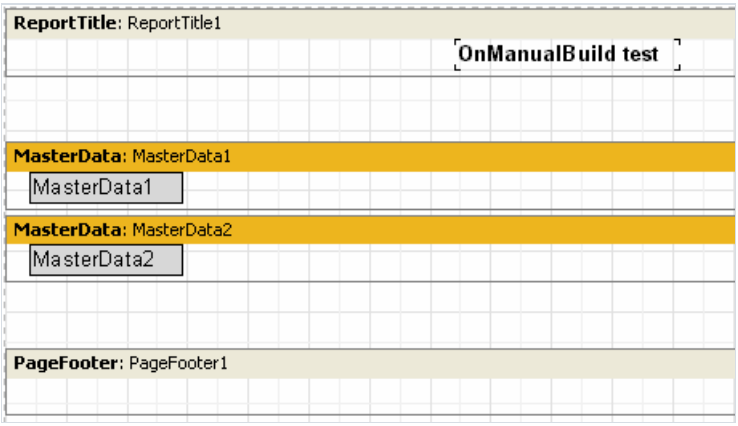
The FastReport core is usually responsible for report construction. It displays the report bands in a specific order, as many times as required by the data, thus creating a complete report.

Sometimes it is necessary to display a report in a non-standard form which the FastReport core is unable to accomplish. In this case it is possible to construct a report manually using the “OnManualBuild” event of the report's design page. If the handler for this event is defined then the FastReport core transfers control to it when data output is required.

At the same time the FastReport core automatically handles the display of those bands which are located on the page, such as “Report title”, “Page title”, “Column title”, “Report footer”, “Page footer”, “Column footer” and “Background”. The core also handles the creation of new pages and columns. The purpose of the “OnManualBuild” event handler is to display data bands, their titles and their footers in a user controlled order.

That is to say the essence of the “OnManualBuild” handler is to give commands to the FastReport core for displaying bands at particular times. The core does the rest itself : it creates new pages as soon as there is no free space on the current one, handles scripts attached to events, etc.

Let's demonstrate a handler using a simple example. This report has two master data bands which are not connected to data:



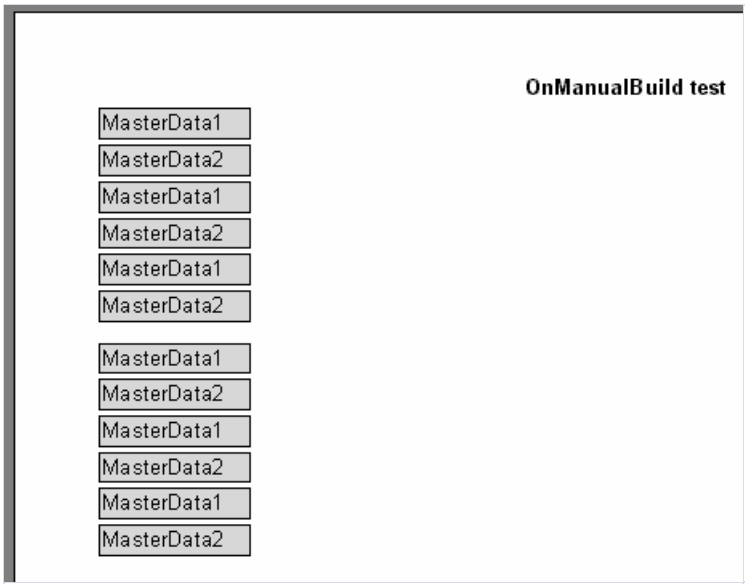
The handler will display these bands in alternate order (six times for each one). After six bands have been displayed a small gap will be inserted.

PascalScript:

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

C++ Script:

```
void Page1OnManualBuild(TfrxComponent Sender)
{
  int i;
  for (i = 1; i <= 6; i++)
  {
    // show two bands
    Engine.ShowBand(MasterData1);
    Engine.ShowBand(MasterData2);
    // make a gap
    if (i == 3)
      Engine.CurY = Engine.CurY + 10;
  }
}
```



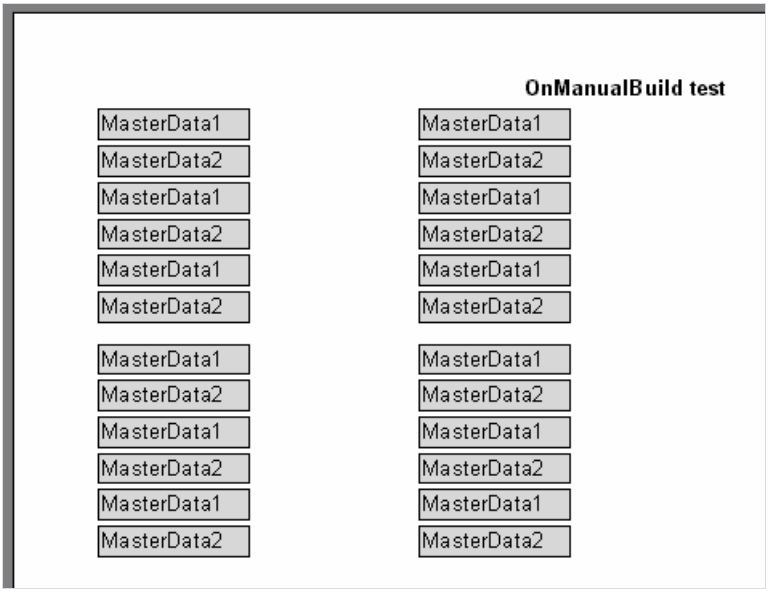
The following example displays the same bands, with a second copy shifted to the right.

PascalScript:

```
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;
```

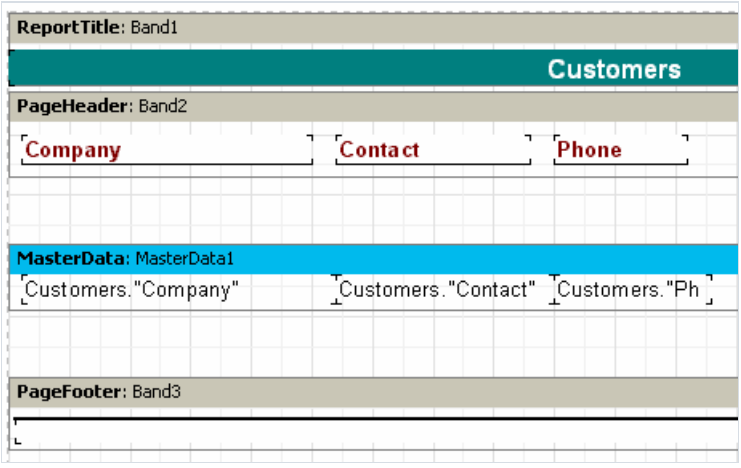
C++Script:

```
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;
  }
}
```



As you can see, in these examples we controlled only the output of data bands. The rest of the bands (in our case the “Report title”) were output automatically.

Finally we will show how to construct a report with a “List of clients” (we have shown several versions before) using the “OnManualBuild” event. This time connect the data band to the data source.



And use this script:

PascalScript:

```
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;
```

C++Script:

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

Preview the report to make sure that the script produces a report identical to the standard report. Note how we got a link to the Dataset - we connected a dataset variable to the data source using this code:

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

If the MasterData band is not connected to a data source then the link to the required data source can be made in the following way:

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

Of course, the data source we are interested in must be enabled in the menu “Report > Data...” dialogue.

Creation of objects in the script

New objects can be added to a report using a script. Let's show how this is done with a simple example. Create a blank report and enter this code in the script's main procedure:

PascalScript:

```
var
  Band: TfrxReportTitle;
  Memo: TfrxMemoView;
begin
  Band := TfrxReportTitle.Create(Page1);
  Band.Height := 20;
  Memo := TfrxMemoView.Create(Band);
  Memo.SetBounds(10, 0, 100, 20);
  Memo.Text := 'This memo is created in code';
end.
```

C++ Script:

```
TfrxReportTitle Band;
TfrxMemoView Memo;
{
  Band = TfrxReportTitle.Create(Page1);
  Band.Height = 20;
  Memo = TfrxMemoView.Create(Band);
  Memo.SetBounds(10, 0, 100, 20);
  Memo.Text = "This memo is created in code";
}
```

Preview the report:



> Note that we did not destroy the FastReport objects we created in this example. This is not required as FastReport objects are automatically destroyed by the Delphi application after the report is completed.

> Also note that when we create standard Delphi objects in a script (such as `TStringLists`) we also have to destroy them in the script, as this will not be done automatically by the application.

Cross-tab reports

This kind of report has a tabular structure, which means that it consists of rows and columns. At design time it is not known how many lines and columns the output table will have. This is why a report grows not only downwards (as in the types of report previously described) but also sideways. A typical example of a cross-tab report is shown below.

Let's look at the elements in the table:

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

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

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

You can see that the first field contains a line letter, the second a column number and the third the contents of the cell at the intersection of the specified line and column. When outputting the report FastReport creates a table in memory and fills it with data. So the table expands dynamically, creating lines and columns where they do not already exist.

Titles can occur at more than one level, as illustrated here:

	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 is generated from 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
```

Here the first field contains the line index, as before, the second and third fields contain column indexes and the last field contains the cell value. Look at how FastReport constructs the memory table when handling cross-tab data

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

When outputting the report from this memory table FastReport joins those title cells which have the same value and are located at the same level.

Here is a more complex cross-tab report, incorporating intermediate and grand 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 derived from the same data as before. The values in the cells highlighted in the new color are automatically calculated and are not present in the original data set.

Constructing a cross-tab report

Now let's turn from theory to practice. We will construct a simple cross-tab report displaying employees' salaries over four years. To do this we need the "crosstest" table, which is located in the FastReport "DEMOS\MAIN" folder. The table contains data like:

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

Create a new project in Delphi, place `TTable`, `TfrxDBDataSet` and `TfrxReport` components on the form and set their properties:

```
Table1:
DatabaseName = 'c:\Program Files\FastReport 4\Demos\Main'
TableName = 'crosstest.db'
// the DatabaseName property must correspond
// to the path of your FastReport installation folder!

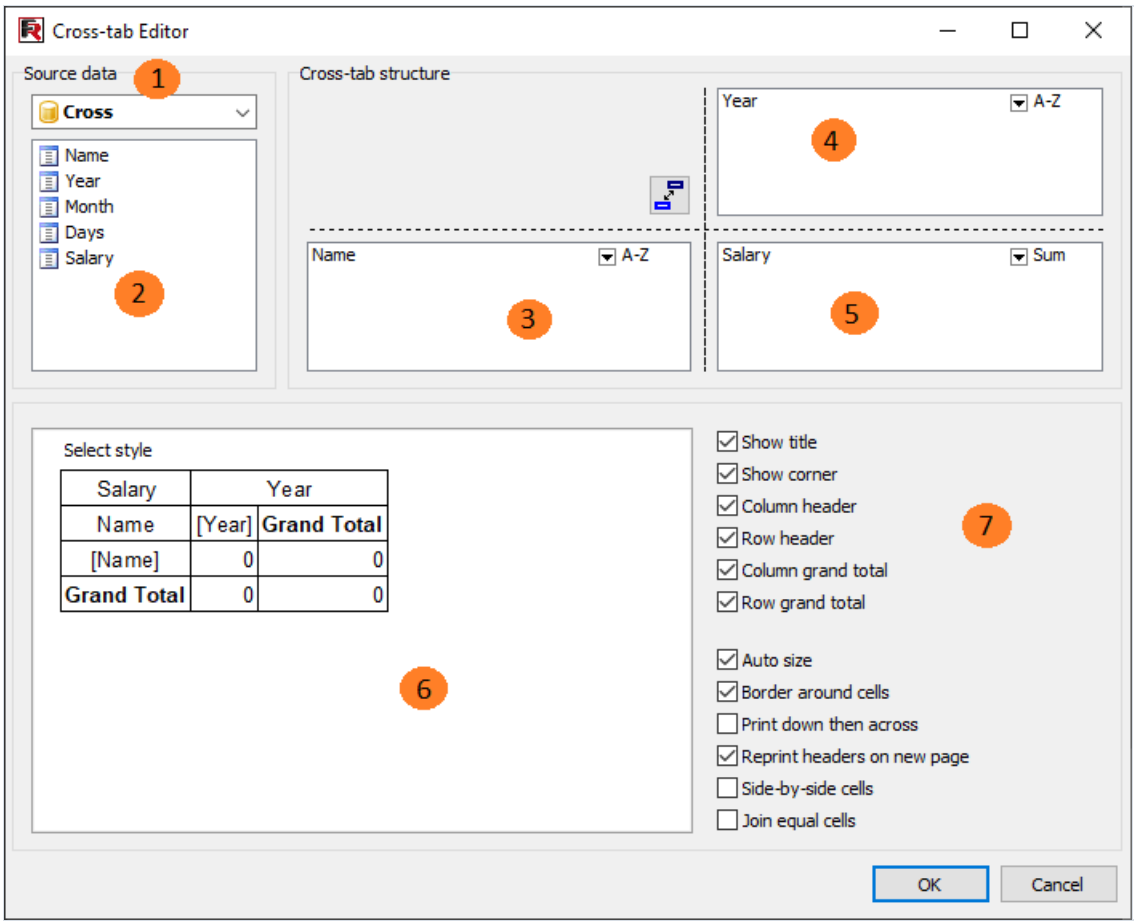
frxDBDataSet1:
DataSet = Table1
UserName = 'SimpleCross'
```

The `TfrxCrossObject` component  from the FastReport component palette is used to design a cross-tab report. Just place it on the Delphi form, it does not need any properties to be changed. The "frxCross" unit, containing the cross-tab functionality, is added to the "uses" list when the Delphi application is compiled.

Open the report designer. Firstly connect the data source using the "Report > Data..." menu item. Then select the "DB cross-tab" object  from the designer's object toolbar and click on the design page to place the object there:



All settings are made using the cross-tab editor. Open it by double-clicking on the object:



Key to the items shown above:

1 – a drop-down list of available data sources

2 – the list of fields in the selected data source; the fields from this list can be dragged to the lists numbered 3, 4 or 5

3 – the list of fields which generate line (row) headers

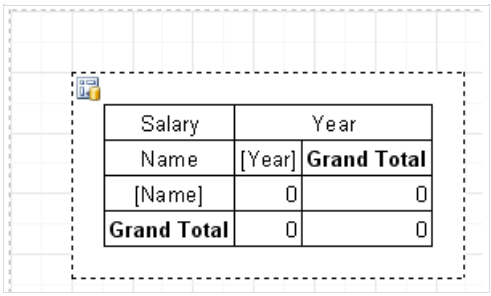
4 – the list of fields which generate column headers

5 – the list of fields which generate table cells

6 – table structure preview

7 – structure options : display of titles, totals, etc

You can only use the mouse in this editor to make changes. For our example it is only necessary to drag fields from the list 2 to lists 3, 4 and 5 (in the diagram above). After that close the editor by clicking the OK button. The cross-tab object now shows its structure:

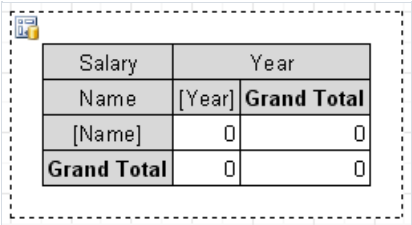


When the report is previewed you will see a table resembling this:

Salary	Year				
Name	1999	2000	2001	2002	**Grand Total**
Ann	3300	2700	3100	1700	10800
Ben	3900	2100		1800	7800
Catherine	6100	3200			9300
Den		3999	8100		12099
Grand Total	13300	11999	11200	3500	39999

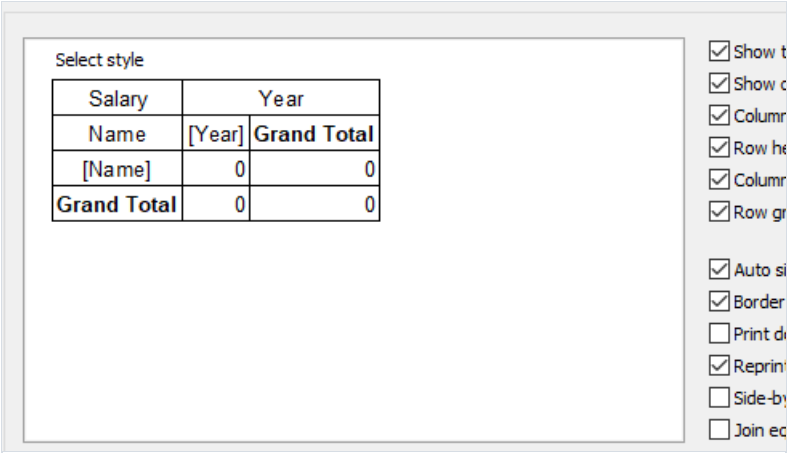
Changing the appearance

Let's modify the cross-tab object's appearance. The first thing we want to do is to change the title colors and display "Total" instead of "Grand total". This is very easy - to change the title color to gray click on the "Year", "Name" and "Grand Total" cells in turn and select gray using the background button on the toolbar.

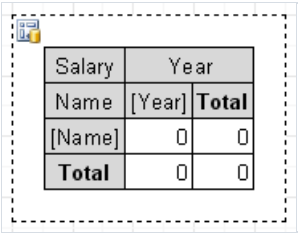


Salary	Year	
Name	[Year]	**Grand Total**
[Name]	0	0
Grand Total	0	0

We can also use a set of predefined styles. These are available in the cross-tab editor - click "Select style" and choose one.

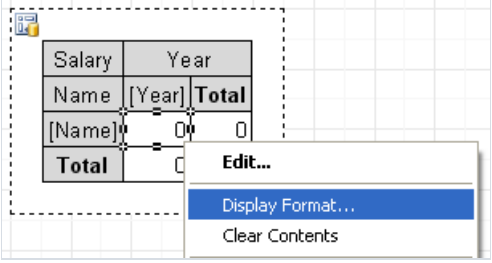


To change the two "Grand Total" texts double-click on each cell, which opens the familiar text editor where we can type "Total":



Salary	Year	
Name	[Year]	**Total**
[Name]	0	0
Total	0	0

To format the currency values select the first cell (intersection of `[Name]` and `[Year]` in our example), right-click to display the context menu and select "Display Format...":



Select the required format and close the format editor. All this produces the report:

Salary	Year				
Name	1999	2000	2001	2002	**Total**
Ann	$3 300,00	$2 700,00	$3 100,00	$1 700,00	$10 800,00
Ben	$3 900,00	$2 100,00		$1 800,00	$7 800,00
Catherine	$6 100,00	$3 200,00			$9 300,00
Den		$3 999,00	$8 100,00		$12 099,00
Total	$13 300,00	$11 999,00	$11 200,00	$3 500,00	$39 999,00

Using functions

In our example we see the sum of each employee's salary over four years in the "Total" line. Any of the following aggregate functions can be used:

`SUM` – sum of values

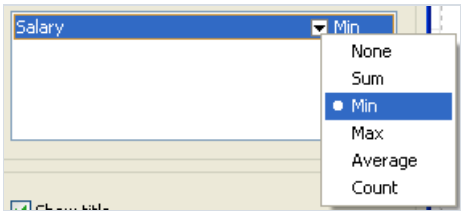
`MIN` – minimal value

`MAX` – maximal value

`AVG` – average value

`COUNT` – number of values

Let's use the `MIN` function in our example. Open the cross-tab editor and click the arrow next to the "Salary" field item.



Select the "Min" function from the drop-down list. Now we can change the text in the total cells from "Total" to "Minimum." The finished report looks like this:

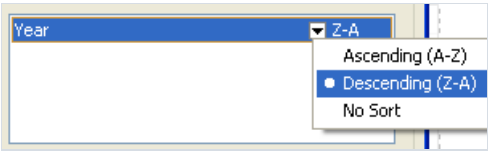
	1999	2000	2001	2002	**Minimum**
Ann	3300	2700	3100	1700	**1700**
Ben	4300	2400		2000	**2000**
Catherine	6100	3200			**3200**
Den		3999	8100		**3999**
Minimum	**3300**	**2400**	**3100**	**1700**	**1700**

Sorting values

Rows and columns by default are arranged in ascending order, either numerically or alphabetically depending on the type of the data. Sort modes can be set independently for rows and columns. The sort modes are :

- ascending
- descending
- no sort. With no sorting the rows/columns are displayed in default database order.

Let's change the column sorting in our example. Let the years be arranged in decreasing order. To do this, open the cross-tab editor, select the "Year" column element and change the sorting mode by clicking on the down arrow and selecting Descending:



Close the editor and preview the report. It will look like this:

	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**

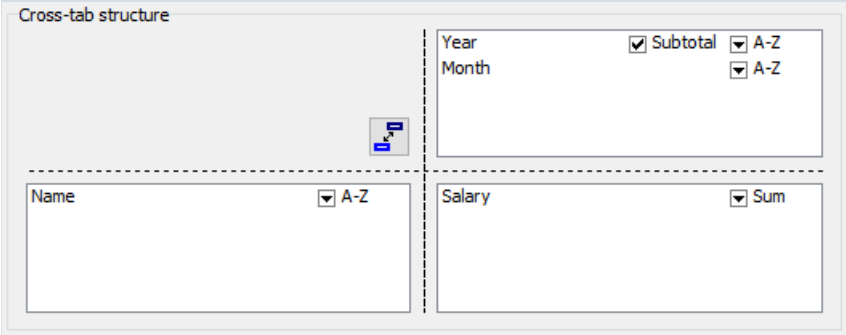
Tables with composite headers

Our previous example contained a single value per line and single column headers. Let's look at a cross-tab design having a composite header with more than one value. The source data has data in the following format:

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

We have added the "Month" and "Days" fields containing the month number and the number of working days respectively. Several different reports can be constructed from this data, for example : salaries of employees over each year, broken down in months.

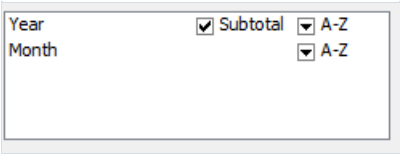
What kind of a report will we get? It must resemble the report from the previous example but having the annual data subdivided by month. The cross-tab object must be set up in the same manner as before. This time we will also drag the "Month" field into the column header list, as shown here:



On preview we see 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 has automatically added a column of intermediate totals, displayed after each year. This can be turned off in the cross-tab editor by de-selecting the "Subtotal" flag of the "Year" column element:



Also note that the last column element in the column header list never has a "Subtotal" flag (including the case of a single element). In our example we do not need intermediate totals for each month, so the "Subtotal" flag can be turned off.

There is another feature of intermediate totals, if they are used: it might be preferable to head the intermediate total as "Year + year total" instead of "Total". In the cross-tab object on the report page double-click the intermediate total cell and in the text editor type:

```
Total for [Value]
```

On report construction the `[Value]` expression is replaced by the actual value of the column header in the cell 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

Looking at the previous illustration it is obvious that FastReport automatically adjusts cell widths so that larger cell values do actually fit within the cells. This may not be desirable in some cases, however, as extra wide columns can appear ugly. What can be done about this? Let's look at three methods of controlling cell width.

The simplest method of controlling cell width is to add line breaks to the text of intermediate totals, i.e.:

```
Total
for [Value]
```

The resulting table is more compact:

	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, there are circumstances where it is difficult or impossible to sensibly break lines manually. The cross-tab object has therefore been given both `MinWidth` and `MaxWidth` properties (referring to cell widths). Both of these properties are only accessible via the object inspector.

By default `MinWidth` is 0 and `MaxWidth` is 200, which is adequate in most cases. The second method of controlling cell width is to alter these values according to special requirements.

So in our example we can set both `MinWidth` and `MaxWidth` to 50. This means that a data cell must be at least 50 pixels wide, even if the cell value would fit into fewer pixels. For large cell values the cell width is limited to the `MaxWidth` value and the text in the cell is broken as required. So our example now looks like this:

	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**

The third method of controlling cell width is to change cell widths manually. To be able to do this the `AutoSize` property must be set to False. The cross-tab cell width can then be changed using the mouse. In the report page cross-tab object, the mouse cursor changes shape over cell borders so allowing the borders to be dragged. Here is an example of what can be achieved:

Salary	Year, Month		
Name	[Year]		**Total**
	[Month]	**Total**	
[Name]	$0,00	$0,00	$0,00
Total	$0,00	$0,00	$0,00

Remember that if auto-size is turned off then the cross-tab will not automatically adjust the cell widths and heights and you may see something like this in the report preview:

Salary			
Name		1999	
	2	10	11
Ann	$1 000,00		$1 100,00
Ben		$1 900,00	$2 000,00
Catherine		$3 000,00	$3 100,00

If so then just increase the cell widths a little.

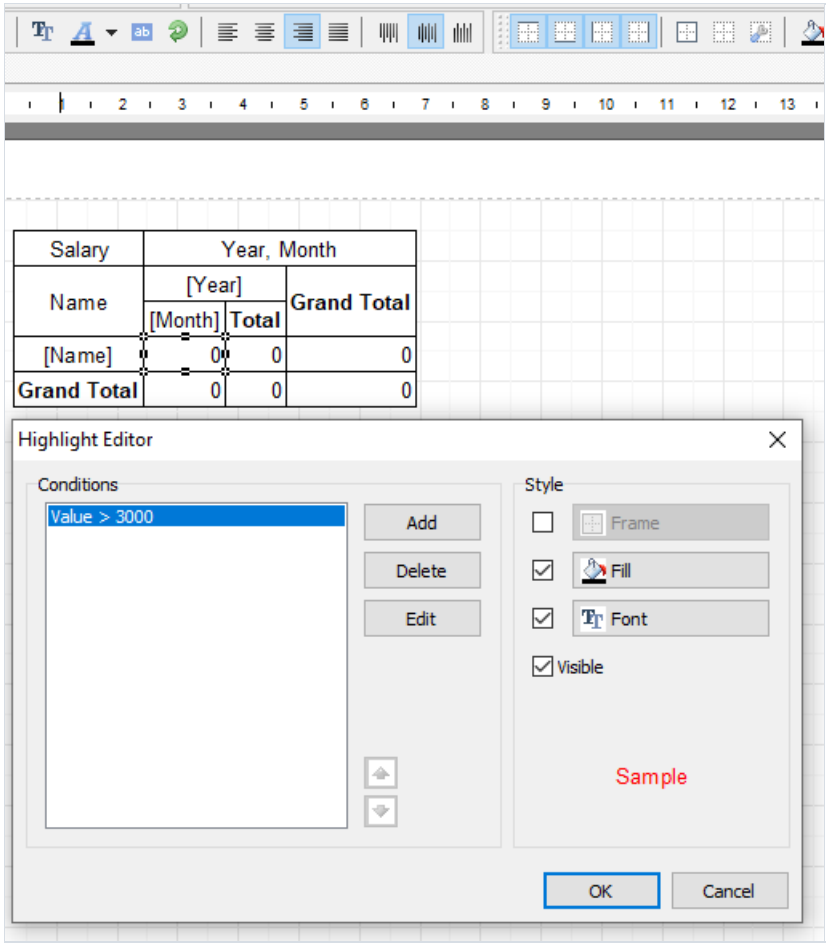
Font colors and highlighting

Sometimes it is necessary to highlight values and/or change the font color. We have looked at this in the group report example, where we used conditional highlighting for "Text" objects. Conditional highlighting can also be useful for us here.

To add highlighting to our example report, assuming we need to change the font color for values greater than 3000 - select the object representing the table cell and set the highlighting parameters by clicking the highlight button on the toolbar. The familiar highlighting editor window will open where this condition can be set:

```
Value > 3000
```

and the font color set to red:



This is all that is required. Close the editor by clicking on the OK button and preview the 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**

Likewise total values can be highlighted, if required, and also cells and lines using the background and frame color buttons.

Managing a cross-tab in script

If any of the methods shown above do not achieve the required report look then a report script can be used instead. The "Cross-tab" 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 height of a row in the table the event handler can be set to either the required height or to "0" if the row needs to be hidden
`OnCalcWidth`	event is called before calculating column width in a table the event handler can be set to either the required width or to "0" if the column needs to be hidden
`OnPrintCell`	event is called before displaying a table cell the event handler can modify the cell design or its contents
`OnPrintColumnHeader`	event is called before displaying the column title the event handler can modify the design or content of the title cell
`OnPrintRowHeader`	event is called before displaying the row title the event handler can modify the design or content of the title cell

We can use the following methods of the "Cross-tab" object in these events:

Method	Description
`function ColCount: Integer`	returns the number of columns in the table
`function RowCount: Integer`	returns the number of rows in the table
`function IsGrandTotalColumn(Index: Integer): Boolean`	returns "True" if the `Index` column is a total
`function IsGrandTotalRow(Index: Integer): Boolean`	returns "True" if the `Index` row is a total
`function IsTotalColumn(Index: Integer): Boolean`	returns "True" if the `Index` column is a sub-total
`function IsTotalRow(Index: Integer): Boolean`	returns "True" if the `Index` row is sub-total
`procedure AddValue(const Rows, Columns, Cells: array of Variant)`	adds a value to the table

Let's show how to highlight the third column (in our example the "November 1999" date). Select the cross-tab object on the report design page, in the object inspector click on the events tab, locate the "OnPrintCell" event and create a handler on the code page by double-clicking in the empty list to the right of the event name. The script editor will appear with the basic declaration created for you, then add the code required in the empty `begin...end`

block of the declaration:

Pascal script:

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

C++ Script:

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

We will see the following when the report is previewed:

	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 handler in a similar way :

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;
```

C++ Script:

```
void Cross1OnPrintColumnHeader(TfrxMemoView Memo,
  Variant HeaderIndexes, Variant HeaderValues, Variant Value)
{
  if ((VarToStr(HeaderValues[0]) == "1999") &&
     (VarToStr(HeaderValues[1]) == "11"))
  {
    Memo.Color = clRed;
  }
}
```

The report preview is now:

	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**

This is how the script works: the "OnPrintCell" event handler is called before printing a cell in the table's data area (note that cells in the table titles call either the "OnPrintColumnHeader" or the "OnPrintRowHeader" handler). The "OnPrintCell" handler parameters include: a link to the "Text" object which represents the table cell (the `Memo` parameter) and the cell's "address" as the location of the row, column and cell (cell is relevant if your cross-tab contains multi-leveled cells) as the `RowIndex`, `ColumnIndex`, and `CellIndex` parameters respectively. The parameter list also has the header's values specified as Variants (the `RowValues` and `ColumnValues` parameters) and the `Value` Variant parameter which holds the cell contents.

In our example it is easier to specify the "address" using the `RowIndex` and `ColumnIndex`. Numbering of columns and rows begins at 0 so "ColumnIndex = 2" refers to the third column. We could also specify the correct column by looking at its data content (we need the 11th month of 1999):

Pascal script:

```
procedureCross1OnPrintCell(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;
```

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;
  }
}
```

The `RowValues` and `ColumnValues` parameters are arrays of the Variant type, having a zero base. The "0" element is at the highest level of the table title, the "1" element is at the next level, etc. In our example "ColumnValues[0]" contains years and "ColumnValues[1]" contains months.

Why is the `VarToStr` function required? This prevents errors during type conversion. When working with the Variant type FastReport tries to automatically cast strings to number format, which in turn can lead to errors when casting the "Total" and "Grand Total'" column values.

The "OnPrintColumnHeader" event handler is called during output of the column title cells. The parameter list is

similar to that of the "OnPrintCell" handler, though in this case the cell's "address" (`HeaderIndexes` and `HeaderValues` parameters) is in a different form. The `HeaderValues` parameter holds the same values as the `ColumnValues` and `RowValues` in the "OnPrintCell" handler. The `HeaderIndexes` parameter is also an array of values of Variant type, and contains the address of the title cell in a different form: the "0" element is the index at the highest level in the table title, the "1" one is at the next level, etc. To clarify the principle of cell numbering 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 example it was easier to use the `HeaderValues` parameter, but the following handler could be used instead:

Pascal script:

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

C++ Script:

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

Adjusting row/column size

The width and height of the table columns and rows can be adjusted by using the "OnCalcWidth" and "OnCalcHeight:" event handlers. Let's see how to increase the width of the column holding the 11th month of 1999 with the following example. Create an "OnCalcWidth" event handler:

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;
```

C++ Script:

```
void Cross1OnCalcWidth(int ColumnIndex, variant ColumnValues, Extended &Width)
{
  if ((VarToStr(ColumnValues[0]) == "1999") &&
    (VarToStr(ColumnValues[1]) = "11"))
    {
      Width = 100;
    }
}
```

And the report becomes:

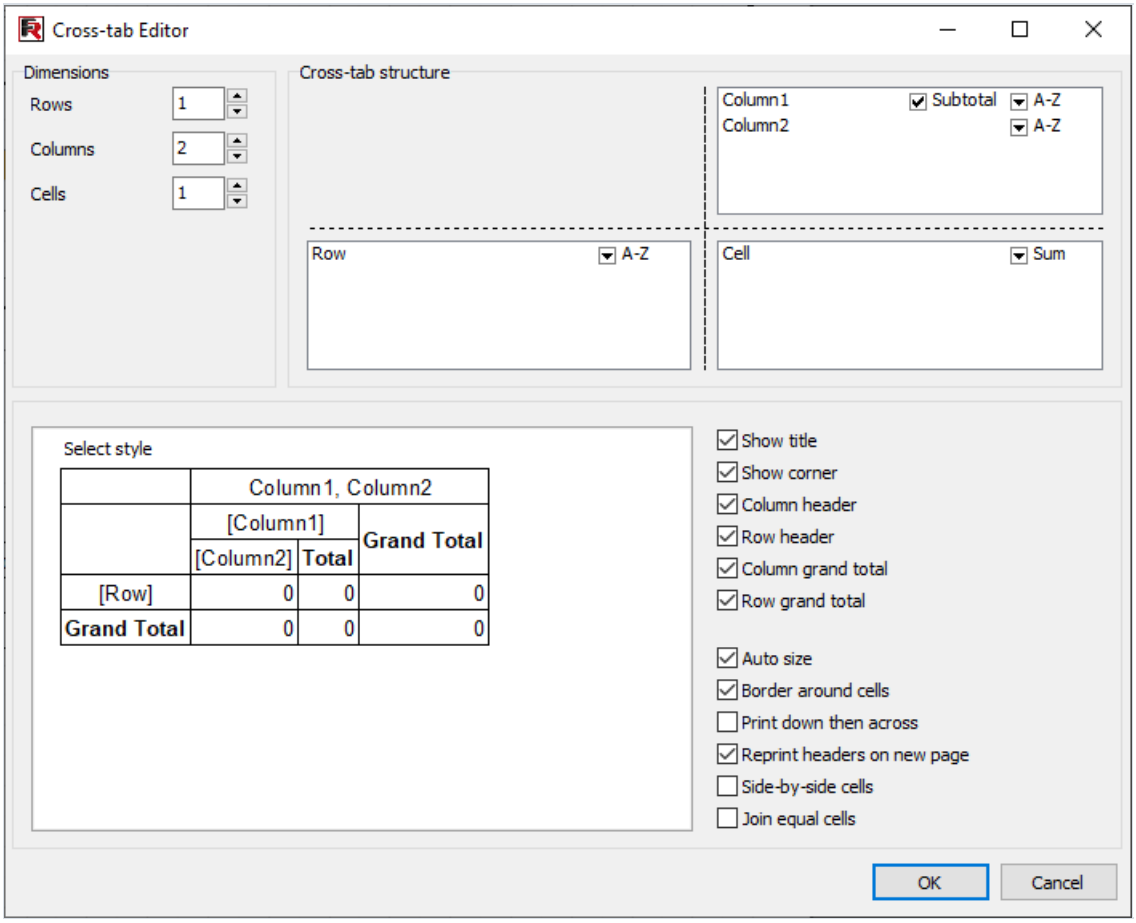
	1999				
	2	10	11	12	**Total**
Ann	1000		1100	1200	**3300**
Ben		2100	2200		**4300**
Catherine		3000	3100		**6100**
Den					**0**
Grand Total	**1000**	**5100**	**6400**	**1200**	**13700**

To hide a column in our example just set the Width to zero. Note that the totals are not recalculated as the table is already filled with values at this point.

Filling a table manually

There are two versions of the cross-tab object, "DB cross-tab" and "Cross-tab". So far we have worked with the first object, attached to data from a DB table. The object fills itself automatically as soon as the report runs. Let's look at the second object, "Cross-tab".

The "Cross-tab" object is not attached to a DB table. Therefore it has to be filled with data manually. The "Cross-tab" object has a similar editor to the "DB cross-tab" object, though it differs in that the dimensions of the table's titles and cells have to be specified, instead of being set by the DB fields:



Let's show the use of a "Cross-tab" object with an example. Place a "Cross-tab" object on the report design page and set its properties as shown above : the number of levels in the rows title is 1, in the columns title is 2 and in the cells is 1. Let's fill the table with data using the "OnBeforePrint" event handler:

PascalScript:

```
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;
```

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]);
}
```

In the handler, data is added to the table through the `TfrxCrossView.AddValue` method. This method has three parameters, each of them an array of Variant type. The first parameter is the row value, the second the column value and the third the cell value.

> Note that the number of values in each array must match the settings for the object!

In our example the object has one level in the row title, two levels in the column title and one level of cells, so the `AddValue` method's Variant array parameters need one value for the rows, two values for the columns and one value for the cells.

On preview the report output is:

	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**

The `AddValue` method can also be used for the “DB cross-tab” object. This allows the insertion of data which is not derived from the data source attached to the object. If any data is added in this way it is also summarized with the data from the data source.

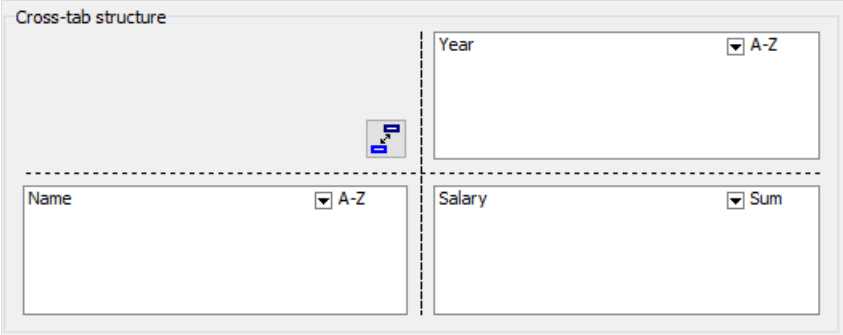
Adding external objects to the table

External objects (such as lines, shapes, pictures) can be placed in the cross-tab. You may need to show, for example, some data in a graphical form. Let's look at an example that uses shapes to display a rudimentary progress bar:

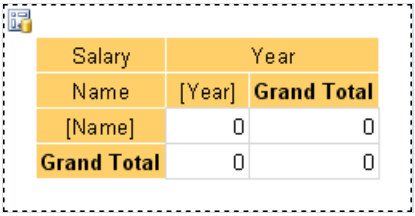
Salary	Year				
Name	1999	2000	2001	2002	**Grand Total**
Ann	3300	2700	3100	1700	10800
Ben	3900	2100		1800	7800
Catherine	6100	3200			9300
Den		3999	8100		12099
Grand Total	13300	11999	11200	3500	39999

A dark red bar is displayed if the cell value is less than 100, yellow if less than 3000 or green if more that 3000.

Let's start with our report. Place a "DB Cross-tab" object on the report page and set its properties like this:

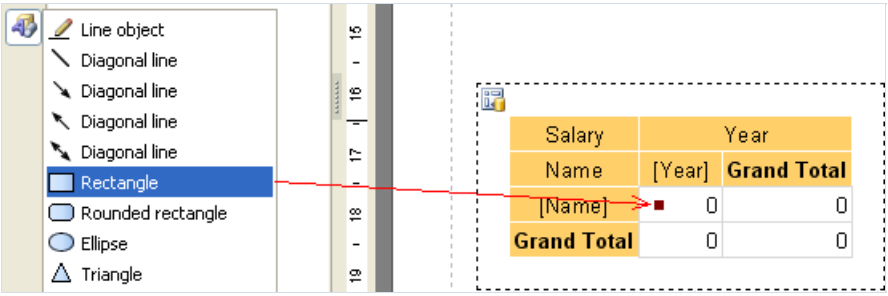


Turn off the `AutoSize` property and set the column widths as shown below:



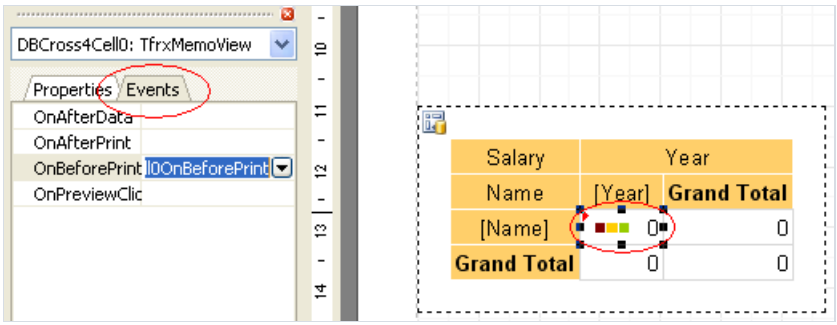
Salary	Year	
Name	[Year]	**Grand Total**
[Name]	0	0
Grand Total	0	0

Now add the shape object to our table by selecting the “Rectangle” object on the object toolbar and placing it inside the cell:



Change its height and width to 0.2cm and set its top and left properties. Add two more similar rectangles.

Now create a script that will show the correct number of colored shapes (depending on the cell value). To do this select the cell and create an “OnBeforePrint” event handler:



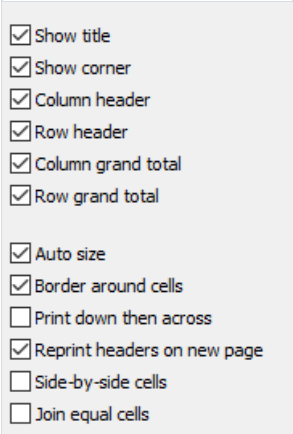
Write the following code in the event handler (pay attention to the shape names, so that they match your objects):

```
procedure DBCross1Cell0OnBeforePrint(Sender: TfrxComponent);
begin
  // 'Value' is the current cell's value
  if Value < 100 then
  begin
    // first shape object
    DBCross1Object1.Color := clMaroon; // dark red
    // second shape object
    DBCross1Object2.Color := clWhite;
    // third shape object
    DBCross1Object3.Color := clWhite;
  end
  else if Value < 3000 then
  begin
    DBCross1Object1.Color := $00CCFF;  // yellow
    DBCross1Object2.Color := $00CCFF;
    DBCross1Object3.Color := clWhite;
  end
  else
  begin
    DBCross1Object1.Color := $00CC98;  // green
    DBCross1Object2.Color := $00CC98;
    DBCross1Object3.Color := $00CC98;
  end;
end;
```

That's all - preview the report, which will be similar to that shown at the top of this section.

Some useful settings

Let's look at some other settings available in the cross-tab editor.



The first six options allow you to show or hide various table elements.

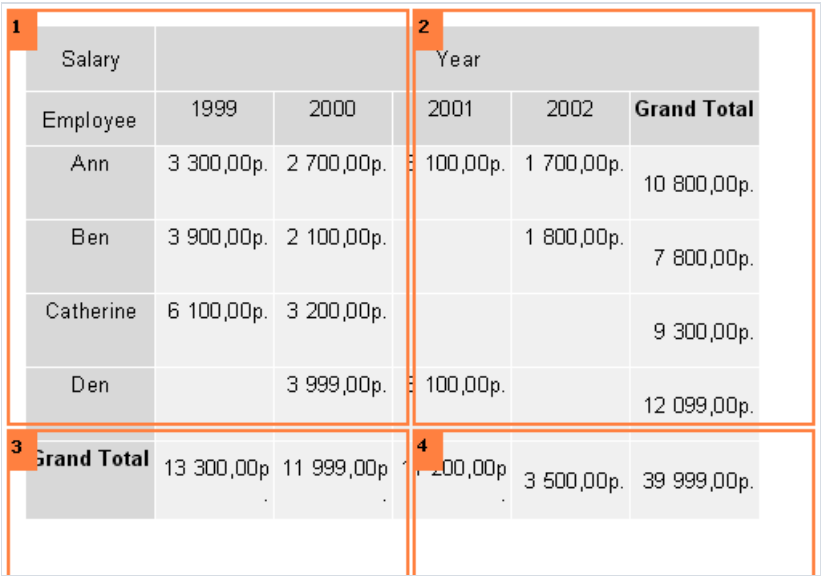
The "Auto size" option is already well known. It allows us to set the table width and height manually.

The "Border around cells" option allows the drawing of a frame around the cell elements. Here is an example table with a border (note that the cells themselves don't have borders):

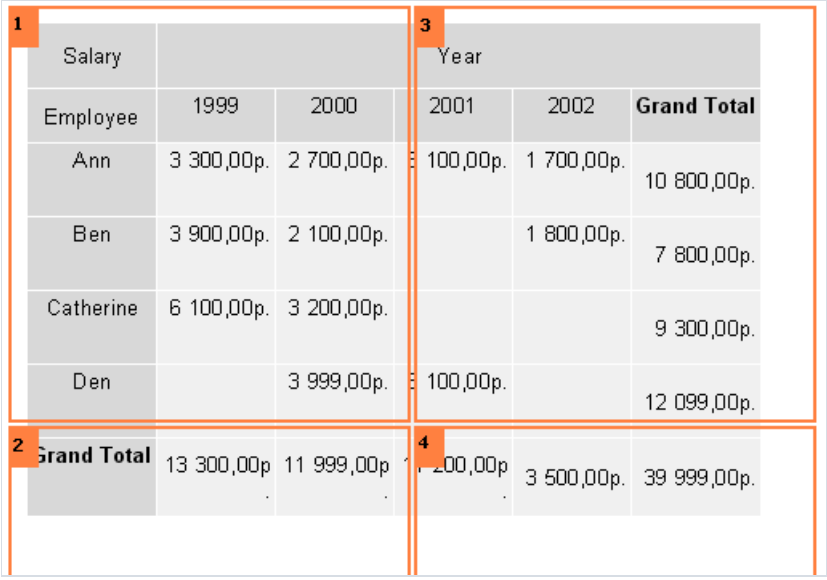
Salary	Year				
Name	1999	2000	2001	2002	**Grand Total**
Ann	3300	2700	3100	1700	10800
Ben	3900	2100		1800	7800
Catherine	6100	3200			9300
Den		3999	8100		12099
Grand Total	13300	11999	11200	3500	39999

The "Print down then across" option determines how a table is printed across several pages. Here are two examples showing how this option works (note the page order):

1. "Print down then across" is off:



2. "Print down then across" is on:



Salary			Year		
Employee	1999	2000	2001	2002	**Grand Total**
Ann	3 300,00р.	2 700,00р.	100,00р.	1 700,00р.	10 800,00р.
Ben	3 900,00р.	2 100,00р.		1 800,00р.	7 800,00р.
Catherine	6 100,00р.	3 200,00р.			9 300,00р.
Den		3 999,00р.	100,00р.		12 099,00р.
Grand Total	13 300,00р.	11 999,00р.	200,00р.	3 500,00р.	39 999,00р.

The “Reprint headers on new page” option determines whether table headers are printed on each new preview page.

The “Side-by-side cells” option is used if you have two or more values in a table cell. It determines if these cell values are printed side-by-side or stacked one above the other (the default).

The “Join equal cells” option merges cells horizontally if they contain the same value:

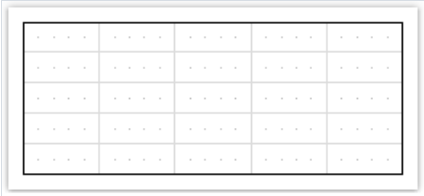
Days	Year				
Name	1999	2000	2001	2002	**Grand Total**
Ann	3		4	2	12
Ben	4	2		2	8
Catherine	6	3			9
Den		4	7		11
Grand Total	13	12	11	4	40

Some other properties are also available in the object inspector:

- `AddWidth`, `AddHeight`: adds a specified amount of space to the cell width or height. It is taken into account when the FastReport engine calculates the cell size (the “Auto size” option must be on)
- `NextCross`: a pointer to another cross-tab object that will be printed to the side of this one
- `NextCrossGap`: the gap between the two adjacent cross-tab objects

Table-type reports

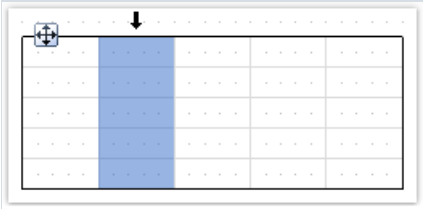
The "Table" object is made up of rows, columns and cells. It is a simplified analog of Microsoft Excel table. It looks in the following way:



Configuring columns

You can add or delete columns with the help of the context menu. For that:

- select the table or any of its elements and place the cursor on the needed column. The cursor's form changes to a small black arrow.



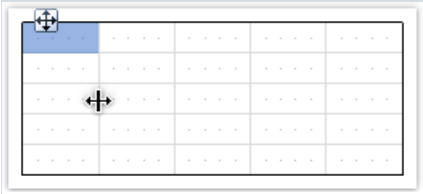
- left click in order to select the column;
- right click in order to show the column's context menu;
- if you need to select several adjacent columns, left click and drag the mouse to the right or to the left, in order to select adjacent columns.

Column's context menu can also be called in the "Report Tree" window. Open the window, select the needed column and right click the mouse.

Managing the size of the column

You can set up the width of the column by using one of the following methods:

- select the table or any of its elements and place the cursor on the border between two columns. The form of a cursor changes into a horizontal splitter:



- select a column and indicate the needed width in the "Width" properties. This property is accessible in the "Properties" window.

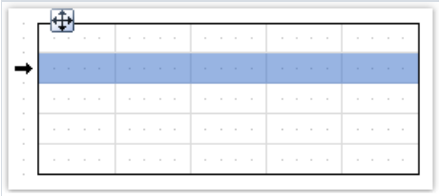
You can also enable the `AutoSize` column property. When running the report, the width of the column will be calculated automatically. In order to limit the width of the column, you can indicate the `MinWidth` and `MaxWidth` properties.

> Width of the column should never be larger than the page's width.

Configuring rows

Rows are configured in the same way. In order to select a row, do the following:

- select the table or any of its elements and place the cursor to the left of the needed row. The cursor's form changes to a small black arrow:



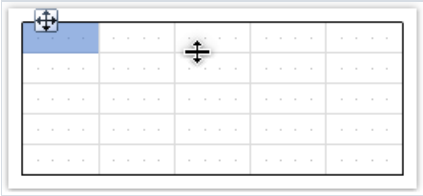
- left click the mouse, in order to select the row;
- right click the mouse in order to show the row's context menu.

If you need to select several adjacent rows, left click the mouse and drag the mouse to the top or to the bottom, so as to select the adjacent rows.

Managing the size of the row

You can set up the height of the row by using one of the following methods:

- select the table or any of its elements and place the cursor on the border between two rows. The cursor's form changes into a vertical splitter:



Left click and move the mouse, in order to change the size of the row.

- select the row and indicate the needed height in the `Height` property. This property is accessible in the "Properties" windows.

You can also enable the row's `AutoSize` property. When the report is run, the height of the row will be calculated automatically. In order to limit the height of the row, you can use the minimum height (`MinHeight`) and maximum height (`MaxHeight`) properties:

Height of the row should never be larger than the page's height.

Configuring cells

Cells are text objects. In essence, the cells class is inherited from the "Text" object. Everything which has been said above about the "Text" object applies to the table's cells as well.

Editing the cells' text can be done just like the "Text" object. Besides, you can drag and drop an element from the "Data" window into the cells.

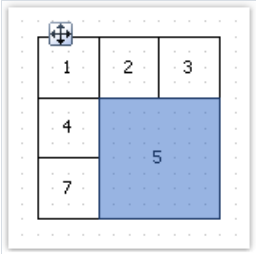
Border and fill of a cell can be configured with the help of the "Border and Fill" toolbar.

In order to call a cell's context menu, just right click on the cell.

Joining and splitting cells

You can join adjacent cells of the table. As a result, there will be one big cell. In order to do this:

- select the first cell with the help of a mouse;
- left click and, without leaving it, move the mouse in order to select the group of cells;
- on the selected region, right click the mouse in order to show the context menu of the cells;
- in the context menu, choose the "Join cells" item.



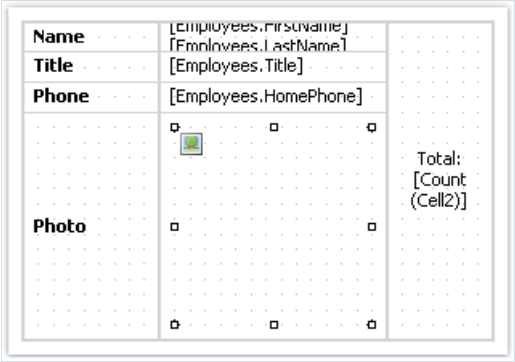
In order to split a cell, call its context menu and choose the "Split cell" item.

Inserting objects in cells

In the cells, you can insert other objects, for example, the picture. The following objects can never be added into the cells:

- "Table";
- "Cross-tab";
- "Subreport".

In order to add an object into the cell, simply drag it inside the cell. You can freely move an object between cells, and also take it back beyond the table's boundary.



A cell serves as a container to objects placed into it. This means that, you can use the `Align` property of an object inside the cell. This allows changing the size of the object when the size of the cell is changing.

Charts

FastReport can insert charts into a report. The `TfrxChartObject`  object from the FastReport component palette in Delphi is used for this purpose.

> The component is based on the "TeeChart" library which is included in Delphi distribution kit. Alternatively the "TeeChartPro" library can be purchased separately.

Let's make an example report with a simple chart. The chart will use the "Country" table from the `DBDEMOS` database that comes with Delphi . The table contains data about countries, their areas and populations:

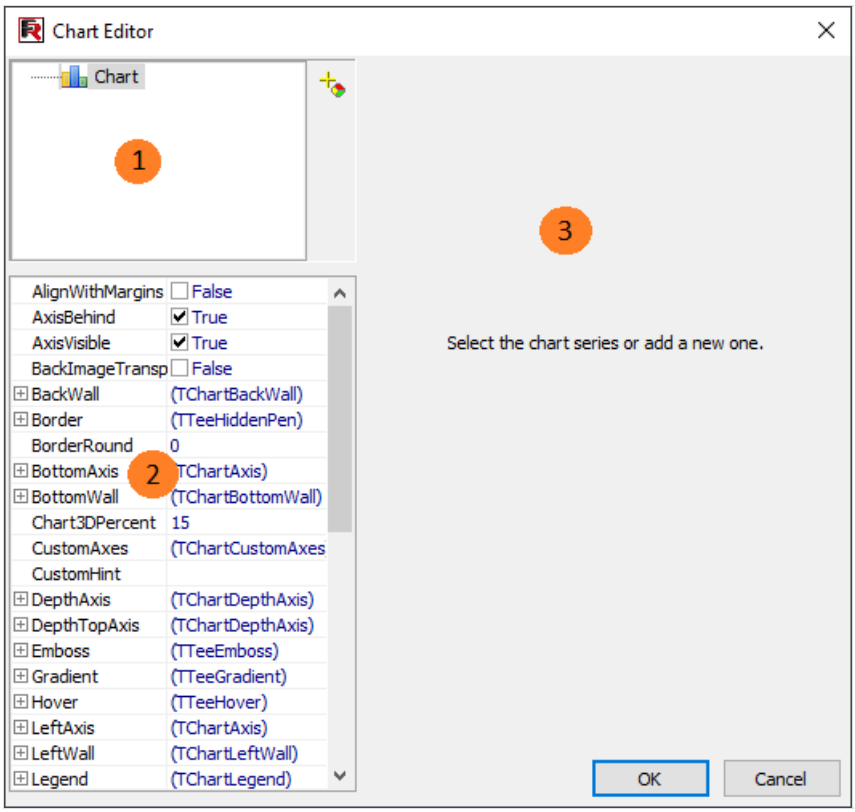
```
Name       Area  Population
Argentina  2777815  32300003
Bolivia    1098575  7300000
...
```

Create a new project in Delphi. Place `TTable` , `TfrxDBDataSet` , `TfrxChart` and `TfrxReport` components on the form and set these properties:

```
Table1:
DatabaseName = 'DBDEMOS'
TableName = 'country.db'

frxDBDataSet1:
DataSet = Table1
UserName = 'Country'
```

Open the report designer, create a new report and connect the data source in the "Report > Data..." dialogue. Add a "Chart" object  to the report design page and set its size to 18 cm wide x 8 cm high. Open its object editor by double-clicking on it.



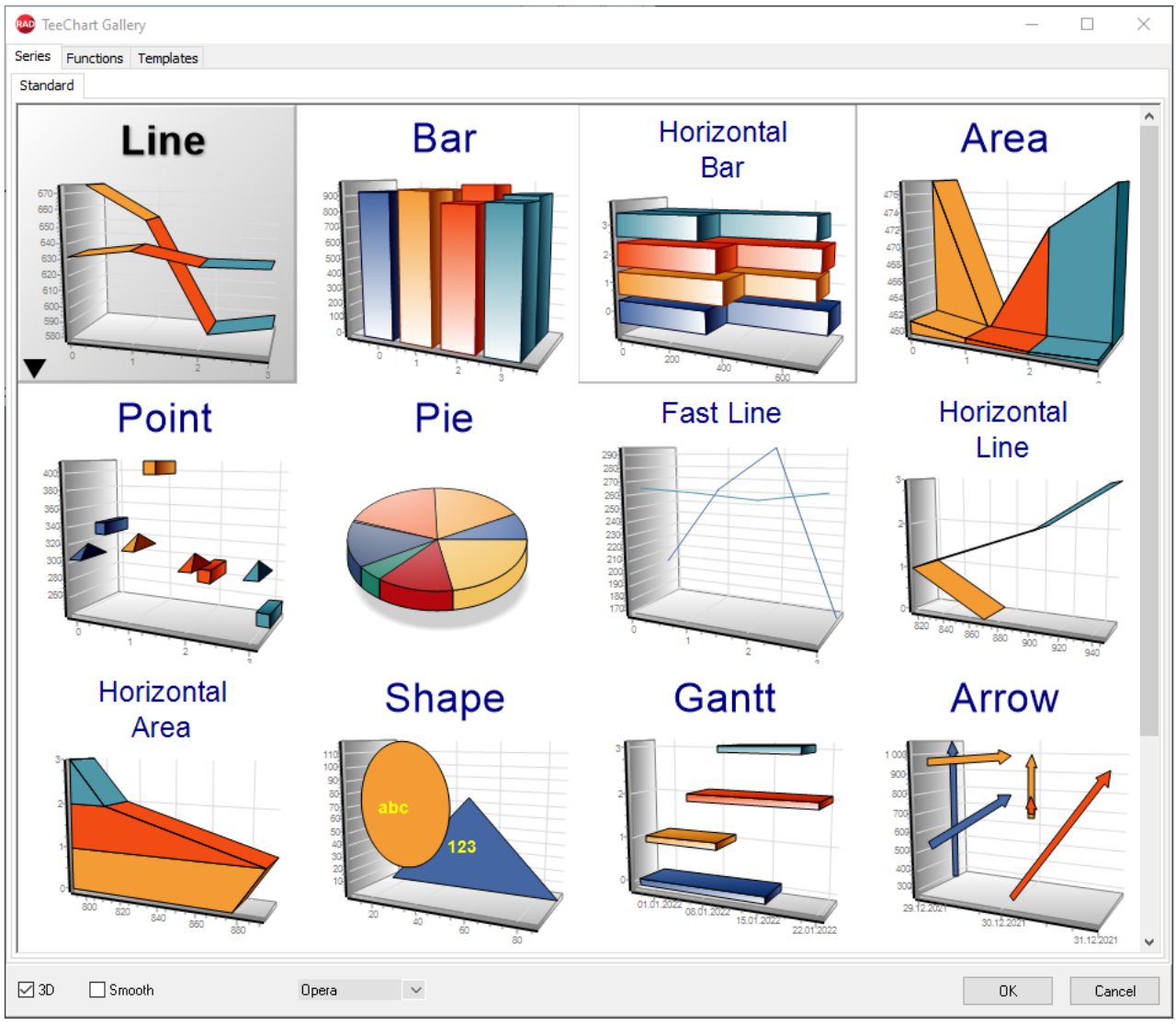
Key to the chart editor above:

1 – chart structure; a chart can contain one or more series

2 – object inspector showing the properties of the element selected in the window; set the chart properties here

3 – options area for connection of the series to data; it is activated once a series in area 1 has been selected

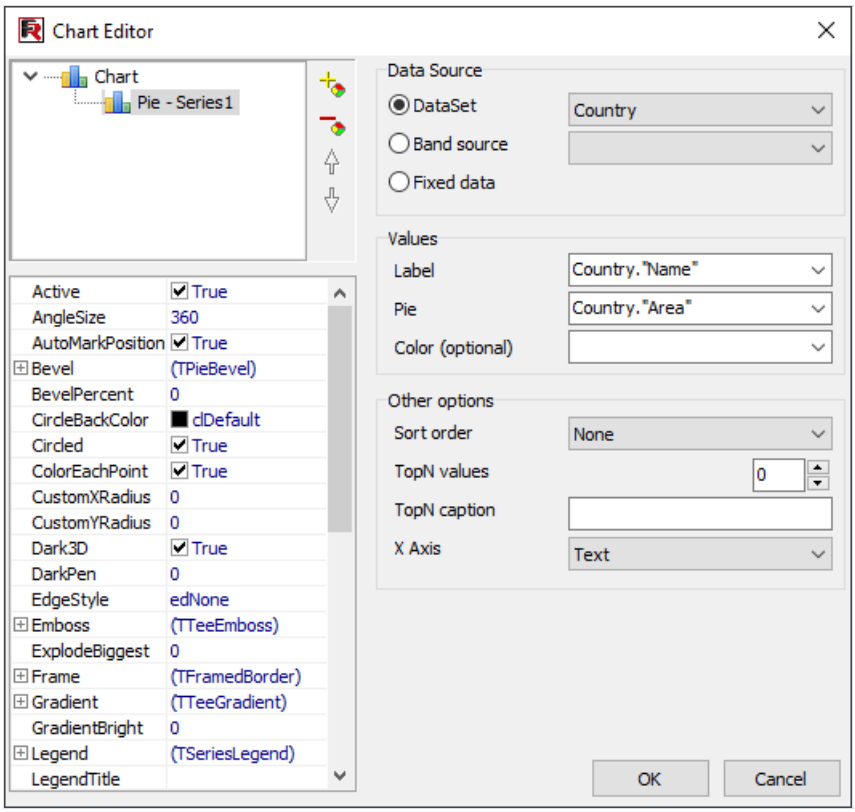
When first opened the chart editor will appear as in the image shown above. The first task is to add one or more series to the chart (just one series in our example). Do this by clicking the add button  and selecting the pie chart in the Gallery:



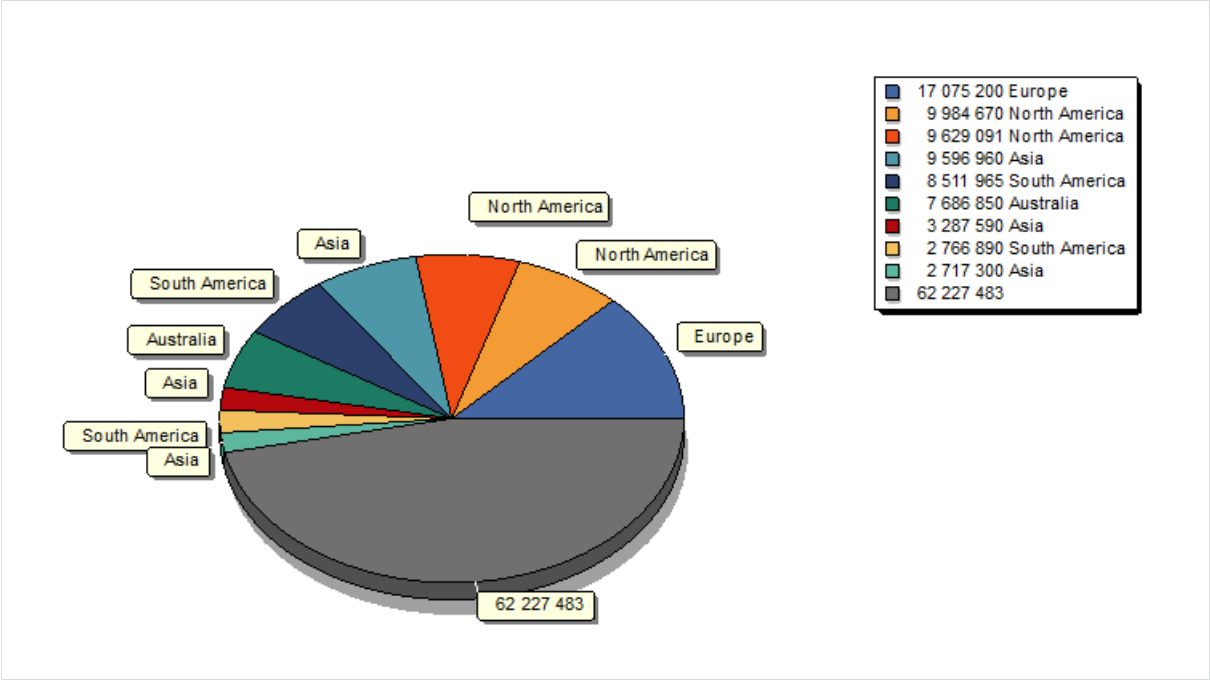
The vertical arrows change the order of the series in the list. To change the name of a series select the series, then one second later click on it again (note that this is not a double-click).

There are many different types of series available. After a series has been added the options area 3 becomes active. Here you specify which data should be used for plotting the chart.

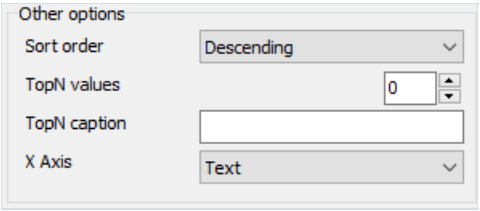
First let's choose the dataset in the “DataSet” drop-down list. Then choose the “Label” and “Pie” fields using their respective drop-down lists, as shown below:



Click OK to close the editor and preview the report:



What can be improved in this report? It would be nice to sort the populations in descending order. Open the chart editor again, select the series in the chart structure and change the sort order from "None" to "Descending":

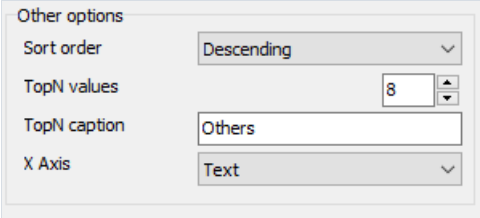


If we previewed the report now, we would see that the data in the legend table has been sorted in descending order.

Limiting the number of chart values

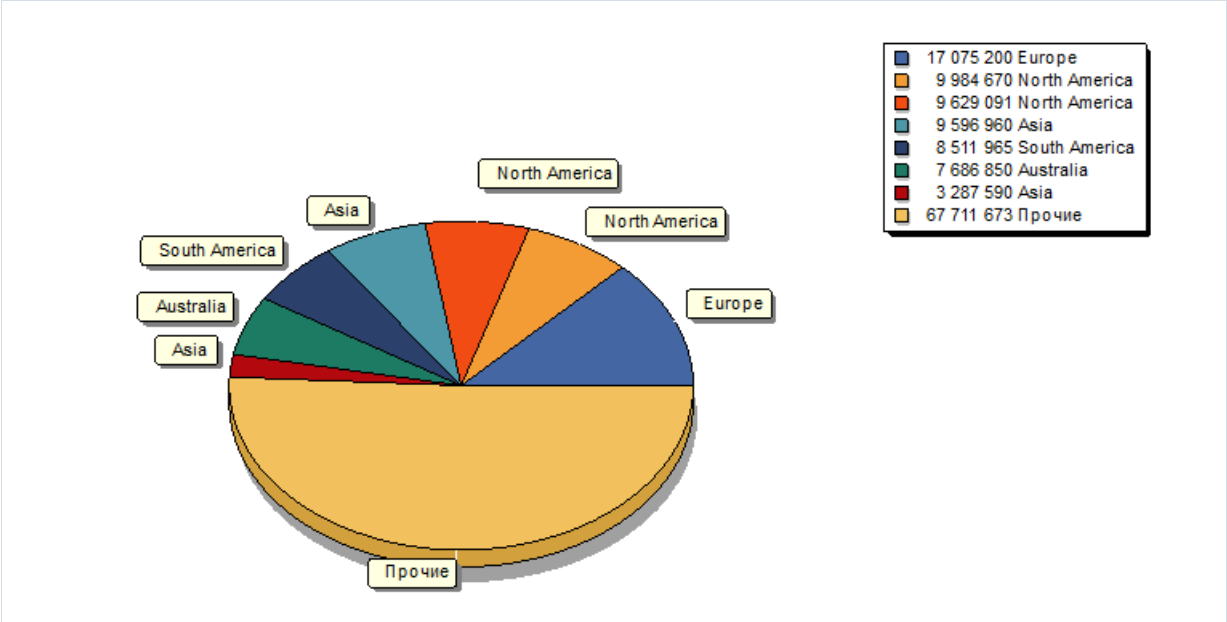
Our chart is over-crowded with many small values, which has resulted in many becoming invisible. FastReport allows you to limit the number of values displayed in the chart. All the values in excess of a set number are displayed as a single composite value, consisting of the sum of all these values.

In our example the chart has 18 values but practically only 8 of them can be seen. Open the chart editor and set the limit to 8:



If "TopN" is set to zero then there will be no limitation. A name should be entered in "TopN caption" so that the aggregated values are identified correctly in the legend list. The Sort mode is irrelevant as the values will always be sorted descending by default.

The report will look like this:



Some useful settings

Let's look at some useful settings for controlling the chart's appearance. These settings can only be accessed through the object inspector of the chart editor.

These basic properties are available when 'Chart' is selected in the Chart Structure:

- `Gradient`: settings for gradient background fill. Enable the `Gradient.Visible` property to show gradients
- `Legend`: settings for the Legend List. The List can be hidden through the `Legend.Visible` property and its position set through the `Legend.Alignment` property.

The following properties are available when a series has been selected:

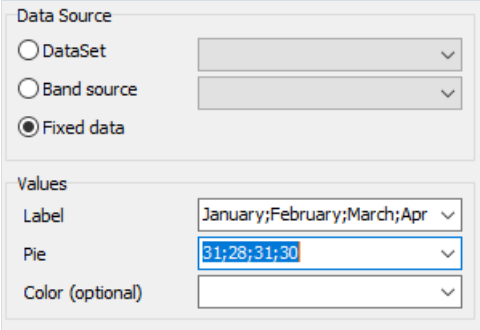
- `ColorEachPoint`: color each value with a different color
- `ExplodeBiggest`: outset the largest value (only works for pie charts)
- `Marks`: settings for the look of chart labels
- `ValueFormat`: the line for formatting values

Note that all charting capabilities are accessible in the "TeeChart Pro" library (which can be bought separately from www.teechart.com). This library contains many types of chart and inludes a convenient chart and series editor.

Chart with manually entered values

In the previous example we created a chart using data from a DB table. A chart can also be created from manually entered data. This method can be convenient if the chart is small.

Let's demonstrate how this works with a simple example. Place a chart on the report design page and open the chart editor. Add a series of “Bar chart” type and set these properties, using semi-colons to separate individual values:



The resulting chart is:

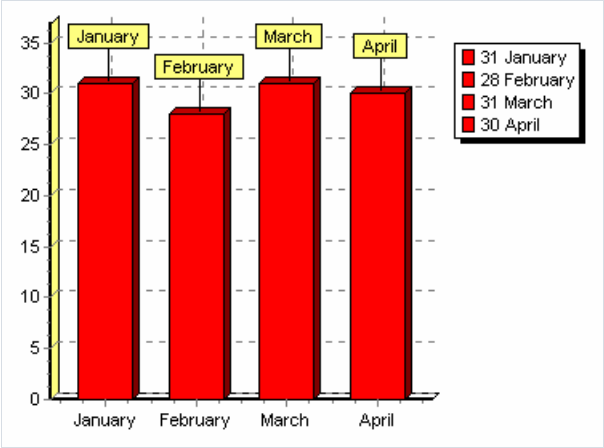


Chart completion from Script

Let's look at how to construct the previous chart using a script. Leave the Label and Y fields empty in the chart editor. In the report script enter the following code:

PascalScript:

```
begin
  Chart1.SeriesData[0].XSource := 'Jan;Feb;Mar;Apr';
  Chart1.SeriesData[0].YSource := '31;28;31;30';
end.
```

C++Script:

```
{
  Chart1.SeriesData[0].XSource = "Jan;Feb;Mar;Apr";
  Chart1.SeriesData[0].YSource = "31;28;31;30";
}
```

"SeriesData[0]" in this case allows us to set parameters for the first series in the chart. If the chart has more than one series then refer to them with "SeriesData[1]", etc.

Printing a chart built in Delphi

If you have already built a chart in your Delphi code and want to print it in the report you need to use a “Picture” object from the FastReport object toolbar. Place the object in the required place on the report design page and create the following "TfrxReport.OnBeforePrint" event handler in the Delphi application:

```
procedure TForm1.frxReport1BeforePrint(Sender: TfrxReportComponent);
begin
  if Sender.Name = 'Picture1' then
    TfrxPictureView(Sender).Picture.Assign(
      Chart1.TeeCreateMetafile(False,
        Rect(0, 0, Round(Sender.Width), Round(Sender.Height))));
end;
```

where Picture1 is the “Picture” object's name and Chart1 is your Delphi chart.

> Note: When you have code assigned to the event handlers of the `TfrxReport` component within a Delphi application you must preview the report by running the compiled Delphi application. You cannot preview the report from within the FastReport report designer.

Maps

The "Map" object can display two-dimensional maps in the ESRI shapefile format (.shp/.dbf), Open Street Map (.osm), GPS-track (GPS Exchange File, .gpx). See more details about these formats here:

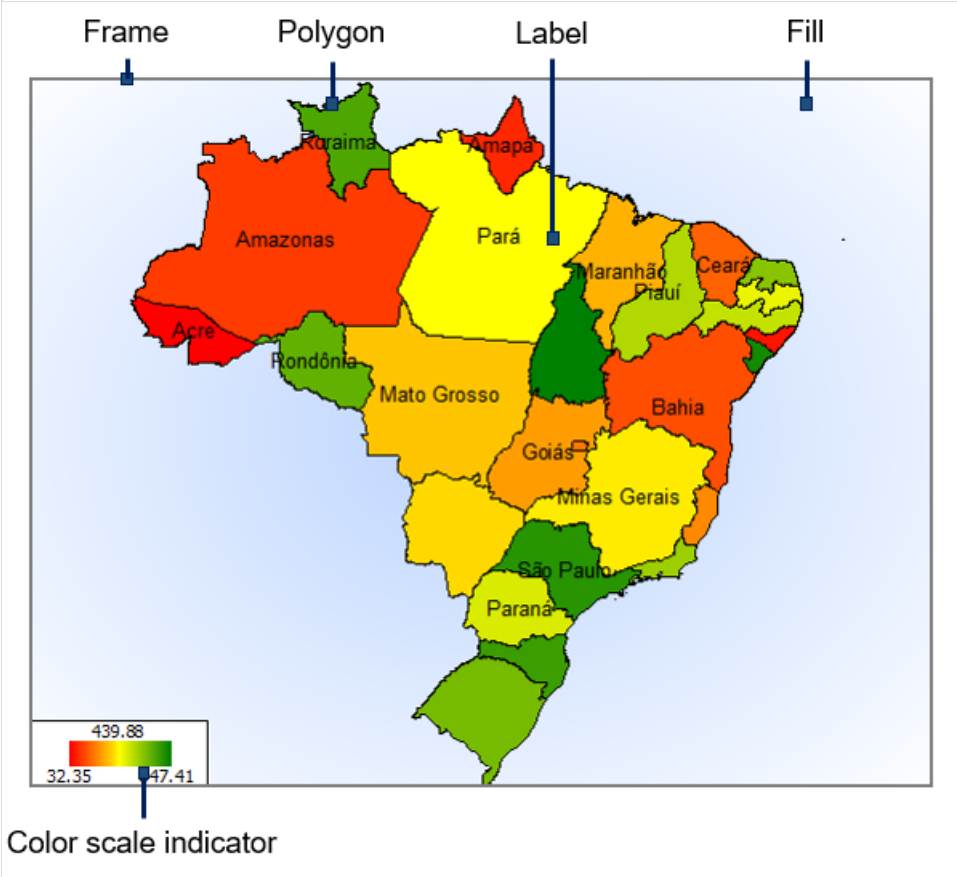
http://wikipedia.org/wiki/Shapefile

https://wikipedia.org/wiki/OpenStreetMap

https://wikipedia.org/wiki/GPX

Map elements

The "Map" object consists of the following elements:



One "Map" object can display one or more layers. Each layer contains its own map.

Controlling the map with the mouse

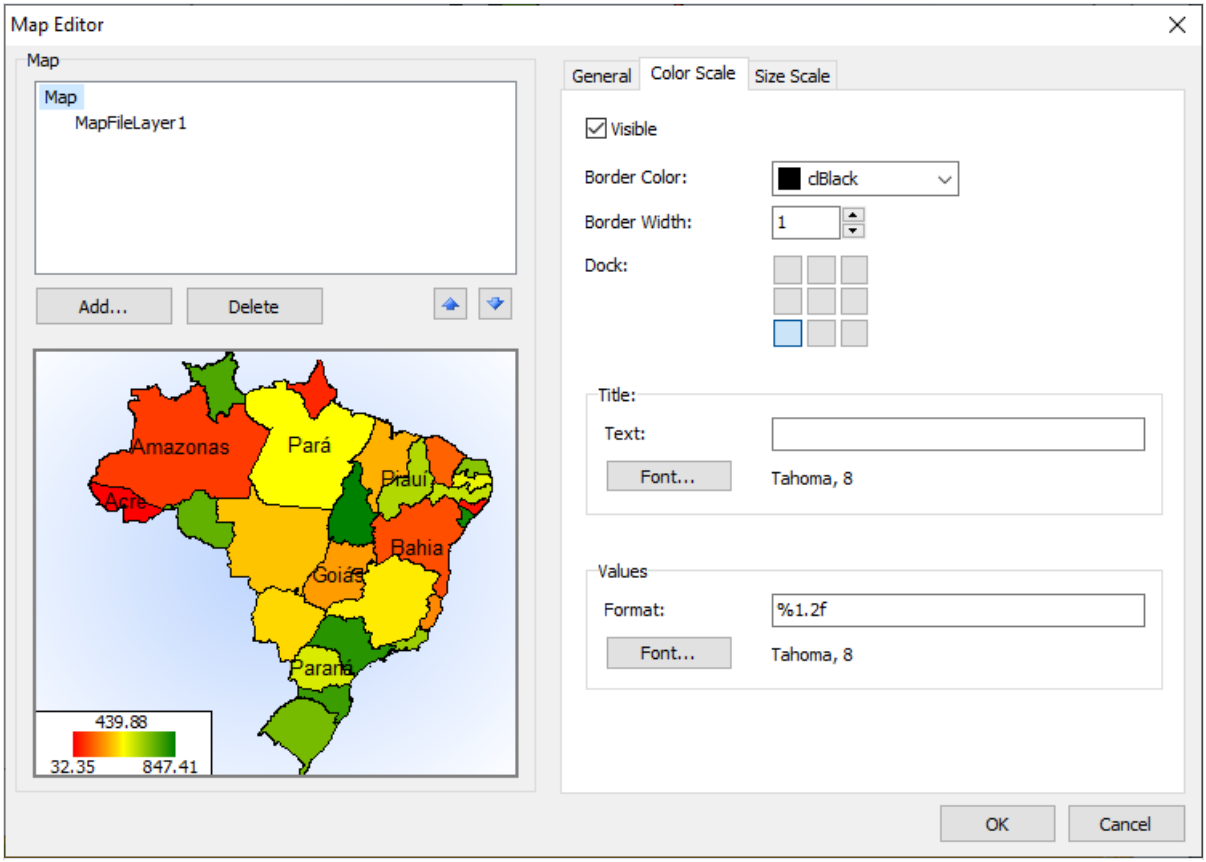
In the report designer and in the preview window the map can be controlled by the mouse:

- zoom in and out using the mouse wheel
- pan the map by dragging the left mouse button
- change a polygon's properties in the "Object Inspector" window by clicking the left mouse button inside a map polygon

The minimum and maximum zoom values can be set in the `MinZoom` and `MaxZoom` properties. These properties are changed in the "Object Inspector" window.

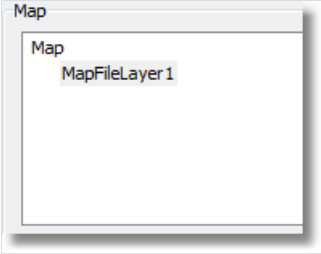
The "Map" object editor

The "Map" object has many properties, which can be set in the object editor. The editor is opened by double-clicking on the object, or from its context menu:

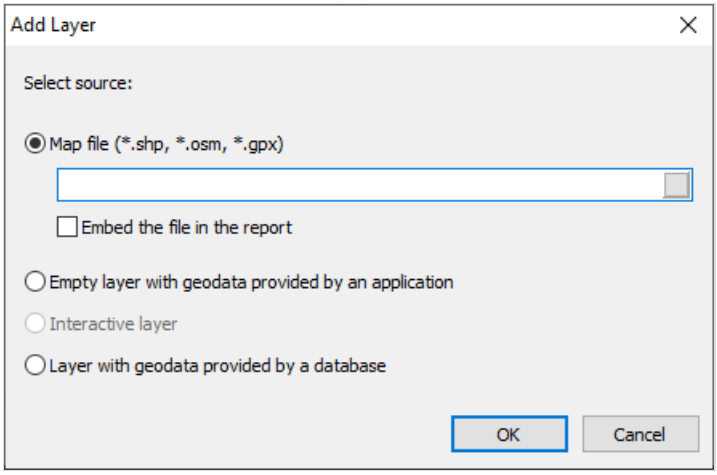


Adding map layers

The "Map" object can contain one or more layers. The layer structure is displayed in the upper left part of the editor window:



To add a new layer, click on the "Add..." button, which opens the following dialog:



At this point, choose the type of the layer, from:

- the map from an ESRI shapefile. This is the most commonly used type for maps. For example, you can display the world map and highlight some countries in color, depending on the sales level for these countries;
- the geodata from your application. Your application must provide the following data: latitude, longitude, name and value. This data is displayed as a small point on the layer. The point can have a caption and can also have a variable size/color, dependent on the value provided. In practice, this type of layer is used together with a base layer of "ESRI shapefile" type. The base layer (the first one) is configured to display a country map (for example) and the second layer (geodata from an application) displays points - city names where sales occurred. The color and size of a point can be configured to reflect sales level.

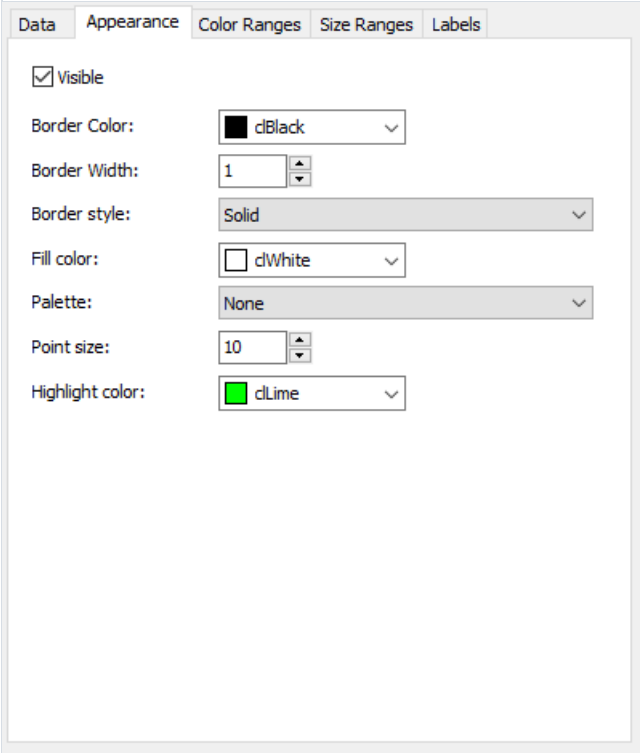
If the "ESRI shapefile" layer type is chosen, select how the map data is stored, from:

- the whole map data is embedded in the report file. In this case the report file (.fr3) may be large;
- the report file holds a reference to the map files (.shp/.dbf). This mode is useful if several reports use the same map files.

> Note: Large map files (more than 30Mb) or map files containing a lot of polygons (more than 20,000) will slow down report generation.

Setting up the layer appearance

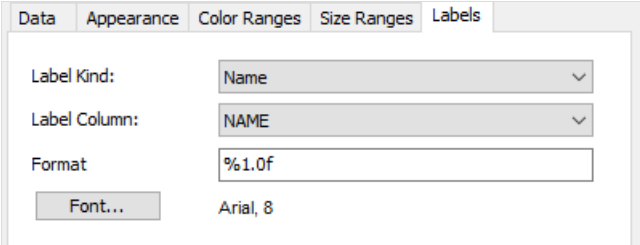
To set the layer appearance, choose the layer and switch to the "Appearance" tab:



Set the border color and style of the map polygons and choose the color palette. Note that the palette is ignored if you configure the color scale (more about this later).

Setting up label display

The map can display labels, such as country names. Set the label type and appearance on the "Labels" tab:



If the "ESRI shapefile" layer type is chosen set the field name which contains the displayed information. In most cases it's a "NAME" field. The world map included in the FastReport demo program contains the following fields:

- `NAME` (eg: Germany)
- `ABBREV` (eg: Ger.)
- `ISO_A2` (eg: DE)
- `ISO_A3` (eg: DEU)

Other maps will have a different set of fields.

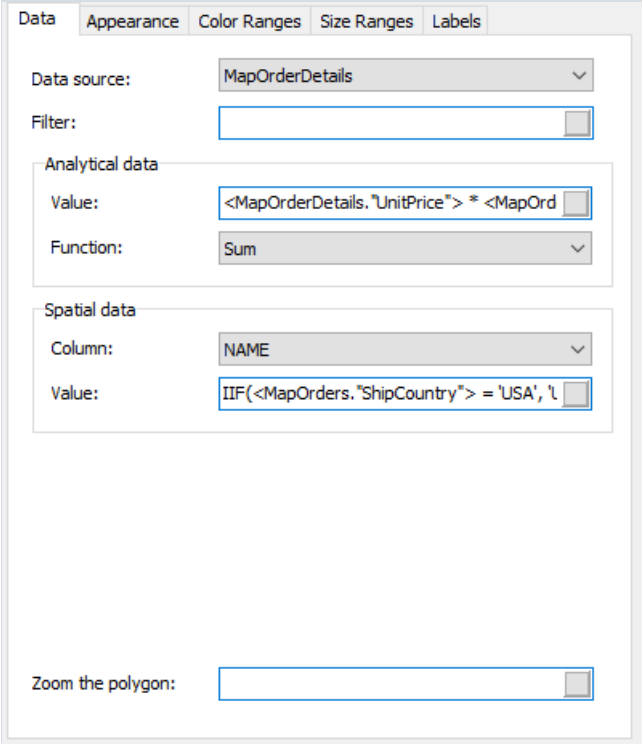
If the "application geodata" layer type is chosen set the minimum zoom value for displaying the labels. The default value is 1, meaning the labels are always displayed.

Connecting the map layer to data

Most reports use the "Map" object to display analytical data, for example sales levels in various countries. So a map layer must be connected to a data source using the map object editor window. Select the layer from the layers tree and switch to the "Data" tab. Data appropriate for the map layer type must be provided:

ESRI shapefile

For map layer of type "ESRI shapefile" the "Data" tab looks like:



The data required is:

- name (eg: the country name);
- numeric value (eg: sales volume in this country).

For example, a "Sales" data source may have this data:

```
Country  SalesTotal
-------  ----------
USA          500000
Germany     1200000
Russia       300000
```

Set up the "Data" tab in the following way:

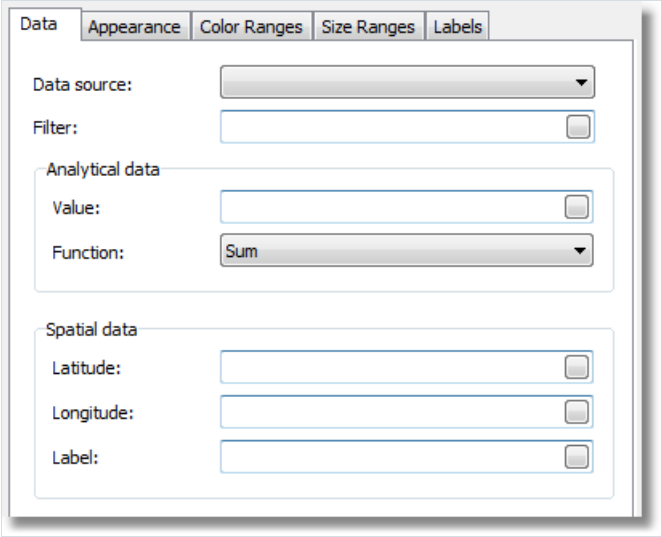
- data source: `Sales`
- spatial data, column name: select the column containing country names; usually this is the "NAME" column
- spatial data, value: `[Sales.Country]`

- analytical data, value: `[Sales.SalesTotal]`
- analytical data, function: `Sum`; this function is used if you have several values per country.

The "Zoom the polygon" edit box allows zooming of the polygon with a given name, so it occupies the whole "Map" object workspace. For example, to zoom Germany on the world map, type 'Germany' (with quotes) in this edit box.

Geodata from an application

For map layer of type "Geodata from an application" the "Data" tab looks like:



And the data required is:

- spatial data : latitude and longitude;
- label (eg: a city name);
- numeric value (eg: sales volume in this city).

For example, a "Sales" data source may have this data:

```
Latitude  Longitude  CityName  SalesTotal
--------  ---------  --------  ----------
48.13641  11.57753   München        50000
50.94165  6.95505    Köln           36000
```

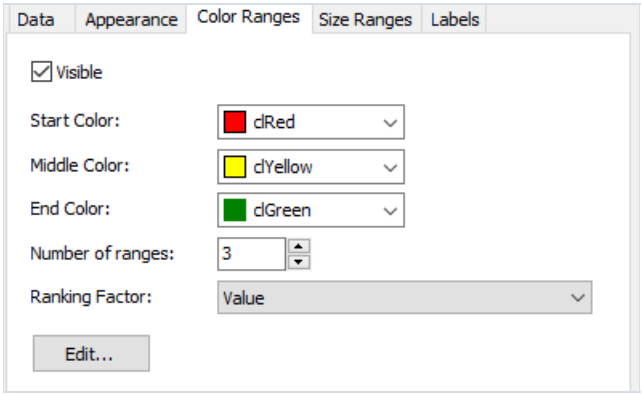
Set up the "Data" tab in the following way:

- the data source: `Sales`
- spatial data, latitude: `[Sales.Latitude]`
- spatial data, longitude: `[Sales.Longitude]`
- spatial data, label: `[Sales.CityName]`
- analytical data, value: `[Sales.SalesTotal]`
- analytical data, function: `Sum`; this function is used if you have several values per city.

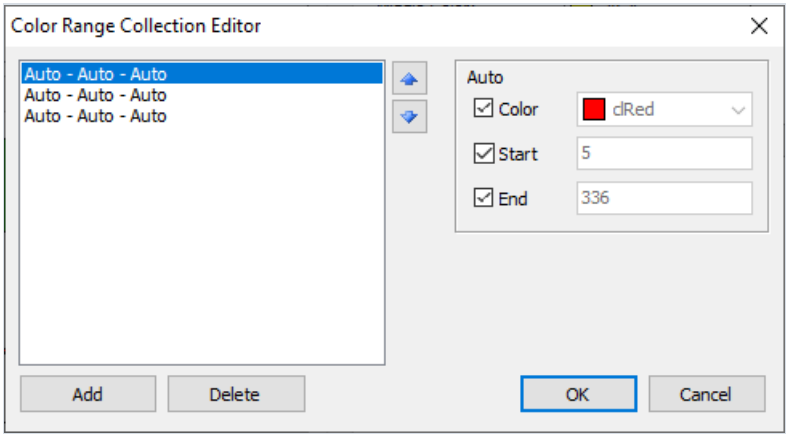
Highlighting data using colors

After you have finished connecting a layer to its data, the next question is - how to display analytical data (for example the sales volumes in various countries)? The easiest way is to set up labels displaying a country name and its sales (see "Setting up label display").

More visual impact can be achieved by painting each country with a color, dependent on its sales volume. For example, low sales as red, mid sales as yellow and high sales as green. This is done by setting up the color scale on the "Color scale" tab:

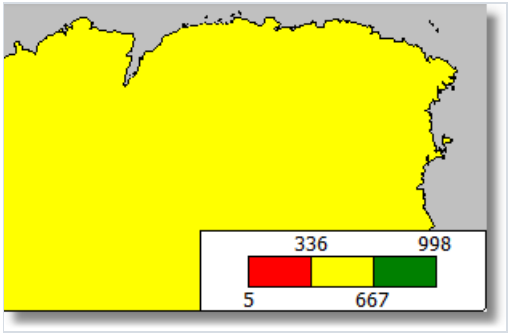


The color scale consists of several ranges. Each range has the following properties: min value, max value and color. You can use as many ranges as you need. To set up the color scale set the number of ranges first and then the properties for each range.

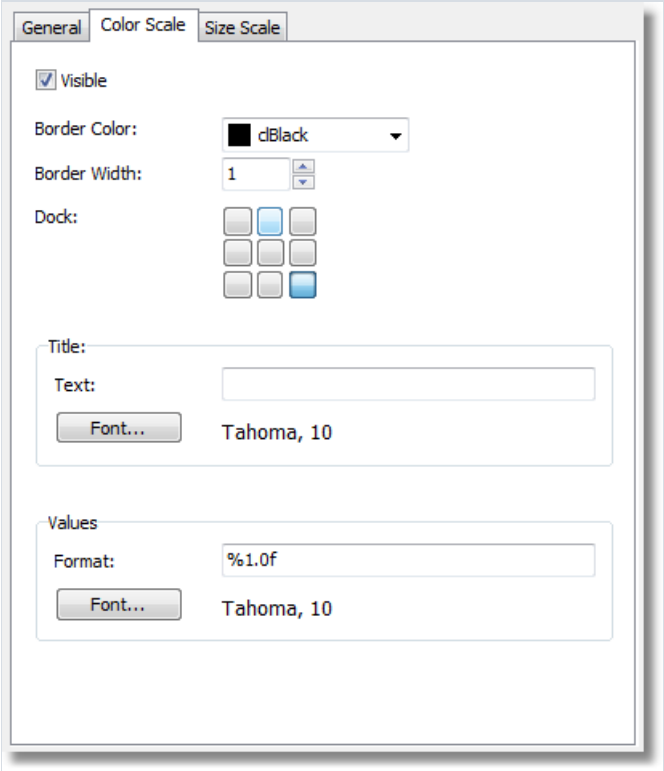


By default all range properties are set to "Auto", meaning FastReport calculates the minimum and maximum values for each range automatically. The auto color is chosen from three presets ("Start color", "Middle color", "End color"). The "Auto" mode may be suitable in most cases.

When a color scale is set up, an indicator control is displayed in the bottom part of the "Map" object:

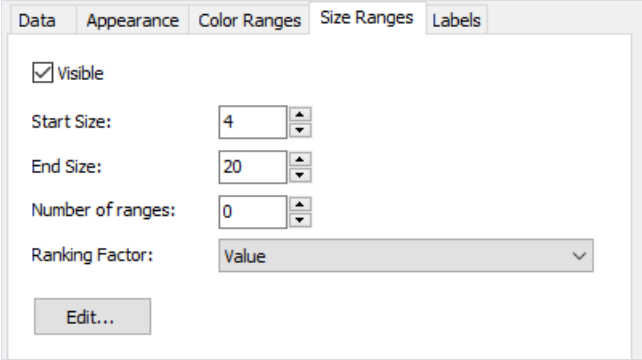


To set the appearance and position of the indicator, select the "Map" element in the layers tree control and switch to the "Color scale" tab:



Highlighting data using size

When using layer type “geodata from an application”, the data is displayed as a small circle with a caption. The size of the circle can be bound to the data in the same way as highlighting data using colors. To do this set up the size ranges on the "Size ranges" tab:



The size ranges have the following properties: min value, max value and size (in pixels). You can use as many ranges as you need. Set the number of ranges first and then set the properties for each range.

By default all range properties are set to "Auto", meaning FastReport calculates the minimum and maximum values for each range automatically. The auto size is chosen from two presets ("Start size", "End size"). The "Auto" mode may be suitable in most cases.

Interactive reports

A FastReport's prepared report can be made interactive. This means that, it will react to the user's actions in the preview window. You can use the following interaction:

- when clicking on the report object, some kind of operation is performed. For example, you can run detailed report and show it in a separate window;
- preview window can show the report outline, which can be used for navigating on the report.

Hyperlink

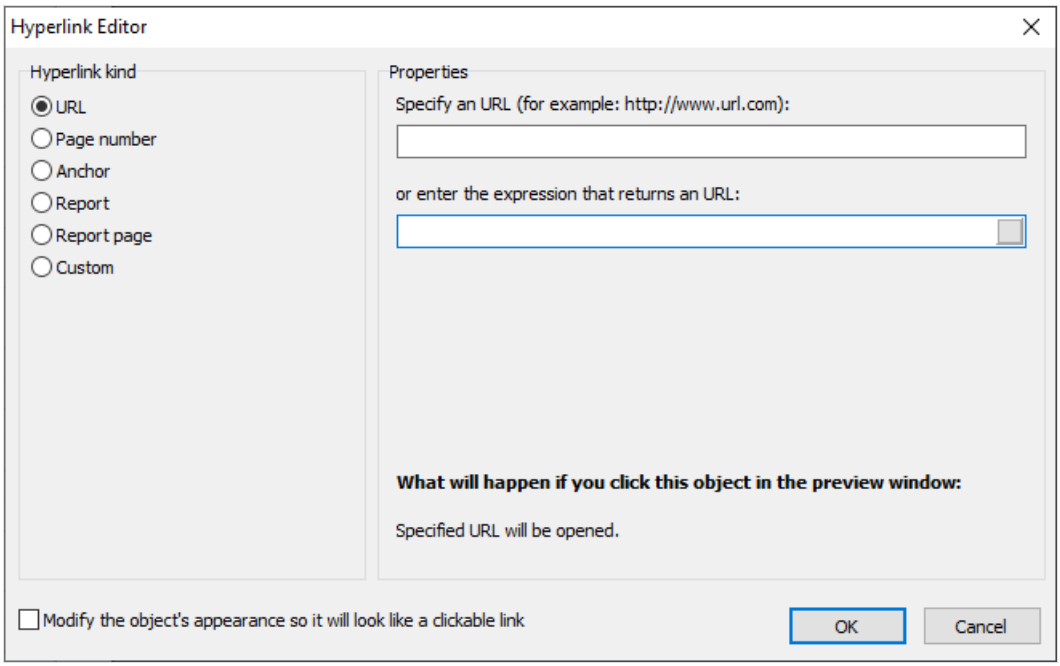
Almost all report objects have the `Hyperlink` property. Using this property, you can define an object's reaction to the mouse click in the preview window.

When clicking such an object, one of the following can occur:

- navigate to the URL address;
- send e-mail;
- execute any kind of system command;
- navigate to the report page with the indicated number;
- navigate to the anchor, added in a script;
- run detailed report in a separate preview window;
- custom action, defined in a script.

Hyperlink configuration

To configure a hyperlink, select the object which you want to make interactive, and right click on it. In the context menu, select the "Hyperlink..." item. The hyperlink editor window will open:



Choose the type of hyperlink by selecting the radiobutton in the left side of the window. After you have done, you may click the "Modify the object's appearance..." checkbox at the bottom of the window. The appearance of the object will change in the following way:

- blue color will be set for the text and it will underlined;
- a hand cursor form will be set.

In some cases hyperlink needs to be shown in the preview window, but there is no need to print it. This is easy to do, if you configure the "Visibility" object property. This can be done in the "Object Inspector" window.

Link to the URL

Using this type of link, you can:

- navigate to the given internet address;
- execute some kind of system commands, for example, "mailto:" for sending an email.

You can indicate the value of the link by using two methods:

- indicate the value directly, for example, "http://www.fast-report.com";
- indicate an expression, which returns the value of the link. This expression will be calculated when you run the report.

Link to the page number

By using this link type, you can organize navigation on the pages of a prepared report. Most often, navigation to the first page is used. For this, indicate the page number (1 in the given case) as a link value.

You can indicate the page number by using two methods:

- indicate the number directly, for example,1;
- indicate an expression which returns the page number. This expression will be calculated when you run the report.

Link to an anchor

By using this type of link, you can navigate to an anchor added in a script. An anchor has got a name and a position in a prepared report (page number and position on the page). When moving to an anchor by its name, you navigate onto the indicated position.

In order to use this link type, you first need to define an anchor. In order to do this, use the script and `Engine.AddAnchor` method (see more in the Anchors topic). Now you can indicate its name in the hyperlink configurations window. This can be done by using two methods:

- indicate the name of the anchor directly;
- indicate an expression which returns the name of the anchor. For example, this can be a data column. This expression will be calculated when the report is run.

Link to a detailed report

Using this link type, you can execute another report and show it in a separate preview window.

You must set the following parameters for this type of hyperlink:

- detailed report's name;
- name of the report's variable, which will take the hyperlink's value;
- hyperlink value.

When the link is clicked, the following will take place:

- the indicated report will be loaded;
- the report's variable will be set to the hyperlink's value;
- the report will be built and run in a separate preview window.

Report variable's value can be indicated by using two methods:

- indicate the value directly;
- indicate an expression, which returns the value. This expression will be calculated when the report is run.

Link to a detailed page

This link type works in the same way, except that, another page in the current report is used as a detailed report. For this, your report must contain at least two pages: one with the main report, another with detailed one.

You must set the following parameters for this type of hyperlink:

- page name in that report;
- name of the report's variable, which will take the hyperlink's value;
- hyperlink value.

When the link is clicked, the following will take place:

- the report's variable will be set to the hyperlink's value;
- the indicated report page will be built and shown in a separate preview window.

Report variable's value can be indicated by using two methods:

- indicate the value directly;
- indicate an expression, which returns the value. This expression will be calculated when the report is run.

When you choose a report page, its `Visible` property resets to false. This means that, when the main report will be built, this page will be skipped.

Custom link

Using this type of link you can define own reaction to the clicking of the mouse. For this, use the "OnClick" event handler of the object.

Dot-Matrix reports

Earlier we looked at reports intended for printing on standard modern printers (laser, etc.). A report sent to a dot-matrix printer will be printed very slowly. FastReport allows the creation of special reports intended for dot-matrix printers, where only standard font symbols and no graphic elements are output; this results in a faster printing speed.

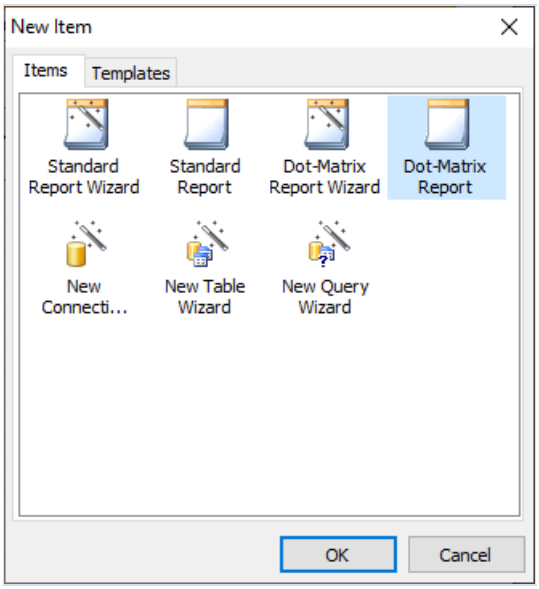
Let's see how to build a report of "List" type intended for dot-matrix printing. Earlier we created this kind of report, see the "List of clients" report. We will use the same data for the report.

So, start a new project in Delphi, place `TTable`, `TfrxDBDataSet`, `TfrxReport` and `TfrxDotMatrixExport` components on the form and set their properties:

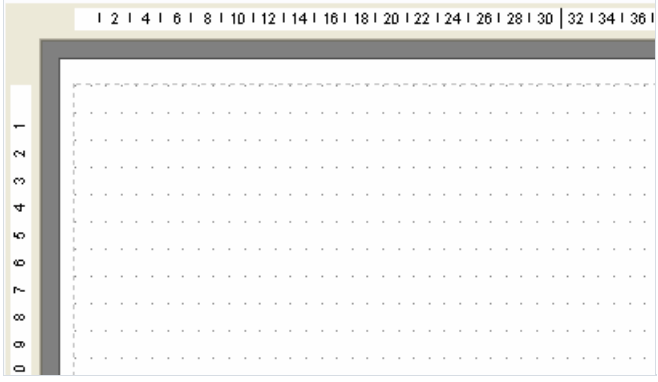
```
TTable:
DatabaseName = 'DBDEMOS'
TableName = 'Customer.db'

TfrxDBDataSet:
DataSet = Table1
UserName = 'Customers'
```

Open the report designer. Select "File > New..." to open the report wizard dialogue and select the "Dot-Matrix Report" item:



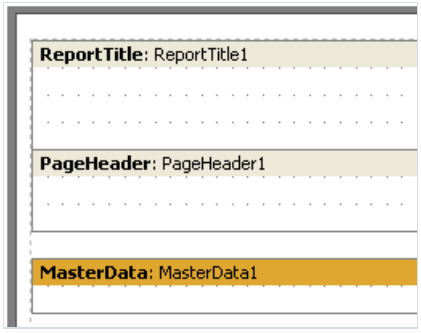
On clicking OK an empty design page is shown, marked out for a dot-matrix font:



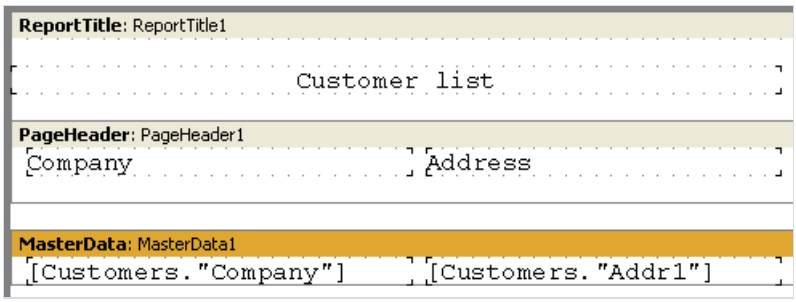
The object toolbar also changes to show the objects available for dot-matrix printing, these are “Band”, “Text”, “Line”, “ESC-Command”, “Subreport”, “Cross-tab” and “DBCross-tab” objects. Other objects cannot be used with a dot-matrix printer.



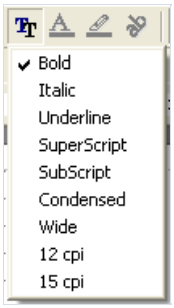
Place “Report Title”, “Page Header” and “Master Data” bands on the report page:



Place “Text” objects on the bands as follows:



The placement of Dot-matrix objects is similar to that in ordinary reports. However, these objects are strictly limited in position and appearance. The objects snap to the grid, the font size (height) cannot be changed and they cannot be colored. But some font attributes can be modified by selecting the “Text” object and clicking the “Tt” button on the toolbar:



These font attributes are specific to dot-matrix printing. The report page and all the dot-matrix objects, excepting bands, have these attributes.

> Note: in the designer and in the preview only “Bold”, “Italic” and “Underline” attributes are implemented on screen. The whole set of attributes is implemented only when printing.

Let's modify our report using the “Bold” style for the headings. The report is finished and ready for previewing:

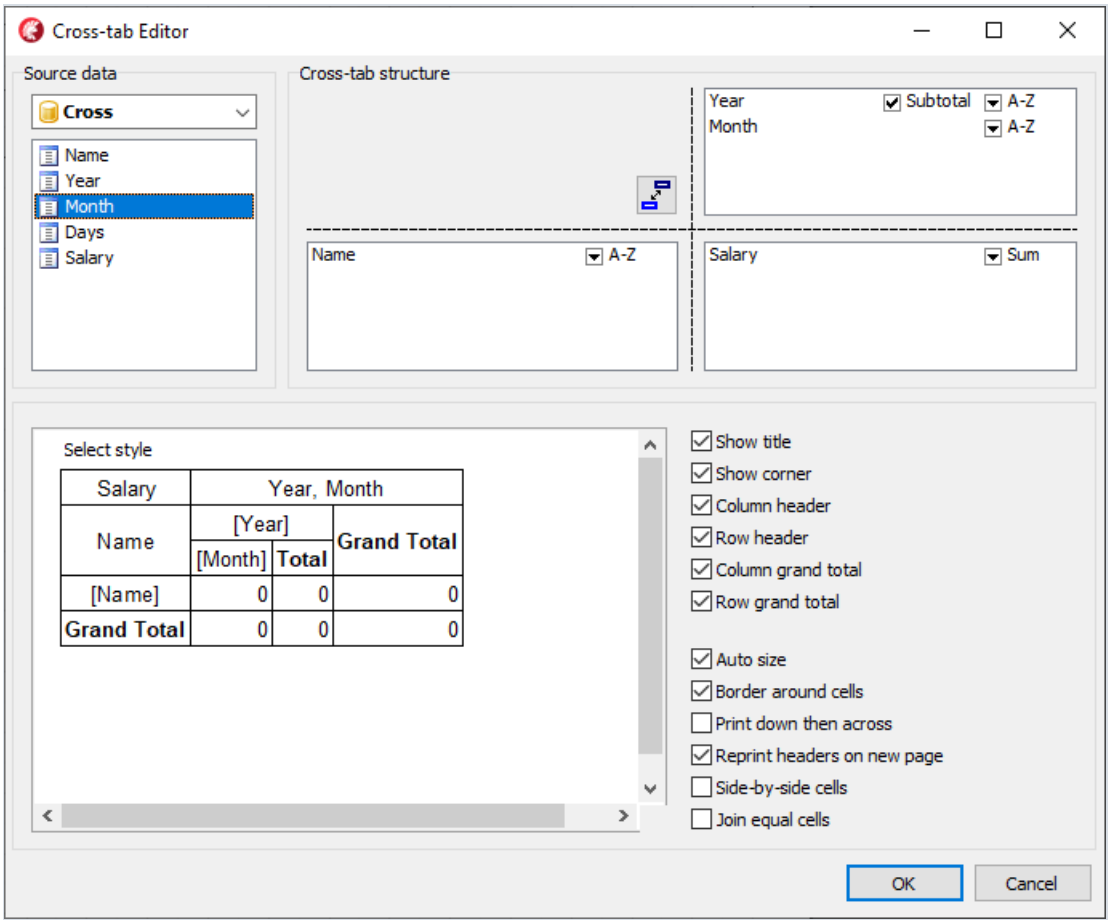
Customer list

Company	Address
Action Club	PO Box 5451-F
Action Diver Supply	Blue Spar Box #3
Adventure Undersea	PO Box 744
American SCUBA Supply	1739 Atlantic Avenue
Aquatic Drama	921 Everglades Way

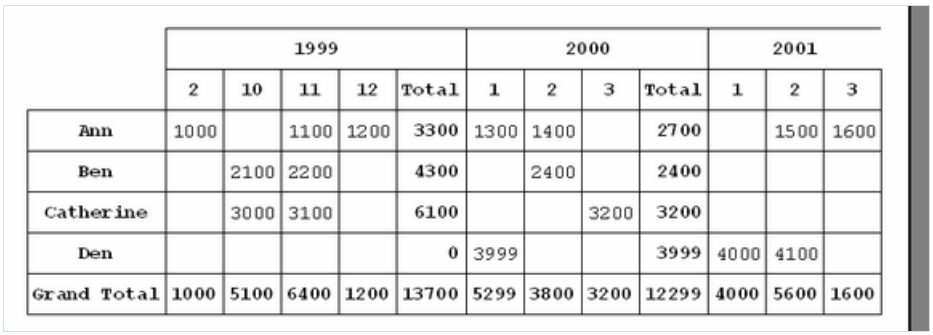
Cross-tab in dot-matrix

The range of objects usable in dot-matrix reports is restricted to those that can be displayed in a textual form. Among these are the two “Cross-tab” objects. Let's look at a simple cross-tab report similar to the one built earlier in the “Tables with composite headers” section.

Create a dot-matrix report as in the previous section, choosing a “Dot-Matrix Report”. Place a “DB cross-tab” object on the report page and open its editor:



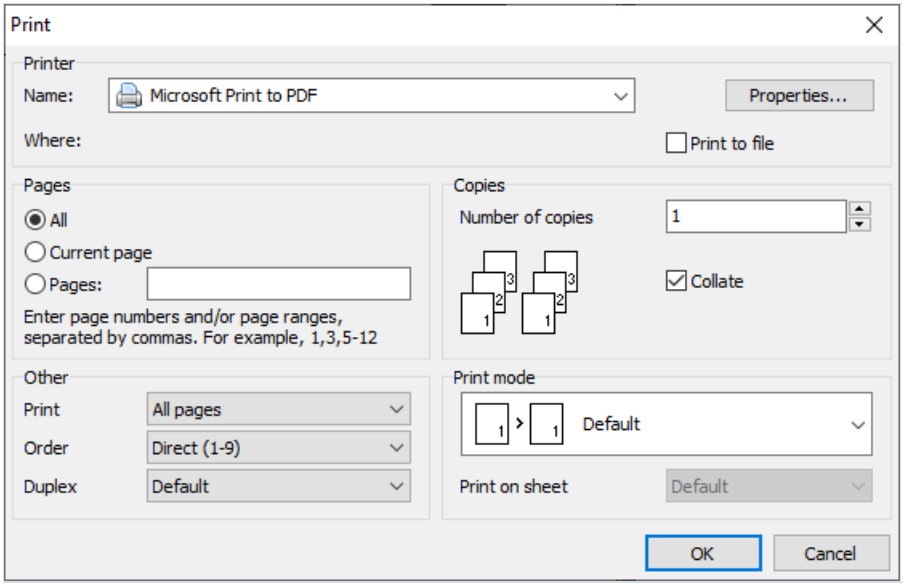
The cross-tab editor shows the structure of the output table in dot-matrix mode. The cell font attributes can be set using the “Tt” button in the toolbar. In all other respects the cross-tab objects behave as previously described. The previewed report looks like this:



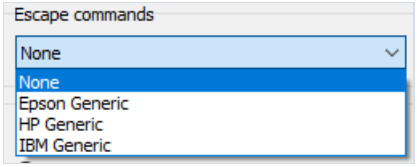
	1999					2000				2001		
	2	10	11	12	Total	1	2	3	Total	1	2	3
Ann	1000		1100	1200	3300	1300	1400		2700		1500	1600
Ben		2100	2200		4300		2400		2400			
Catherine		3000	3100		6100			3200	3200			
Den					0	3999			3999	4000	4100	
Grand Total	1000	5100	6400	1200	13700	5299	3800	3200	12299	4000	5600	1600

Dot-matrix report printing

To enable the actual printing of a previewed dot-matrix report it is necessary to first place a `TfrxDotMatrixExport`  component from the "FastReport" component palette on your Delphi project form. This component handles the conversion of the preview report into the text format required by dot-matrix printers. It also replaces the normal printer dialogue with a special one for dot-matrix printers:



The printer dialogue is similar to the normal one, but has dot-matrix printer specifics. The printer's commands system must be chosen from the following list before printing (the ESC-commands):

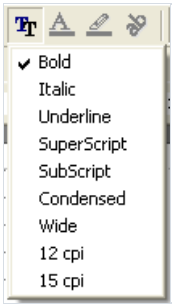


There are also some other options for dot-matrix printing:

- Print to file : whether to send the print stream to a file on the hard disk. If enabled then the normal 'Save As...' dialogue appears
- Page breaks : whether a "Page break" control command is sent on reaching the bottom of the page. If disabled then printing is unbroken on continuous stationery
- OEM-codepage : whether to perform symbol conversion
- Pseudographic : specifies how to draw vertical and horizontal lines. If disabled then lines are drawn using -, | and + symbols

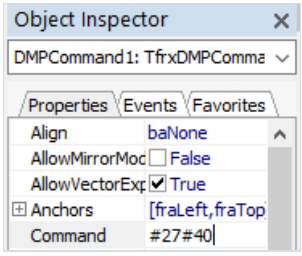
“Command” object

As described earlier, these attributes can be set for dot-matrix report objects:



This is a standard set that is understood by all models of dot-matrix printer. Certain printer models can support other commands that are not present in the standard set, for example printing at 20 character per inch resolution. To be able to send these extra attributes to the printer use the “ESC-Command” object in the report.

Place the “ESC-Command” object at the correct place on the report page, before any objects which use the non-standard attributes (e.g. in the top left corner of the page). To set a command edit the Command property of the object in the object inspector:

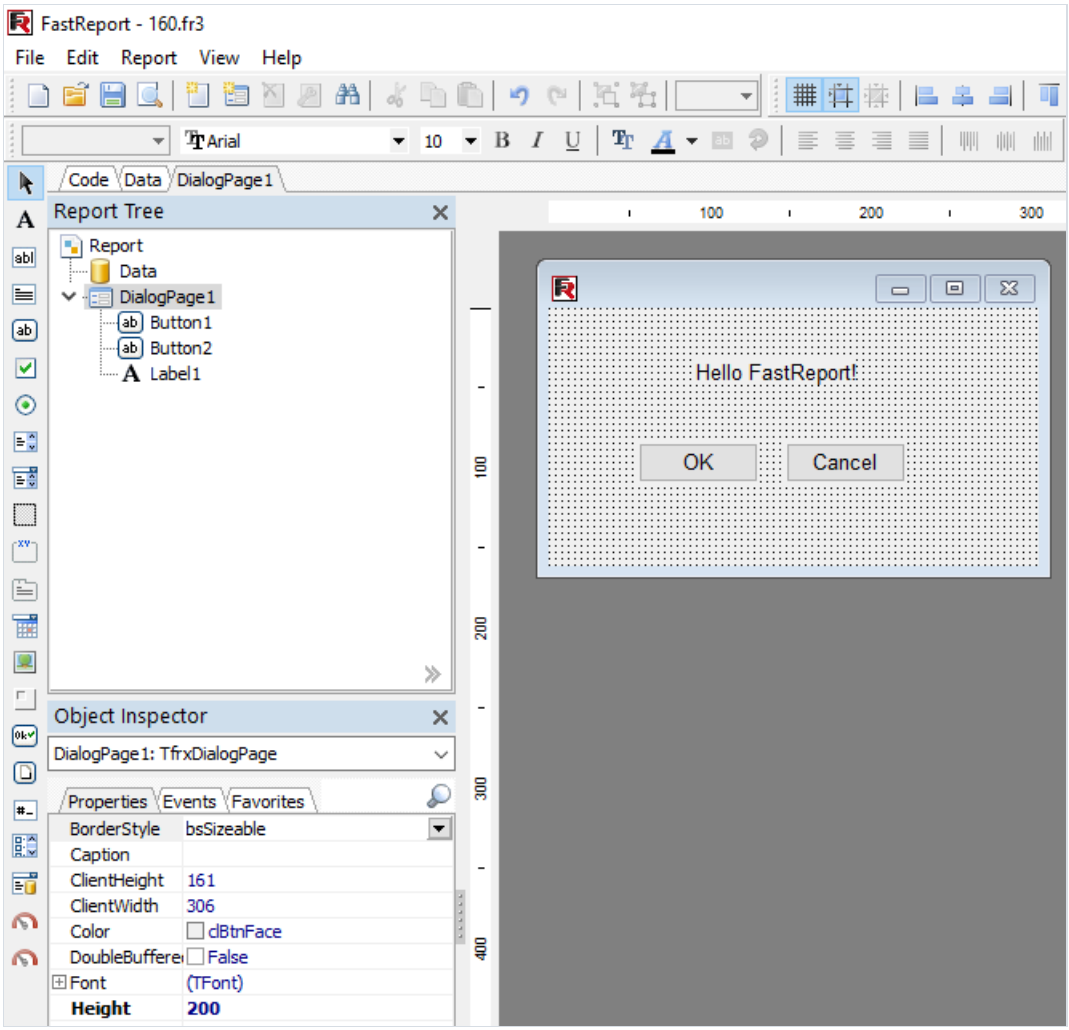


The property can be set using one of the two formats :

decimal (e.g. `#27#40`) or hexadecimal (e.g. `1B28`).

Dialogue forms

As well as the usual report design pages, you can use dialogue forms in a report. Dialogue forms are created in the usual report designer using the button in the designer toolbar: the button adds a new dialogue page tab to the report. When switching to the dialogue page tab the designer workspace changes to show the form, and the object toolbar changes to show the controls which can be placed on the form:



Controls

To use Dialogue form controls in a report place a `TfrxDialogControls`  component from the FastReport component palette on the Delphi project form. Alternatively add "frxDCtrl" to the "Uses" list. The following controls then become available for use in the report:

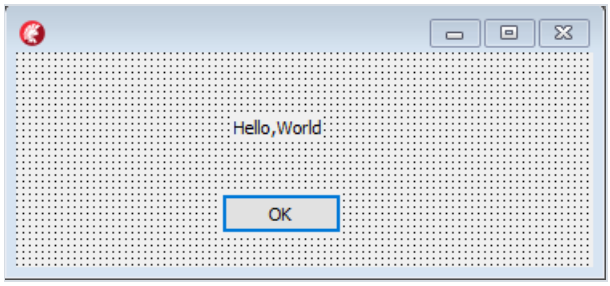
Element	Name	Description
	TfrxLabelControl	control for displaying fixed texts on the dialogue form
	TfrxEditControl	control for entering text using the keyboard
	TfrxMemoControl	control for entering multiple lines of text using the keyboard
	TfrxButtonControl	control representing a button
	TfrxCheckBoxControl	control representing a flag, having two states - enabled or disabled; has an adjacent label
	TfrxRadioButtonControl	control representing a radio button. There must be more than one radio button control on the form
	TfrxListBoxControl	control representing a list of items from which one can be selected
	TfrxComboBoxControl	control representing a drop-down list of items from which one can be selected
	TfrxDateEditControl	control representing a field with a drop-down calendar for date entry
	TfrxGroupBoxControl	control representing a labelled box that can contain other objects
	TfrxPanelControl	control representing a panel which can contain other objects
	TfrxBitBtnControl	control representing a button containing a picture
	TfrxSpeedButtonControl	control representing a speed button containing a picture
	TfrxMaskEditControl	control for entering text using the keyboard and conforming to a template
	TfrxCheckListBoxControl	control representing a list of items, selected by means of a checkbox
	TfrxBevelControl	control used for showing lines and frames on the form
	TfrxImageControl	control representing a picture in "BMP", "ICO", "WMF" or "EMF" format

All the controls are similar to those used in Delphi. See the FastReport component help for information on the properties, events and methods of each control.

"Hello, World!" report

In this example we will create a report which, before creating the preview, displays a greeting window using a dialogue form.

Create a new project in Delphi and place `TfrxReport` and `TfrxDialogControls` components on the form. Open the FastReport designer by double-clicking on the `TfrxReport` component and add a dialogue form to the report. Place "Label" and “Button” controls on the form:



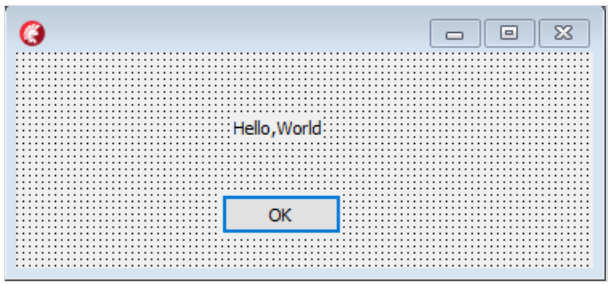
Set the object properties:

```
Label:
Caption = 'Hello, World!'

Button:
Caption = 'OK'
Default = True
ModalResult = mrOk
```

Set the form's `BorderStyle` property to `bsDialog`. Both the controls and the form have the same set of properties as for the corresponding Delphi controls.

When the dialogue form design is complete return to the report design Page1 tab and place a “Text” object containing the greeting on the page. Preview the report and you will see the dialogue form:

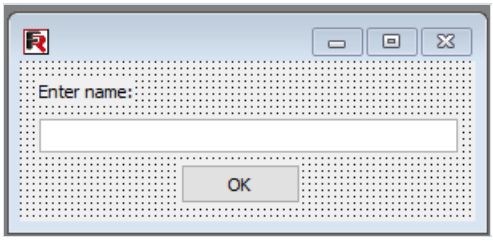


After clicking on the OK button the dialogue closes and the report is constructed and displayed. If the dialogue is closed by clicking the system menu “X” button then the report will not be constructed.

FastReport works like this: if there are dialogue forms in a report, the report is only constructed after each dialogue has been closed with ModalResult = mrOk, i.e. in this example by clicking the OK button,. That is why the “ModalResult” property of the button had to be set to “mrOk.”

Entering parameters and transferring them into a report

Let's make this example more complicated, to show how to transfer values entered in the dialogue to the report. Modify the dialogue as shown below:



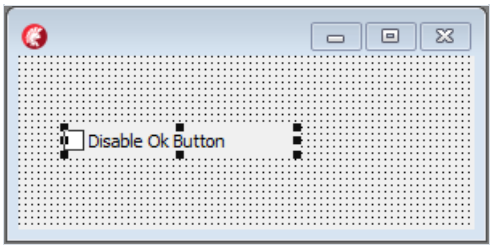
Place a “Text” object containing this text on the report Page1, and set its `AutoWidth` to True:

```
You entered:
[Edit1.Text]
```

Preview the report and make sure that the text typed in is successfully displayed in the report. Other objects in the dialogue can be accessed in a similar way. Since each object has a name that is unique within the whole report it can be used anywhere within the same report.

Interaction with controls

By using a script, complex logic can be incorporated into a dialogue's operation. Let's illustrate this with a simple example. Modify the form like this:



Double-click on the “CheckBox” object to create an “OnClick” event handler and enter the following script:

PascalScript:

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

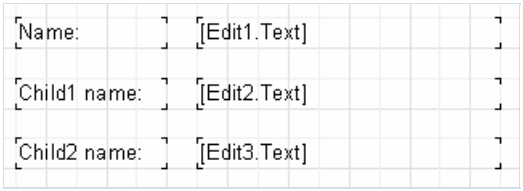
C++ Script:

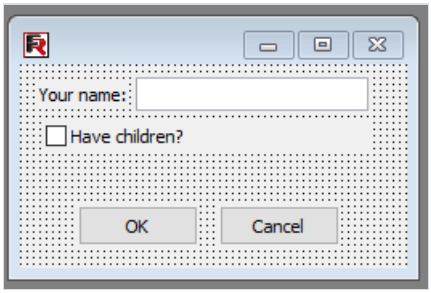
```
void CheckBox1OnClick(TfrxComponent Sender)
{
  Button1.Enabled = !CheckBox1.Checked;
}
```

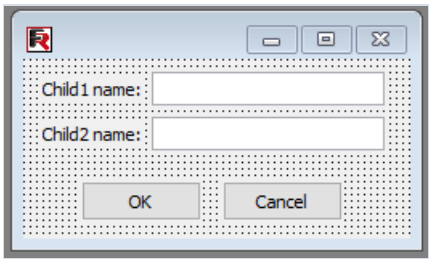
This code is the same as is used in Delphi. On running the report the button responds to the state of the check box.

Several dialogue forms

Let's see how a report with two dialogues works. Create a report with two dialogues and one design page:







Set the `ModalResult` property for the OK and Cancel buttons to `mrOk` and `mrCancel` respectively. Now run the report.

First of all the user will be asked to answer questions from the first dialogue (name, any children), then after clicking OK, from the second one (childrens' names). After clicking OK in the second dialogue the report will be built.

FastReport handles multiple dialogues in this way: the dialogues are displayed in their creation order; each dialogue is displayed only after the previous one has been closed with "ModalResult = mrOk" (in this example by clicking each OK button). If any dialogue is closed using the Cancel button or the system menu "X" button then building of the report is prevented.

Managing dialogue forms

In the previous example the second dialogue is displayed irrespective of whether "Have children" was checked. Let's show how to hide the second dialogue when this flag is unchecked. Create an "OnClick" handler for the OK button on the first dialogue (double-click on the button to create the handler):

PascalScript:

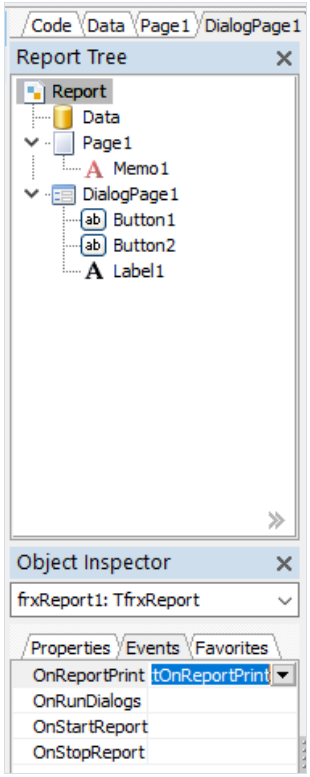
```
procedure Button1OnClick(Sender: TfrxComponent);
begin
  DialogPage2.Visible := CheckBox1.Checked;
end;
```

C++Script:

```
void Button1OnClick(TfrxComponent Sender)
{
  DialogPage2.Visible = CheckBox1.Checked;
}
```

This code hides the second dialogue (DialogPage2) if the flag is not checked. Preview the report to see that this works correctly.

Another way of managing the forms is to use the "OnRunDialogs" report event. To create this event handler select the Report object in the report tree or object inspector and switch to the "Events" tab in the object inspector. Double-click on the "OnRunDialogs" event to create a handler:



Write the following code in the handler:

PascalScript:

```
procedure frxReport1OnRunDialogs(var Result: Boolean);
begin
  Result := DialogPage1.ShowModal = mrOk;
  if Result then
  begin
    if CheckBox1.Checked then
      Result := DialogPage2.ShowModal = mrOk;
  end;
end;
```

C++Script:

```
void frxReport1OnRunDialogs(bool &Result);
{
  Result = DialogPage1.ShowModal == mrOk;
  if (Result)
  {
    if (CheckBox1.Checked)
      Result = DialogPage2.ShowModal == mrOk;
  }
}
```

The handler works like this:

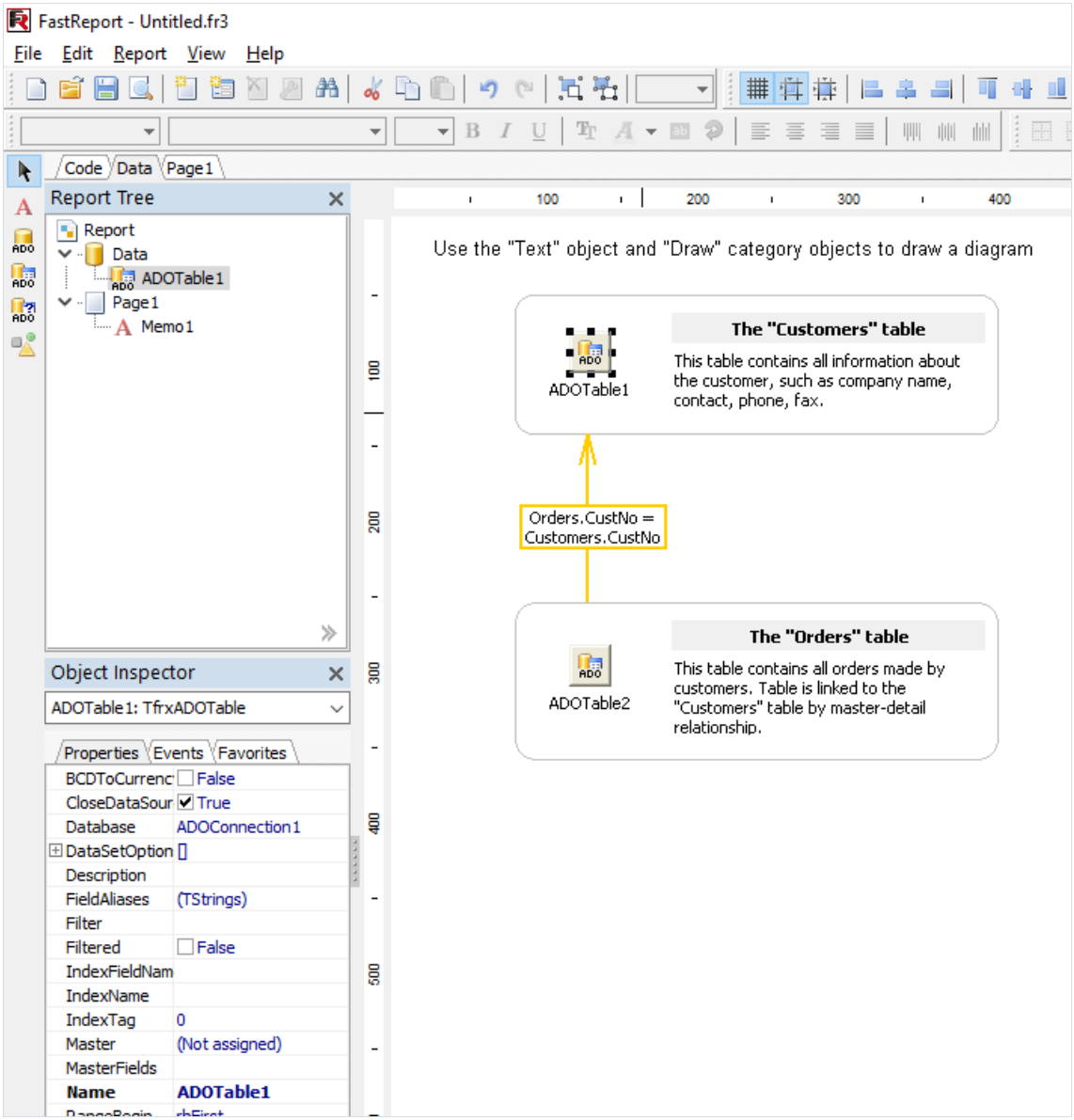
- the first dialogue is shown
- if it is closed via the OK button then look at the state of CheckBox1
- if this state is Checked then show the second dialogue
- if the second dialogue is closed via the OK button then set Result to True.
- If the handler returns Result = True then the preview is built; if Result = False then the report stops running without building a preview.

Data access components

As a rule most reports are based on data sourced from a DB. Delphi provides a variety of components for linking to DB data and FastReport makes use of these links. Here we discuss the use of `TTable` and `TQuery` components as data sources for reports, but in general any `TDataSet` descendant can be used.

Apart from this, a report in FastReport can contain its own data access components which you can create and configure at run-time. The principles of data access in FastReport are much the same as those used in the Delphi environment. Just as in Delphi a component is placed on a dialogue form and its properties are set in the object inspector.

Component design is very flexible and new components to support different database engines can easily be created (see the Developers Manual).



Component descriptions

Let's see how components are used for data access via ADO. They are made available in FastReport by adding the `TfrxADOComponents`  component from the Delphi FastReport palette to the project form.

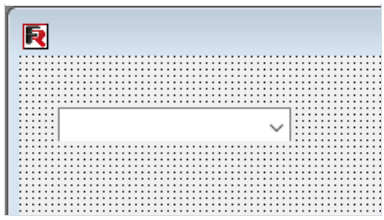
The following objects then appear on the object toolbar when you switch to the "Data" tab in the report designer : `TfrxADOTable` , `TfrxADOQuery` and `TfrxADODataBase` . These components are similar to the corresponding Delphi components (`TADOTable` , `TADOQuery` and `TADOConnection`) in terms of their functionality. There is also a `TfrxDBLookupComboBox` control which can be used in dialogue forms.

Icon	Name	Description	Used in
	TfrxDBLookupComboBox	control used for selecting a value from a directory	dialogue forms
	TfrxADOTable	control used for accessing a DB table	"Data" tab
	TfrxADOQuery	control used for executing a SQL query	"Data" tab
	TfrxADODataBase	control used for connecting to a DB	"Data" tab

Let's look at each component.

TfrxDBLookupComboBox

This component is used for selecting a value from a look-up dataset.



The component has the following properties:

Property	Description
`DataSet`	data source to which the control is connected
`ListField`	name of DB field displayed in the control
`KeyField`	name of DB key field identifying the selected record
`KeyValue`	value of DB key field returned by the selection from the list
`Text`	value of DB list field displayed in the list
`AutoOpenDataSet`	when True the connected data source is opened automatically after the dialogue's OnActivate event

To connect the control to the look-up dataset enter values for the three properties : `DataSet` , `ListField` and `KeyField` .

> Note that the returned value is available via either the `Text` or the `KeyValue` properties, neither of which appears in the object inspector. They are accessible only through code. The initial position of the cursor in the look-up dataset can be set in code using `KeyValue` .

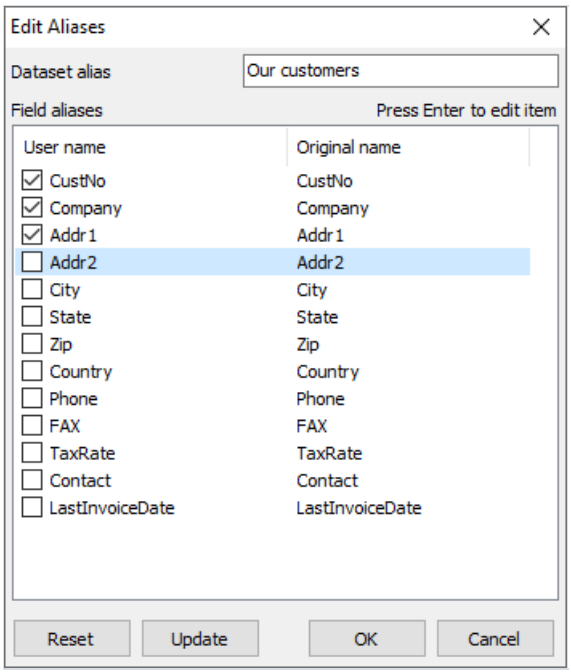
TfrxADOTable

This component is used for accessing a DB table using ADO. The component has the following properties:

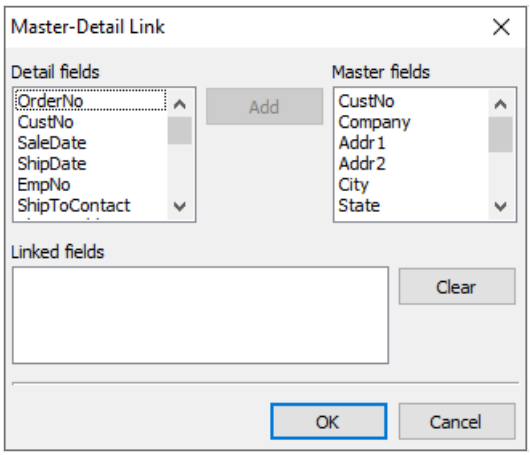
Property	Description
`DatabaseName`	connection name (name of the TfrxADODatabase component)
`FieldAliases`	enables aliases to be set for the dataset fields
`Filter`	expression for filtering records
`Filtered`	whether filtered or not
`IndexFieldNames`	names of index fields (for sorting)
`IndexName`	secondary index name
`MasterFields`	fields connected with master dataset
`Master`	master dataset
`TableName`	DB table name
`UserName`	alias for the dataset, used in code

These properties are similar to those of Delphi's `TADOTable` component. To connect the component to a DB table just set the `DatabaseName` and `TableName` properties. The Table is opened either by setting `Active` property to True or by calling the `Open` method.

The `FieldAliases` property editor is opened from the object inspector or by double-clicking on the component. The editor allows the selection of which fields will become available for use in code and the setting of aliases for these fields and for the dataset as a whole.



The `MasterFields` property editor is used for creating master-detail connections between two tables. To connect two tables with the master-detail relation a second table must be set in the `Master` property of the dependent table and the `MasterFields` property editor opened. If the table has secondary indexes which are to be used then set the `IndexName` property beforehand.



The editor visually binds the “master” and the “detail” fields of the datasets. When two datasets are connected in a “Master-Detail” relationship movement within the master dataset automatically filters the detail dataset so that only records belonging to the current record of the master dataset are shown.

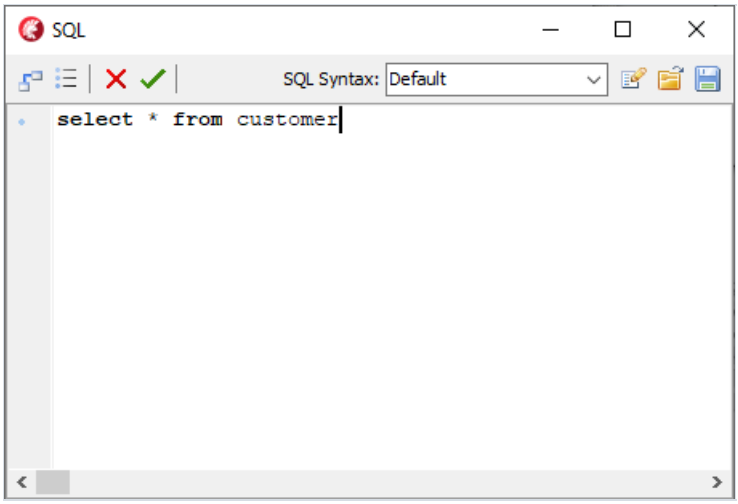
To connect fields in the two datasets select a field from the list on the left (the detail dataset) then a field from the list on the right (the master dataset) and click the Add button. The link between the two fields is then displayed in the bottom list. To clear the bottom list use the Clear button. The linked fields must of compatible type and be indexed.

TfrxADOQuery

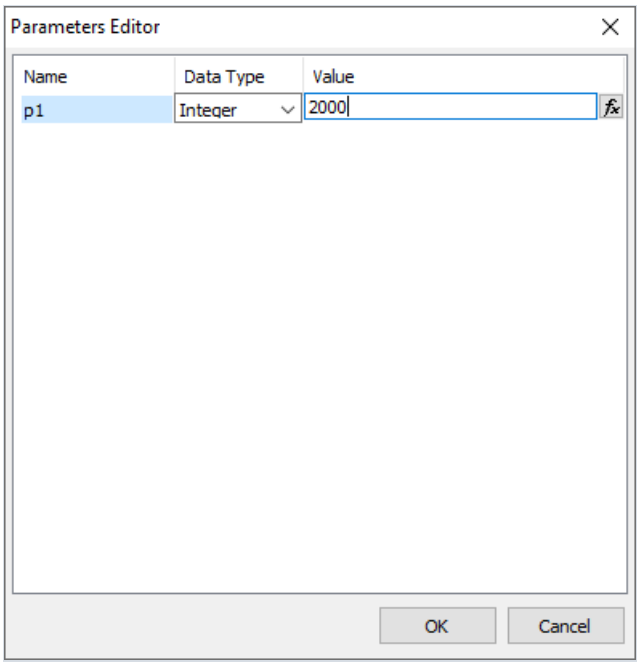
This component executes SQL queries on a DB. It has the following properties:

Property	Description
`DatabaseName`	connection name (name of the `TfrxADODatabase` component)
`FieldAliases`	enables aliases to be set for the dataset fields
`Filter`	expression for filtering records
`Filtered`	whether filtered or not
`Master`	master dataset
`Params`	list of query parameters
`SQL`	Query text
`UserName`	alias for the dataset, used in code
`IgnoreDupParams`	when True allows parameters with duplicate names. The name of the Query parameters will not be edited in the parameter editor

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



The `Params` property also has its own editor. It is enabled when a Query text contains parameters.



A parameter can be one of two types: its value either derived from the master-source or set as a discrete value (either an absolute value, as shown above, or linked to a variable or to an object property).

When a parameter is derived from the data master-set the `TfrxADOQuery.Master` property has to be set. This dataset must contain a field of the same name as the parameter. Neither the parameter type nor its value has to be specified.

TfrxADODataBase

This component is used to connect to a database. Its function is similar to the `TADOConnection` Delphi component. The component has the following properties:

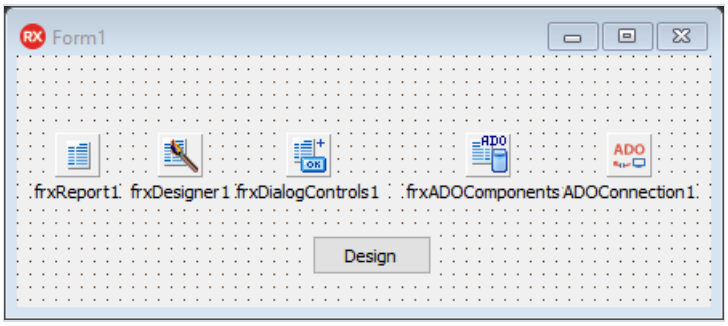
Property	Description
`Connected`	when True the connection is activated
`DatabaseName`	the ADO connection string
`LoginPrompt`	whether to prompt for the password when connecting to the DB

The `LoginPrompt` property defines whether to prompt for the password when connecting to the DB. When `LoginPrompt` is False a user name and password must be included in the ADO connection string.

Example of usage

Let's look at the design of a simple report using data access components at runtime. It will use the Demo.mdb database that comes with FastReport as the data source - "FR\Demos\Main\demo.mdb".

Create a new Delphi project and add one each of `TfrxReport`, `TfrxDesigner`, `TfrxDialogControls`, `TfrxADOComponents`, `TADOConnection` and `TButton` components to the form.



Setup the database connection by double-clicking on the `TADOConnection`, and choosing "Build connection string", the provider ("Microsoft Jet 4.0 OLE DB Provider") and the database ("FR\Demos\Main\demo.mdb"). Close the connection dialogue with OK and set the component properties here:

```
ADOConnection1:
LoginPrompt = False

frxADOComponents1:
DefaultDatabase = ADOConnection1
```

Define the following handler for the “Design” button:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  frxReport1.DesignReport;
end;
```

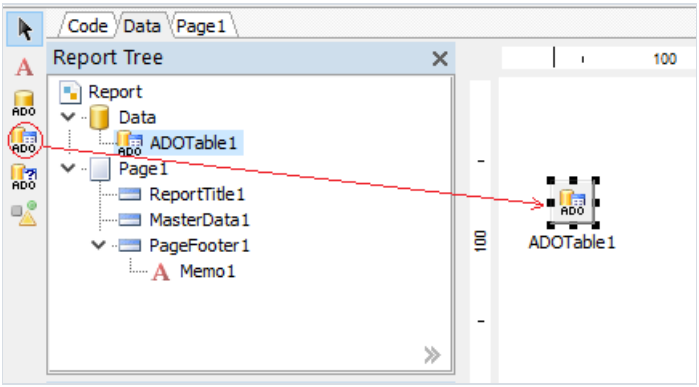
After that, compile and run the project. This is all you need to do to create an end-user runtime reports designer.

When the “Design” button is clicked the FR report designer opens, containing a blank report. Let's look at the design of a simple report in this environment.

Simple report of “List” type

This report will contain data from a single DB table. To construct the report, take these steps:

Click on the “New report” button on the designer toolbar - FastReport creates an empty report containing “Code”, “Data” and “Page1” tabs. Switch to the “Data” tab and place an “ADO Table” component on page:

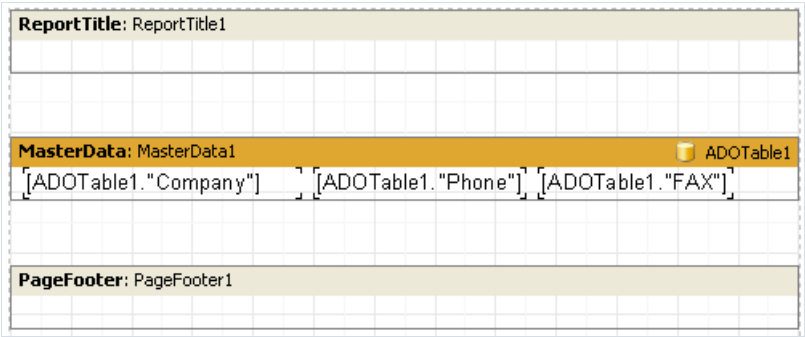


Note that the `Database` property is already connected to our database, because this was specified in the `TfrxADOComponents.DefaultDatabase` property. But the table name has to be set now:

```
TableName = 'Customer'
```

Switch to the Page1 tab. Connect the “MasterData” band to the table by double-clicking on it and selecting “ADOTable1” in the dialogue.

Drag the fields shown below from the “Data tree” window to the report page, which will then look roughly like this:



Preview the report with the “Preview” button on the toolbar.

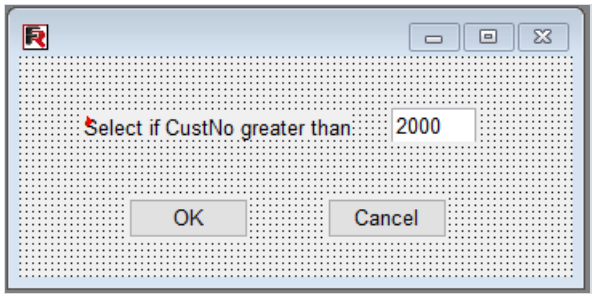
Report with parametric query

Let's design a more complicated report in which the query parameters are requested in a dialogue before the report is created. Use the same project as in the previous section, and click the New Report button in the report designer to clear the old one.

Switch to the “Data” tab and place an “ADO Query” component on the page. Double-click on it to open its editor and enter the following SQL text:

```
select * from Customer where CustNo > :p1
```

Add a dialogue form to the report and place a “Label”, an “Edit” and two “Button” components on the dialogue form:



Set the component properties:

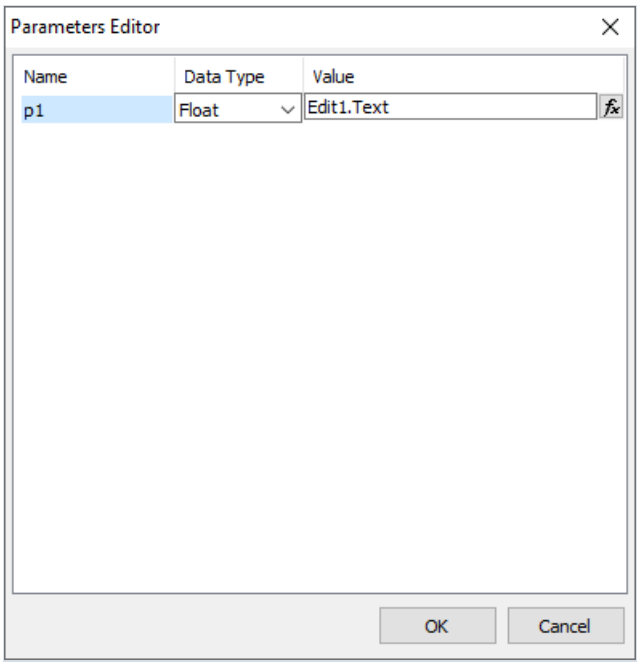
```
Label1:
Caption = 'Select if CustNo greater than'

Edit1:
Text = '2000'

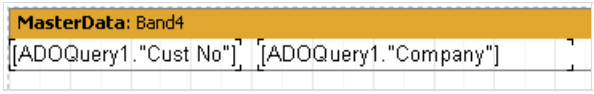
Button1:
Caption = 'OK'
ModalResult = mrOk

Button2:
Caption = 'Cancel'
ModalResult = mrCancel
```

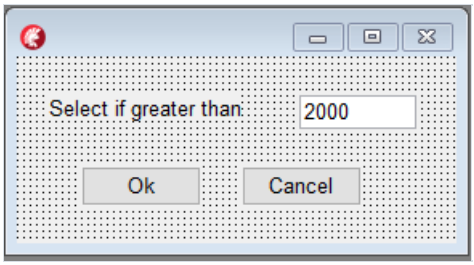
Open the `Params` property editor of the “Query” component and set the parameter:



After that switch to the report design Page1 and create the report as in the previous example:

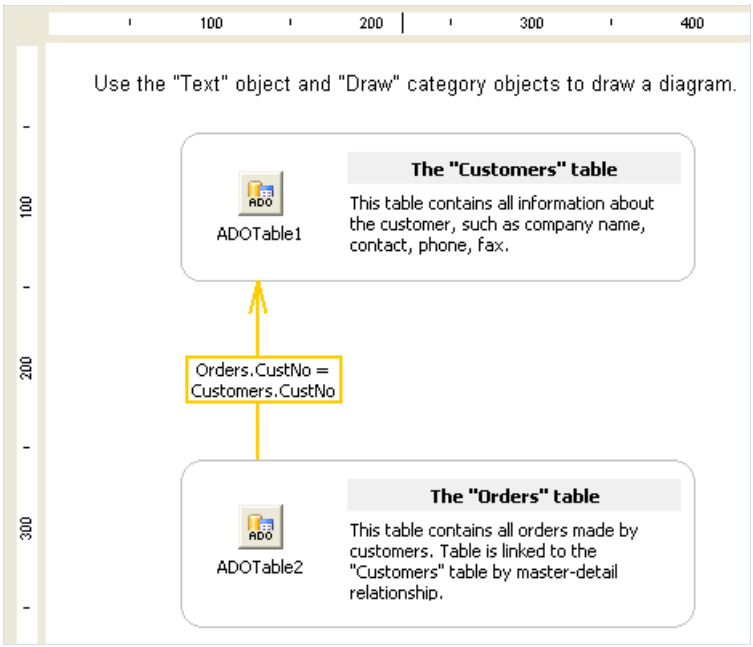


When the report is previewed the dialogue prompting for a customer number is displayed. After a figure is entered and the dialogue has been closed with the OK button the report is created. All customers with CustNos larger than that entered are shown.



Other useful features

“Text” and “Draw” elements can be placed on the “Data” page. Using these elements simple explanatory diagrams can be drawn, as shown below:



Report inheritance

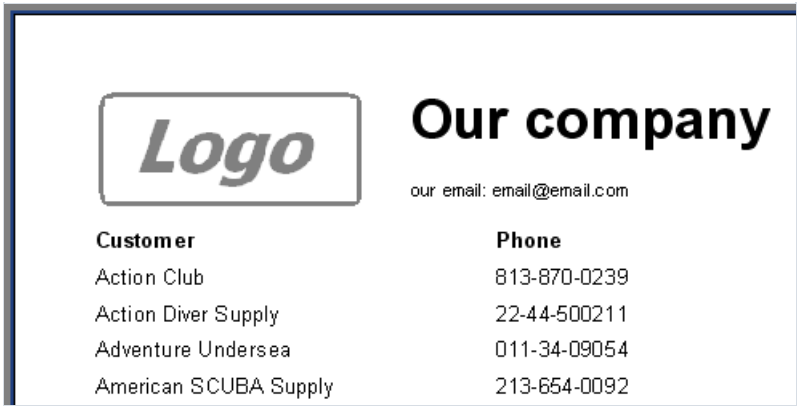
Often a group of reports share some common data - for example, the header/footer with company logo or other data, like email or address etc. Should the situation arise that some of this company data needs changing, for example the email address, then it would have to be done in each report! To avoid this tedious task, report inheritance can be used. What is report inheritance?

As an example, reports commonly have elements (logo, company name, email etc) typically placed in the report title and/or page header. A base report can be designed that contains only these common elements. All other reports can then use the base report and thus contain all of the common elements, as well as other elements specifically added to each report.

Should something (e.g. logo or email) need changing then the base report would be opened and the necessary changes made there. All reports inheriting from the base report would then be amended automatically. In fact, when a report based on inheritance is opened the base report is opened first of all, followed by the derived one.

Creating a base report

Let's create a simple report that uses inheritance. Our finished report should look like this:

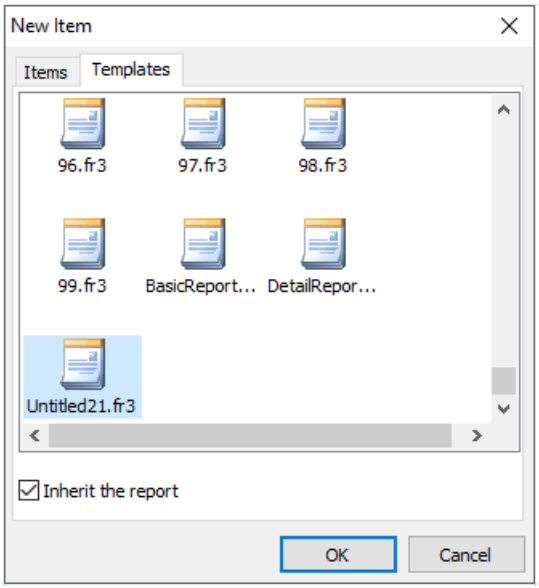


First the base report has to be created. Which elements must it contain? They are the logo bitmap, "Our company" title and email address. Create a new report and place the common objects in the “ReportTitle”:



Save the report as `base.fr3`. In which folder? This depends on how you setup the `TfrxDesigner` component. By default FastReport searches for base reports in the folder that contains the application's .exe file. Alternatively a folder for templates can be specified in the `TfrxDesigner.TemplateDir` property.

Now create the derived report. To do this use “File > New”, select the “Templates” tab in the dialogue and search for the base report (`base.fr3`). Click the "Inherit the report" checkbox and press OK:



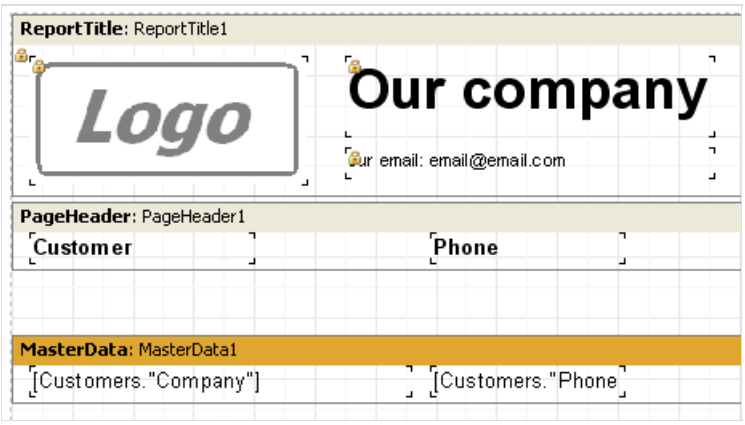
FastReport will create a report containing all of the objects from the base report. They are tagged with the "lock" symbol:



What does "lock" symbol mean? It means that these objects cannot be renamed or deleted, nor can they be moved to another band. Changes to any other property (such as text, color or frame) can be made. Note that if you change some property of a locked object (for example color) this change will be stored in the derived report. If you subsequently change the color of this object in the base report the change will be ignored by the derived report.

For example: open the derived report, change the color of "Our company" to red and save the report. Now open the base report and set the color for "Our company" to green. When the derived report is opened again the color of "Our company" is still red. It is therefore preferable to change the properties of objects having the "lock" symbol back in the base report and not in the derived report.

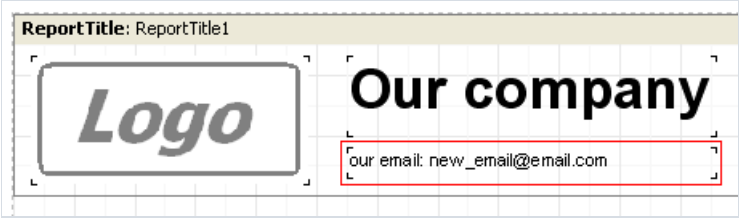
Let's finish our report. All that is needed is to add page header and master data bands:



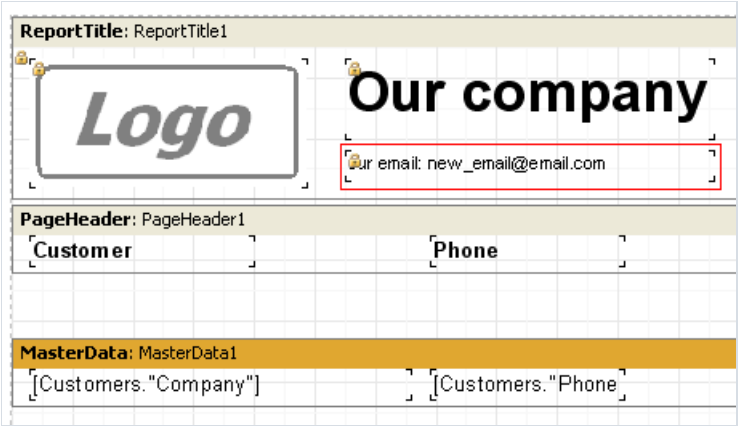
And the report is finished.

Changing a base report

Let's look at changing a base report. Open the base report (`base.fr3` in our example) and change some fields. Let's alter the email address:



Save the report. Now open the derived report and see that the email address has been changed in this report as well:

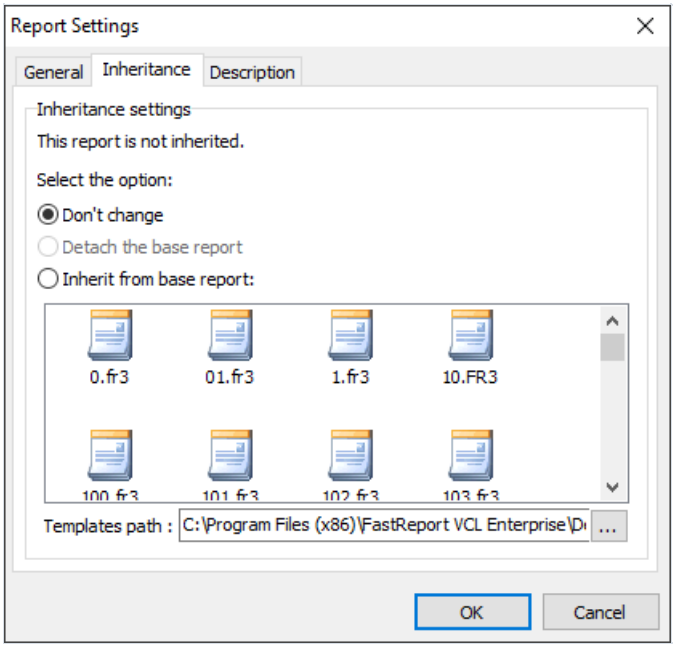


What if some objects have to be added to the base report? There is a simple rule: the base and derived reports cannot contain any objects having the same name. While changing base reports, it may not be known how many reports use the base report, nor what object names have been used in these derived reports. So a simple strategy is: when adding objects to base reports name the objects using a template like `ReportName_ObjectName`. In our example add a “Text” object to our report and set its name to 'BaseMemo3'.

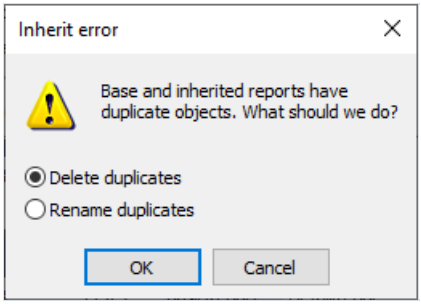
There is no restriction on deleting objects from a base report nor on moving them.

Inheritance control

We have followed the creation of a derived report from scratch. What happens if an existing report is to be changed to inherit from a different report? To do this, open the report and from “Report > Options...” choose the “Inheritance” tab:



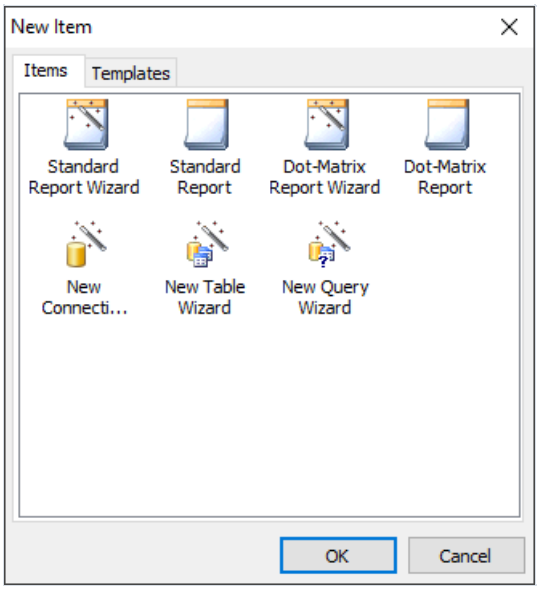
Select the "Inherit from base report" option and choose the base report from the list. If necessary change the Template path to see a different set of templates. Press OK and FastReport will combine the two reports. The following error message may appear:



This happens if the two reports contain objects having the same name. Objects with duplicated names can be deleted from the derived report or can be renamed in the derived report.

Wizards

FastReport provides some wizards that simplify the report creation process. Wizards are found under "File > New...":



New report wizard

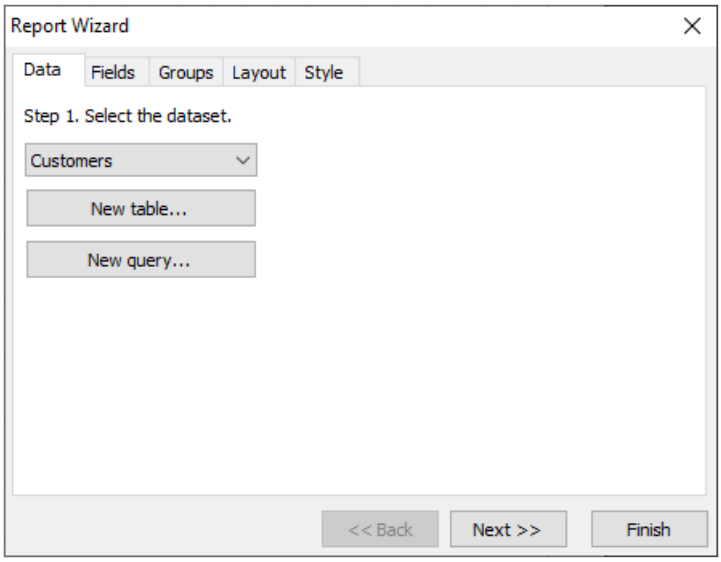
There are two icons for wizards helping to create new reports and two icons for the straight forward creation of new reports:

- Standard report wizard
- Dot-matrix report wizard
- Standard report
- Dot-matrix report

Wizards of type "Standard report" and "Dot-matrix report" create the empty standard and dot-matrix reports respectively (there is more about dot-matrix reports in previous sections). These new reports contain one empty page.

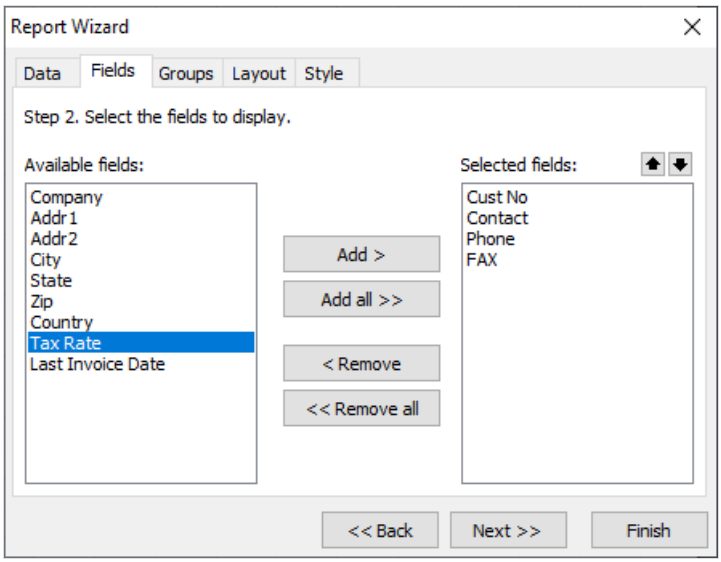
Wizards of type "Standard report wizard" and "Dot-matrix report wizard" guide the process of choosing the dataset and fields required for the report, the optional creation of groups and the selection of data layout. Let's look at a report created with the help of the "Standard report wizard".

Choose "File > New..." and the "Standard report wizard" icon. The report wizard dialogue appears:



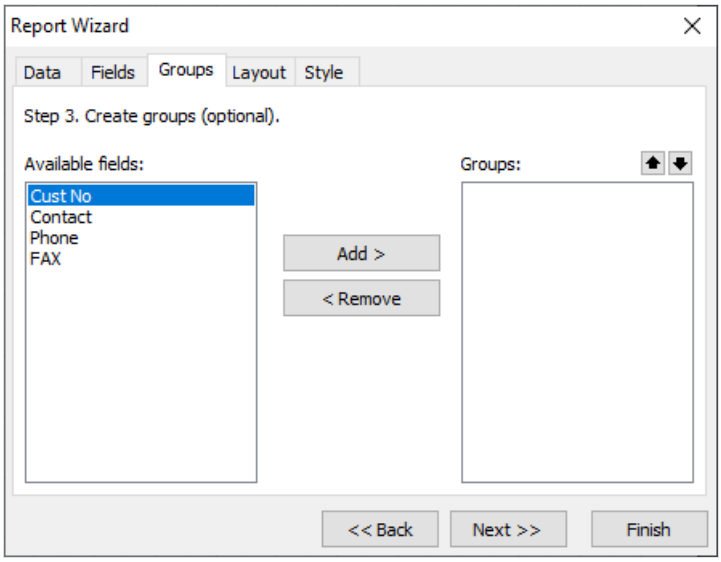
The dialogue has several tabs. On the first tab choose the data source for the report. All data sources available in your application are listed here (`TfrxDBDataSet` components). New data sources can be created, either tables or queries, using the "New table" or "New query" buttons. These buttons open the "New table/query" wizard (described later in this chapter). Let's choose the Customers table. Then press the "Next >>" button.

On the next tab select the fields to be displayed in the report:



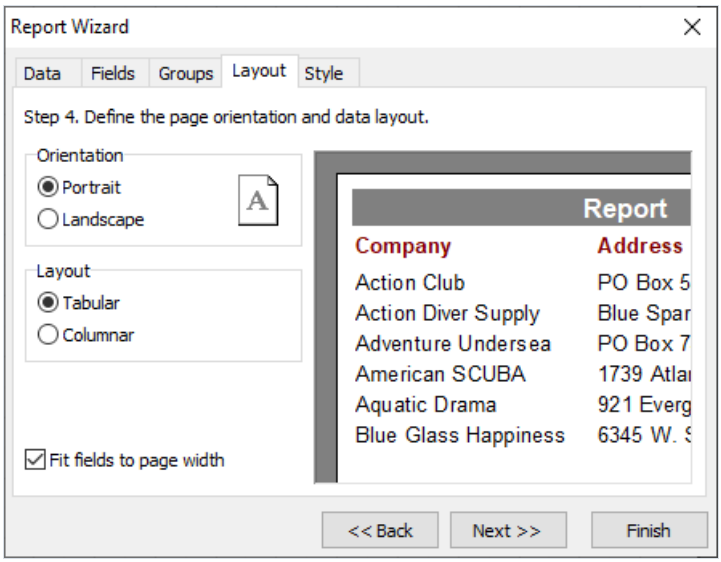
On the left side is a list of available fields; on the right side a list of fields already selected to appear in the report. Use the "Add >", "Add all >>", "< Remove" and "<< Remove all" buttons to move fields from one list to another. Use the  buttons to move selected fields up or down in the list. Let's add the "Company", "Contact", "Phone" and "FAX" fields to the selected fields list and press the "Next >>" button.

On the next tab one or more groups can be created. FastReport will add the Group header and Group footer bands to the report.



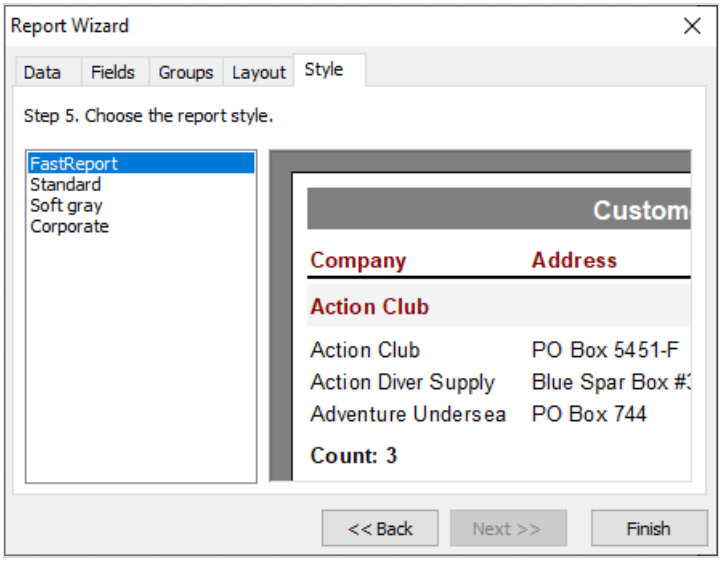
Group creation is optional. Skip it here by pressing the "Next >>" button.

The next tab sets the page orientation and one of two data layouts - tabular and columnar:

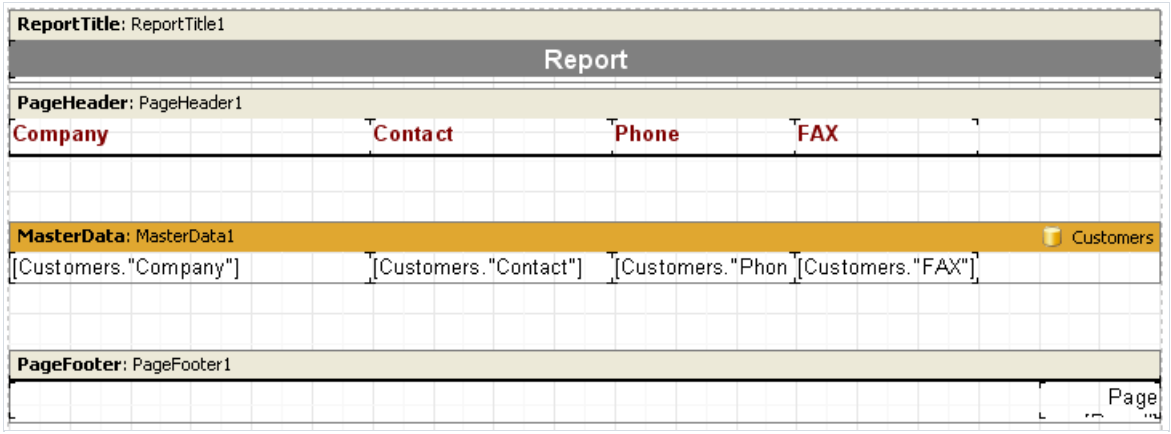


The chosen layout is illustrated on the right side of the dialogue.

Finally, the last tab lists the available color schemes for your report, again illustrated on the right side of the dialogue..



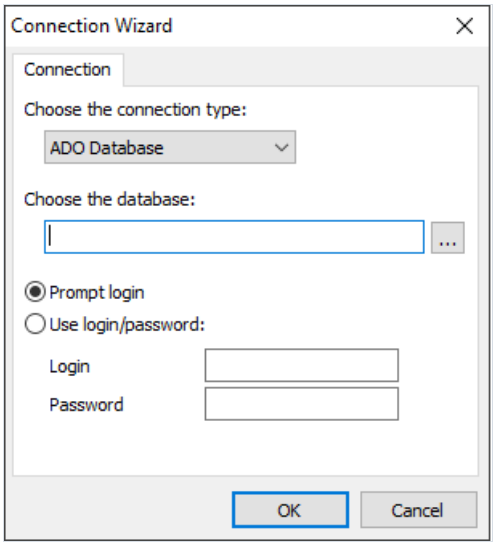
When the "Finish" button is pressed the wizard will create the following report:



The report can now be previewed.

New connection wizard

This wizard adds a new database connection to an existing report. Two or more connections may be needed if data from two or more databases is to appear in the report. The wizard adds the database component (for example `TfrxADODatabase`) to your report.

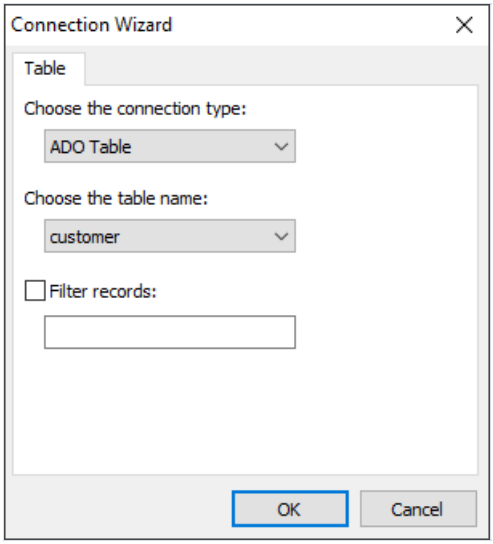


The connection string must be created using the ... button which opens the standard Windows connection dialogue for setting the database and connection parameters. After this the user name and password can be set, if necessary.

Note: a new connection can be made manually - just place a `TfrxADODatabase` component on the report's Data tab.

New table wizard

This wizard adds a new database table to an existing report.



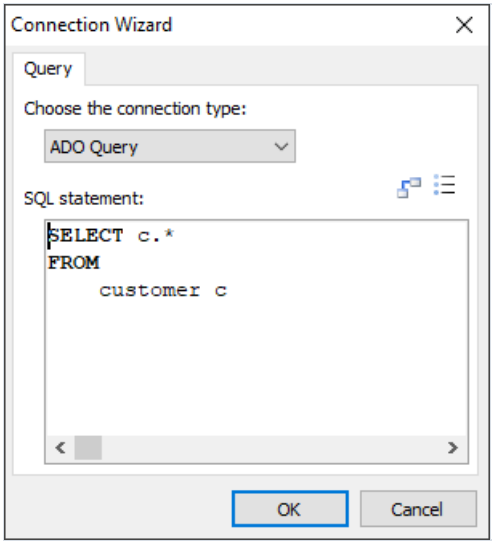
Select the table name. If required, a filter can also be defined, for example:

```
(CustNo > 2000) and (CustNo < 3000)
```

> A new table can be created manually by placing a `TfrxADOTable` component on the report's Data tab.

New query wizard

This wizard adds a new SQL query to an existing report.

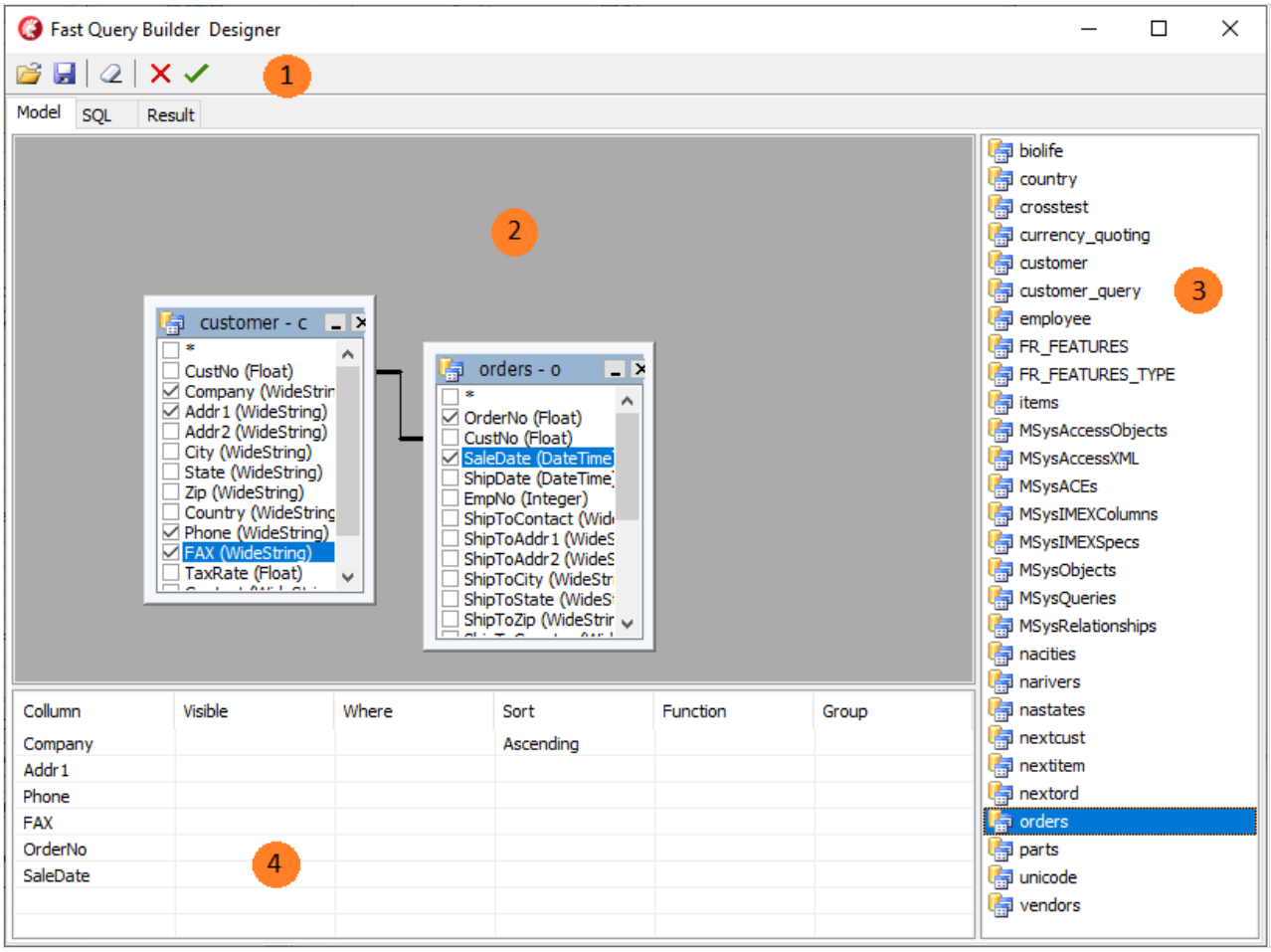


The SQL query must be composed here. The visual query builder can be used to do this - click the button. The query builder is described later in this chapter.

A new query can be created manually by placing a `TfrxADOQuery` component on the report's Data tab.

Query construction

FastQueryBuilder can be used to compose queries visually. FastQueryBuilder is included in FastReport Professional and Enterprise editions, or alternatively it can be purchased as an independent product. The query builder builds queries visually in the SQL language. The builder is illustrated below:



1 - toolbar

2 – builder workspace

3 – list of available tables

4 – selected table field properties

Toolbar:

- open SQL file

- save query to file (query diagram is also saved in the file)

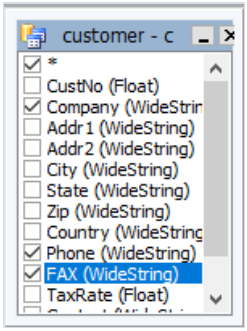
- clear builder workspace

- OK button : save and exit builder

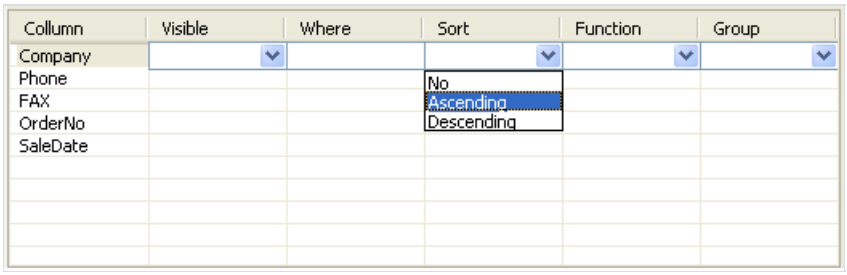
- Cancel button : exit builder without saving

The builder's workspace and the list of available tables support drag&drop, i.e. tables can be dragged onto the workspace with the mouse. Alternatively double-click on a table in the list of available tables.

To include a field from the table in the query select it in the list:

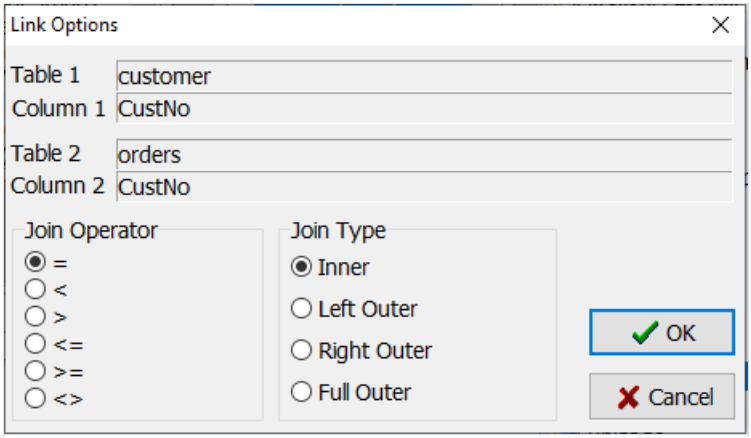


Selected fields appear in the field properties space (4):



- `Visible` : whether field appears in the output
- `Where` : field selection condition, e.g. '> 5'
- `Sort` : sorting direction for field
- `Function` : function applied to field
- `Group` : grouping on the field

By "dragging" fields between the tables in the workspace (2) "Join lines" appear. When fields are joined the compatibility of the types of the joined fields is checked. The builder prevents joins between type-incompatible fields. To change the join parameters hold the cursor over the "join line", right-click and select the Options item. The Link Options dialogue will open, where the join can be configured, as below:

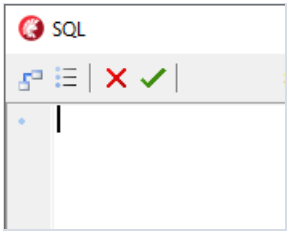


Using the Query constructor

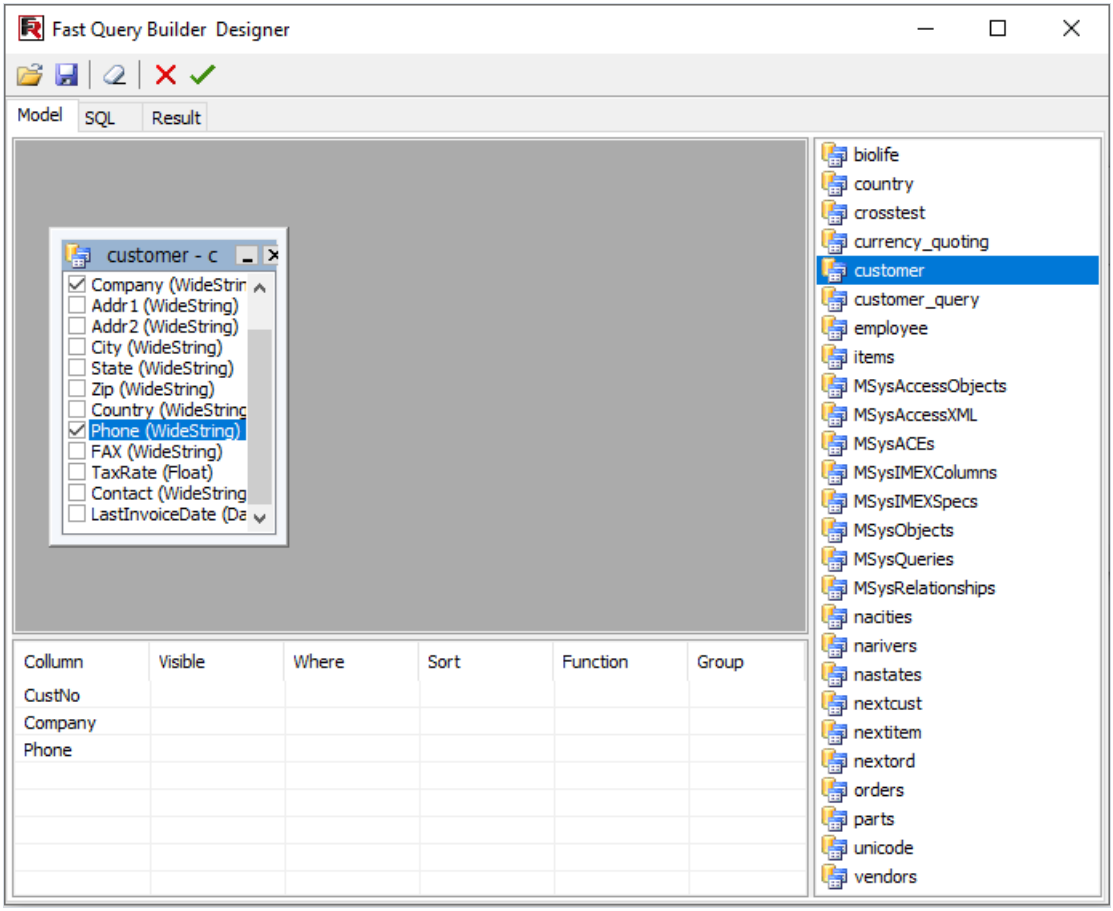
Let's create a simple report using the query builder.

Click “New report” on the designer toolbar to create a report page having “Report header”, “First level data” and “Page footer” bands.

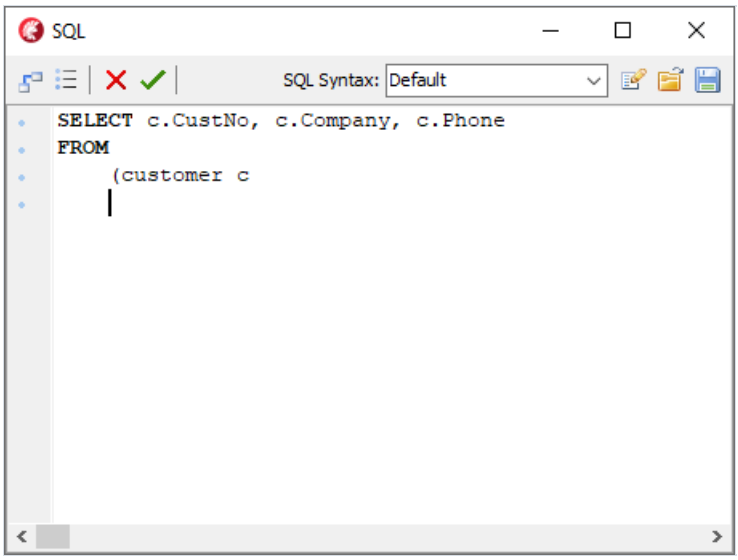
Place an “ADO Query” component on the “Data” tab. Double-click on the component to open the query editor.



Click the button in the query editor to open the query builder window. Select the Customer table in the list of tables and drag it onto the workspace (alternatively double-click on the table). Select the CustNo, Company and Phone fields:



That is all that is required for query building. The query text appears on the SQL tab and the Result tab shows the data returned by the query. Click to close the builder and to return to the query editor, where the query text is now displayed:



If the query text is altered in the query editor then the query diagram of tables and joins will be lost. Do not alter the query text manually, always open the query builder and modify the diagram visually.

Clicking in the query editor returns to the report designer. All that is left to do is to connect the “MasterData” band to the data source and place the required fields on the “MasterData” band.

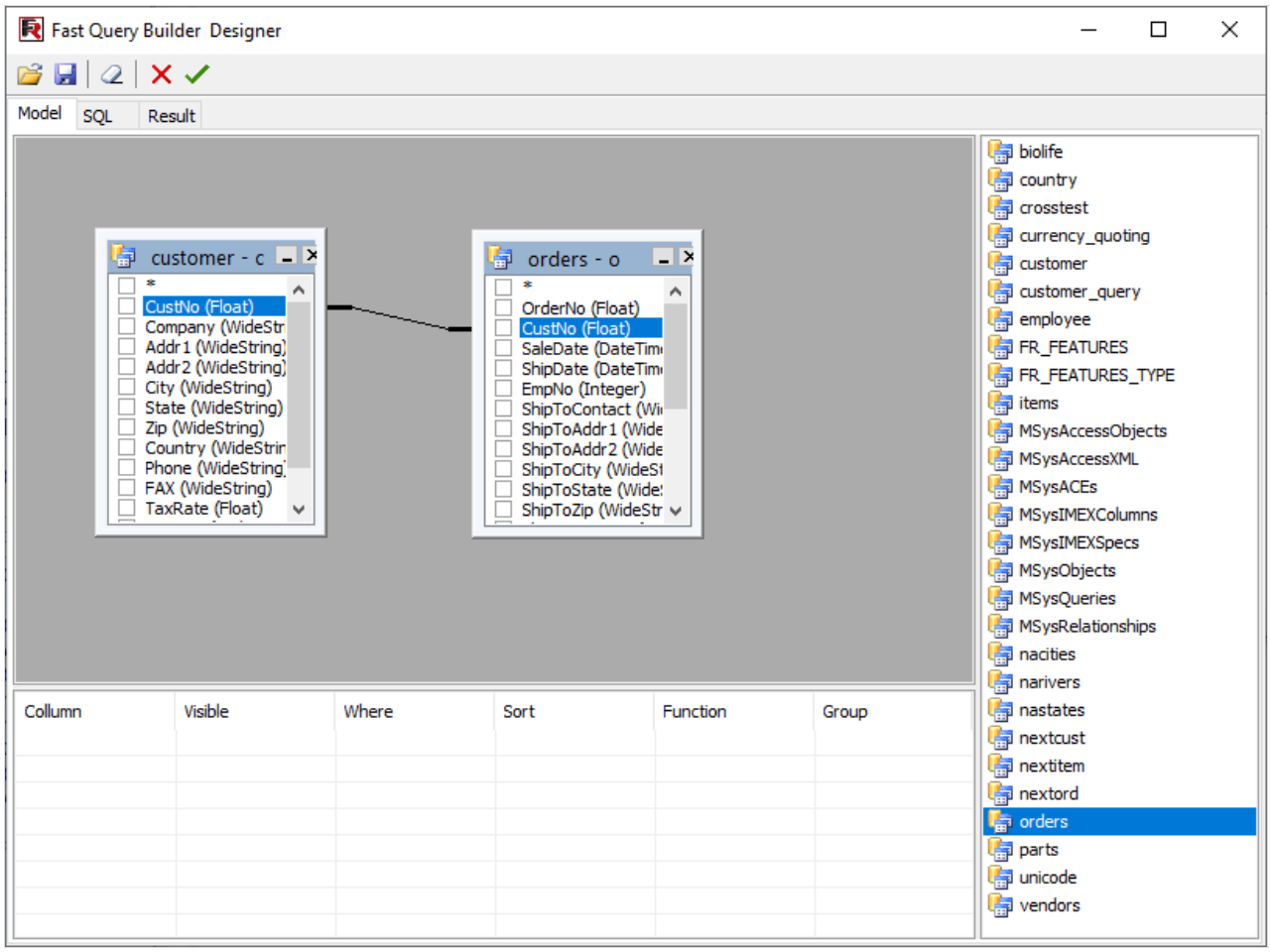
Complex query building

In the last example we built a report based on one table. Let's now look at query building where data comes from two tables.

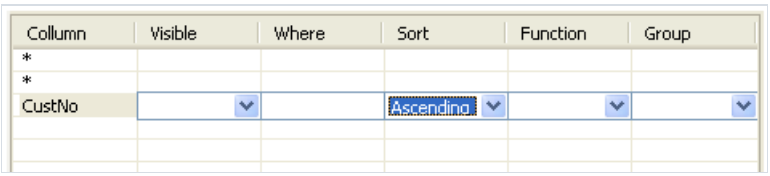
Earlier we looked at a report working with groups. Let's build a query for this report using the query builder. We need to compose a query in SQL which will return data from both tables, with the data grouped on a specific condition. In our example the condition will be CustNo fields in both tables.

As in the previous example, create a new report and put a `TADOQuery` component on the page. Open the query editor and then the query builder.

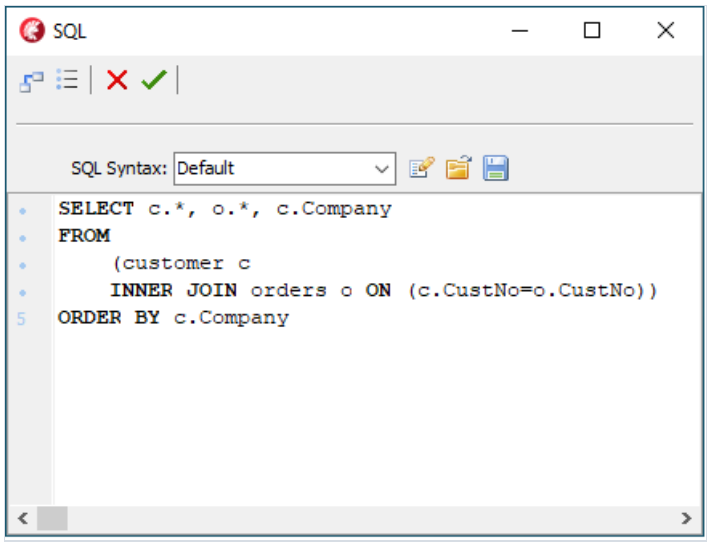
Drag two tables to the work area – Customers and Orders. Both tables have a CustNo field which we will use to join them. Drag the CustNo field from one table to the other table to create a join between the two tables:



Now the fields to be displayed and the sort field need to be set. Check the "*" field in both tables and check the CustNo field in the Customer table. The selected fields appear in the field parameters list. Select the sort order for the CustNo field:

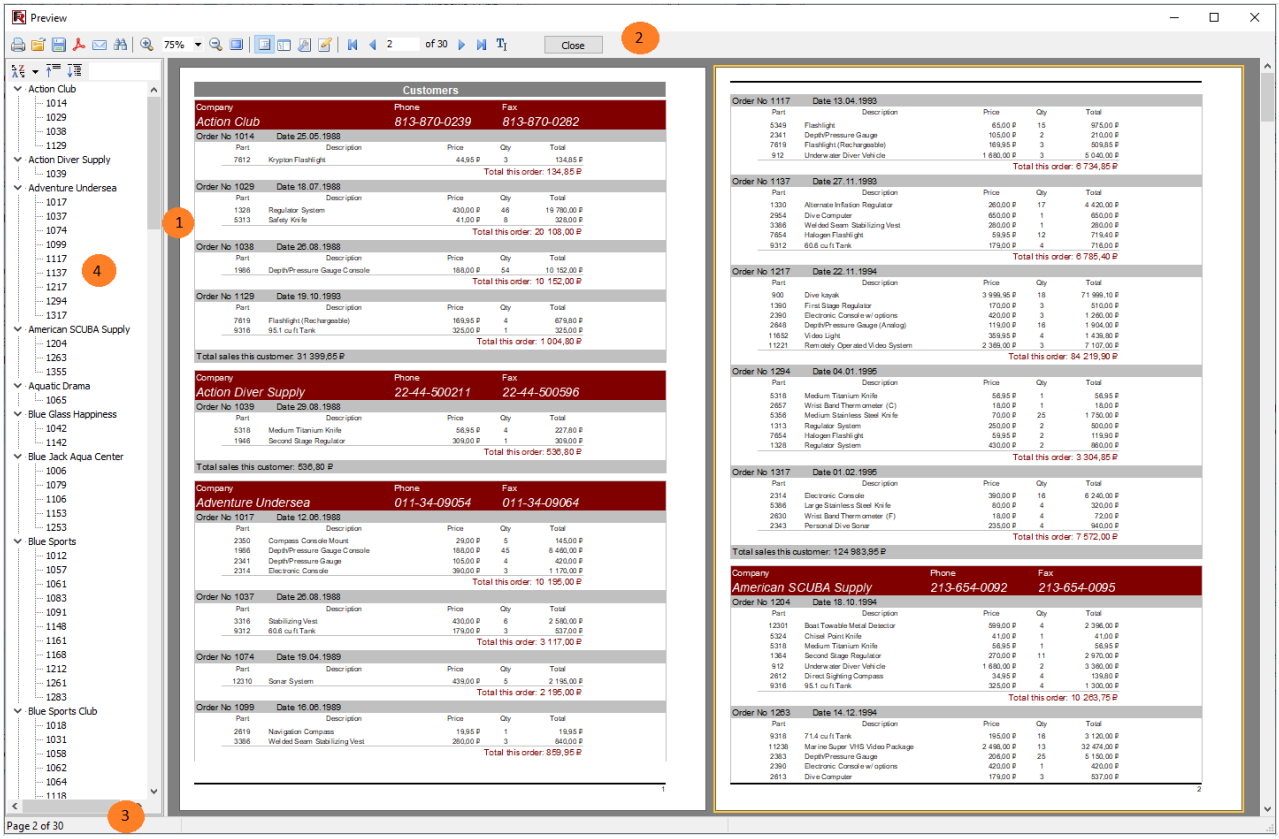


That is all that is needed to complete the Query. The SQL code looks like this:


SQL
SQL Syntax: Default
SELECT c.*, o.*, c.Company
FROM
(customer c
INNER JOIN orders o ON (c.CustNo=o.CustNo))
ORDER BY c.Company

Preview, print, export

A built report can be displayed and printed or exported into one of the supported formats. Everything can be done in the preview window.



Key:

1 – finished report pages

2 – toolbar

3 – status bar

4 – outline space for either the outline tree (as shown above) or for thumbnails

The toolbar has these buttons:



Icon	Name	Description
	Print report	prints report - Hotkey : Ctrl+P
	Open report	opens file containing finished report (*.fp3)
	Save report	saves report to file (*.fp3) or exports report to one of the supported formats

Icon	Name	Description
	Export to PDF	exports report to Adobe Acrobat file (*.pdf) visible if corresponding export filter is installed
	Send via e-mail	exports report to one of the supported formats and sends it via e-mail as enclosure visible if corresponding export filter is installed
	Text search	text search in report - Hotkey : Ctrl+F
	Zoom in	zooms in on the preview
	Scale	selects arbitrary scale for zoom
	Zoom out	zooms out of the preview
	Full screen	displays report as full screen return to normal size by double-clicking on report
	Outline	shows or hides report outline
	Thumbnails	shows or hides thumbnail view
	Page properties	opens dialogue with page settings
	Edit page	edits current page
	To beginning	jump to first report page
	Previous page	jump to previous report page
	Page number	jump to report page number type number and press Enter
	Next page	jump to next report page
	To end	jump to last report page
	Close window	close preview

Control keys

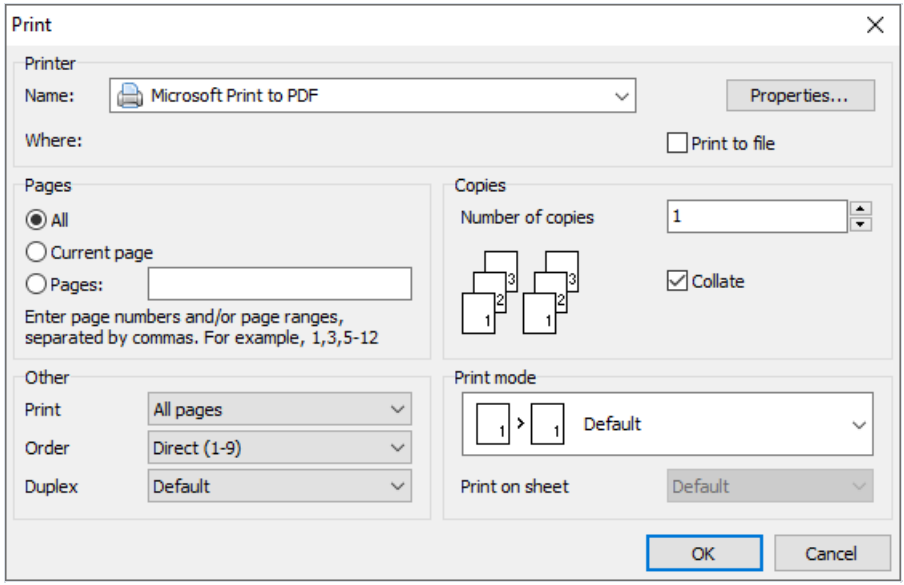
Keys	Description
Ctrl+S	save report to *.fp3 file
Ctrl+P	print report
Ctrl+F	text search
F3	search again
Arrows	smooth report scrolling
PageUp, PageDown	up/down report scrolling
Ctrl+PageUp, PageDown	next/previous page report scrolling
Home	report beginning
End	report end

Mouse control

Action	Description
Left button	click on selected object (in interactive report); report scrolling in “hand” mode (move mouse with button held down); zoom in when in “magnifier” mode
Right button	context menu; zoom out in “magnifier” mode
Double-click	return to normal size when in full-screen mode
Mouse wheel	report scrolling

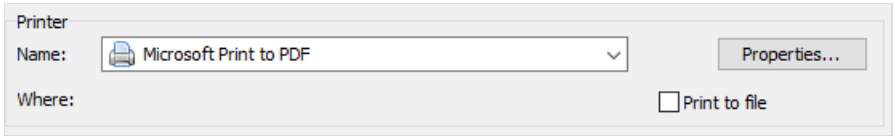
Report printing

To print a report click on the button (or Ctrl+P hotkey). The standard print dialogue opens.

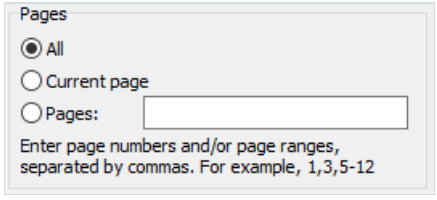


Let's look at the options available in this dialogue.

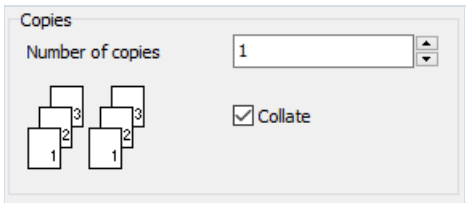
"Printer" panel : select printer on which to print the report; set printer properties, for example printing quality; choose to print to file.



"Pages" panel : select which pages to print (all, current or selected range).



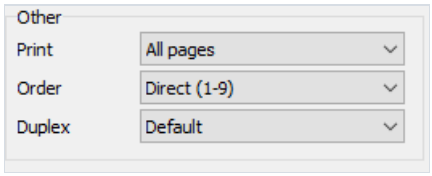
"Copies" panel : set number of copies to print. If printing more than one copy and Collate is checked then the first copy is printed in full, then the second in full, etc. If Collate is not checked then all copies of the first page are printed followed by all copies of the second page, etc.



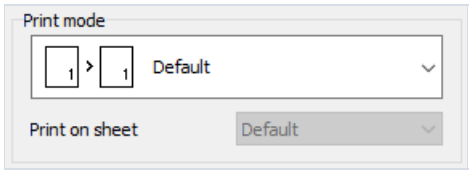
"Other" panel :

- Print : select which pages to print (All pages, Even pages, Odd pages)
- Order : print pages in direct or reverse order (first page to last, last page to first)

- Duplex : handle duplex by default (report settings are used) or choose one of duplex options: vertical, horizontal, simplex



"Print mode" panel : select the printing mode.





- Default mode : prints on the sheet defined in the report. One preview page is printed on each sheet



- Split big pages : this mode is useful if printing an A3 report on an A4 sheet. One preview page is printed on more than one sheet. When this mode is chosen the sheet size ("Print on sheet") must also be specified.



- Join small pages : this mode is useful if printing an A4 report on an A3 sheet. Two or more preview pages are printed on one sheet. When this mode is chosen the sheet size ("Print on sheet") must also be specified.

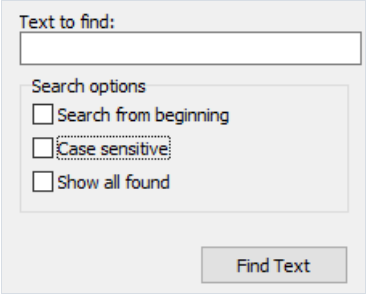


- Scale mode : report is printed on specified size of sheet. All report output is scaled. One preview page is printed on one sheet. When this mode is chosen the sheet size ("Print on sheet") must also be specified.

When OK is clicked the report printing begins. If "Print to file" is checked then the standard "Save As..." dialogue opens. The report is saved to this file with *.prn extension. The file contains a copy of the information sent to the printer.

Text search in reports

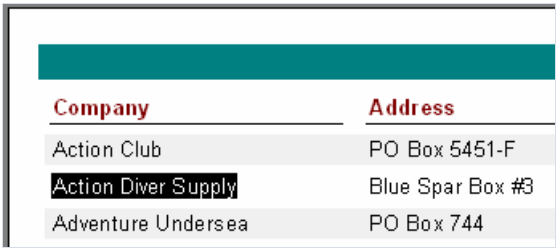
FastReport can search for a given phrase within the text of a previewed report. Search using the button on the toolbar (or Ctrl+F hotkey). The search dialogue opens:



Enter the phrase to be searched for and select options if required:

- Search from beginning : search report from the beginning; otherwise searching is from current page on
- Case sensitive : match the case of the search phrase (lower or upper) when searching

On clicking OK the search is started and the first occurrence (if any) is highlighted:



To continue searching click F3. The next occurrence (if any) will be highlighted

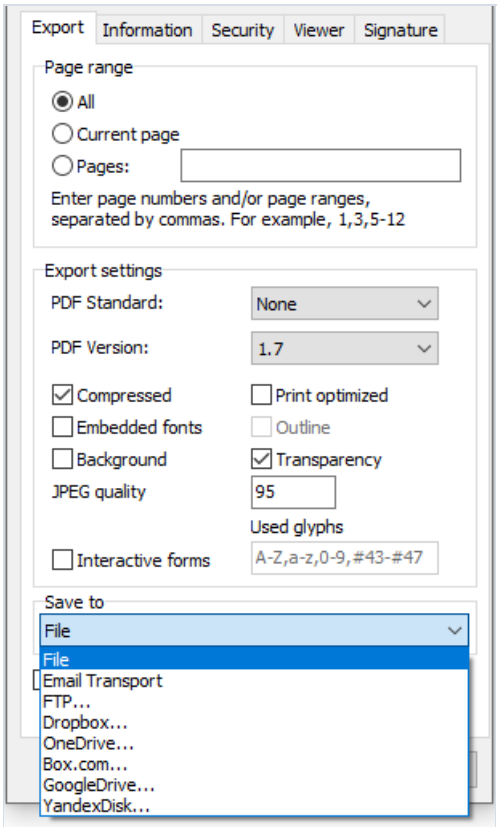
Saving the report

FastReport uses so-called transports to save prepared reports or exported files to one of the following places:

- file;
- email;
- FTP;
- Dropbox;
- OneDrive;
- Google Drive;
- Box.com.

To save the prepared report (.fp3 file), press the "Save" button in the preview window, then select the "Prepared report" menu item.

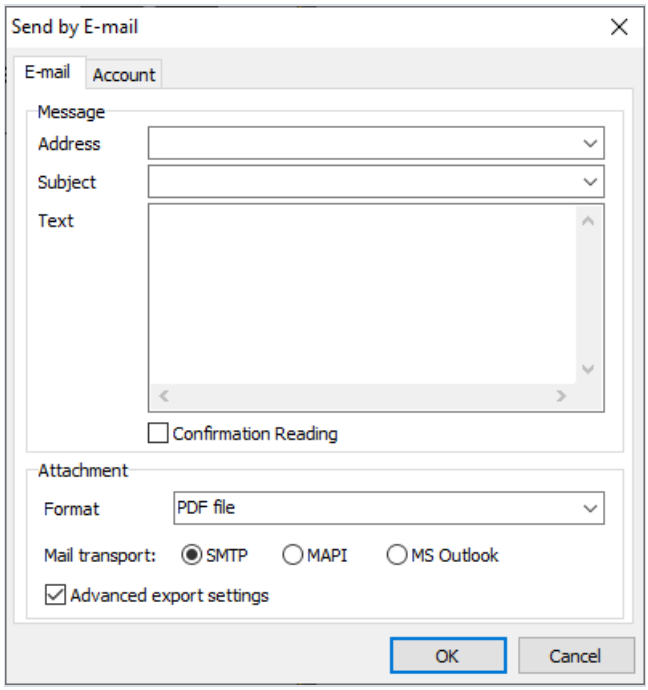
To export the report and save the result, press the "Save" button in the preview window, then select the export you need. In the export dialogue window, choose the "Save to" option:



Sending a Report via E-mail

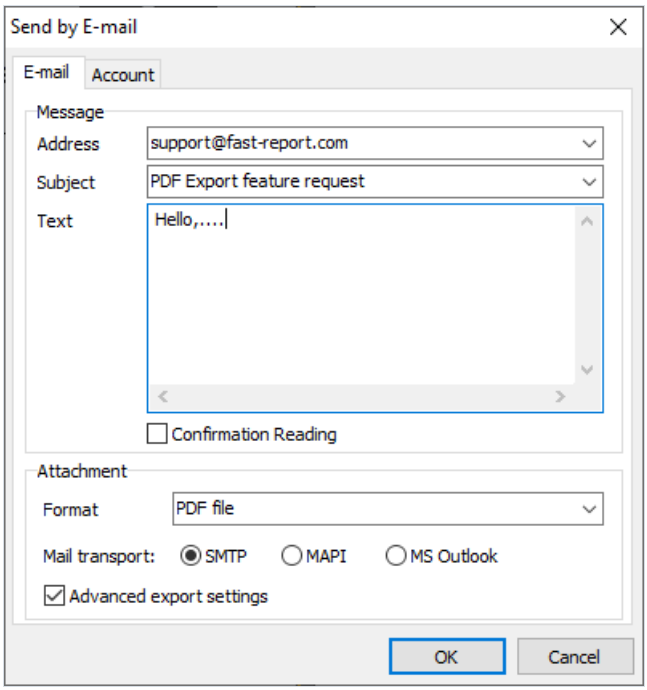
FastReport can send a finished report by e-mail in any format required without the need for a separate mail application.

When exporting by e-mail the E-mail setup dialogue is opened. Before exporting begins the sender's email account details should be set on the “Account” tab:



- From Name : sender’s name
- From Address : sender’s e-mail address
- Organization : sender’s organization
- Signature : signature for e-mail; it can be generated by clicking on the “Build” button once the earlier fields have been filled in
- Host : SMTP server service
- Port : SMTP server port
- Login : access name for authorization on SMTP server, if required by the specified SMTP server
- Password : authorization password, if required by the specified SMTP server
- Remember properties : remember all fields for further usage

After filling in the necessary fields on the “Account” tab, the message fields on the "E-mail” tab must be completed:

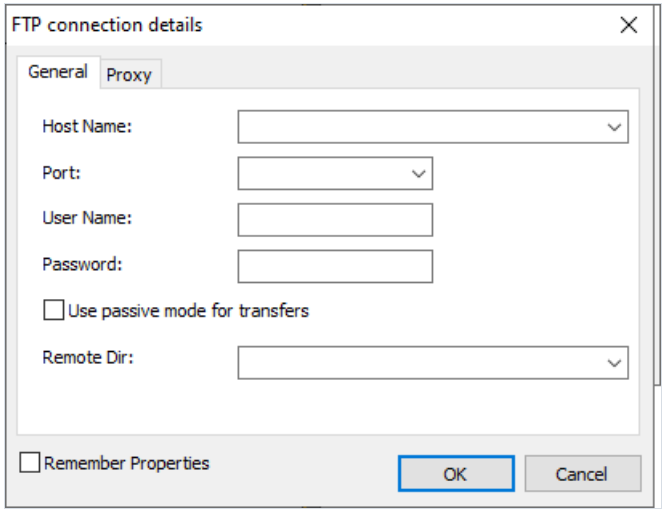


- Address : e-mail address of recipient; previously used addresses can be selected from the drop-down list
- Subject : message subject; previously used subjects can be selected from the drop-down list
- Text : message text
- Format : format of exported report attached to the e-mail; select one of the available export formats; the FastReport (FR3) format is also available for selection
- Advanced export settings : when enabled the OK button opens the appropriate export format settings dialogue; when disabled default export settings are used

Export via e-mail features: only plain authentication on SMTP servers is supported. If authentication is not required then it is not necessary to enter “Login” and “Password”.

Export to FTP

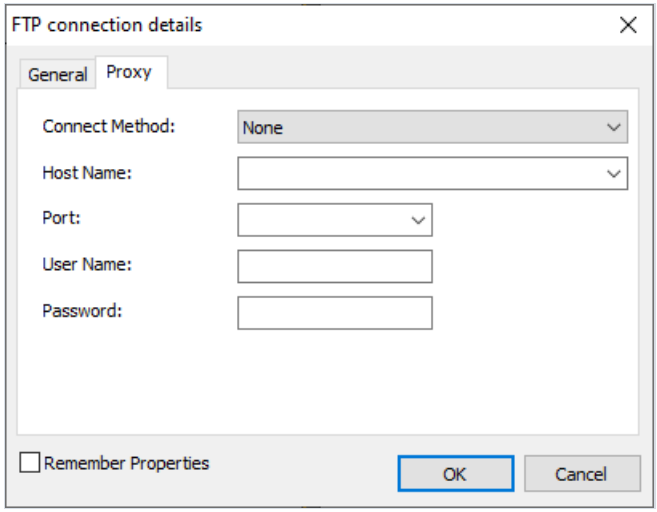
The following dialogue window will be displayed when you save the report to FTP:



The "General" tab has the following fields:

- Host name: enter the URL-address of the FTP server.
- Port: enter the port.
- User Name and Password: enter your username and password.
- Use passive mode: choose FTP connection mode.
- Remote dir: choose remote directory on a server (optional).

If a proxy server is used then the URL-address, port, username and password of the proxy server are set on the Proxy tab:



When all settings have been made click the OK button to save the file to the FTP server.

Export to Dropbox

Before saving a report to a Dropbox an application must be created in your Dropbox account. Do this by logging in to your Dropbox account and taking the following steps:

- click the “More” button: it is located at the bottom of the Dropbox homepage
- choose “Developers” in the drop-down list: you will be directed to the page for developers
- go to “App Console”: it directs you to the list of applications
- click the “Create App” button: the Dropbox will check your email: click Send Email. In your mail system inbox you will find an email having a “Confirm” button: click the button to confirm your email address.

Finally you will be directed to “Create a new Dropbox Platform app”.

Select "Dropbox API app" and answer the question "What type of data does your app need to store on Dropbox?" by selecting "Files and datastores".

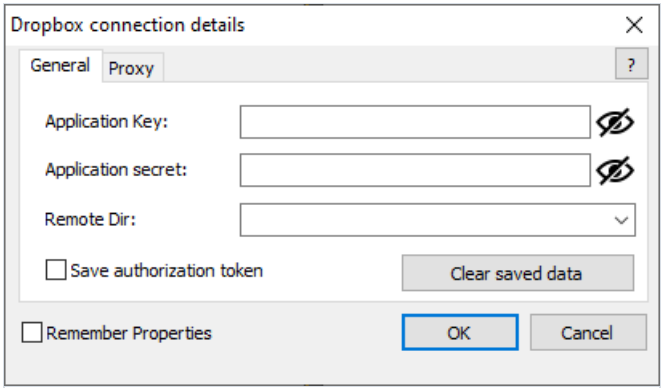
Answer the question "Can your app be limited to its own, private folder?" by choosing either of the two proposed answers.

This page also requires you to enter the name of the application.

After clicking the “Create app” button the system will create the application.

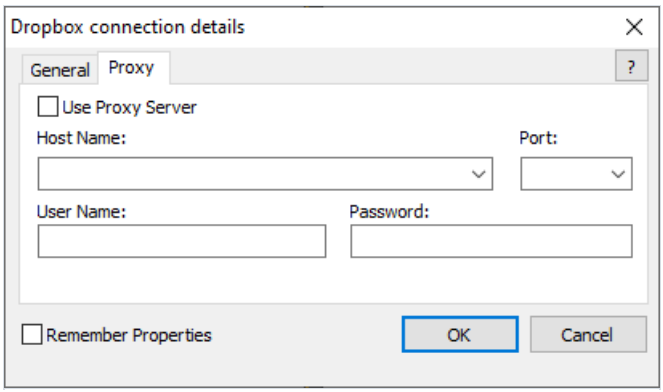
As a result the application settings page is opened. Here you can see the “App key” and the “App secret”, which are required when exporting to a Dropbox.

When exporting to a Dropbox the following dialogue window will be displayed:



Enter the “Application Key” obtained above. You may also choose remote directory on a server.

If a proxy server is used then the URL-address, port, username and password of the proxy server are set on the Proxy tab:



When all settings have been made click the “OK” button. The browser window will be opened to login into Dropbox. Then the file will be saved to the Dropbox.

Export to OneDrive

Before saving a report to OneDrive an application must be created in your OneDrive account. Do this by going to the OneDrive home page and clicking "Developers".

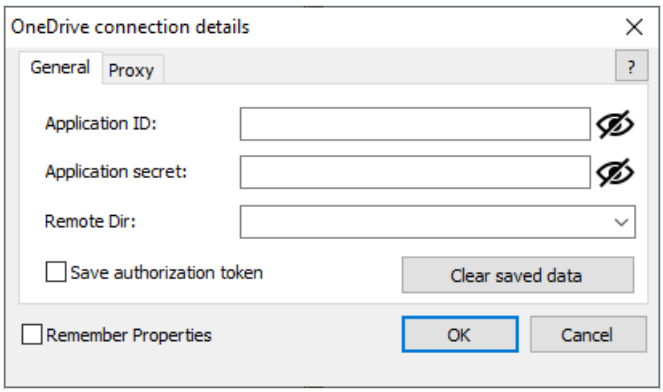
In the Development Center click "My Apps" and then click "Create Application".

Enter the name of the application and select a language. Read the "Terms of Use" and "Privacy Statement" and click "I accept".

As a result the next page is the Settings page of the application which shows the "Client ID" and "Client Secret".

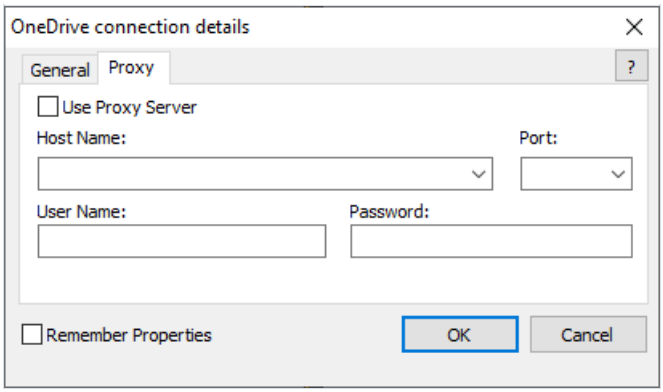
Enter the redirection domain and click "Save".

When exporting to OneDrive the following dialogue window will be displayed:



Enter the “Application ID” obtained above. You may also choose remote directory on a server.

If a proxy server is used then the URL-address, port, username and password of the proxy server are set on the Proxy tab:



When all settings have been made click the “OK” button. The browser window will be opened to login into OneDrive. Then the file will be saved to the OneDrive.

Export to Google Drive

Before saving a report to Google Drive an application must be created in your Google Drive account.

Do this by going to https://code.google.com/apis/console/

- this page has the license agreement and leads to the Project Settings page.

Go to the “Services” tab and activate the “Drive API”.

Go to the “API Access” tab and click "Create an OAuth 2.0 client ID".

In the “Branding Information” section enter the name of the application and click “Next”.

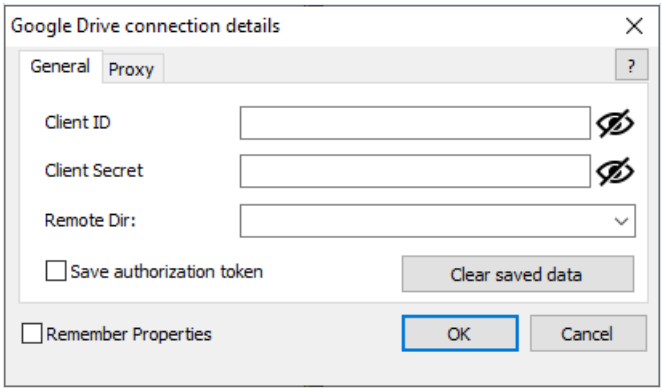
In “Client ID Settings”, select the following:

- “Installed application” for Application type
- “Other” for Installed application type.

Click “Create Client ID”.

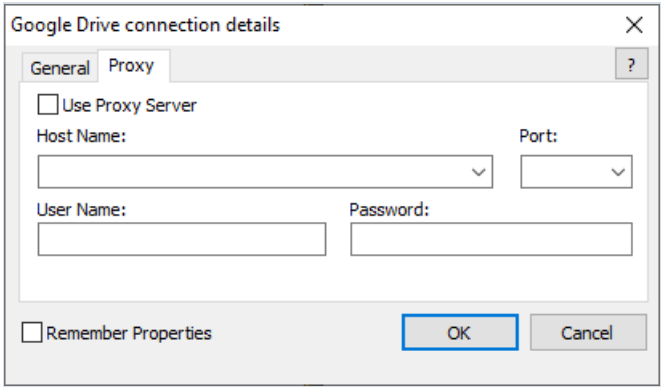
As a result the next page shows the “Client ID” and “Client secret”.

When exporting to a Google Drive the following dialogue window will be displayed:



Enter the “Client ID” and “Client Secret” obtained above.

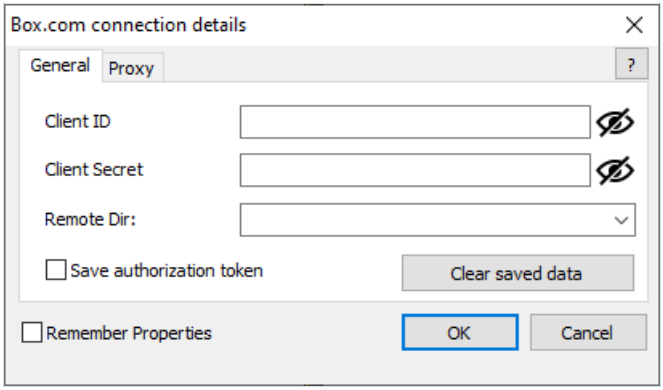
If a proxy server is used then the URL-address, port, username and password of the proxy server are set on the Proxy tab:



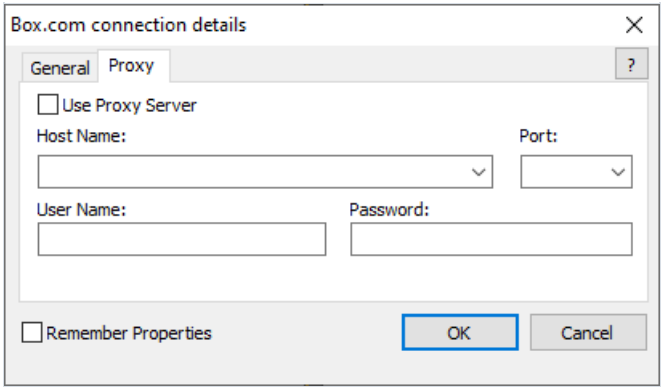
When all settings have been made click the “OK” button to save the file to Google Drive.

Export to Box.com

When exporting to a Box.com the following dialogue window will be displayed:



If a proxy server is used then the URL-address, port, username and password of the proxy server are set on the Proxy tab:

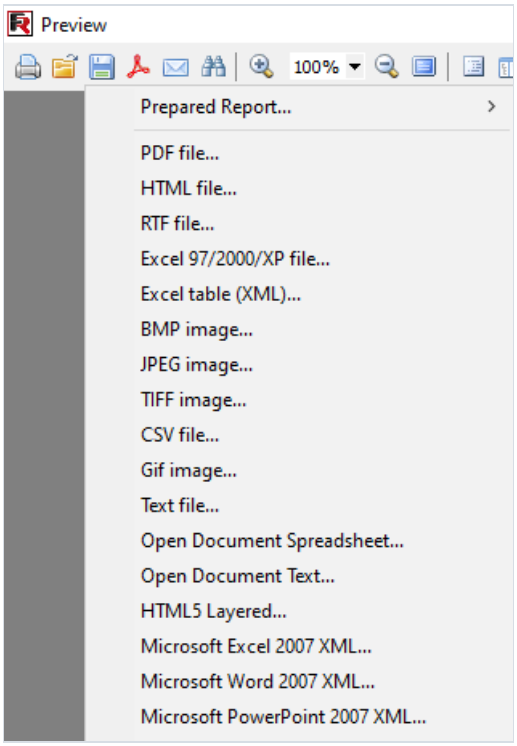


When all settings have been made click the “OK” button to save the file to Box.com.

Report Export

FastReport can export a previewed report to various formats for further editing, archiving or sending by e-mail, etc.. To enable export the appropriate FR export components must be added to the Delphi form.

FastReport can export to the following formats: PDF, Open Document Spreadsheet, Open Document Text, Excel 97/2000/XP, Excel XML, Excel 2007, RTF, Word 2007, PowerPoint 2007, HTML, TXT, CSV, BMP, JPG, TIFF, and GIF. Also, reports can be sent by e-mail in any of these listed formats.



FastReport uses one of the three following methods to export reports:

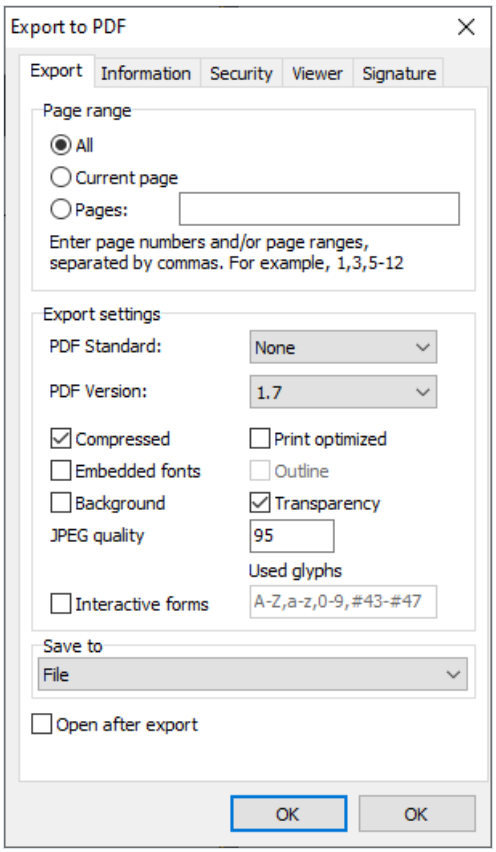
- `Layer` : each report object is exported to a separate layer. The exported output approximates to the original preview.
- `Table` : export of objects to the output file is by creation of a transitional grid in memory and then output of this grid. The exported output closely matches the original preview, based on the assumption that the principles of good report design were followed (see "Report Design considerations" chapter).
- `Drawing` : exported objects are captured from the page image. The exported output is a direct copy of the preview. This method is used when exporting to graphic formats.

Export to PDF Format

PDF (Portable Document Format): a platform-independent format of electronic documents created by Adobe Systems. The free Adobe Reader package is used for viewing. This format is flexible – it allows inclusion of required fonts, vector and bitmapped images; it is a means to distribute and store documents intended for viewing and/or printing.

The Export method is `Layer`.

When exporting to PDF format a dialogue opens requesting output file settings.



Export settings:

- Compressed : output file is compressed, file-size is reduced but export time is increased
- Embedded fonts : all fonts used in report are contained in the PDF output file, allows accurate rendering on computers where the fonts are absent; output file size is considerably increased
- Background : graphic image assigned to page background is exported to PDF file; output file size is considerably increased
- Print optimized : graphic images output in high resolution for accurate printing; this option is only necessary when a document contains graphics and will be printed; output file size is considerably increased
- Outline : option is enabled only when report contains an outline; outline is exported to PDF file
- Open after export : exported file is opened immediately after export using default PDF viewer installed on the computer (for example, Adobe Reader).

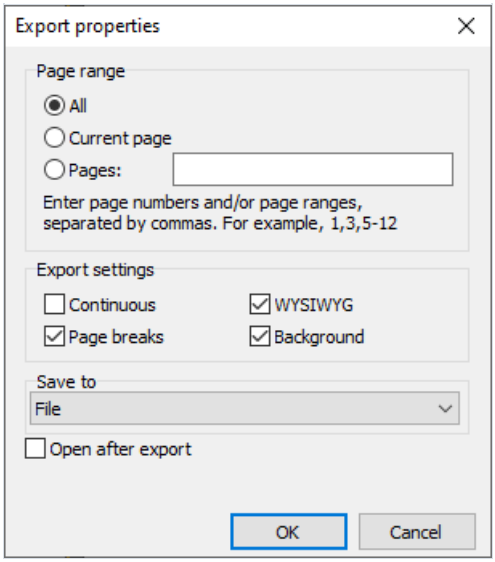
Export to Open Document

Open Document Format (ODF, OASIS Open Document Format for Office Application) was designed by OASIS and based on the XML format used in OpenOffice.

FastReport supports export to table (.ods) and text (.odt) files. These files can be opened in OpenOffice.

The Export method is `Table`.

When exporting to ODF format a dialogue opens requesting output file settings.



Export settings:

- Continuous : export as a continuous document, without page breaks and without page headers/footers
- Page breaks : enables page breaks in the document
- WYSIWYG : accurate rendition of previewed report; when disabled optimization is allowed, reducing the number of lines and columns in the export file
- Background : graphic image assigned to page background(s) is exported to ODF file; output file size is considerably increased
- Open after export : exported file is opened immediately after export.

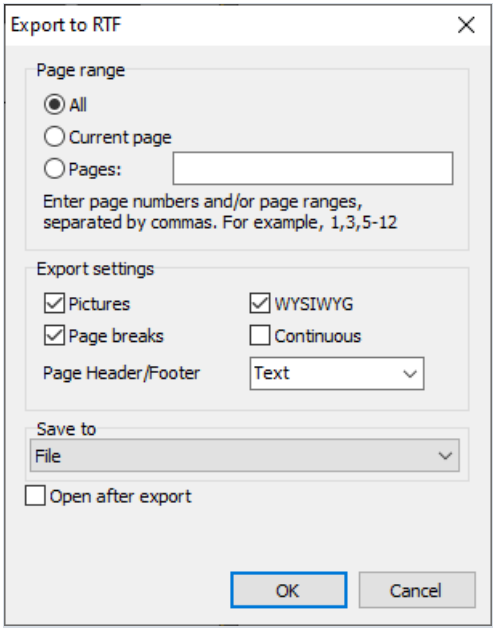
Export feature : RichText objects are exported as simple text, export of graphic images is supported.

Export to RTF Format

RTF (Rich Text Format) was developed by Microsoft as a standard for the interchange of text documents. RTF documents are supported by many modern text editors and operating systems.

The Export method is `Table`.

When exporting to RTF format a dialogue opens requesting output file settings.



Export settings:

- Pictures : enables export of graphic images to file
- Page breaks : enables page breaks in the document
- WYSIWYG – accurate rendition of previewed report; when disabled optimization is allowed, reducing the number of lines and columns in the export file
- Continuous : export as a continuous document, without page breaks and without page headers/footers
- Page header/footer : header/footer export mode; modes are: Text (h/f exported as normal text), Header/Footer (h/f exported) and None (h/f not exported)
- Open after export : exported file is opened immediately after export using default RTF viewer installed on the computer (for example, Microsoft WordPad)

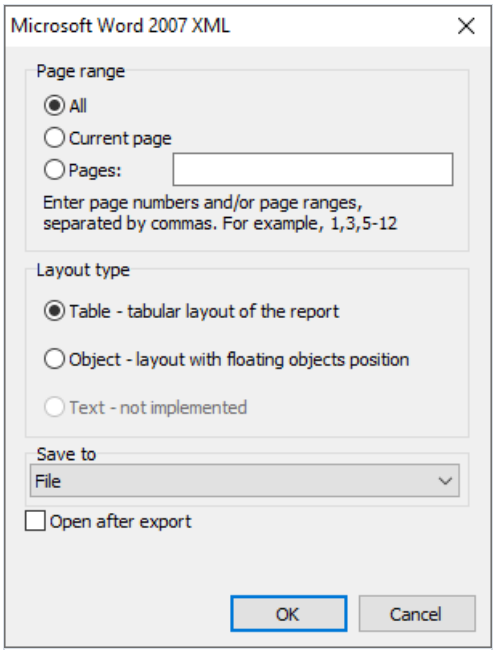
Export feature : RichText objects are fully integrated in RTF format; accuracy of rendering and file-size depend on how the report was designed, see the “Report Design considerations” chapter.

Export to Word 2007

Word 2007 is an application for working with text documents. It is included into Microsoft Office 2007.

The export method is `Table`.

When exporting to Word 2007 format a dialogue opens requesting output file settings.



Export settings:

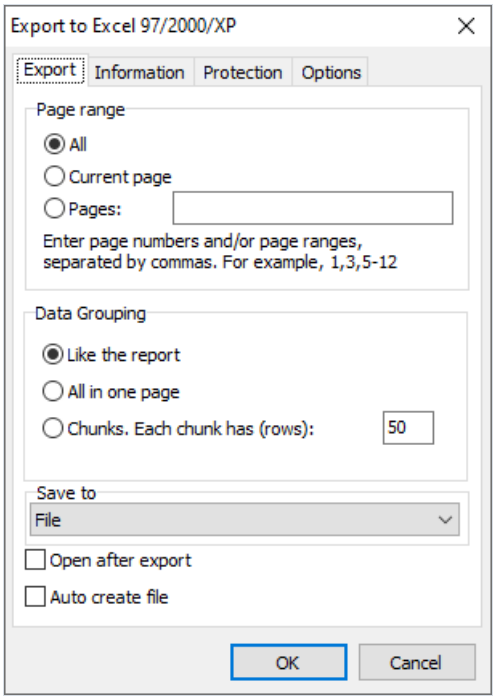
- Open after export – output file will be opened immediately after export.

Export to Excel 97/2000/XP

Excel – application for working with electronic spreadsheets. It is included in Microsoft Office.

The Export method is `Table`.

When exporting to Excel format a dialogue opens requesting output file settings.



Data grouping:

- Like the report - each page of prepared report is exported on a separate Excel sheet;
- All in one page - generate continuous document without page breaks and page headers/footers;
- Chunks. Each chunk has (rows) - each chunk is exported on a separate Excel sheet.

Export settings:

- WYSIWYG : full compliance to report appearance; when disabled optimization is allowed, reducing the number of lines and columns in the export file
- Pictures – includes graphic images export into output table;
- Grid lines - turns on/off Excel grid lines;
- Adjust page size - adjusts cell size to fit the content;
- Delete empty rows - deletes empty rows from the output table;
- Export formulas - if a cell text starts with "=" symbol, it is exported as an Excel formula.
- Open Excel after export : exported file is opened immediately after export.

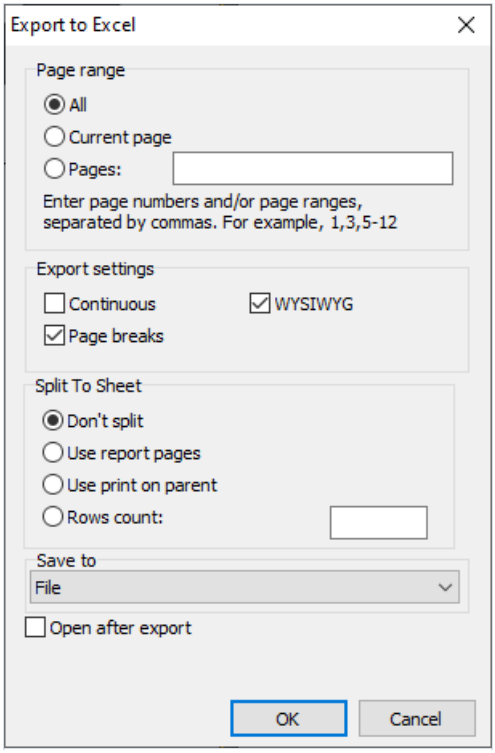
Export features: RichText objects are exported as simple text.

Export to Excel XML Format

XML (Extensible Markup Language) is an extendable markup language. XML is intended for structured data storage and also for information exchange between different programs. FastReport uses XML format for data transfer into Excel ver. 2003 and later.

Export method is `Table`.

On exporting to XML format the dialogue box for output file parameter settings appears.



Export parameters:

- Continuous - generate continuous document without page breaks and page headers/footers
- Page breaks – enables page breaks in the document
- WYSIWYG – accurate rendition of previewed report; when disabled optimization is allowed, reducing the number of lines and columns in the export file
- Background : background color of report page(s) exported to spreadsheet
- Open Excel after export : exported file is opened immediately after export.

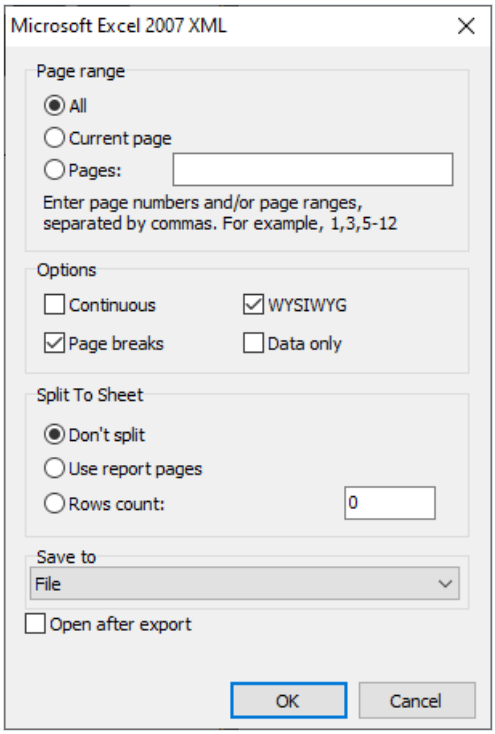
Export features : RichText objects are exported as simple text; graphic images are not supported.

Export to Excel 2007

Excel 2007 is an application for working with electronic worksheets. It is included into Microsoft Office 2007.

The export method is `Table`.

When exporting to Excel format a dialogue opens requesting output file settings.



Split to sheet:

- Don't split - all report pages are exported to a single Excel sheet;
- Use report pages - each page of prepared report is exported on a separate Excel sheet;
- Rows count - each set of rows is exported on a separate Excel sheet.

Export parameters:

- Continuous - generate continuous document without page breaks and page headers/footers;
- Page breaks – includes page breaks in resulting document;
- WYSIWYG – accurate rendition of previewed report; when disabled optimization is allowed, reducing the number of lines and columns in the export file;
- Open Excel after export - exported file is opened immediately after export.

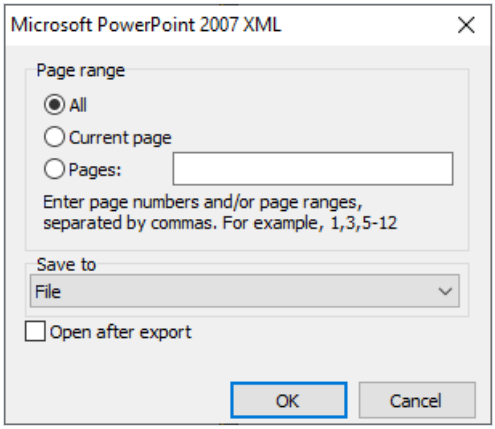
Export features: RichText objects are exported as simple text.

Export to PowerPoint 2007

PowerPoint 2007 is an application for working with electronic presentations. It is included into Microsoft Office 2007.

Export method is `Layer`.

When exporting to PowerPoint format a dialogue opens requesting output file settings.



Export settings:

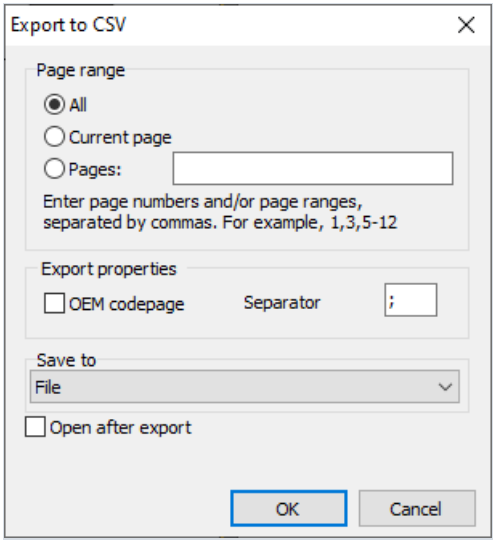
- Open after export – exported file will be opened immediately after export.

Export to CSV Format

CSV files contain tabular data (numbers and text) in plain-text form. A specified separator is inserted between column values and each row starts on a new line. This format can be opened by various table/diagram editors.

The Export method is `Table`.

When exporting to CSV format a dialogue opens requesting output file settings.



Export settings:

- OEM codepage : OEM coding for exported file
- Separator : separator used between columns
- Open after export : exported file is opened immediately after export using default CSV viewer installed on the computer.

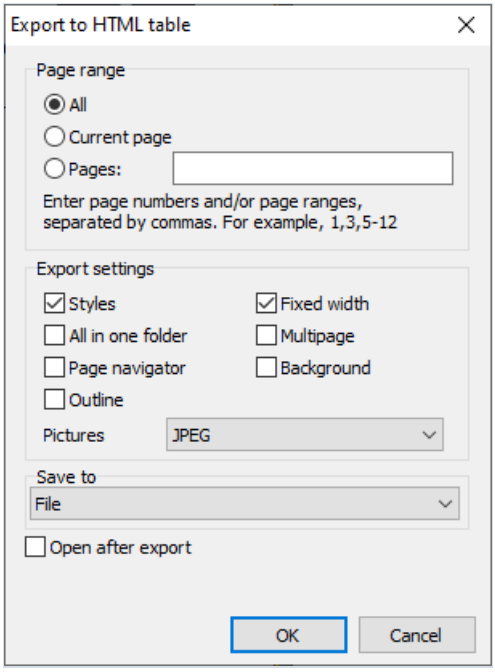
Export features : no layout information (i.e. report design) is included in the export file; graphic images are not supported.

Export into HTML Format

HTML (Hypertext Markup Language) is regarded as the standard language for documents on the Internet. It is used for creating relatively simple but well designed documents. HTML supports hypertext linking as well as simple document layout.

The Export method is `Table`.

When exporting to HTML format a dialogue opens requesting output file settings.



Export settings:

- Styles : design styles for text objects exported; when disabled exporting speed is increased, but detracts from spreadsheet appearance
- All in one folder : all additional files are saved in the same folder as the main file
- Page navigator : special navigator for fast jumping between pages created
- Fixed width : blocks automatic table/diagram width adjustment when changing display window size
- Multipage : each page exported as a separate file
- Background : graphic image assigned to page background is exported to HTML file
- Pictures : graphic image exported to HTML file
- Open after export : exported file is opened immediately after export using default HTML viewer installed on the computer.

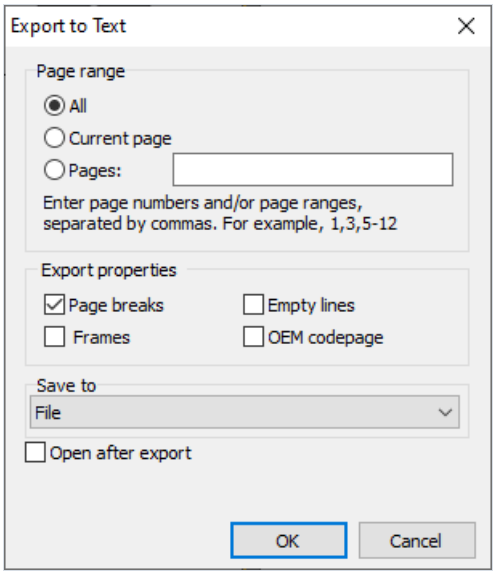
Export features : export may consist of several files; each graphic image exported to its own file; RichText objects are exported as simple text; accuracy of rendering and file-size depend on how the report was designed, see the Report Design considerations chapter.

Export to Text Format

Normal plain text file without any graphics.

The Export method is `Table`.

When exporting to Text format a dialogue opens requesting output file settings.



Export settings:

- Page breaks : export of page breaks to resulting file
- Empty lines : enables page breaks in the document
- Frames : enables text object frame export
- OEM codepage : OEM coding for exported file
- Open after export : exported file is opened immediately after export using default text viewer installed on the computer.

Export features: no layout information (i.e. report design) is included in the export file; graphic images are not supported: page width is automatically set dependent on the type of text objects on report page.

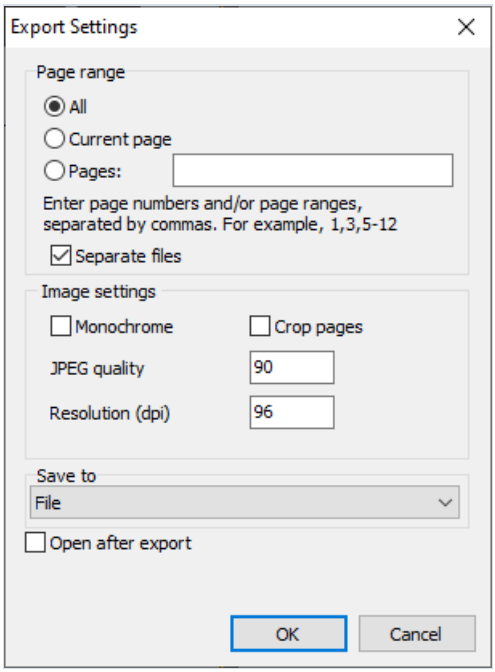
Export to Jpeg, BMP, Gif, Tiff Graphic Formats

FastReport exports to graphic formats.

- JPEG (Joint Photographic Experts Group) : a compressed format based on an algorithm that records differences between pixels. It is characterized by high compression at the expense of graphic accuracy.
- BMP (Windows Device Independent Bitmap) : used for storage of bitmap images used in Windows. A standard file format for computers under Windows control.
- GIF (Graphics Interchange Format) : hardware independent format was developed for transmission of bitmap images through networks. Well suited for compressing homogeneous content (logos, inscriptions, schemes).
- TIFF, TIF (Target Image File Format) : hardware independent format. Today it is one of the most widespread and reliable in polygraphy and facsimile transmission.

The Export method is `Drawing`.

When exporting in the above graphic formats a dialogue opens requesting output file settings.



Export settings:

- Separate files : when enabled each report page is exported to a separate file; filenames are derived from the specified filename by suffixing an underscore + page number
- Monochrome : exports as a monochrome image
- Crop pages : blank space round the page edges are cropped
- JPEG quality : JPEG compression ratio; only enabled when exporting to JPEG format
- Resolution : graphic resolution of exported image

Export features: when the Separate files is disabled one very large file is created.

Report design considerations

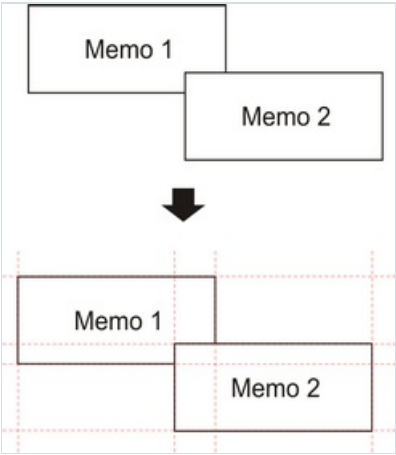
The quality of the export output in any format is highly dependent on the quality of the original report design. FastReport can manipulate objects in a large number of ways during report creation. This has the advantage of fast development of reports and subsequent printing. Printed documents look just as previewed. This is the primary objective of the FastReport report engine.

The downside of this development freedom is the complexity of exporting FastReport documents to different data formats, each of which has its own, sometimes complex, requirements and limitations. In this chapter, special design requirements for reports intended for export will be discussed.

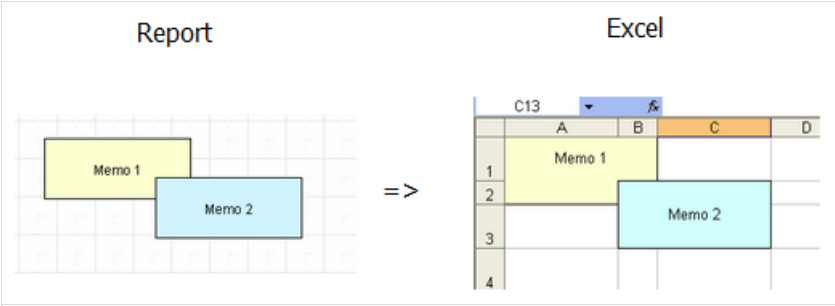
Many formats use tabular data presentation, such as HTML, XLS, XML, RTF and CSV. In contrast to the freedom allowed in FastReport page design, when exporting to these formats the output cells created for these tables cannot intersect or be arranged in layers.

Export filters take into account these requirements when objects are exported by FastReport, by the use of special algorithms dealing with intersections and the optimal placing of cells. At object intersections new columns and lines are created in the output table. This is necessary to enable FastReport to position objects exactly and to obtain the best correspondence with the original preview page.

A large number of intersecting objects in a report design leads to a large number of additional columns and rows in the output table. This in turn can lead to the need for editing of the exported file in its own editor before suitable for further use.



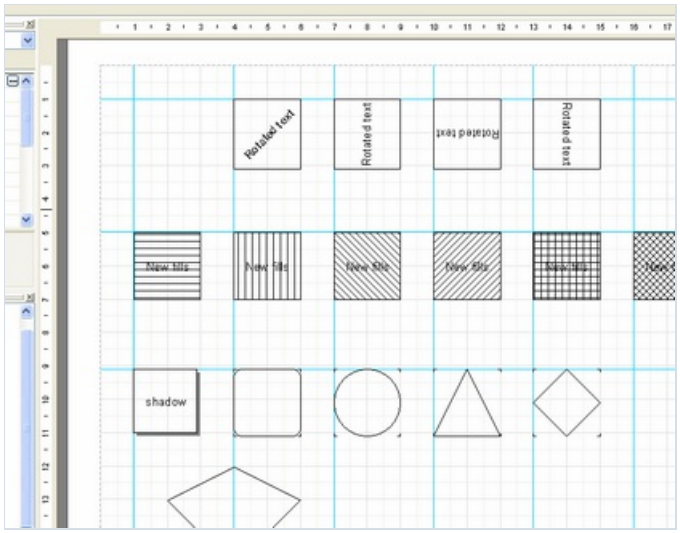
For example, take a report where the design has a slight overlap of two objects in the same band and the number of records in the report is 150. On export to RTF format 450 lines will be created (150 rows for each object and 150 rows for the intersection). If we remove the overlap there would only be 300 rows. For large reports with a large number of objects the difference would be much greater. This, of course, affects the size of the output file.



Bear this in mind when designing reports intended for export to any of the formats that use the 'table' output method.

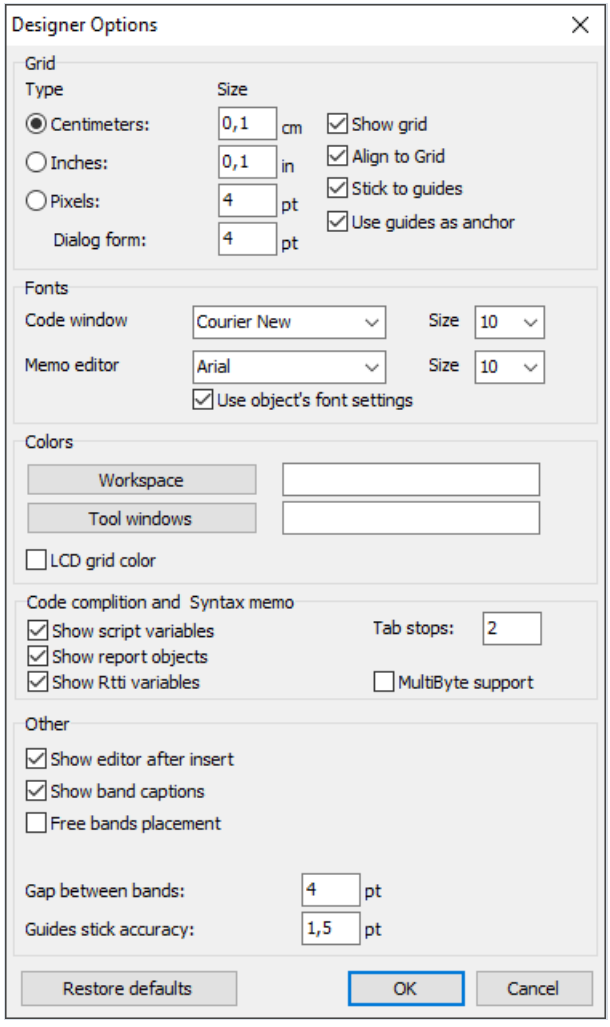
When designing tables in reports be aware of the borders of neighboring cells. It is important that cells do not overlap and are not arranged in layers. The export algorithm may deal with the cells in an unexpected way and give a result far from that intended.

It is best to arrange objects in such a way that they are placed in line both vertically and horizontally. Guidelines can help to achieve this.



To use guidelines in FastReport designer just click on the horizontal or vertical ruler at the top or left edge of the report page and drag the ruler to the required position on the page, where a guideline will be displayed. You will then be able to place objects aligning to these horizontal and vertical guidelines.

Grid alignment can also be helpful in placing “Text” objects to avoid them overlapping. The grid is enabled in the designer Options, where the pitch can also be adjusted : “View > Options > Grid”.

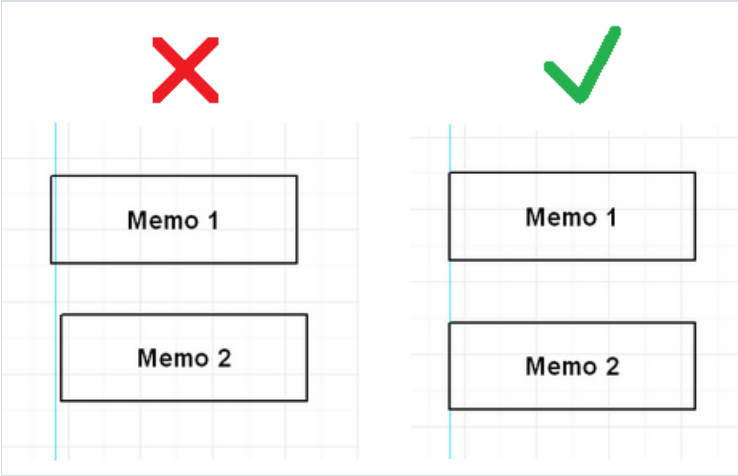


When using frames around “Text” objects it is best to use the object's frame properties, rather than adding graphic objects like lines and rectangles around the text. Also, try not to use objects in the background beneath transparent text objects.

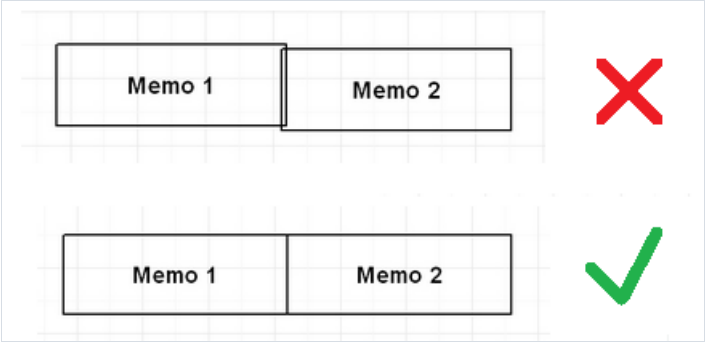
Keeping these simple rules in mind will help you to create a report which will look perfect after export to any format that uses the table-based output method.

Below are some examples of good and bad arrangement of objects.

The objects are displaced horizontally - they do not line up horizontally with the vertical guideline:



The objects are overlapping - on export to a table/diagram format additional unnecessary rows and columns and also three additional cells in the overlap zone are created:



Study of the demo reports included in the FastReport installation is recommended, to help master the basic principles of good report design.