

# **FastReport.Net Programmer's manual**



# Table of contents

<b>Chapter I</b>	<b>General information</b>	<b>6</b>
Installing into VS Toolbox		6
Troubleshooting		6
Deployment		7
Compiling the source code		7
<b>Chapter II</b>	<b>Working with Windows.Forms</b>	<b>10</b>
Using the Report component in Visual Studio		10
Working with report in a code		11
Storing and loading a report		12
Registering data		13
Passing a value to a report parameter		14
Running a report		14
Designing a report		15
Exporting a report		15
Configuring the FastReport.Net environment		16
Replacing the "Open" and "Save" dialogs		18
Replacing the standard progress window		19
Passing own connection string		20
Passing custom SQL		21
Reference to a report object		21
Creating a report by using code		21
Using own preview window		24
Filtering tables in the Data Wizard		24
<b>Chapter III</b>	<b>Working with ASP.NET</b>	<b>26</b>
Using the WebReport component		26
Storing and loading a report		27
Registering data		28
Passing a value to a report parameter		28
Working in the "Medium Trust" mode		29



# Chapter

---



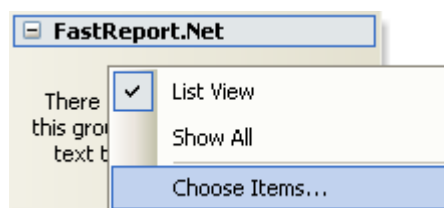
## General information

## General information

### Installing into VS Toolbox

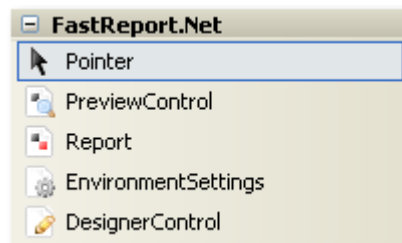
The FastReport.Net installation program automatically adds FastReport components into the Visual Studio Toolbox. If you have disabled this feature when installing, you may add the components manually. To do this:

- delete the "FastReport.Net" tab from the Toolbox, if it is there;
- create a new tab (to do this, right-click the Toolbox and select "Add Tab" item), or select an existing tab you would like to add FastReport components to;
- right-click on a tab and select "Choose Items...":



- in the dialog, press the "Browse..." button and choose FastReport.dll, FastReport.Web.dll files (they are located in the "C:\Program Files\FastReports\FastReport.Net" folder);
- close the dialog with OK button.

After this, you will see FastReport.Net components in a chosen tab:



- Report;
- PreviewControl;
- DesignerControl;
- EnvironmentSettings;
- WebReport (this component will be visible in ASP.NET project only).

### Troubleshooting

If you face a problem when working with report designer (for example, some toolbars or tool windows are damaged), you should delete the configuration file. This file is created when you start FastReport.Net. It is located in the following folder:

Windows XP:

*C:\Documents and Settings\user\_name\Local Settings\Application Data\FastReport\FastReport.config*

Windows Vista:

`C:\Users\user_name\AppData\Local\FastReport\FastReport.config`

The following information is stored in the config:

- sizes and locations of dialog windows;
- toolbar settings;
- recent used data connections;
- email settings (if you use the "Send Email" feature in the preview).

## Deployment

You may redistribute the following files along with your application:

- FastReport.dll - the main FastReport.Net library;
- FastReport.Web.dll - the library that contains ASP.Net WebReport component;
- FastReport.Bars.dll - the toolbars and docking windows library;
- FastReport.Editor.dll - the code editor with syntax highlight. This library is not needed if you don't provide an end-user designer;
- FastReport.xml - comments for classes, properties and methods used in FastReport. This file is used by the script editor, and also by the hint panels in the "Data" and "Properties" windows. It's not obligatory to distribute this file.

You may distribute the User's Manual which is contained in the FRNetUserManual.chm file. This file can be shown from the report designer, if you select the "Help|Contents..." menu item.

If your reports are stored in files, you have to deploy them as well.

## Compiling the source code

Source code is shipped with FastReport.Net Professional edition. It includes source code of FastReport.dll, FastReport.Web.dll libraries. You may include it in your application's solution file. Let us demonstrate how to do this:

- open your project in the Visual Studio;
- open the Solution Explorer and right-click on the "Solution" item;
- choose the "Add/Existing Project..." item;
- add the "FastReport.csproj" file (it is located in the "C:\Program Files\FastReports\FastReport.Net\Source\FastReport" folder);
- add the "FastReport.Web.csproj" file (it is located in the "C:\Program Files\FastReports\FastReport.Net\Source\FastReport.Web" folder).

Turn off assembly signing for FastReport and FastReport.Web projects. To do this:

- right-click the "FastReport" project in the Solution Explorer;
- choose the "Properties" item;
- switch to the "Signing" tab and uncheck the "Sign the assembly" checkbox;
- do the same steps for FastReport.Web project.

Update the references to other FastReport.Net assemblies. To do this:

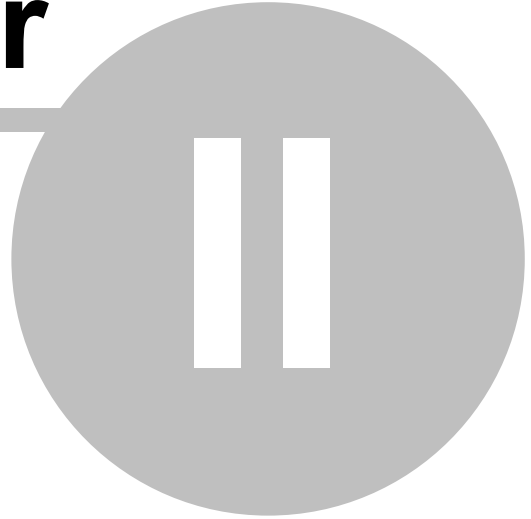
- expand the "FastReport\References" item in the Solution Explorer;

- remove the "FastReport.Bars", "FastReport.Editor" references;
- right-click the "References" item and choose the "Add Reference..." item;
- add references to the "FastReport.Bars.dll", "FastReport.Editor.dll" and "System.Windows.Forms.DataVisualization.dll" files. These files are located in the "C:\Program Files\FastReports\FastReport.Net" folder.



# Chapter

---



## **Working with Windows.Forms**

# Working with Windows.Forms

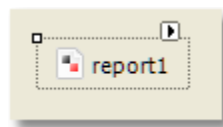
## Using the Report component in Visual Studio

Let us consider the typical use of the Report component in Visual Studio. We will use the data from a typed dataset.

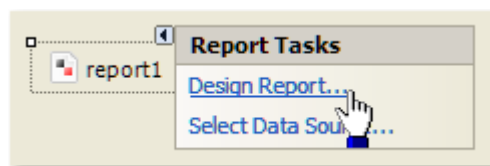
- create a new Windows Forms application;
- add a dataset into it ("Data|Add New Data Source..." menu item);
- switch to the Form designer;
- add the "DataSet" component on a form and connect it to the typed dataset that you have created.

To create a report, perform the following steps:

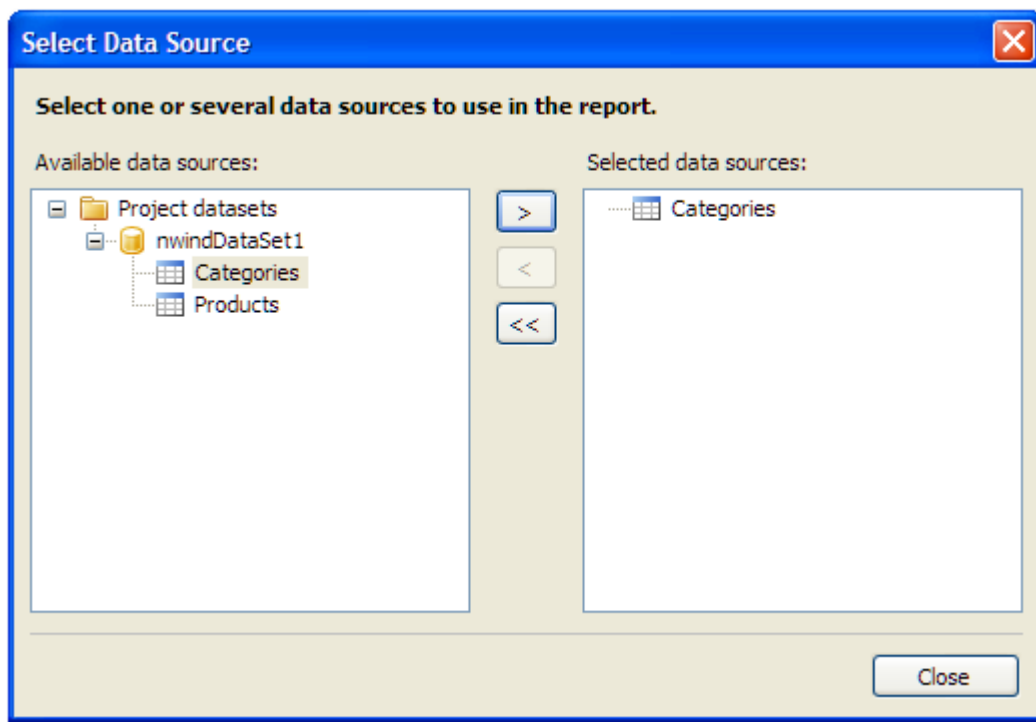
- put the "Report" component on a form:



- right-click it (or click on a smart tag button) and select the "Design Report..." item:



- choose the data source to use in a report:



- create your report. Read more about this in the User's Manual;
- close the report designer;
- add a "Button" control on your form;
- double-click it and write the following code in the button\_Click event handler:

```
report1.Show();
```

- save the project and run it. When you click on a button you will see the prepared report.

## Working with report in a code

To work with Report component in a code, you need to do the following:

- create a report instance;
- load a report file into it;
- register the application-defined data in a report;
- pass the values into the report parameters, if needed;
- run the report.

The following example demonstrates how to do this:

```
using (Report report = new Report())
{
    report.Load("report1.frx");
    report.RegisterData(dataSet1, "NorthWind");
    report.Show();
}
```

We will consider these steps in details in the following sections of this manual.

## Storing and loading a report

You may store a report in the following ways:

Method	Description
in the application's resources	<p>The typical scenario of using the Report, which we looked at before, uses this method. The <code>StoreInResources</code> property of the Report object is responsible for this. This property is set to <b>true</b> by default. This method has the following pros and cons:</p> <ul style="list-style-type: none"><li>+ a report is embedded into your application, you don't need to deploy extra files;</li><li>- if you need to change a report, you have to recompile your application.</li></ul> <p>Loading a report is performed automatically. To do this, FastReport.Net adds a code into the <code>InitializeComponent</code> method of your form.</p>
in the .FRX file	<p>This method is useful if you want to give your users the ability to change a report. In this case, set the report's <code>StoreInResources</code> property to <b>false</b>.</p> <p>To load the report from a file, use the <code>Load</code> method of the Report object:</p> <pre>report1.Load("filename.frx");</pre>
in the database	<p>You may store a report in the database, either as a string or in a blob-stream.</p> <p>To load the report from a string, use the <code>LoadFromString</code> method of the Report object. To load the report from a stream, use the overloaded version of the <code>Load</code> method:</p> <pre>report1.Load(stream);</pre> <p>To support the load/save operations in the report designer, you need to replace the "Open File" and "Save File" dialogs in the designer. Read here how to do this.</p>
as a C#/VB.NET class	<p>To work with a report as a class, design your report and save in to the .cs/.vb file. To do this, select "file type" in the "Save" dialog. The file type maybe either .cs or .vb - it depends on the script language in the report (it may be changed in the "Report Options..." menu). Include that file into your project. This method has the following pros and cons:</p> <ul style="list-style-type: none"><li>+ you can work with a report as a class;</li><li>+ you may debug a report;</li><li>+ this is the only way to use a report in ASP.NET project running on medium trust environment;</li><li>- you cannot edit such a report. To do this, you need the original .FRX file;</li><li>- if you need to change a report, you have to recompile your application.</li></ul> <p>To work with a report, create an instance of the report's class:</p> <pre>SimpleListReport report = new SimpleListReport(); report.Show();</pre>

## Registering data

If your report uses data from an application (for example, the typed dataset or a business-object), you have to register such data in a report. This can be done using the `RegisterData` method of the `Report` object.

When you use the Report as described in the "Using the Report component in Visual Studio" section, you don't need to register the data. FastReport.Net does it automatically (it adds the `RegisterData` call in the `InitializeComponent` method of your form).

The `RegisterData` method must be called after you have loaded the report:

```
report1 = new Report();
report1.Load("report.frx");
report1.RegisterData(dataSet1, "NorthWind");
```

The `RegisterData` method is overloaded and allows to register the following data:

Method	Description
<code>void RegisterData(DataSet data)</code>	Registers the dataset. This method registers all tables, views and relations as well.  Attention: if you register more than one dataset, use the <code>RegisterData(DataSet data, string name)</code> method instead.
<code>void RegisterData(DataSet data, string name)</code>	Registers the dataset. Specify any name in the <i>name</i> parameter (it must be persistent and unique if you register more than one dataset).
<code>void RegisterData(DataTable data, string name)</code>	Registers the data table.
<code>void RegisterData(DataView data, string name)</code>	Registers the data view.
<code>void RegisterDataAsp(IDataSource data, string name)</code>	Registers the ASP.NET data source such as <code>AccessDataSource</code> .
<code>void RegisterData(DataRelation data, string name)</code>	Registers the relation.
<code>void RegisterData(IEnumerable data, string name, BOConverterFlags flag, int maxNestingLevel)</code>	Registers the business object. Specify what items (properties, fields) should be used, in the <i>flags</i> parameter. Specify the maximum nesting level in the <i>maxNestingLevel</i> parameter (typically you need no more than 3 levels). Several nested objects may slow down the report.

## Passing a value to a report parameter

The report may have parameters. Read more about this in the User's Manual. To pass a value to the parameter, use the `SetParameterValue` method of the Report object:

```
report1.Load("report.frx");  
report1.SetParameterValue("MyParam", 10);  
report1.Show();
```

This method is declared as follows:

```
public void SetParameterValue(string complexName, object value)
```

Specify the parameter's name in the *complexName* parameter. To access a nested parameter, use its full name, for example:

```
"ParentParameter.ChildParameter"
```

## Running a report

To run a report, use one of the following methods of the Report object:

Method	Description
<code>void Show()</code>	<p>Runs a report and shows it in the preview window. This method is equal to:</p> <pre>if (Prepare())     ShowPrepared();</pre>
<code>bool Prepare()</code>	<p>Runs a report. If the report was prepared successfully, returns <b>true</b>. After this method, you need to call one of the following methods: <code>ShowPrepared</code>, <code>PrintPrepared</code>, <code>SavePrepared</code>, <code>Export</code>:</p> <pre>if (Prepare())     ShowPrepared();</pre>
<code>bool Prepare(     bool append)</code>	<p>Runs a report. If the <i>append</i> parameter is set to <b>true</b>, the prepared report will be added to the existing one. So you can build several reports and display them in the preview as one report:</p> <pre>report1.Load("report1.frx"); report1.Prepare(); report1.Load("report2.frx"); report1.Prepare(true); report.ShowPrepared();</pre>
<code>void ShowPrepared</code>	<p>Shows a prepared report in the preview window. The report must be either prepared using the <code>Prepare</code> method, or loaded from the .FPX file using the <code>LoadPrepared</code> method:</p> <pre>if (Prepare())     ShowPrepared();</pre>

<code>void ShowPrepared bool modal)</code>	Shows a prepared report in the preview window. The <i>modal</i> parameter determines whether the preview should be shown modally.
<code>void ShowPrepared bool modal, Form owner)</code>	The same as the previous method. The <i>owner</i> parameter determines a window that owns the preview window.
<code>void ShowPrepared Form mdiParent)</code>	The same as the previous method. The <i>mdiParent</i> parameter determines the main MDI window.

## Designing a report

You can use the report designer in your application. This is possible for all FastReport.Net editions except the Basic. To do this, use the Design method of Report object:

```
report1 = new Report();
report1.Load("report1.frx");
report1.Design();
```

The Design method is overloaded:

Method	Description
<code>bool Design()</code>	Shows the designer.
<code>bool Design( bool modal)</code>	Shows the designer. The <i>modal</i> parameter determines whether it is necessary to show the designer modally.
<code>bool Design( Form mdiParer</code>	Shows the designer. The <i>mdiParent</i> parameter defines the main MDI window.

## Exporting a report

The prepared report can be exported to one of the supported formats. At this moment, the following formats can be used:

- PDF
- HTML
- RTF
- Excel XML (Excel 2003+)
- Excel 2007
- CSV
- TXT
- OpenOffice Calc
- Pictures (Bmp, Png, Jpeg, Gif, Tiff, Metafile)

The export is performed by the export filter. To do this:

- prepare a report using the Prepare method;
- create an instance of export filter and set up its properties;

- call the Export method of the Report object.

The following example exports a prepared report in the HTML format:

```
// prepare a report
report1.Prepare();

// create an instance of HTML export filter
FastReport.Export.Html.HTMLExport export = new FastReport.Export.Html.HTMLExport();

// show the export options dialog and do the export
if (export.ShowDialog())
    report1.Export(export, "result.html");
```

In this example, export settings are made using the dialog window.

## Configuring the FastReport.Net environment

Using the EnvironmentSettings component which is available in the Toolbox, you can control some FastReport.Net environment settings. To do this, put the component on your form and set up its properties using the Properties window.

The EnvironmentSettings.ReportSettings property contains some report-related settings:

Property	Description
Language DefaultLanguage	The default script language for new reports.
bool ShowProgress	Determines whether it is necessary to show the progress window.
bool ShowPerformance	Determines whether to show the information about the report performance (report generation time, memory consumed) in the lower right corner of the preview window.

The EnvironmentSettings.DesignerSettings property contains some designer-related settings:

Property	Description
Icon Icon	The icon for the designer window.
Font DefaultFont	The default font used in a report.

The EnvironmentSettings.PreviewSettings property contains some preview-related settings:

Property	Description
PreviewButtons Button	Set of buttons that will be visible in the preview's toolbar.
int PagesInCache	The number of prepared pages that can be stored in the memory cache during preview.
bool ShowInTaskbar	Determines whether the preview window is displayed in the Windows taskbar.
bool TopMost	Determines whether the preview window should be displayed as a topmost form.



Icon Icon	The icon for the preview window.
string Text	The text for the preview window. If no text is set, the default text "Preview" will be used.

The EnvironmentSettings.EmailSettings property contains email account settings. These settings are used in the "Send Email" feature in the preview window:

Property	Description
string Address	Sender address (e.g. your email address).
string Name	Sender name (e.g. your name).
string MessageTemplate	The message template that will be used to create a new message. For example, "Hello, Best regards, ...".
string Host	SMTP host address.
int Port	SMTP port (25 by default).
string UserName, string Password	User name and password. Leave these properties empty if your server does not require authentication.
bool AllowUI	Allows to change these settings in the "Send Email" dialog. Settings will be stored in the FastReport.Net configuration file.

UI style settings are available in the following properties of EnvironmentSettings component:

Property	Description
UIStyle UIStyle	The style of designer and preview form. 6 styles are available - VisualStudio2005, Office2003, Office2007Blue, Office2007Silver, Office2007Black, VistaGlass.  The default style is Office2007Black.
bool UseOffice2007For	This property affects the designer and preview form. It determines whether the Office2007-style form should be used if one of the following styles is selected: Office2007Blue, Office2007Silver, Office2007Black, VistaGlass.  Default value is true.

Besides these properties, the EnvironmentSettings component has some events. Using such events, you may do the following:

- replace standard "Open file" and "Save file" dialogs in the designer;
- replace standard progress window;
- pass own connection string to a connection defined in the report.

These tasks will be described in the following sections of this manual.

## Replacing the "Open" and "Save" dialogs

If you decided to store a report in the database, you may need to change the designer in such a way that it can open and save reports from/to a database. That is, you need to replace standard "Open" and "Save" dialogs with your own dialogs that work with database. To do this, use the EnvironmentSettings component (see the previous section). This component has the following events:

Event	Description
CustomOpenDialog	<p>Occurs when the report designer is about to show the "Open" dialog. In the event handler, you must display a dialog window to allow user to choose a report file. If dialog was executed successfully, you must return <code>e.Cancel = false</code> and set the <code>e.FileName</code> to the selected file name.</p> <p>The following example demonstrates how to use this event:</p> <pre>private void CustomOpenDialog_Handler(     object sender, OpenSaveDialogEventArgs e) {     using (OpenFileDialog dialog = new OpenFileDialog())     {         dialog.Filter = "Report files (*.frx) *.frx";          // set e.Cancel to false if dialog         // was successfully executed         e.Cancel = dialog.ShowDialog() != DialogResult.OK;         // set e.FileName to the selected file name         e.FileName = dialog.FileName;     } }</pre>
CustomSaveDialog	<p>Occurs when the report designer is about to show the "Save" dialog. In the event handler, you must display a dialog window to allow user to choose a report file. If dialog was executed successfully, you must return <code>e.Cancel = false</code> and set the <code>e.FileName</code> to the selected file name.</p> <p>The following example demonstrates how to use this event:</p> <pre>private void CustomSaveDialog_Handler(     object sender, OpenSaveDialogEventArgs e) {     using (SaveFileDialog dialog = new SaveFileDialog())     {         dialog.Filter = "Report files (*.frx) *.frx";         // get default file name from e.FileName         dialog.FileName = e.FileName;          // set e.Cancel to false if dialog         // was successfully executed         e.Cancel = dialog.ShowDialog() != DialogResult.OK;         // set e.FileName to the selected file name         e.FileName = dialog.FileName;     } }</pre>

CustomOpenReport	<p>Occurs when the report designer is about to load the report. In the event handler, you must load the report specified in the e.Report property from the location specified in the e.FileName property. The latter property contains the name that was returned by the CustomOpenDialog event handler. It may be the file name, the database key value, etc.</p> <p>The following example demonstrates how to use this event:</p> <pre>private void CustomOpenReport_Handler(     object sender, OpenSaveReportEventArgs e) {     // load the report from the given e.FileName     e.Report.Load(e.FileName); }</pre>
CustomSaveReport	<p>Occurs when the report designer is about to save the report. In the event handler, you must save the report specified in the e.Report property to the location specified in the e.FileName property. The latter property contains the name that was returned by the CustomSaveDialog event handler. It may be the file name, the database key value, etc.</p> <p>The following example demonstrates how to use this event:</p> <pre>private void CustomSaveReport_Handler(     object sender, OpenSaveReportEventArgs e) {     // save the report to the given e.FileName     e.Report.Save(e.FileName); }</pre>

## Replacing the standard progress window

The progress window is shown during the following actions:

- running a report
- printing
- exporting

You may turn off the progress by setting the ReportSettings.ShowProgress property of the EnvironmentSettings component to **false**. Besides that, you may replace the standard progress window with your own. To do this, use the following events of the EnvironmentSettings component (see the "Configuring the FastReport.Net environment" section):

Event	Description
StartProgress	Occurs once before the operation. In this event, you have to create your own progress window and show it.
Progress	Occurs each time when current report page is handled. In this event, you have to show the progress state in your window.
FinishProgress	Occurs once after the operation. In this event, you have to destroy the

progress window.

The Progress event takes e parameter of ProgressEventArgs type. It has the following properties:

Property	Description
<code>string</code> Message	The message text.
<code>int</code> Progress	The index of current report page being handled.
<code>int</code> Total	The number of total pages in a report. This parameter may be 0 when preparing a report, because the number of total pages is unknown.

In most cases, you need to display the text from the e.Message property, in the Progress event handler. Other parameters may be useful if you want to display a progress bar.

## Passing own connection string

If you use data sources that are defined inside a report, you may need to pass an application-defined connection string to a report. This can be done in three ways.

The first method: you pass a connection string directly to the Connection object in a report. Do the following:

```
report1.Load(...);  
// do it after loading the report, before running it  
// assume we have one connection in the report  
report1.Dictionary.Connections[0].ConnectionString = my_connection_string;  
report1.Show();
```

The second method: you pass a connection string using the report parameter. Do the following:

- run the report designer;
- in the "Data" window, create a new report parameter (with "MyParameter" name, for example). See the User's Manual for more details;
- in the "Data" window, select the "Connection" object that contains a data source;
- switch to the "Properties" window and set the ConnectionStringExpression property to the following:

**[MyParameter]**

- pass the connection string to the MyParameter parameter:

```
report1.SetParameterValue("MyParameter", my_connection_string);
```

The third method: use the DatabaseLogin event of the EnvironmentSettings component (see the "Configuring the FastReport.Net environment" section). This event occurs each time when FastReport opens the connection. Here is an example of this event handler:

```
private void environmentSettings1_DatabaseLogin(  
    object sender, DatabaseLoginEventArgs e)  
{  
    e.ConnectionString = my_connection_string;  
}
```

Keep in mind that the DatabaseLogin event is global, it works with all reports.

## Passing custom SQL

The report may contain data sources that are added using the Data Wizard (via "Data|Add Data Source..." menu). Sometimes it is needed to pass custom SQL to that data source from your application. To do this, use the following code:

```
using FastReport.Data;

report1.Load(...);
// do it after loading the report, before running it
// find the table by its alias
TableDataSource table = report1.GetDataSource("MyTable") as TableDataSource;
table.SelectCommand = "new SQL text";
report1.Show();
```

## Reference to a report object

When you work with a report as a class (see the "Storing a report and loading it" section), you may refer to the report objects directly. The following example demonstrates how to change the font of the "Text1" object contained in a report:

```
SimpleListReport report = new SimpleListReport();
report.Text1.Font = new Font("Arial", 12);
```

In other cases, you have to use the FindObject method of the Report object, if you need to get a reference to an object:

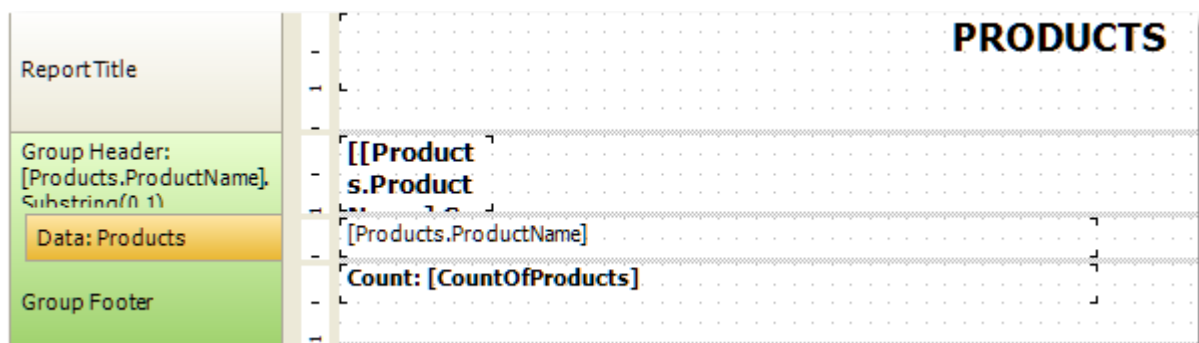
```
TextObject text1 = report1.FindObject("Text1") as TextObject;
text1.Font = new Font("Arial", 12);
```

To reference to a data source defined in a report, use the GetDataSource method of the Report object. This method takes a data source's alias as a parameter:

```
DataSourceBase ds = report1.GetDataSource("Products");
```

## Creating a report by using code

Let us consider how to create a report in code. We will create the following report:



```

Report report = new Report();

// register the "Products" table
report.RegisterData(dataSet1.Tables["Products"], "Products");
// enable it to use in a report
report.GetDataSource("Products").Enabled = true;

// create A4 page with all margins set to 1cm
ReportPage page1 = new ReportPage();
page1.Name = "Page1";
report.Pages.Add(page1);

// create ReportTitle band
page1.ReportTitle = new ReportTitleBand();
page1.ReportTitle.Name = "ReportTitle1";
// set its height to 1.5cm
page1.ReportTitle.Height = Units.Centimeters * 1.5f;

// create group header
GroupHeaderBand group1 = new GroupHeaderBand();
group1.Name = "GroupHeader1";
group1.Height = Units.Centimeters * 1;
// set group condition
group1.Condition = "[Products.ProductName].Substring(0, 1)";
// add group to the page.Bands collection
page1.Bands.Add(group1);

// create group footer
group1.GroupFooter = new GroupFooterBand();
group1.GroupFooter.Name = "GroupFooter1";
group1.GroupFooter.Height = Units.Centimeters * 1;

// create DataBand
DataBand data1 = new DataBand();
data1.Name = "Data1";
data1.Height = Units.Centimeters * 0.5f;
// set data source
data1.DataSource = report.GetDataSource("Products");
// connect databand to a group
group1.Data = data1;

// create "Text" objects

// report title
TextObject text1 = new TextObject();
text1.Name = "Text1";
// set bounds
text1.Bounds = new RectangleF(0, 0,
    Units.Centimeters * 19, Units.Centimeters * 1);
// set text
text1.Text = "PRODUCTS";
// set appearance
text1.HorzAlign = HorzAlign.Center;
text1.Font = new Font("Tahoma", 14, FontStyle.Bold);
// add it to ReportTitle
page1.ReportTitle.Objects.Add(text1);

// group
TextObject text2 = new TextObject();
text2.Name = "Text2";
text2.Bounds = new RectangleF(0, 0,
    Units.Centimeters * 2, Units.Centimeters * 1);
text2.Text = "[[Products.ProductName].Substring(0, 1)]";
text2.Font = new Font("Tahoma", 10, FontStyle.Bold);

```

```

// add it to GroupHeader
group1.Objects.Add(text2);

// data band
TextObject text3 = new TextObject();
text3.Name = "Text3";
text3.Bounds = new RectangleF(0, 0,
    Units.Centimeters * 10, Units.Centimeters * 0.5f);
text3.Text = "[Products.ProductName]";
text3.Font = new Font("Tahoma", 8);
// add it to DataBand
data1.Objects.Add(text3);

// group footer
TextObject text4 = new TextObject();
text4.Name = "Text4";
text4.Bounds = new RectangleF(0, 0,
    Units.Centimeters * 10, Units.Centimeters * 0.5f);
text4.Text = "Count: [CountOfProducts]";
text4.Font = new Font("Tahoma", 8, FontStyle.Bold);
// add it to GroupFooter
group1.GroupFooter.Objects.Add(text4);

// add a total
Total groupTotal = new Total();
groupTotal.Name = "CountOfProducts";
groupTotal.TotalType = TotalType.Count;
groupTotal.Evaluator = data1;
groupTotal.PrintOn = group1.Footer;
// add it to report totals
report.Dictionary.Totals.Add(groupTotal);

// run the report
report.Show();

```

The prepared report looks as follows:



## Using own preview window

Using the EnvironmentSettings component (see the "Configuring the FastReport.Net environment" section), you may tune up the standard preview window. The related properties are contained inside the EnvironmentSettings.PreviewSettings property.

If you don't want to use the standard preview window for some reason, you may create your own. To do this, use the PreviewControl control that can be added on your form. To show a report in this control, connect it to the Report object by the following code:

```
report1.Preview = previewControl1;
```

To prepare a report and show it in the PreviewControl, use the Show method of the Report object:

```
report1.Show();  
your_form.ShowDialog();
```

or the following code:

```
if (report1.Prepare())  
{  
    report1.ShowPrepared();  
    your_form.ShowDialog();  
}
```

In these examples, the *your\_form* is your form that contains the PreviewControl.

Using the methods of PreviewControl component, you can handle it from your code. You may even turn off the standard toolbar and/or statusbar, using the ToolbarVisible, StatusbarVisible properties. This is demonstrated in the Demos\C#\CustomPreview example project.

## Filtering tables in the Data Wizard

The Data Wizard can be called from the "Data|Add Data Source..." menu. Here you can set up the connection and choose one or several data tables. By default, the wizard displays all tables available in the selected connection. If you want to filter unnecessary tables, use the Config.DesignerSettings.FilterConnectionTables event. The following example shows how to remove the "Table 1" table from the tables list:

```
using FastReport.Utils;  
Config.DesignerSettings.FilterConnectionTables += FilterConnectionTables;  
  
private void FilterConnectionTables(  
    object sender, FilterConnectionTablesEventArgs e)  
{  
    if (e.TableName == "Table 1")  
        e.Skip = true;  
}
```



# Chapter

---



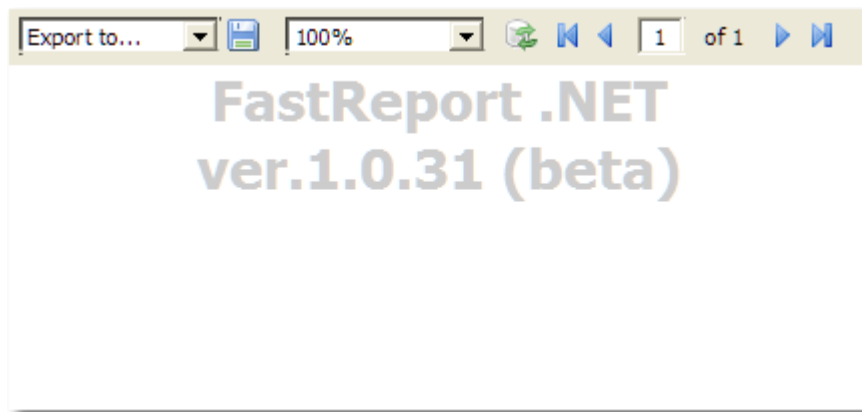
## Working with ASP.NET

## Working with ASP.NET

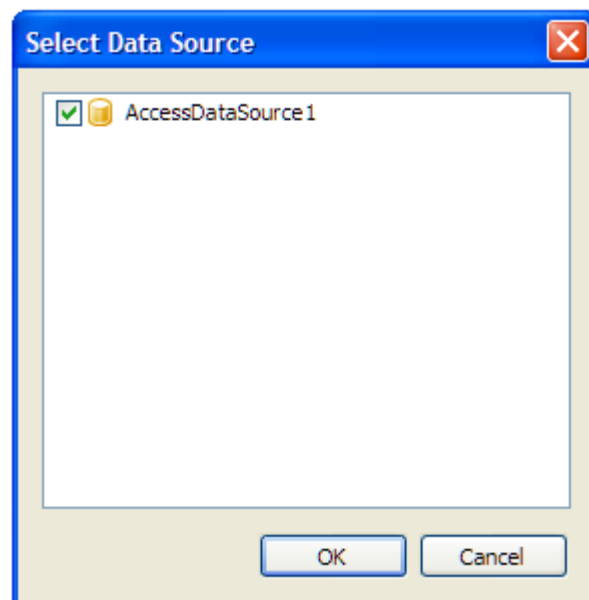
### Using the WebReport component

Let us consider the typical case of using the WebReport component.

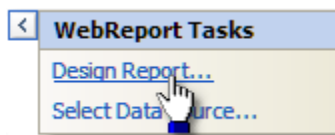
- assuming that you have a web project with all necessary data sources (for example, AccessDataSource);
- put the WebReport component on your web form:



- in the "smart tag" menu, select the "Select Data Source..." item and choose one or several data sources which you want to use in a report:



- in the "smart tag" menu, select the "Design Report..." item to run the report designer:



- create a report. Read more about this in the User's Manual;
- close the designer;
- save the changes in your project and run it. You will see a window with a prepared report.

## Storing and loading a report

You may store a report in the following ways:

Method	Description
in a web form	<p>Typical scenario that we have looked at before, uses this method. The report is stored in the ReportResourceString property of the WebReport component. This method has the following pros and cons:</p> <ul style="list-style-type: none"> <li>+ it's a simplest way to work with FastReport.Net;</li> <li>- the report template is stored in the ViewState of your web form. It will be transferred on a client side. It may slow down the work if the report has a big size;</li> <li>- this method is not compatible with "Medium Trust" mode.</li> </ul> <p>The report loading is performed automatically.</p>
in the .FRX file	<p>This method assumes that the report is stored in a file in a special folder "App_Data". To do this:</p> <ul style="list-style-type: none"> <li>• run the report designer;</li> <li>• create a report and save it to the .FRX file;</li> <li>• in the Solution Explorer, select the "App_Data" folder, right-click it and choose the "Add Existing Item..." item. Select the report file that you just saved;</li> <li>• select the WebReport component and clear its ReportResourceString property;</li> <li>• select the "ReportFile" property, invoke its editor and choose the report from "App_Data" folder.</li> </ul> <p>This method has the following pros and cons:</p> <ul style="list-style-type: none"> <li>+ the report is not transferred to a client machine;</li> <li>- this method is not compatible with "Medium Trust" mode.</li> </ul> <p>The report loading is performed automatically.</p>
as a C#/VB.NET class	<p>In this method, you work with the report as a class. To do this:</p> <ul style="list-style-type: none"> <li>• design your report and save in to the .cs/.vb file. To do this, select "file type" in the "Save" dialog. The file type maybe either .cs or .vb - it depends on the script language in the report (it may be changed in the "Report Options..." menu);</li> </ul>

- include that file into your project. It's better to save it in the "App\_Code" folder;
- clear both ReportResourceString and ReportFile properties of the WebReport component.

This method has the following pros and cons:

- + you can work with the report as a regular class;
- + you can debug the report in the Visual Studio;
- + it's the only way to use a report in the "Medium Trust" mode;
- you cannot edit such a report. To do this, you need the original .FRX file.

To work with a report, create the WebReport.StartReport event handler. In this handler, you should do the following:

- create an instance of your report class;
- register the data;
- set the report to the Report property of the WebReport component.

Example of the StartReport event handler:

```
SimpleListReport report = new SimpleListReport();
report.RegisterDataAsp(your_data, "your_data_name");
WebReport1.Report = report;
```

## Registering data

If you select the data source using the "smart tag" menu of the WebReport component, you don't need to register the data manually. In this case, FastReport.Net stores the names of data sources in the ReportDataSources property of the WebReport component.

In case you don't want to use such method of registering data, you need to do it manually. It can be done by using the StartReport event of the WebReport component. In this event handler, you can call the RegisterData and RegisterDataAsp methods of the report. The report can be accessed through the WebReport.Report property:

```
webReport1.Report.RegisterData(myDataSet);
```

Read more about registering data in this section.

## Passing a value to a report parameter

To pass a value to the report parameter, use the SetParameterValue method of the Report object. This method was described in details in the "Working with Windows.Forms" chapter.

To use this method in ASP.NET, you need to create the event handler for the StartReport event of the WebReport component. The report can be accessed through the WebReport.Report property:

```
webReport1.Report.SetParameterValue("MyParam", 10);
```

## Working in the "Medium Trust" mode

This mode is used by many shared hosting providers. In this mode, the following actions are restricted:

- report compilation is impossible;
- impossible to use MS Access data source;
- impossible to use the RichObject;
- impossible to use some export filters that use WinAPI calls or temp files (PDF, Open Office);
- there may be other restrictions, depending on the provider.

To work with a report in this mode, you need to store a report as a C#/VB.NET class, as described in the "Storing and loading a report" section. In this case, the report compilation is not required.

Besides that, it is necessary to add System.Windows.Forms.DataVisualization.dll assembly into the GAC. This assembly is a part of Microsoft Chart Control and is used in FastReport to draw charts. Consult with your shared-hosting provider regarding adding this assembly into the GAC.