FastReport WPF Programmers Manual

Version 2023.2

© 2008-2023 Fast Reports Inc.

General information

Installing into VS Toolbox

Troubleshooting

Deployment

Compiling the source code

Credits

Installing into VS Toolbox

Working with FastReport.Net* packages Toolbox is included in the package and should be automatically loaded by Visual Studio. Manual installation of Toolbox is also possible.

To add manually FastReport .NET components to the Visual Studio toolbox, you need to do the following steps:

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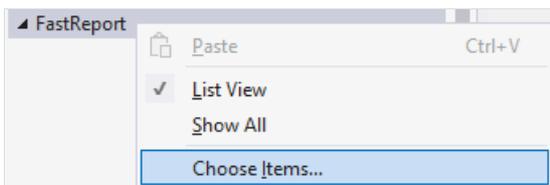
- in the dialog, press the "Browse..." button and choose FastReport.dll, FastReport.Web.dll files (they are located in the "C:\Program Files\FastReports\FastReport.Net" folder);
- close the dialog with OK button.

After this, you will see FastReport .NET components in a chosen tab:
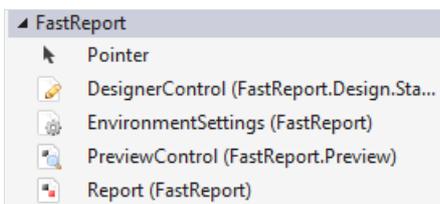
- Report;
- PreviewControl;
- DesignerControl;
- EnvironmentSettings;
- WebReport (this component will be visible in ASP.NET project only).

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. It is located in the following folder:

Windows XP:

```
C:\Documents and Settings\user_name\Local Settings\Application Data\FastReport
```

Windows Vista +:

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

The config file name varies depending on product:

- FastReport.config for FastReport .NET;
- FastReport.Mono.config for FastReport Mono;
- FastReport.WPF.config for FastReport WPF.

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 (FastReport.Mono.dll, FastReport.WPF.dll) - the main FastReport library;
- FastReport.Web.dll - the library that contains ASP.Net WebReport component;
- FastReport.Bars.dll - the toolbars and docking windows library (related to FastReport .NET);
- FastReport.Editor.dll - the code editor with syntax highlight. This library is not needed if you don't provide an end-user designer (related to FastReport .NET);
- FastReport.WPF.RoslynPad.dll - the code editor with syntax highlight for .Net 6.0+ (related to FastReport WPF);
- FastReport.Forms.WPF.dll - the forms library (related to FastReport WPF);
- FastReport.SkiaDrawing.dll - the Skia support library;
- FastReport.Compat.dll (FastReport.Compat.Skia.dll) - the library for cross-platform compatibility;
- FastReport.DataVisualization.dll (FastReport.DataVisualization.Skia.dll) - the library for drawing charts;
- FastReport.xml (FastReport.Mono.xml, FastReport.WPF.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.

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

FastReport NuGet packages:

- **FastReport.Core (demo on nuget.org)** - package containing the main logic of the program (obtaining the necessary data, rendering reports, exports, etc.). Some of the functionality from FastReport.NET is missing due to the cross-platform nature of this package.
- **FastReport.Net (demo on nuget.org)** - package with FastReport.dll library for .NET Framework 4.x, which is part of the 'Pro' and 'Demo' editions - for .NET Core 3.1, .NET 5 and .NET 6 exclusively for Windows. This package is available in several editions:
  - FastReport.NET.Demo - This package is available at nuget.org and in our official Trial installer, and it allows you to evaluate the product capabilities for various target frameworks on OS Windows (so-called FastReport.CoreWin). It has restrictions that are present in our other Demo edition products (5 page limit for exports, watermarked export pages, etc.).
  - FastReport.NET - This package is available in our official installer for licensed owners of the following editions: FastReport .NET WinForms and FastReport.Net Web. It includes FastReport .NET for .NET Framework 4.0 and higher.
  - FastReport.NET.Pro - This package is available in our official installer for licensed owners of the following editions: FastReport.NET Professional and Enterprise. It includes FastReport .NET for .NET Framework 4.0 or higher, FastReport.CoreWin for .NET Core 3.1 and for .NET 5.0 and higher.

- **FastReport.WPF (demo on nuget.org)** - package with FastReport WPF library;
- **FastReport.Web (demo on nuget.org)** - a package for integrating FastReport into scripts for working with web applications (rendering a report in a browser, exporting and printing from a browser, working with Online Designer) for ASP.NET Core. Includes components for Blazor Server and is used only with FastReport.Core.
- **FastReport.Core3.Web (demo on nuget.org)** - the same principle as FastReport.Web, but compatible with FastReport.CoreWin, which is included in the FastReport.Net.Demo/ FastReport.Net.Pro package.
- **FastReport.Localization (nuget.org)** - package with a set of FastReport localizations. Add it to your project if you need German or Russian localization, for example.
- **FastReport.Compat and FastReport.DataVisualization** - packages with basic logic (report compilation, MSChart support, etc.). You don't need to include them in your project, they are dependency packages.

- **FastReport.Data.** * - packages with connector plugins for FastReport to work with various databases, the connectors of which are not included in the source library. These packages are �common� for different FastReport editions and are suitable for both FastReport .NET and FastReport.Core and FastReport.CoreWin. Limitations: FastReport version 2021.4.0+ and NuGet Client 3.4.4+ are required.
  - FastReport.Data.ClickHouse
  - FastReport.Data.Couchbase
  - FastReport.Data.Firebird
  - FastReport.Data.Json
  - FastReport.Data.MongoDB
  - FastReport.Data.MsSql
  - FastReport.Data.MySql
  - FastReport.Data.OracleODPCore
  - FastReport.Data.Postgres
  - FastReport.Data.RavenDB
  - FastReport.Data.SQLite

Compiling the source code

FastReport's source code is shipped with Professional and Enterprise editions. You may include it in your application's solution file. Let us demonstrate how to do this with FastReport .NET project:

- 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);

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" and "FastReport.Editor.dll" files. These files are located in the "C:\Program Files\FastReports\FastReport.Net" folder.

To build FastReport WPF from source code, open the file "C:\Program Files\FastReports\FastReport WPF\Sources\FastReport.WPF\FastReport.WPF.sln". This solution file includes all FastReport WPF projects, as well as some sample projects.

Credits

Toolbars and docking - DevComponents (http://www.devcomponents.com)

Code editor with syntax highlight - Quantum Whale (http://www.qwhale.net)

Silk icons set (https://github.com/legacy-icons/famfamfam-silk)

Working with Windows.Forms and WPF

Using the Report component in Windows.Forms

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.

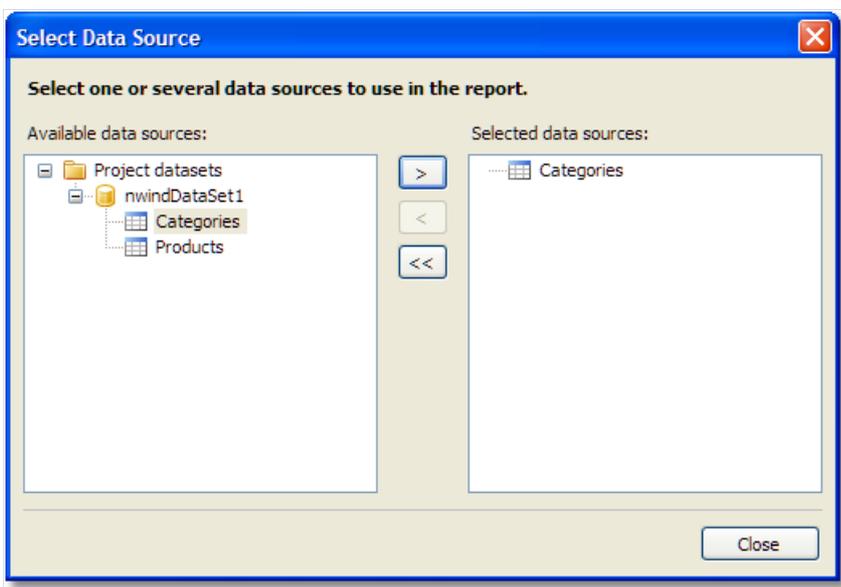To create a report, perform the following steps:

- put the "Report" component on a form:
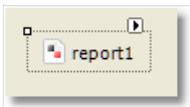
- right-click it (or click on a smart tag button) and select the "Design Report..." item:
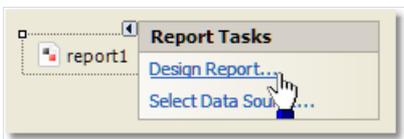
- choose the data source to use in a report:

- 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 (FastReport .NET/Mono)	The typical scenario of using the Report, which we looked at before, uses this method. The StoreInResources property of the Report object is responsible for this. This property is set to true by default. This method has the following pros and cons: + a report is embedded into your application, you don't need to deploy extra files; - if you need to change a report, you have to recompile your application. Loading a report is performed automatically. To do this, FastReport .NET adds a code into the InitializeComponent method of your form.
in the .FRX file	This method is useful if you want to give your users the ability to change a report. In this case, set the report's StoreInResources property to false. To load the report from a file, use the Load method of the Report object: `report1.Load("filename.frx");`
in the database	You may store a report in the database, either as a string or in a blob-stream. To load the report from a string, use the LoadFromString method of the Report object. To load the report from a stream, use the overloaded version of the Load method: `report1.Load(stream);` 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.

as a C#/VB.NET class	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:
	+ you can work with a report as a class;
	+ you may debug a report;
	+ this is the only way to use a report in ASP.NET project running on medium trust environment;
	- you cannot edit such a report. To do this, you need the original .FRX file;
	- if you need to change a report, you have to recompile your application.
	To work with a report, create an instance of the report's class:
	`SimpleListReport report = new SimpleListReport();`
	`report.Show();`

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
`void RegisterData(DataSet data)`	Registers the dataset. This method registers all tables, views and relations as well.Attention: if you register more than one dataset, use the RegisterData(DataSet data, string name) method instead.
`void RegisterData(DataSet data, string name)`	Registers the dataset. Specify any name in the name parameter (it must be persistent and unique if you register more than one dataset).
`void RegisterData(DataTable data, string name)`	Registers the data table.
`void RegisterData(DataView data, string name)`	Registers the data view.
`void RegisterDataAsp(IDataSource data, string name)`	Registers the ASP.NET data source such as AccessDataSource.
`void RegisterData(DataRelation data, string name)`	Registers the relation.
`void RegisterData(IEnumerable data, string name, BOConverterFlags flags, int maxNestingLevel)`	Registers the business object. Specify what items (properties, fields) should be used, in the flags parameter. Specify the maximum nesting level in the `maxNestingLevel` 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:

```
publicvoid 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

`void Show()`	Runs a report and shows it in the preview window.
	This method is equal to:
	`if (Prepare())`
	`  ShowPrepared();`

`bool Prepare()`	Runs a report. If the report was prepared successfully, returns true. After this method, you need to call one of the following methods: ShowPrepared, PrintPrepared, SavePrepared, Export:
	`if (Prepare())`
	`  ShowPrepared();`

`bool Prepare(bool append)`	Runs a report. If the append parameter is set to true, 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:
	`report1.Load("report1.frx");`
	`report1.Prepare();`
	`report1.Load("report2.frx");`
	`report1.Prepare(true);`
	`report.ShowPrepared();`

`void ShowPrepared()`	Shows a prepared report in the preview window. The report must be either prepared using the Prepare method, or loaded from the .FPX file using the LoadPrepared method:
	`if (Prepare())`
	`  ShowPrepared();`

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

Designing a report

You can use the report designer in your application. 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
`bool Design()`	Shows the designer.
`bool Design(bool modal)`	Shows the designer. The modal parameter determines whether it is necessary to show the designer modally.
`bool Design(Form mdiParent)`	Shows the designer. The mdiParent parameter defines the main MDI window (related to FastReport .NET/Mono).

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 environment

Using the FastReport.Utils.Config global class, you can control some FastReport environment settings.

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

Property	Description
`PreviewButtons Buttons`	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 Config.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).

Property	Description
`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 configuration file.

UI style settings are available in the following properties of Config class:

Property	Description
`UIStyle UIStyle`	The style of designer and preview form.

Besides these properties, there are 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 FastReport.Utils.Config.DesignerSettings global class (see the Configuring the FastReport environment). This class has the following events:

Event `CustomOpenDialog`

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 e.Cancel = false and set the e.FileName to the selected file name. The following example demonstrates how to use this event:

```
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 succesfully executed
    e.Cancel = dialog.ShowDialog() != DialogResult.OK;
    // set e.FileName to the selected file name
    e.FileName = dialog.FileName;
  }
}
```

Event `CustomSaveDialog`

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 e.Cancel = false and set the e.FileName to the selected file name. The following example demonstrates how to use this event:

```
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 succesfully executed
    e.Cancel = dialog.ShowDialog() != DialogResult.OK;
    // set e.FileName to the selected file name
    e.FileName = dialog.FileName;
  }
}
```

Event `CustomOpenReport`

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. The following example demonstrates how to use this event:

```
private void CustomOpenReport_Handler(object sender, OpenSaveReportEventArgs e)
{
  // load the report from the given e.FileName
  e.Report.Load(e.FileName);
}
```

Event `CustomSaveReport`

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. The following example demonstrates how to use this event:

```
private void CustomSaveReport_Handler(object sender, OpenSaveReportEventArgs e)
{
  // save the report to the given e.FileName
  e.Report.Save(e.FileName);
}
```

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 FastReport.Utils.Config.ReportSettings.ShowProgress property to false. Besides that, you may replace the standard progress window with your own. To do this, use the following events of the FastReport.Utils.Config.ReportSettings global class (see the "Configuring the FastReport 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
string Message	The message text.
int Progress	The index of current report page being handled.
int 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 FastReport.Utils.Config.ReportSettings global class (see the "Configuring the FastReport 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
```

```
// 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:

PRODUCTS

A

Aniseed Syrup
Alice Mutton
Count: 2

B

Boston Crab Meat
Count: 1

Using own preview window

Using the FastReport.Utils.Config.PreviewSettings global class (see the "Configuring the FastReport environment" section), you may tune up the standard preview window.

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 (WpfPreviewControl for FastReport WPF) 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; // FastReport .NET/Mono
report1.WpfPreview = previewControl1; // FastReport WPF
```

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 (in FastReport WPF, the demo is located here: Demos\WPF\CustomPreview).

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 FastReport.Utils.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;
}
```

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:
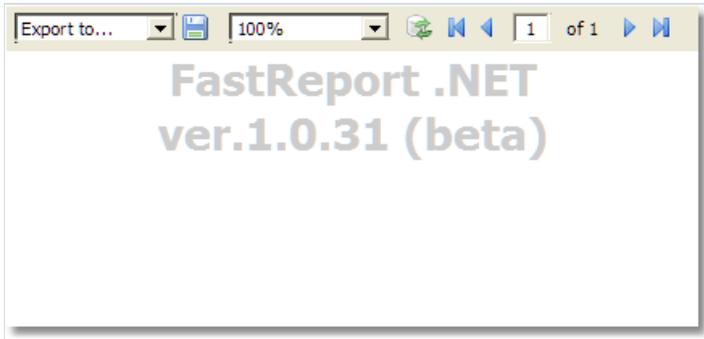
- in the "smart tag" menu, select the "Select Data Source..." item and choose one or several data sources which you want to use in a report:
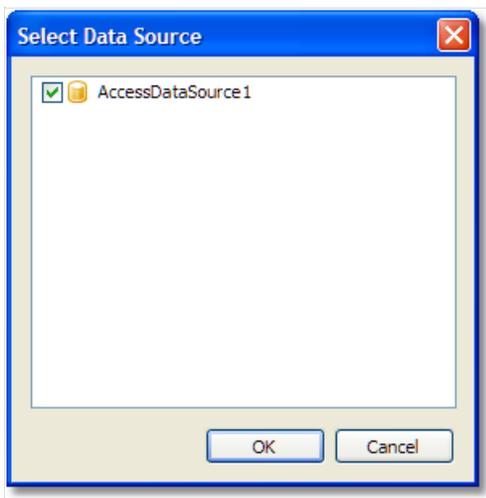
- in the "smart tag" menu, select the "Design Report..." item to run the report designer:
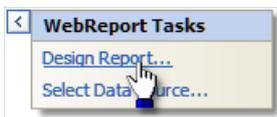
- create a report. Read more about this in the User's Manual;
- close the designer;
- save the changes in your project and run it. You will see a window with a prepared report.

Setting up web handler

WebReport requires a specific handler to be set in the web.config file. When you create a report object in Visual Studio, necessary lines are automatically written to the configuration file. The WebReport component checks the availability of the specified configuration at run time of the application. If the required lines are not found in the web.config file an error is thrown, requesting the file to be changed.

The web.config file should contain the following lines when used with an IIS6 server:

```
<system.web>
…
<httpHandlers>
 <add path="FastReport.Export.axd" verb="*" type="FastReport.Web.Handlers.WebExport"/>
</httpHandlers>
…
</system.web>
```

and with an IIS7 server, these lines:

```
<system.webServer>
…
<handlers>
 <add name="FastReportHandler" path="FastReport.Export.axd" verb="*"
type="FastReport.Web.Handlers.WebExport"/>
</handlers>
…
</system.webServer>
```

The correct WebReport configuration lines in the web.config file must be used when transferring your project from one server to another.

Check for correct working of the WebReport handler by means of the URL:

http://yoursite/app_folder/FastReport.Export.axd

An information message gives the version of FastReport and the server time.

Storing and loading a report

You may store a report in the following ways:

In a web form:

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:

+ it's a simplest way to work with FastReport.Net;
- 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;
- this method is not compatible with "Medium Trust" mode.

The report loading is performed automatically.

In the .FRX file:

This method assumes that the report is stored in a file in a special folder "App_Data". To do this:

- run the report designer;
- create a report and save it to the .FRX file;
- 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;
- select the WebReport component and clear its ReportResourceString property;
- select the "ReportFile" property, invoke its editor and choose the report from "App_Data" folder.

This method has the following pros and cons:

+ the report is not transferred to a client machine;
- this method is not compatible with "Medium Trust" mode.

The report loading is performed automatically.

A report can also be loaded from WebReport.StartReport event handler. Example code in StartReport:

```
(sender as WebReport).Report.Load(this.Server.MapPath("~/App_Data/report.frx"));
```

As a C#/VB.NET class:

In this method, you work with the report as a class. To do this:

- 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. 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;
```

The prepared report can be displayed from WebReport.StartReport event handler using the property WebReport.ReportDone. Example code in StartReport to load and display a prepared report:

```
(sender as WebReport).Report.LoadPrepared(this.Server.MapPath("~/App_Data/Prepared.fpx"));
(sender as WebReport).ReportDone = true;
```

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.

Working in Web Farm and Web Garden architectures

To use the FastReport report generator in a multi-server (Web Farm) or multi-processor (Web Garden) architecture there are additional requirements for creating special storage for data synchronization between WebReport objects.

Add the following lines to the configuration file web.config:

```
 <appSettings>
<add key="FastReportStoragePath" value="\\FS\WebReport_Exchange"/>
<add key="FastReportStorageTimeout" value="10"/>
<add key="FastReportStorageCleanup" value="1"/>
 </appSettings>
```

- FastReportStoragePath : path to the folder for temporary files when working in a multi-server architecture, each server must have access to this folder

- FastReportStorageTimeout : cache time for reports, in minutes

- FastReportStorageCleanup : time for checking the expired cache entries, in minutes

Check for correct configuration by means of the URL:

http://yoursite/app_folder/FastReport.Export.axd

You should see "Cluster mode: ON".

Working with ASP.NET MVC

You will not have any problems when using WebReport in ASPX (MVC 2) – it is only necessary to drag the control from the Toolbox to the page. WebReport will make all the required changes to web.config automatically. Let's look at a demo of WebReport in aspx, to be found in folder \Demos\C#\MvcDemo.

To use WebReport in Razor (MVC 3,4) you will need to add a line with the handler definitions to the web.config file in the root folder of your web-application.

Add this line in section <system.web> for use with IIS6:

```
<add path="FastReport.Export.axd" verb="*" type="FastReport.Web.Handlers.WebExport" />
```

and add this line in section <system.webServer> for use with IIS7:

```
<add name="FastReportHandler" path="FastReport.Export.axd" verb="*" type="FastReport.Web.Handlers.WebExport" />
```

Then modify the web.config file in the folder containing Views. Add these lines in section <system.web.webPages.razor> :

```
<add namespace="FastReport" />
<add namespace="FastReport.Web" />
```

Add these lines to file _Layout.cshtml in tag :

```
@WebReportGlobals.Scripts()
@WebReportGlobals.Styles()
```

Now you can draw the report on the View. Go to the controller and create a WebReport:

```
WebReport webReport = new WebReport(); // create object
```

```
    webReport.Width = 600;  // set width
    webReport.Height = 800; // set height
    webReport.Report.RegisterData(dataSet, "AppData"); // data binding
    webReport.ReportFile = this.Server.MapPath("~/App_Data/report.frx");  // load the report from the file
    ViewBag.WebReport = webReport; // send object to the View
```

Go to View and add the line:

```
@ViewBag.WebReport.GetHtml()
```

Similar code to create WebReport you can also write directly in View.

Let's look at the demo of WebReport in Razor in folder \Demos\C#\MvcRazor. There are various samples for loading into the report, including pre-prepared, and there is an example of using the event StartReport.

Don't forget to add the missing dll in the bin directory.

Example of export in MVC

When using FastReport.Net together with the ASP.Net MVC framework there is an easy method for creating a file in any supported format from a button press on the HTML form.

Add this code in the View:

```
@using (Html.BeginForm("GetFile", "Home"))
{
 <input id="pdf" type="submit" value="Export to PDF" />
}
```

- GetFile : name of the controller handler

- Home : name of the controller (eg: HomeController.cs)

Add the name space in the controller:

```
using FastReport.Export.Pdf;
```

Add method GetFile in the controller:

```
public FileResult GetFile()
{
  WebReport webReport = new WebReport();
  // bind data
  System.Data.DataSet dataSet = new System.Data.DataSet();
  dataSet.ReadXml(report_path + "nwind.xml");
  webReport.Report.RegisterData(dataSet, "NorthWind");

  // load report
  webReport.ReportFile = this.Server.MapPath("~/App_Data/report.frx");
// prepare report
webReport.Report.Prepare();
// save file in stream
Stream stream = new MemoryStream();
webReport.Report.Export(new PDFExport(), stream);
stream.Position = 0;
// return stream in browser
return File(stream, "application/zip", "report.pdf");
}
```

Example for Excel 2007:

```
using FastReport.Export.OoXML;
...
webReport.Report.Export(new Excel2007Export(), stream);
...
return File(stream, "application/xlsx", "report.xlsx");
```

FastReport .Net and jQuery

The WebReport object from FastReport.Net uses the jQuery library. You may already be using this library in your project.

To avoid duplication of jQuery boot scripts and styles in the client browser when working with markup Razor, you must use the following lines in _Layout.cshtml:

```
@WebReportGlobals.ScriptsWOjQuery()
@WebReportGlobals.StylesWOjQuery()
```

replacing these lines, which include all jQuery files:

```
@WebReportGlobals.Scripts()
@WebReportGlobals.Styles()
```

You must set the property ExternalJquery = true (defaults to false) when working with ASPX markup.

Working with WCF

WCF service library FastReport.Service.dll

FastReport .NET contains the library FastReport.Service.dll (only in the .NET 4.0 package). This library is a WCF Service Library and is intended for use in custom applications that perform the functions of the service.
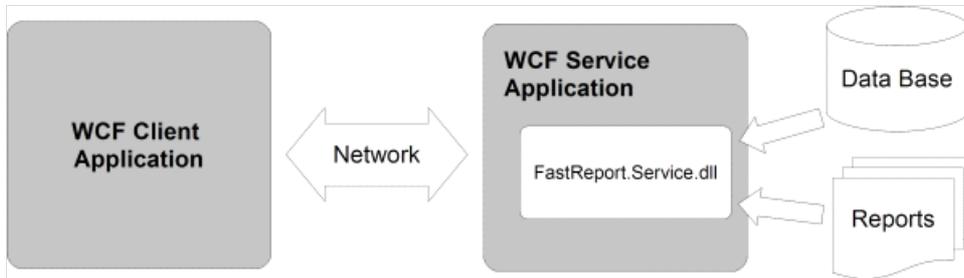
The library contains the following functions:

```
List<ReportItem> GetReportsList();
```

returns a list of available reports. Each item is returned as a ReportItem object. Reports are stored on a hard drive on a server that is running the service. Files are sorted in alphabetical order.

```
List<ReportItem> GetReportsListByPath(string path);
```

returns a list of available reports by path. Files are sorted in alphabetical order.

```
List<GearItem> GetGearList();
```

returns a list of available formats that can generate service reports as elements GearItem.

```
Stream GetReport(ReportItem report, GearItem gear);
```

returns a stream of the result of building a report. Parameters "report" and "gear" can be used from the lists previously obtained, or by creating new objects with the required properties. The returned stream does not support positioning.

Let's look at list elements.

ReportItem

```
public class ReportItem
{
 public string Path;
 public string Name;
 public string Description;
 public Dictionary<string, string> Parameters;
}
```

Path – path to the report file on the server, relative to the root folder for storing reports. The file extension of the report must be *.frx. This property is used to identify a specific report with further queries.

Name – name of the report, taken from the metadata of the report. If the metadata of the report contains an empty

name then the property contains a filename without an extension. This property can be used to build an interactive list of available reports in your application (eg: in a ListBox).

Description – description of the report, taken from the metadata of the report.

Dictionary<string, string> Parameters – dictionary of report parameters, may be filling parameters, which will be subsequently transferred to the report. It supports only the string values that must be considered when designing a report template.

GearItem

```
public class GearItem
{
 public string Name;
 public Dictionary<string, string> Properties;
}
```

Name – name of the format : may contain one of the following strings:

Name	Description
PDF	Adobe Acrobat file
DOCX	Microsoft Word 2007 file
XLSX	Microsoft Excel 2007 file
PPTX	Microsoft PowerPoint 2007 file
RTF	Rich Text file – supported by many text editors
ODS	Open Office Spreadsheet file
ODT	Open Office Text file
MHT	Compressed HTML file together with the images, can be opened in Internet Explorer
CSV	Comma separated values file
DBF	dBase file
XML	Excel XML table – without images
TXT	Text file
FPX	FastReport.Net Prepared report file

Dictionary<string, string> Properties – dictionary of parameters of a report. A complete list of supported parameters with default values is available on requesting the server to list formats.

When creating a service you must add the following lines in your App.config or Web.config:

```
<appSettings>
<add key="FastReport.ReportsPath" value="C:\Program files\FastReports\FastReport.Net\Demos\WCF" />
<add key="FastReport.ConnectionStringName" value="FastReportDemo" />
<add key="FastReport.Gear" value="PDF,DOCX,XLSX,PPTX,RTF,ODS,ODT,MHT,CSV,DBF,XML,TXT,FPX" />
</appSettings>
```

FastReport.ReportsPath – specifies the path to the folder with the reports, a list of which will be transmitted to the client.

FastReport.ConnectionStringName – name of the connection string to the database, which is stored in the configuration section . Used to replace the internal connection string in the report template.

FastReport.Gear – list of available formats. You can select only those required and change the order of the names.

Schematic of using FastReport.Service:
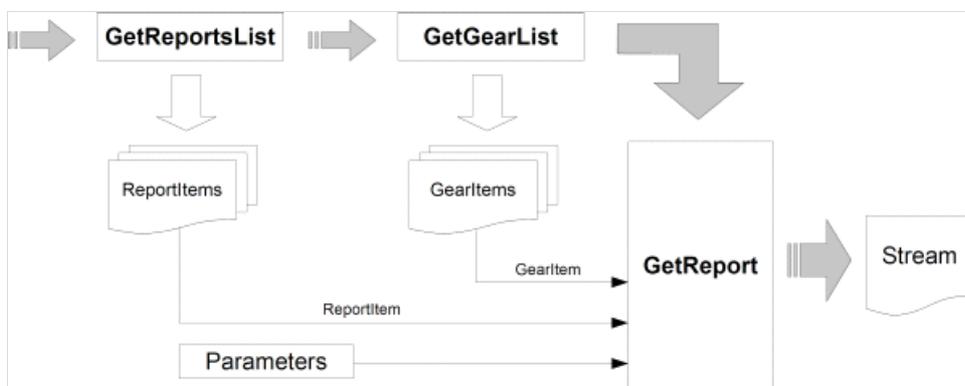
And, if you already know exactly what to report and in which format to receive it (this reduces the number of queries made to the service):

Important points to note when you create report templates for use in the services:

- dialogs in the reports are not supported and will be ignored;

- each report must include an internal DataConnection, whose connection string for the report service is replaced by a string from the configuration.

Examples of use of FastReport.Service.dll can be found in the folders \Demos\C#\WCFWebService , \Demos\C#\WCFWindowsService , \Demos\C#\WCFWebClient , \Demos\C#\WCFClient.

An example configuration file service - FastReport.Service.dll.config.

Simple example of WCF service

This example does not require programming and is intended for testing the library and the configuration file. To complete the task, we will use the program WcfSvcHost.exe, that comes with Visual Studio:

1. Create a folder for our project anywhere on the disk, eg: as C:\WCF\FastReport

2. Copy these files to the folder : FastReport.Service.dll, FastReport.Service.dll.config, FastReport.dll and FastReport.Bars.dll

3. Create two sub-folders \Data and \Reports

4. Copy the database file to the \Data folder from the Demos folder \FastReport.Net\Demos\Reports\nwind.xml

5. Copy the contents of folder \FastReports\FastReport.Net\Demos\WCF to \Reports – it contains test reports with built-in connections to the database, which are essential when used with library FastReport.Service.dll

6. Open the configuration file FastReport.Service.dll.config in any text editor

7. Change the path to the reports in section

```
<add key="FastReport.ReportsPath" value="C:\WCF\FastReport\Reports" />
```

8. Change the connection string in section :

```
<add name="FastReportDemo" connectionString="XsdFile=;XmlFile=C:\WCF\FastReport\Data\nwind.xml"/>
```

9. Create batch file service.bat containing the line:

"C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\WcfSvcHost.exe" /service:C:\WCF\FastReport\FastReport.Service.dll /config:C:\WCF\FastReport\FastReport.Service.dll.config

10. Run service.bat from Explorer with administrator rights ('Run as administrator'). You will see an icon for WCF Service Host in the system tray. Double-click on the icon:
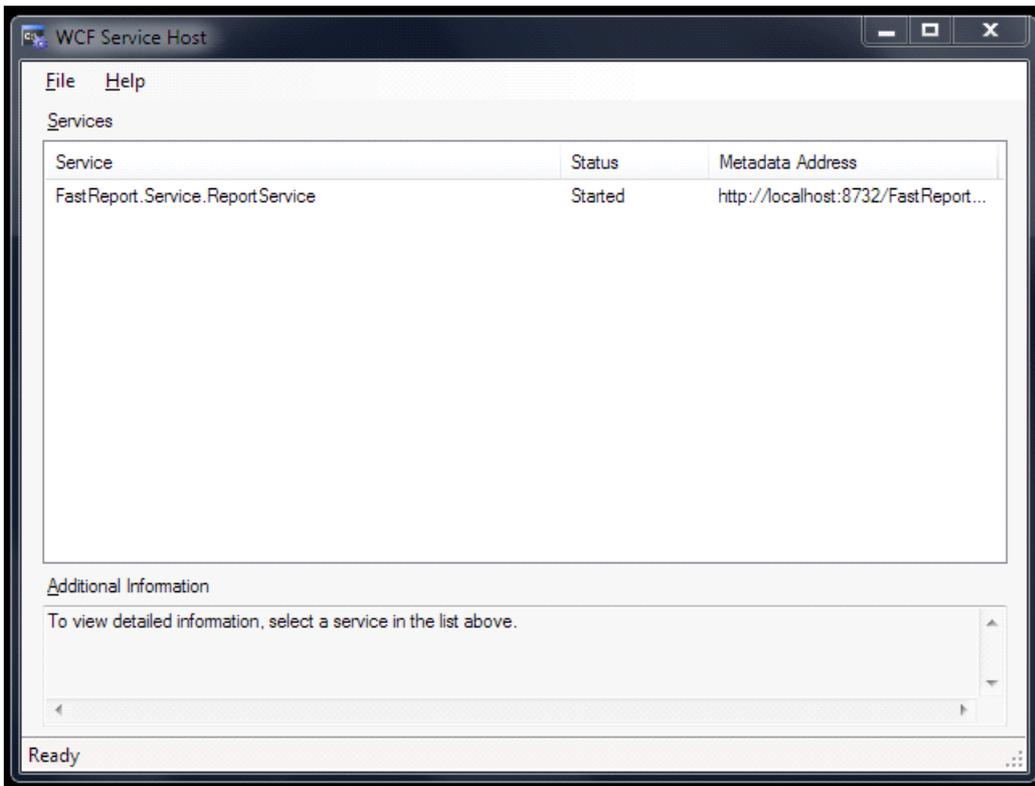
11. Open a web browser and go to address http://localhost:8732/FastReportService/
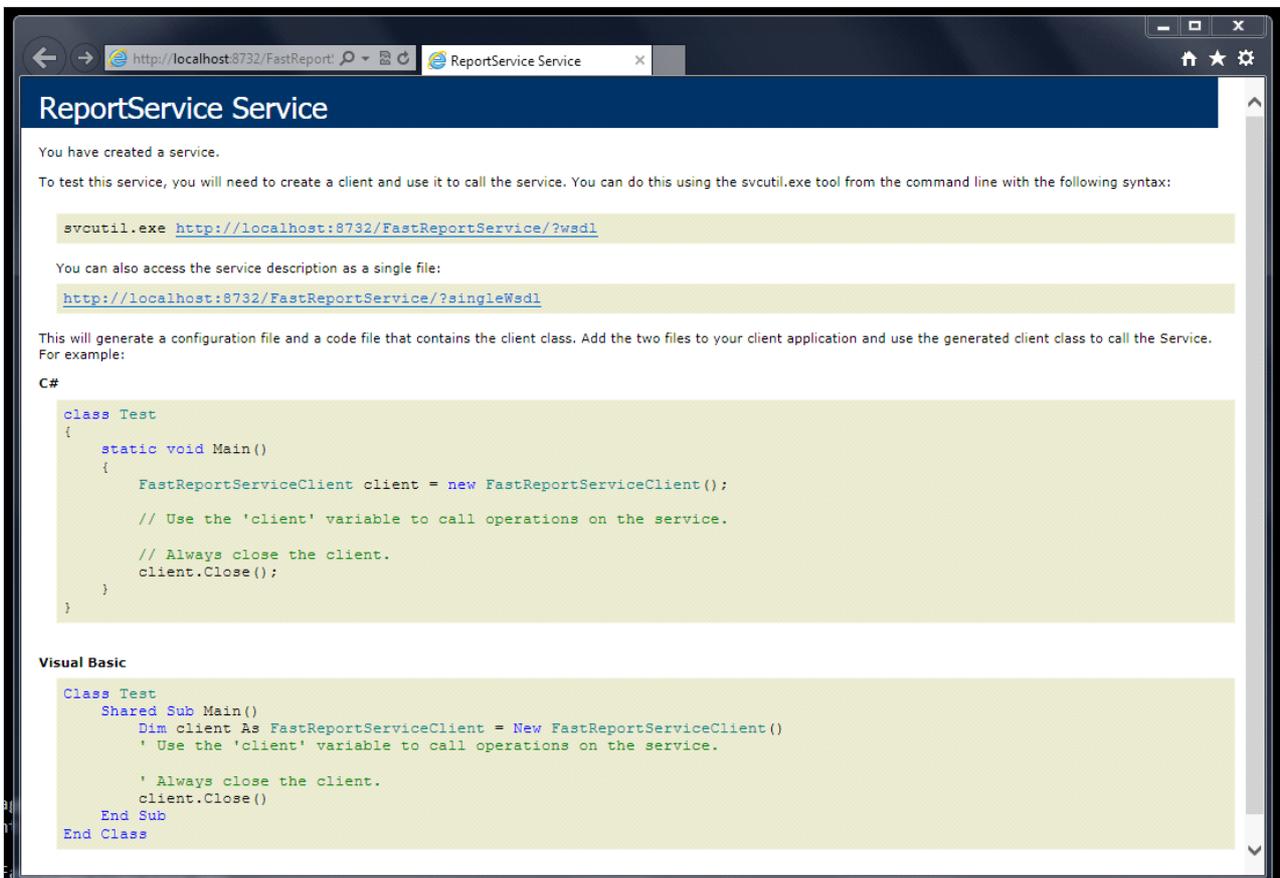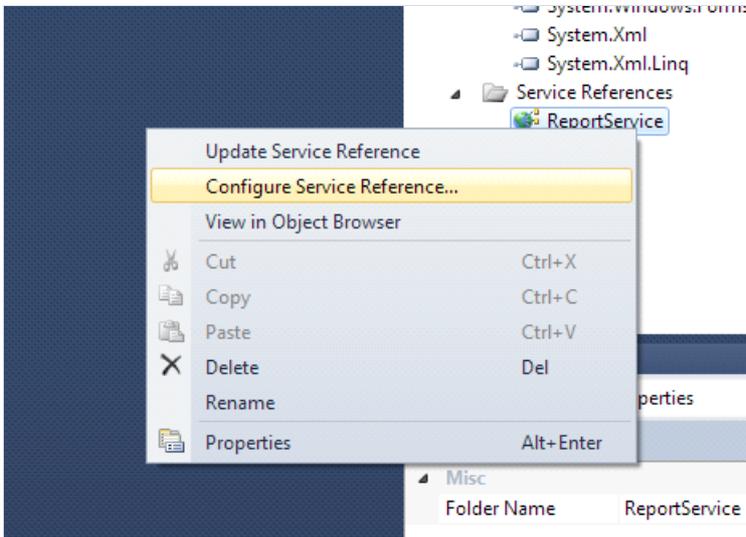

This shows the Service working normally. You can change the port number of the service in the configuration file:

```
<add baseAddress="http://localhost:8732/FastReportService/" />
```
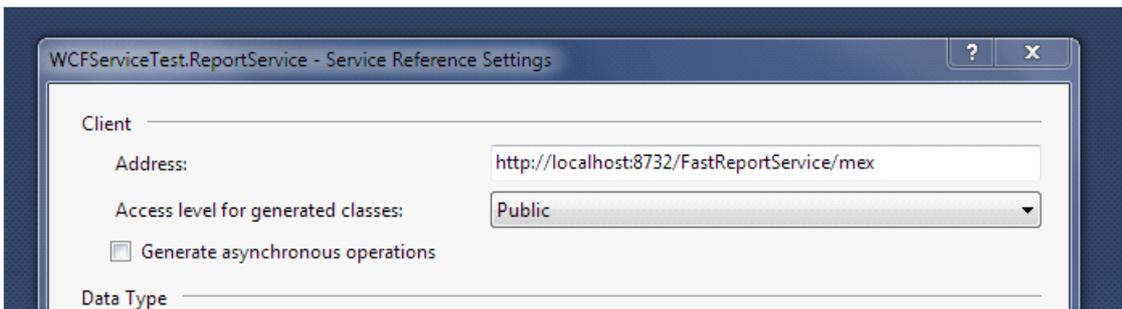
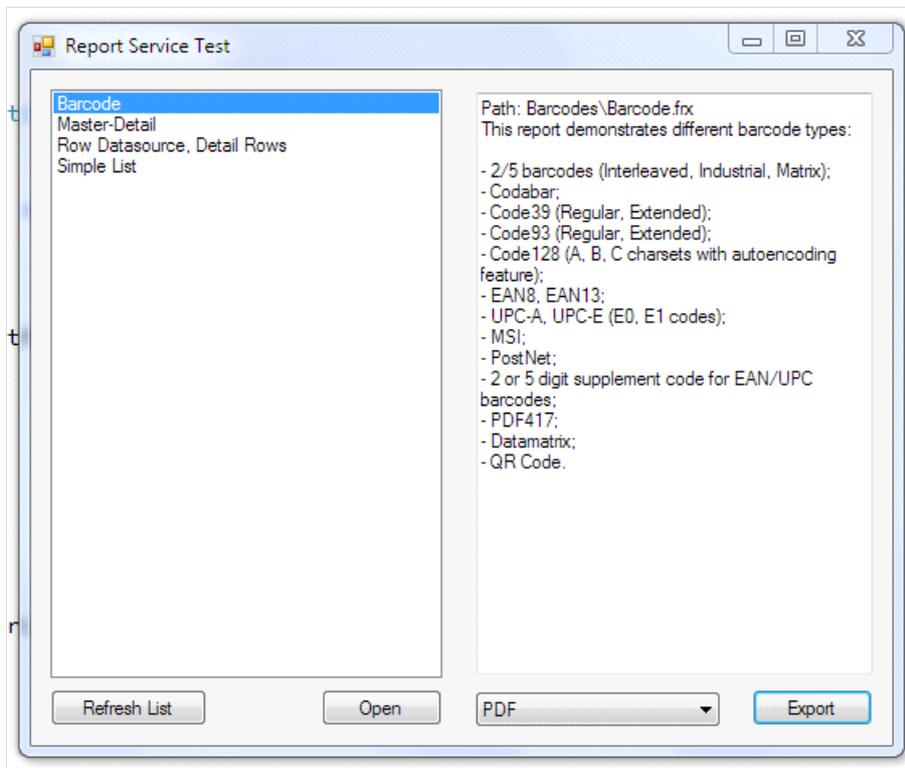Let's connect to our service from the demo example \FastReport.Net\Demos\C#\WCFClient

1. Open WCFServiceClient.csproj in Visual Studio

2. Right-click in Solution Explorer on "Service References:ReportService" and select "Configure Service Reference" in the popup

3. Review the service address, which should end with "/mex" (metadata exchange)

4. Compile and run an example.

Report Service Test

Barcode
Master-Detail
Row Datasource, Detail Rows
Simple List

Path: Barcodes\Barcode.frx
This report demonstrates different barcode types:

- 2/5 barcodes (Interleaved, Industrial, Matrix);
- Codabar;
- Code39 (Regular, Extended);
- Code93 (Regular, Extended);
- Code128 (A, B, C charsets with autoencoding feature);
- EAN8, EAN13;
- UPC-A, UPC-E (E0, E1 codes);
- MSI;
- PostNet;
- 2 or 5 digit supplement code for EAN/UPC barcodes;
- PDF417;
- Datamatrix;
- QR Code.

[Refresh List]   [Open]   PDF ▼   [Export]

Barcode
Master-Detail
Row Datasource, Detail Rows
Simple List

Path: Barcodes\Barcode.frx
This report demonstrates different barcode types:

Creating Web Service using FastReport.Service.dll

There is an easy way to implement a web service using the library FastReport.Service.dll (WCF Service Library), which is supplied with FastReport .Net.

Our example is based on creating a simple web application with web service functions, but you can modify your existing project based on .NET Framework 4.0 or later.

Run Visual Studio and create a new ASP.NET Web Application project under .NET Framework 4.0.
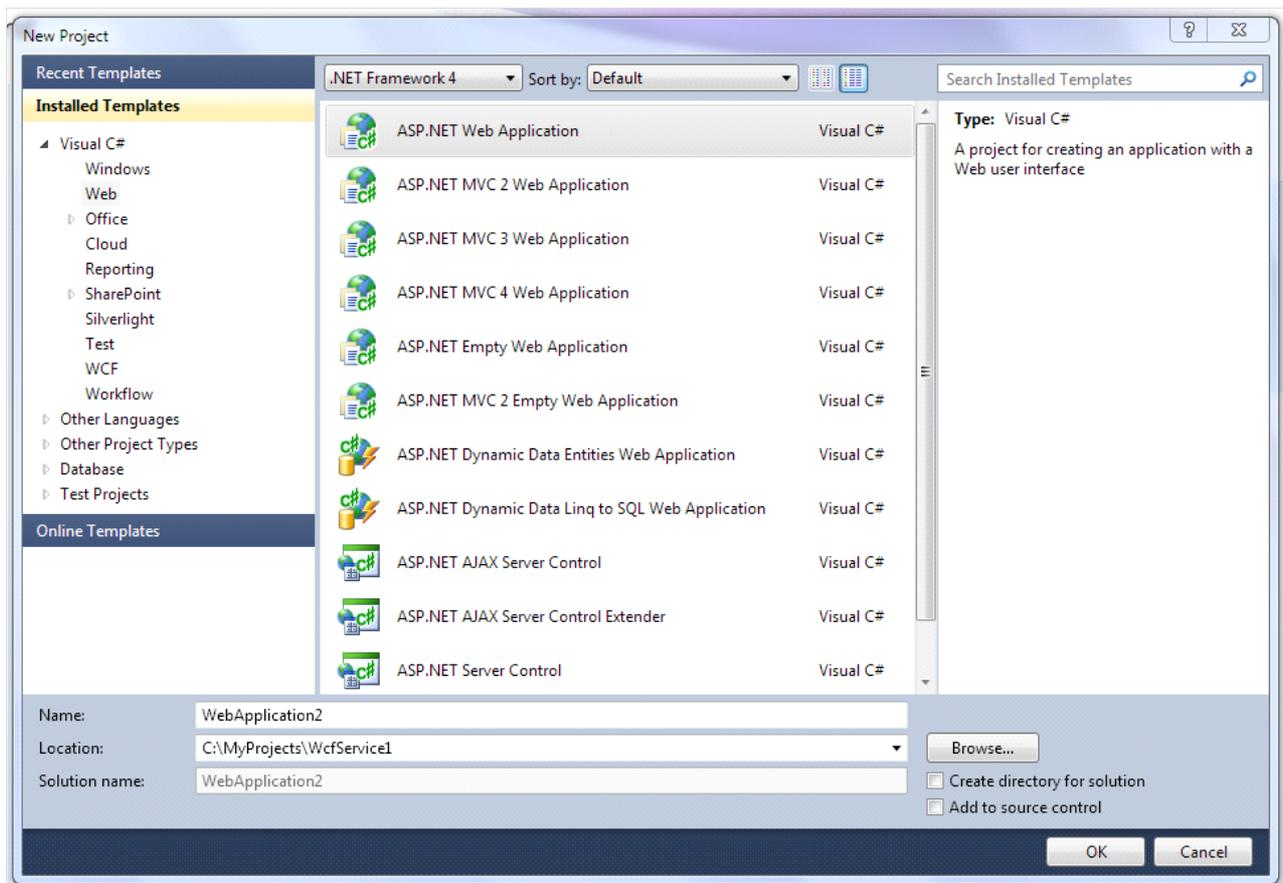
Add references to libraries FastReport.dll, FastReport.Bars.dll, FastReport.Service.dll

Create a new text file with name ReportService.svc in the site root.
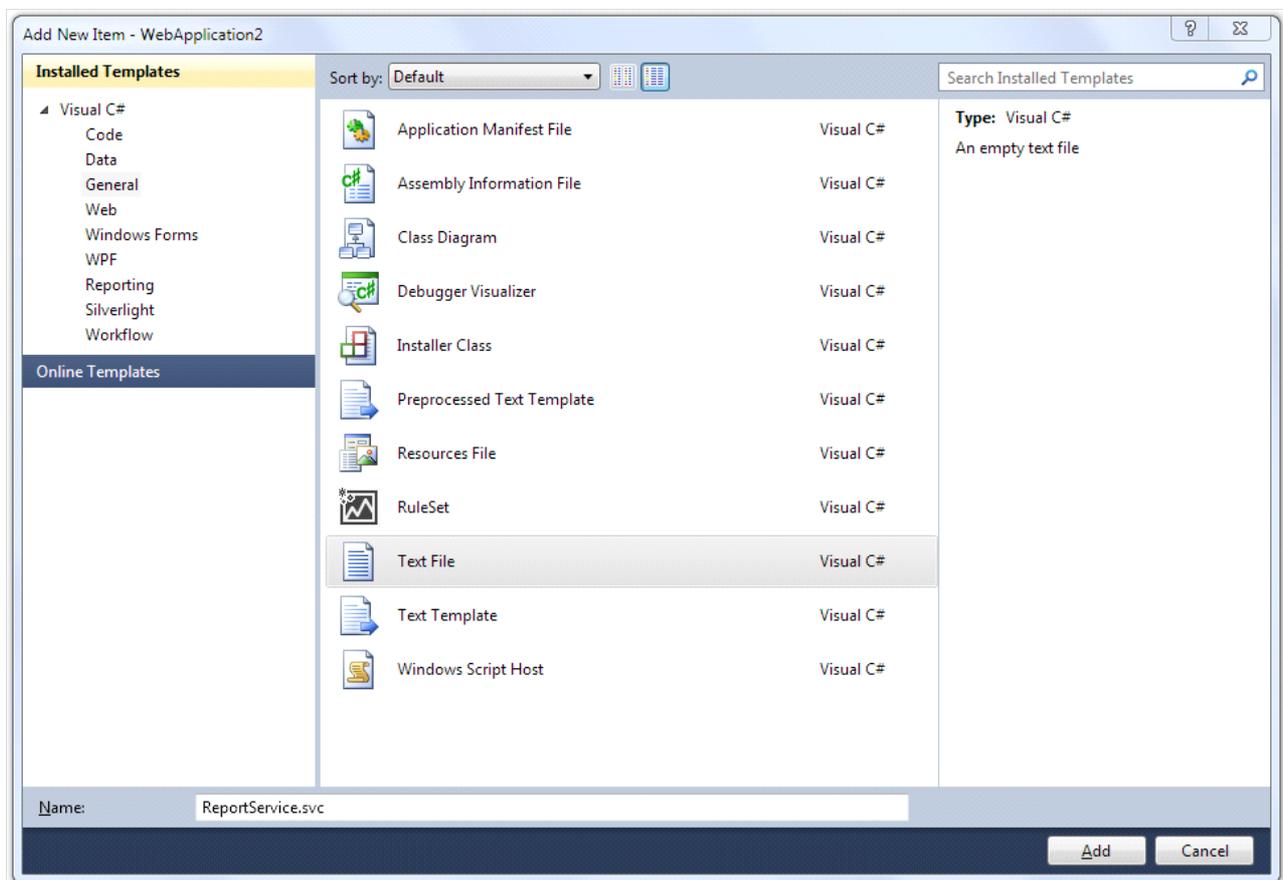
Add these lines to the file:

```
<%@ ServiceHost Service="FastReport.Service.ReportService" %>
<%@ Assembly Name="FastReport.Service" %>
```

Open web.config and add this code in section:

```xml
<appSettings>
<!-- path to folder with reports -->
<add key="FastReport.ReportsPath" value="C:\Program files\FastReports\FastReport.Net\Demos\WCF" />
<!-- name of connection string for reports -->
<add key="FastReport.ConnectionStringName" value="FastReportDemo" />
<!-- Comma-separated list of available formats PDF,DOCX,XLSX,PPTX,RTF,ODS,ODT,MHT,CSV,DBF,XML,TXT,FPX.
You can delete any or change order in this list. -->
<add key="FastReport.Gear" value="PDF,DOCX,XLSX,PPTX,RTF,ODS,ODT,MHT,CSV,DBF,XML,TXT,FPX" />
</appSettings>
<connectionStrings>
<add name="FastReportDemo" connectionString="XsdFile=;XmlFile=C:\Program
Files\FastReports\FastReport.Net\Demos\Reports\nwind.xml"/>
</connectionStrings>
<system.serviceModel>
<services>
<service behaviorConfiguration="FastReportServiceBehavior" name="FastReport.Service.ReportService">
<endpoint address="" binding="wsHttpBinding" contract="FastReport.Service.IFastReportService">
<identity>
<dns value="localhost" />
</identity>
</endpoint>
<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
</service>
</services>
<behaviors>
<serviceBehaviors>
<behavior name="FastReportServiceBehavior">
<serviceMetadata httpGetEnabled="True" />
<serviceDebug includeExceptionDetailInFaults="True" />
</behavior>
</serviceBehaviors>
</behaviors>
<bindings>
<basicHttpBinding>
<binding messageEncoding="Mtom"
closeTimeout="00:02:00" openTimeout="00:02:00"
receiveTimeout="00:10:00" sendTimeout="00:02:00"
maxReceivedMessageSize="67108864" maxBufferSize="65536"
transferMode="Streamed">
<security mode="None">
<transport clientCredentialType="None" />
</security>
</binding>
</basicHttpBinding>
</bindings>
</system.serviceModel>
```

The key "FastReport.ReportsPath" should contain a path to the folder with the reports. You can set it to the demo folder «\FastReport.Net\Demos\WCF», for example.

The key "FastReport.ConnectionStringName" should contain the connection string name. This line should be registered in section .

Let's run our site and check the availability of a Web service by accessing the file ReportService.svc.
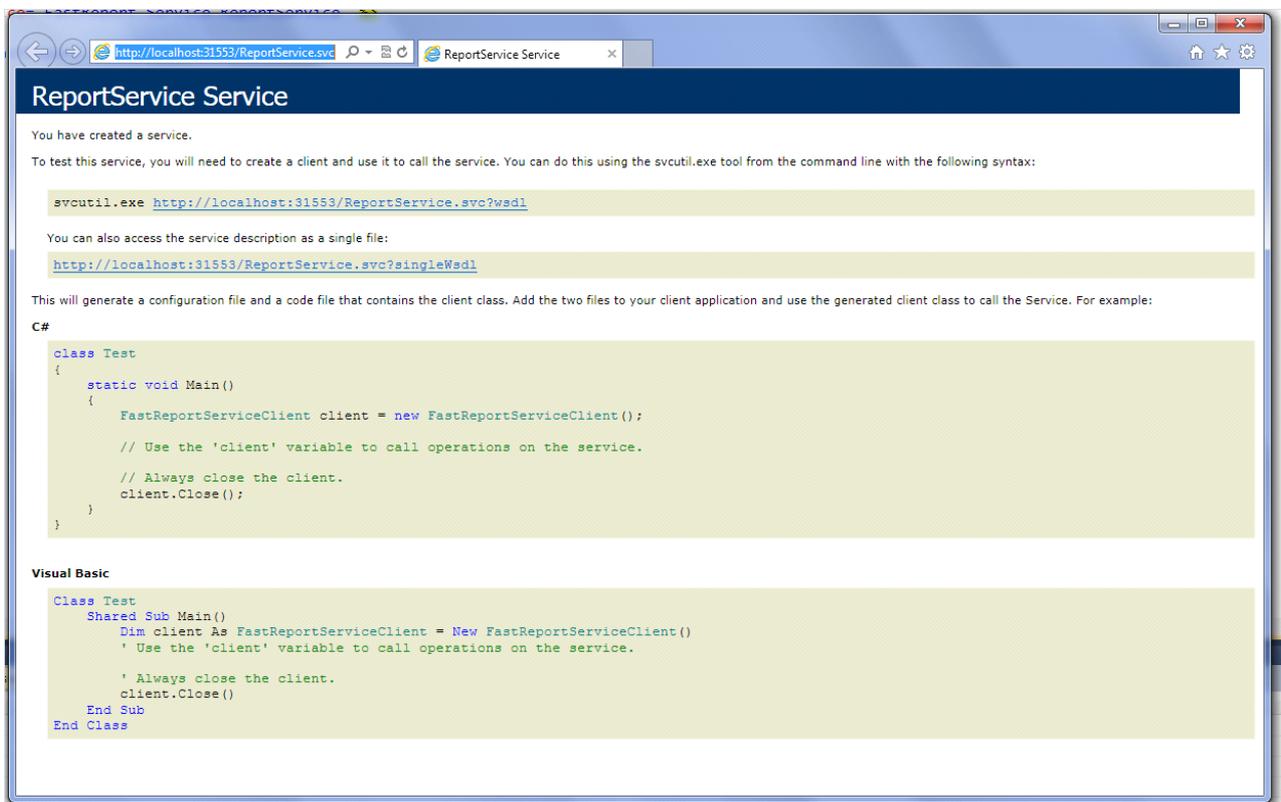
When you deploy the project on the server, be sure to check that files FastReport.dll, FastReport.Bars.dll, FastReport.Service.dll are in the folder \bin.
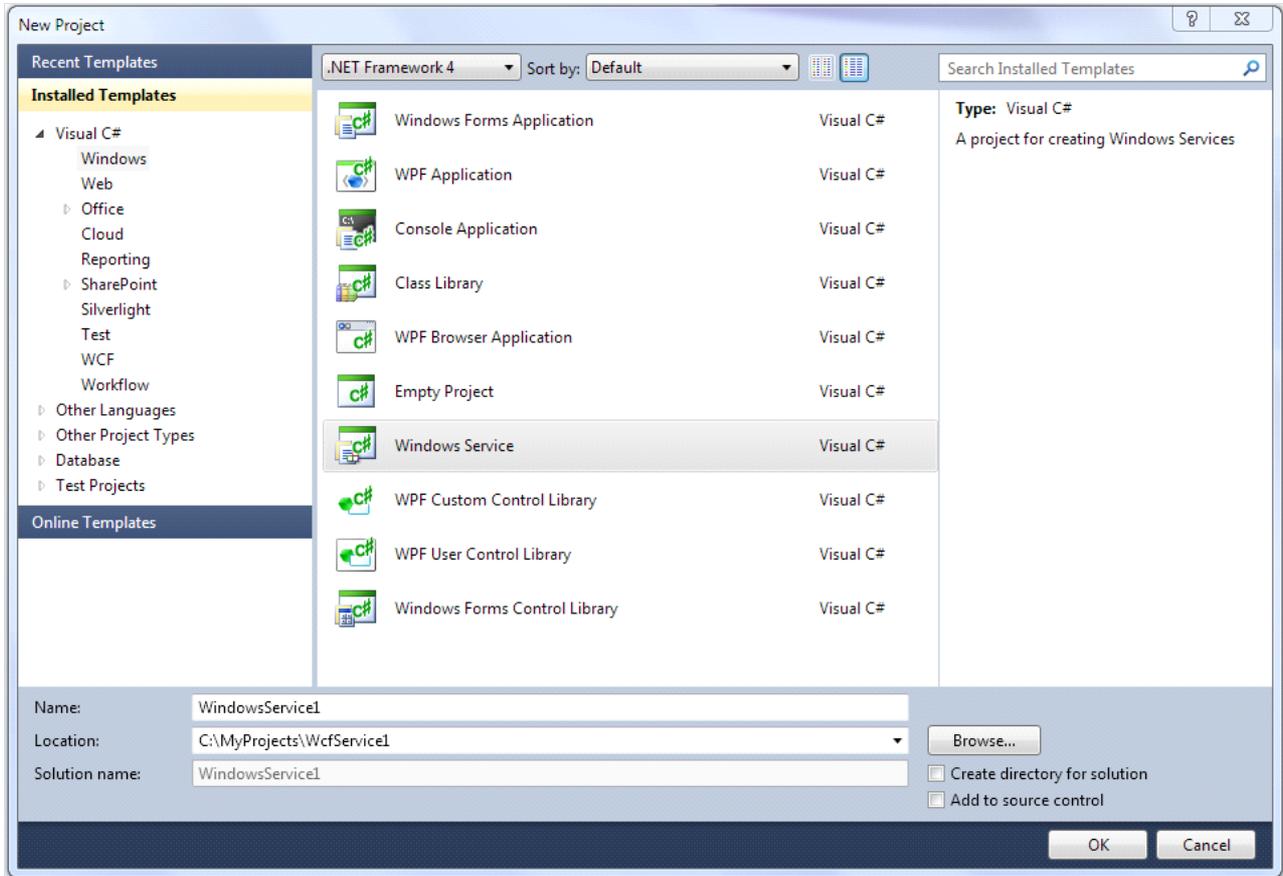
Examples of client programs can be found in the folders \FastReport.Net\Demos\C#\WCFClient and \FastReport.Net\Demos\C#\WCFWebClient. Open each project in Visual Studio, right-click on ReportService and select Configure Service Reference in the popup.
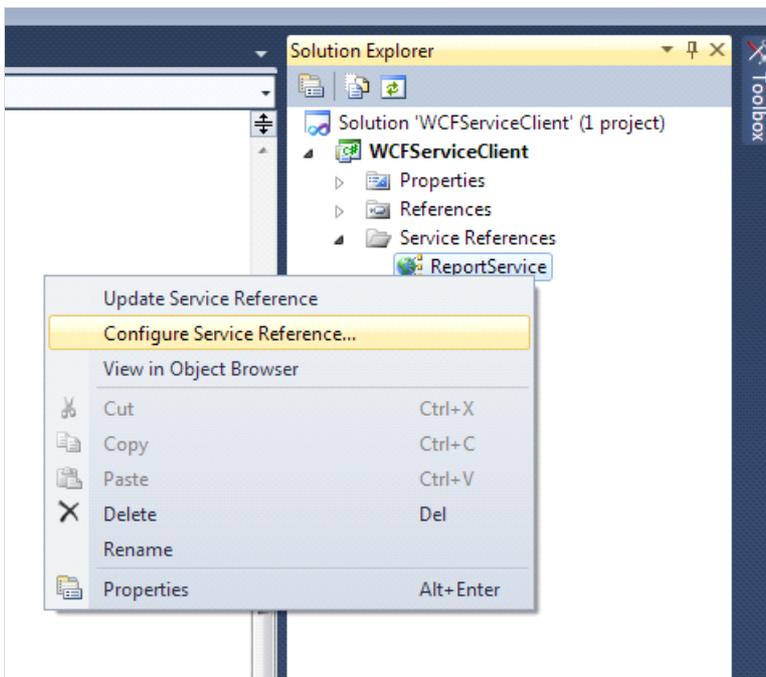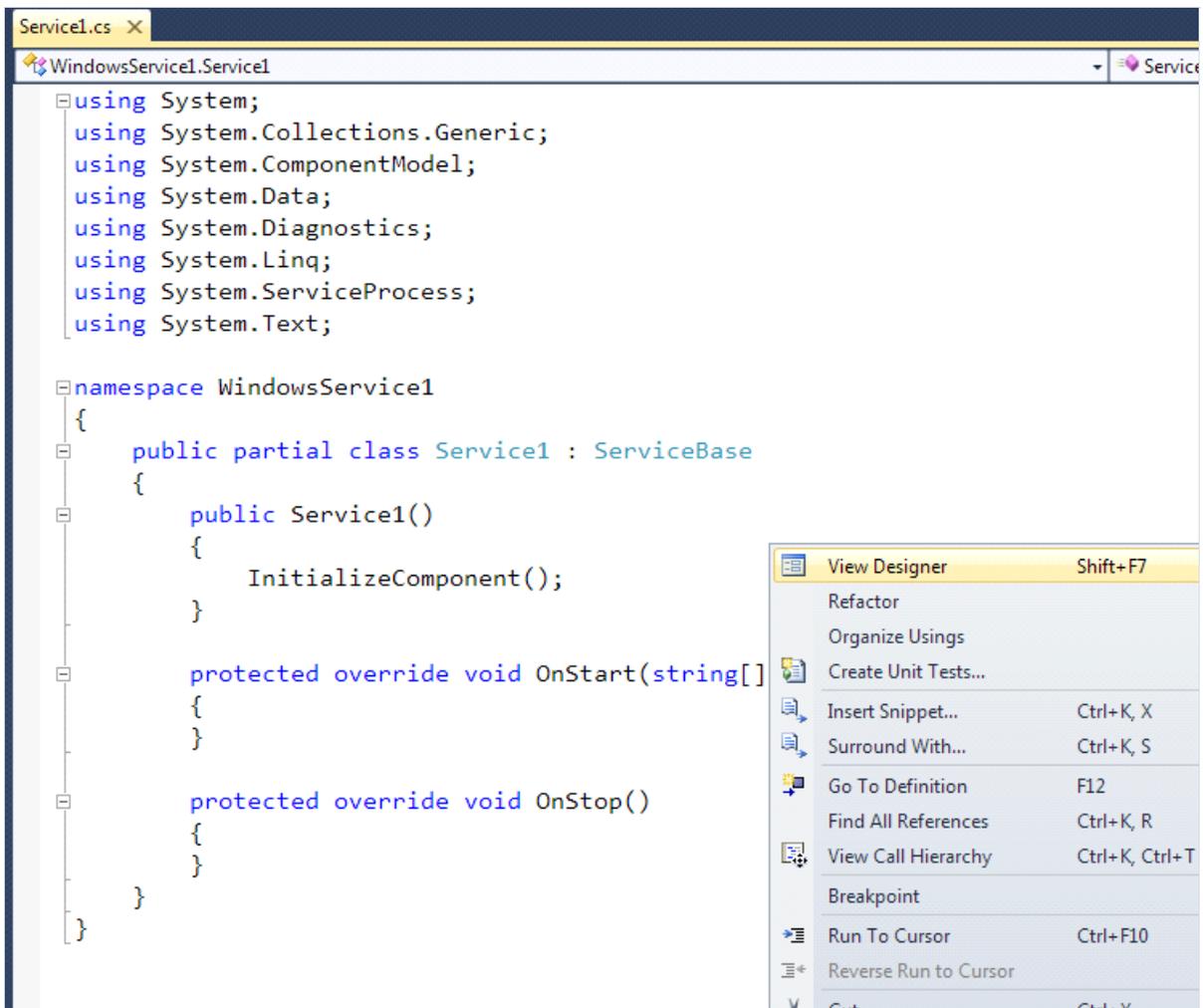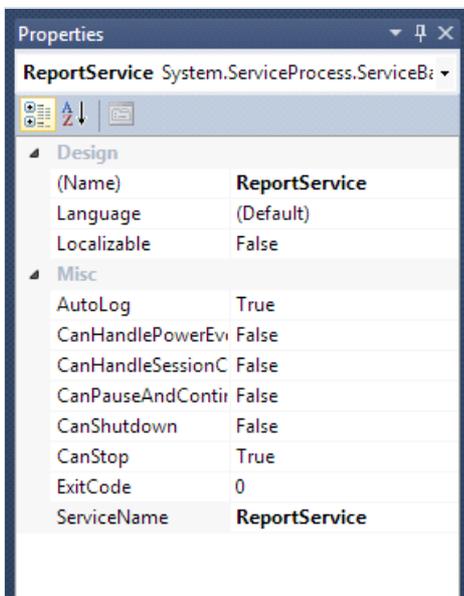

Specify the address of an existing web service in the configuration window.

Creating the WCF service hosted in windows service

Open Visual Studio and create a project WindowsService.


Open the designer of Service1.cs

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Linq;
using System.ServiceProcess;
using System.Text;

namespace WindowsService1
{
    public partial class Service1 : ServiceBase
    {
        public Service1()
        {
            InitializeComponent();
        }

        protected override void OnStart(string[]
        {
        }

        protected override void OnStop()
        {
        }
    }
}
```

	View Designer	Shift+F7
	Refactor	
	Organize Usings	
	Create Unit Tests...	
	Insert Snippet...	Ctrl+K, X
	Surround With...	Ctrl+K, S
	Go To Definition	F12
	Find All References	Ctrl+K, R
	View Call Hierarchy	Ctrl+K, Ctrl+T
	Breakpoint	
	Run To Cursor	Ctrl+F10
	Reverse Run to Cursor	

Change the name of the service to your own choice:

Right-click on window and select "Add Installer" in the popup:

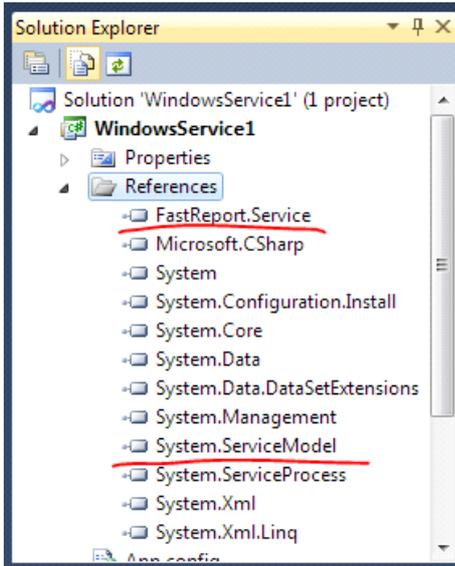To add components to your class, drag them from the Toolbox and use t[...]rope[...]
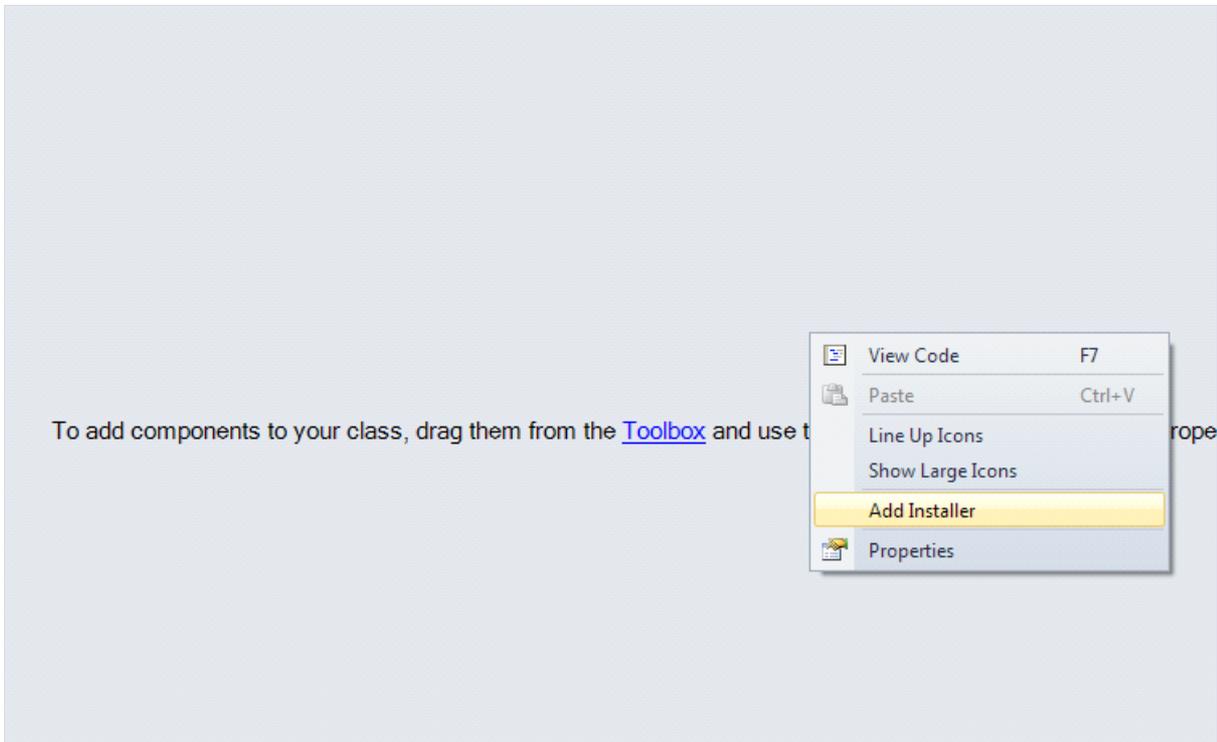
Edit the properties of the component serviceInstaller1 - set up a DisplayName.
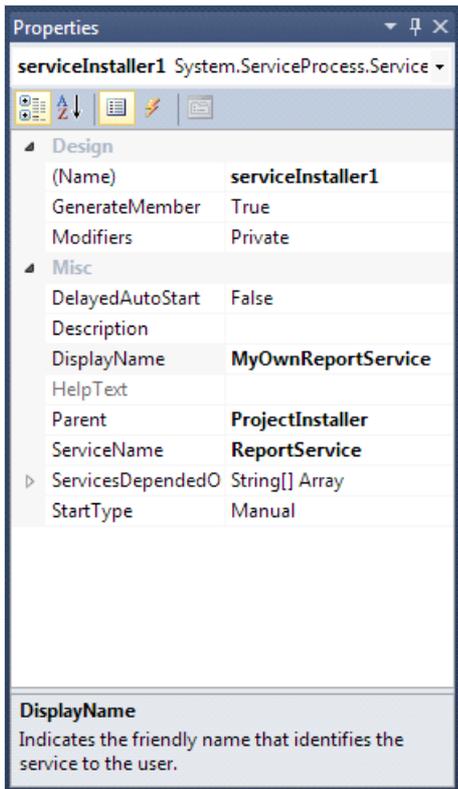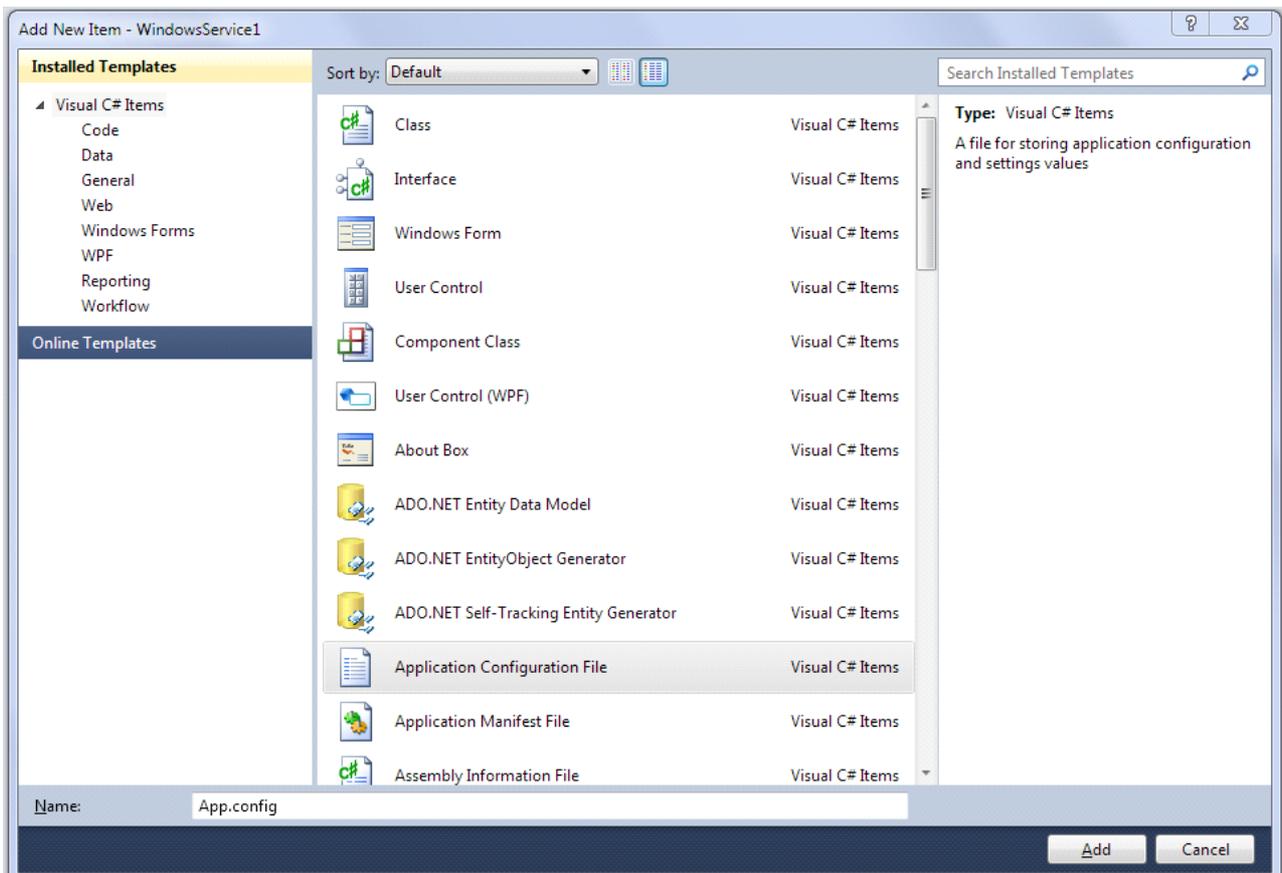

In the component properties of serviceProcessInstaller1 set the type of account for the service as LocalSystem.
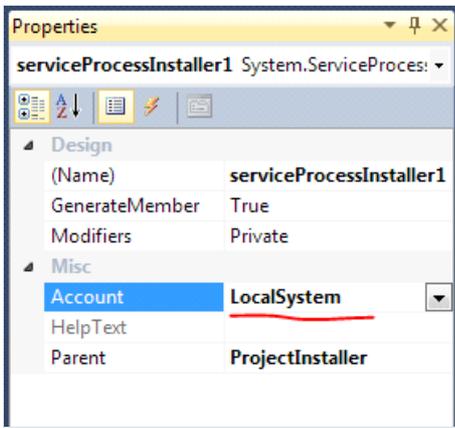
Add references in the project to System.ServiceModel and FastReport.Service.dll :

Create an application configuration file:

Copy the following text into the new app.config file:

```xml
<?xml version="1.0"?>
<configuration>
 <appSettings>
 <!-- path to folder with reports -->
 <add key="FastReport.ReportsPath" value="C:\Program files\FastReports\FastReport.Net\Demos\WCF" />
 <!-- name of connection string for reports -->
 <add key="FastReport.ConnectionStringName" value="FastReportDemo" />
 <!-- Comma-separated list of available formats PDF,DOCX,XLSX,PPTX,RTF,ODS,ODT,MHT,CSV,DBF,XML,TXT,FPX.
 You can delete any or change order in this list. -->
 <add key="FastReport.Gear" value="PDF,DOCX,XLSX,PPTX,RTF,ODS,ODT,MHT,CSV,DBF,XML,TXT,FPX" />
 </appSettings>
 <connectionStrings>
 <add name="FastReportDemo" connectionString="XsdFile=;XmlFile=C:\Program
Files\FastReports\FastReport.Net\Demos\Reports\nwind.xml"/>
 </connectionStrings>
 <system.web>
 <compilation debug="true" />
 <membership defaultProvider="ClientAuthenticationMembershipProvider">
 <providers>
 <add name="ClientAuthenticationMembershipProvider"
type="System.Web.ClientServices.Providers.ClientFormsAuthenticationMembershipProvider,
System.Web.Extensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" serviceUri=""
/>
 </providers>
 </membership>
 <roleManager defaultProvider="ClientRoleProvider" enabled="true">
 <providers>
 <add name="ClientRoleProvider" type="System.Web.ClientServices.Providers.ClientRoleProvider,
System.Web.Extensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" serviceUri=""
cacheTimeout="86400" />
 </providers>
 </roleManager>
 </system.web>
 <!-- When deploying the service library project, the content of the config file must be added to the
host's
 app.config file. System.Configuration does not support config files for libraries. -->
 <system.serviceModel>
 <services>
 <service behaviorConfiguration="FastReportServiceBehavior" name="FastReport.Service.ReportService">
 <endpoint address="" binding="wsHttpBinding" contract="FastReport.Service.IFastReportService">
 <identity>
 <dns value="localhost" />
 </identity>
 </endpoint>
 <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
 <host>
 <baseAddresses>
 <add baseAddress="http://localhost:8732/FastReportService/" />
 </baseAddresses>
 </host>
 </service>
 </services>
 <behaviors>
 <serviceBehaviors>
 <behavior name="FastReportServiceBehavior">
 <serviceMetadata httpGetEnabled="True" />
 <serviceDebug includeExceptionDetailInFaults="True" />
 </behavior>
 </serviceBehaviors>
 </behaviors>
 <bindings>
 <basicHttpBinding>
 <binding messageEncoding="Mtom"
closeTimeout="00:02:00" openTimeout="00:02:00"
 receiveTimeout="00:10:00" sendTimeout="00:02:00"
```

```
    maxReceivedMessageSize="67108864" maxBufferSize="65536"
    transferMode="Streamed">
    <security mode="None">
    <transport clientCredentialType="None" />
    </security>
    </binding>
    </basicHttpBinding>
    </bindings>
    </system.serviceModel>
    <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0" />
    </startup>
</configuration>
```

Go to the editor of Service1.cs and add the line:

```
    using System.ServiceModel;
```

Modify the class of service so it looks like:

```
    public partial class ReportService : ServiceBase
    {
     ServiceHost reportHost;

     public ReportService()
     {
     InitializeComponent();
     }

     protected override void OnStart(string[] args)
     {
     if (reportHost != null)
     reportHost.Close();
     reportHost = new ServiceHost(typeof(FastReport.Service.ReportService));
     reportHost.Open();
     }

     protected override void OnStop()
     {
     reportHost.Close();
     reportHost = null;
     }
    }
```

You can install the service using the command line utility InstallUtil.exe, which comes with .NET Framework, for instance:

C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe
"C:\MyProjects\WcfService1\WindowsService1\bin\Debug\WindowsService1.exe"

And you can start the service with the command:

net start ReportService

Open a web browser and check the address http://localhost:8732/FastReportService/, which was set in app.config in baseAddress. You can change the folder and port to your own choice.

Commands to stop and to uninstall the service:

net stop ReportService

C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe /u
"C:\MyProjects\WcfService1\WindowsService1\bin\Debug\WindowsService1.exe"

This example is located in folder "\Demos\C#\WCFWindowsService".

Working with Blazor

Presenting `FastReport.Web` package with components for embedding a web application into your Blazor Server. Starting from version 2021.3, this library contains Razor components for the Blazor Server (server-side) application, which means that all operations will be performed on the server side, which will then pass the finished display to the client.

The minimum target framework at the moment is .Net Core 3.1 to ensure the highest possible compatibility with the latest LTS (long-term support) version. Also, most users have this version and it is compatible with the latest .NET 5 framework (within this package) and newer.

Preflight Preparation

To use Blazor components in `FastReport.Web` , you need to add a reference in your project file (csproj)
`PackageReference` specifying the id of this package and the `FastReport.Core` package (versions may differ):

```
<ItemGroup>
  <PackageReference Include="FastReport.Core" Version="2021.3.0-demo"/>
  <PackageReference Include="FastReport.Web" Version="2021.3.0-demo"/>
</ItemGroup>
```

Then, to simplify naming, we recommend adding the following namespaces to your project's imports (
`_Imports.razor` file):

```
@using FastReport.Web
@using FastReport.Web.Blazor.Components
@using FastReport.Web.Blazor.Components.Internal
```

In fact, just adding `FastReport.Web.Blazor.Components` may be enough, however, for some cases, you may need
other namespaces as well.

Also, some components are likely to move within these namespaces during the beta version.

In the configurator of your web application, you need to call the `UseFastReport` method with an optional lambda
expression for setting `FastReportOptions` .

Also, for some built-in common styles and SVG images of icons in Toolbar and Tab to work, you need to use the
`UseStaticFiles` call (if you are not going to use Toolbar and Tabs, the `UseStaticFiles` call to use this package is
optional):

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // ...
    app.UseStaticFiles();
    // ...
    app.UseFastReport();
    // ...
}
```

Description of built-in components

The following is a description of the components included in `FastReport.Web` . We do not recommend using components other than `WebReportContainer` , because they can be unstable outside of the standard component tree.

However, to fine-tune the rendering, you may need to use other components.

WebReportContainer

The main and most versatile Blazor component that performs `WebReport` rendering is `<WebReportContainer/>` . It is located in the namespace `FastReport.Web.Blazor.Components` .

The only parameter it takes is an object of the `WebReport` class. This means that to use this component, you must create an object of the `WebReport` class, assign it a `Report` , other necessary parameters, and pass this object to the `WebReportContainer` parameters.

Example

```
<WebReportContainer WebReport="@UserWebReport" />

@code {
    public WebReport UserWebReport { get; set; }

    protected override void OnParametersSet()
    {
        var report = Report.FromFile(
        Path.Combine(
            directory,
            "My report.frx"));

        // Registers the application dataset
        Report.RegisterData(DataSet, "NorthWind");

        UserWebReport = new WebReport();
        UserWebReport.Report = Report;
    }
}
```

This component can define a different Mode (Designer, Dialog, and normal Preview) and can prepare a report, embed default styles and individual styles, display Toolbar, Outline and Tabs, work with interactive reports, etc.

WebReportPreview

It is similar to the previous component but does not take into account the Designer Mode. That is, it always tries to prepare and render a report.

ReportContainer

It is similar to the previous component but does not include loading `WebReport` styles (common and individual for Toolbar/Tabs/Outline).

It is engaged in the preparation of the report and subsequent display together with the Toolbar and Tabs (if necessary).

When working with any interactivity (clicks on the report / working with dialog forms), it is this component that is updated.

ReportBody / ExportComponent

The `ReportBody` calls the Outline rendering (if necessary) and "nests" a component that is the rendering of the report itself (`ExportComponent`), which the `ReportContainer` passes to it. *Not recommended for use.*

BlazorExport

The "lowest" level of a component is not a component at all, but `BlazorExport` itself - a tool for exporting a prepared report to the RenderTreeBuilder build format. Located in `FastReport.Web.Blazor.Export` namespace.

To build this export, you must:

1. Prepare the report;
2. Make sure that this report does not use dialog forms (they are rendered using the `DialogPageComponent` and are not covered in this tutorial);
3. Create your own component and explicitly define the construction method in it (call the override of the `BuildRenderTree` method);
4. In this build method, create a `BlazorExport` instance, set the properties it needs, and call Export passing the following parameters: a Report and a builder instance that is an argument to this overridden method.

```
/// Main function
protected override void BuildRenderTree(RenderTreeBuilder builder)
{
    using (BlazorExport blazor = new BlazorExport())
    {
        blazor.StylePrefix = $"fr{WebReport.ID}";
        blazor.EmbedPictures = true;
        blazor.OnClick += ProcessClick;
        blazor.EnableMargins = WebReport.EnableMargins;
        blazor.SinglePage = true;
        blazor.CurPage = WebReport.CurrentPageIndex;

        blazor.Export(myReport, builder);
    }
}
```

Online Designer

At the moment, Online Designer can work in the iframe element using javascript and it is fully compatible with the Online Designer assembly for Core.

To use only the designer's capabilities, you can call the `<IFrameDesigner/>` component passing it the `WebReport` parameter with the configured Report property and the optional `DesignerLocale` and `DesignerPath` :

```
<IFrameDesigner WebReport="CurrentWebReport" />
```

However, we remind you that the `WebReportContainer` component understands which Mode it is currently working with and it is not at all necessary to call `IFrameDesigner` in this form.

Setting up common styles and SVG

Unlike `FastReport.Web` for Core, SVG images for Toolbar and Tabs, as well as some general display styles of Tabs, Outline, etc. have been moved to `staticWebAssets` for possible customization in your web application (changing colors, sizes, replacing images).

These resources are located in your local storage. At the time of development/assembly of your application, they are located at: `{UserName}/.nuget/packages/fastreport.web/{version}/staticwebassets`

At the time of publishing your web application (dotnet publish), these resources are copied to the directory: `wwwroot/_content/FastReport.Web`

Demo project

You can find a project for demonstrating work with the `FastReport.Web` package on our [GitHub](). An example of using the `WebReportContainer` is in the custom component under `Pages/Index.razor` and `Pages/Index.razor.cs` .

Extending FastReport functionality

Create your own connection types

Creating custom connection types

Data connections are used to add a data source to a report. This allows you to connect to data directly from the report, rather than using data provided by the application.

To create your own connection you need to do the following:

- create a connection class - a descendant of `FastReport.Data.DataConnectionBase` and implement some of its methods;
- create a connection editor - an heir of `FastReport.Data.ConnectionEditors.ConnectionEditorBase` and implement the user interface and methods `GetConnectionString` , `SetConnectionString` ;
- register the connection in FastReport.

These steps will be described below.

DataConnectionBase class

To create your own connection, use the FastReport.Data.DataConnectionBase class. This class has the following set of methods that you must override when creating your own connection:

```
public abstract class DataConnectionBase : DataComponentBase
{
  protected virtual string GetConnectionStringWithLoginInfo(string userName, string password)
  public abstract string QuoteIdentifier(string value, DbConnection connection);
  public virtual DbConnection GetConnection();
  public virtual DbDataAdapter GetAdapter(string selectCommand, DbConnection connection,
    CommandParameterCollection parameters);
  public virtual ConnectionEditorBase GetEditor();
  public virtual Type GetParameterType();
  public virtual string GetConnectionId();
  public virtual string[] GetTableNames();
  public virtual void TestConnection();
  public virtual void FillTableSchema(DataTable table, string selectCommand,
    CommandParameterCollection parameters);
  public virtual void FillTableData(DataTable table, string selectCommand,
    CommandParameterCollection parameters);
}
```

Not all methods need to be overridden - some of them have a default implementation, which will be sufficient for most cases. So, if your connection uses objects of type DbConnection and DbDataAdapter (and this is standard for the vast majority of connections), you need to implement the following methods:

```
public abstract string QuoteIdentifier(string value, DbConnection connection);
protected virtual string GetConnectionStringWithLoginInfo(string userName, string password)
public virtual DbConnection GetConnection();
public virtual DbDataAdapter GetAdapter(string selectCommand, DbConnection connection,
    CommandParameterCollection parameters);
public virtual ConnectionEditorBase GetEditor();
public virtual Type GetParameterType();
public virtual string GetConnectionId();
public virtual string[] GetTableNames();
```

If your connection does not work with such objects, but, for example, is a bridge between the application server and the report, you will need to implement the following methods:

```
public abstract string QuoteIdentifier(string value, DbConnection connection);
public virtual ConnectionEditorBase GetEditor();
public virtual Type GetParameterType();
public virtual string GetConnectionId();
public virtual string[] GetTableNames();
public virtual void TestConnection();
public virtual void FillTableSchema(DataTable table, string selectCommand,
    CommandParameterCollection parameters);
public virtual void FillTableData(DataTable table, string selectCommand,
    CommandParameterCollection parameters);
```

Below is a description of each method.

ConnectionString propery

Each connection type has its own set of parameters, such as database path, SQL dialect, username and password. The ConnectionString property, which has a string type, is used to store the parameters.

This property is used as follows:

- in the GetConnection method when creating an object of type DbConnection;
- In the connection editor. In the latter case, individual parameters are selected from the connection string (e.g., the path to the database file). This uses a class of type `DbConnectionStringBuilder`, specific to each connection type. For example, for MS SQL it is `SqlConnectionStringBuilder`.

QuoteIdentifier method

An identifier (table or column name) is passed to this method. The method must return a quoted identifier. Quotation marks are specific to each database. For example, MS Access uses square brackets:

```
select * from [Table with long name]
```

The method in `MsAccessDataConnection` looks like this:

```
public override string QuoteIdentifier(string value, DbConnection connection)
{
  return "[" + value + "]";
}
```

You must override this method. Refer to the manual for this connection type to find out which characters are used to refer to tables with long names.

GetConnectionStringWithLoginInfo method

This method should take the existing connection string (from the `ConnectionString` property), include the username and password information, and return the modified string. This method is used if the connection settings specify "Ask for password when connecting".

For example, the implementation of this method for `MsSqlDataConnection` looks like this:

```
protected override string GetConnectionStringWithLoginInfo(string userName, string password)
{
  // get the existing connection string
  SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder(ConnectionString);

  // include the username and password
  builder.IntegratedSecurity = false;
  builder.UserID = userName;
  builder.Password = password;

  // return the modified string
  return builder.ToString();
}
```

You have to override this method.

GetConnection method

The method returns a new object of type `DbConnection` , specific to this connection. This object is used to connect to the database when the table needs to be filled with data.

The connection parameters are taken from the `ConnectionString` property. For example, for `MsSqlDataConnection` , the method looks like this:

```
public override DbConnection GetConnection()
{
  return new SqlConnection(ConnectionString);
}
```

Most of the time you have to override this method. It is used in two other methods, `FillTableSchema` and `FillTableData` . If your connection type does not use `DbConnection` and `DbDataAdapter` objects to get information about the table and load data into it, you may not override this method. In that case, you must override the `FillTableSchema` and `FillTableData` methods.

GetAdapter method

The method returns a new object of type `DbDataAdapter` , specific to this connection. This object is used to fill the table with data.

The following parameters are passed to the method:

Parameter	Description
selectCommand	SQL query text.
connection	Object of type `DbConnection` , created in the `GetConnection` method.
Parameters	Query parameters, if defined.

Let's look at an example implementation of this method in `MsSqlDataConnection` :

```
public override DbDataAdapter GetAdapter(string selectCommand, DbConnection connection,
  CommandParameterCollection parameters)
{
  SqlDataAdapter adapter = new SqlDataAdapter(selectCommand, connection as SqlConnection);
  foreach (CommandParameter p in parameters)
  {
    SqlParameter parameter = adapter.SelectCommand.Parameters.Add(p.Name, (SqlDbType)p.DataType, p.Size);
    parameter.Value = p.Value;
  }
  return adapter;
}
```

The method creates an adapter specific to MS SQL, fills the query parameters and returns the adapter. The parameters should be described in more detail. The query text may contain parameters. In this case, a collection of parameters in the parameters variable is passed to the method - these are the parameters defined in the FastReport designer when the query was created. You must add the parameters to the `adapter.SelectCommand.Parameters` list. Each parameter in the parameters collection has the following properties:

Property	Description
Name	Parameter name.
DataType	The data type of the parameter. This is an int type property; you must cast it to the type used for the parameters in this connection.
Size	The size of the parameter data.
Value	The value of the parameter.

Most of the time you have to override this method. It is used in two other methods, `FillTableSchema` and `FillTableData` . If your connection type doesn't use the `DbConnection` and `DbDataAdapter` objects to get information about the table and load data into it, you may not override this method. In that case, you must override the `FillTableSchema` and `FillTableData` methods.

GetEditor method

This method returns an instance of the editor for the given connection type. The implementation of the editor will be discussed in "Connection Editor".

An example implementation of this method in `MsSqlDataConnection` :

```
public override ConnectionEditorBase GetEditor()
{
  return new MsSqlConnectionEditor();
}
```

GetParameterType method

The value returned by this method is used in the query parameter editor to select the data type of the parameter:
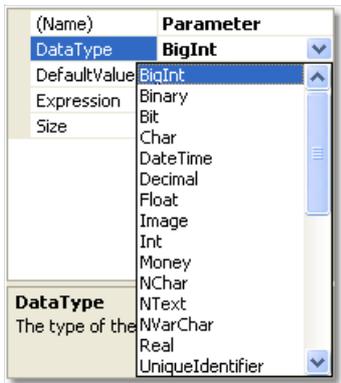
For example, for `MsSqlDataConnection` the parameter is of type `SqlDbType` :

```
public override Type GetParameterType()
{
  return typeof(SqlDbType);
}
```

You have to override this method.

GetConnectionId method

The value returned by this method is used in the "Connection Wizard" to display short information about the connection.

This value usually contains the name of the connection type and the name of the database. In `MsSqlDataConnection` the implementation of the method looks like this:

```
public override string GetConnectionId()
{
  SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder(ConnectionString);
  string info = builder.InitialCatalog;
  if (String.IsNullOrEmpty(info))
    info = builder.AttachDBFilename;
  return "MS SQL: " + info;
}
```

Note that `SqlConnectionStringBuilder` is used to parse the connection string (`ConnectionString` property).

You have to override this method.

GetTableNames method

This method returns a list of tables and views in the database. As a rule, the `GetSchema` method of the `DbConnection` object is used for this, which is returned by the `GetConnection` method.

This method has a standard implementation. In case the standard implementation is incompatible with your connection type (i.e., it does not return a list of database entities), you will have to override this method. Consider the `MsSqlDataConnection` implementation of this method as an example:

```
public override string[] GetTableNames()
{
  List<string> list = new List<string>();
  GetDBObjectNames("BASE TABLE", list);
  GetDBObjectNames("VIEW", list);
  return list.ToArray();
}

private void GetDBObjectNames(string name, List<string> list)
{
  DataTable schema = null;
  using (DbConnection connection = GetConnection())
  {
    connection.Open();
    schema = connection.GetSchema("Tables", newstring[] { null, null, null, name });
  }
  foreach (DataRow row in schema.Rows)
  {
    list.Add(row["TABLE_NAME"].ToString());
  }
}
```

TestConnection method

This method is called when you click the "Test" button in the connection settings.

The method has a default implementation that creates the connection and tries to open it:

```
public virtual void TestConnection()
{
  DbConnection conn = GetConnection();
  if (conn != null)
  {
    try
    {
      conn.Open();
    }
    finally
    {
      conn.Dispose();
    }
  }
}
```

If the test was successful, the method does nothing. Otherwise, an exception occurs when the connection is opened, which is handled in the "Connection Wizard" and gives a window with an error text.

As a rule, you don't need to override this method. You may need it if your connection does not use a `DbConnection` object to access the database (and therefore you do not return it in the `GetConnection` method).

FillTableSchema method

The method fills in the table schema (i.e. field names and types).

The following parameters are passed to the method:

Parameter	Description
table	The DataTable object whose schema you want to fill.
selectCommand	Query text in SQL.
parameters	Query parameters, if defined.

The method has a default implementation that uses objects returned by the `GetConnection` and `GetAdapter` methods:

```
public virtual void FillTableSchema(DataTable table, string selectCommand, CommandParameterCollection
parameters)
{
 using (DbConnection conn = GetConnection())
 {
  OpenConnection(conn);

  // prepare select command
  selectCommand = PrepareSelectCommand(selectCommand, table.TableName, conn);

  // read the table schema
  using (DbDataAdapter adapter = GetAdapter(selectCommand, conn, parameters))
  {
   adapter.SelectCommand.CommandTimeout = CommandTimeout;
   adapter.FillSchema(table, SchemaType.Source);
  }
 }
}
```

In most cases, you do not need to override this method. You may need to do this if you don't use the `DbConnection` and `DbDataAdapter` objects to access the data (and therefore don't implement the `GetConnection` and `GetAdapter` methods).

FillTableData method

The method fills the table with data.

The following parameters are passed to the method:

Parameter	Description
table	The DataTable object to be filled.
selectCommand	Query text in SQL.
parameters	Query parameters, if defined.

The method has a default implementation that uses objects returned by the GetConnection and GetAdapter methods:

```
public virtual void FillTableData(DataTable table, string selectCommand, CommandParameterCollection
parameters)
{
 using (DbConnection conn = GetConnection())
 {
  OpenConnection(conn);

  // prepare select command
  selectCommand = PrepareSelectCommand(selectCommand, table.TableName, conn);

  // read the table
  using (DbDataAdapter adapter = GetAdapter(selectCommand, conn, parameters))
  {
   adapter.SelectCommand.CommandTimeout = CommandTimeout;
   table.Clear();
   adapter.Fill(table);
  }
 }
}
```

In most cases, you do not need to override this method. You may need to do this if you don't use the DbConnection and DbDataAdapter objects to access the data (and therefore don't implement the `GetConnection` and `GetAdapter` methods).

Connection Editor

To edit the connection, use a control of type UserControl, which is displayed in the connection selection window (in the figure the control is highlighted with a red frame).

Each connection type has its own editor. All editors are inherited from the base class - `FastReport.Data.ConnectionEditors.ConnectionEditorBase` :

```
public class ConnectionEditorBase : UserControl
{
  protected virtual string GetConnectionString();
  protected virtual void SetConnectionString(string value);
}
```

To create your own editor, do the following:

- Add a new control to the project (Add->User Control...) and select "Inherited User Control" in the "Add New Item" window:
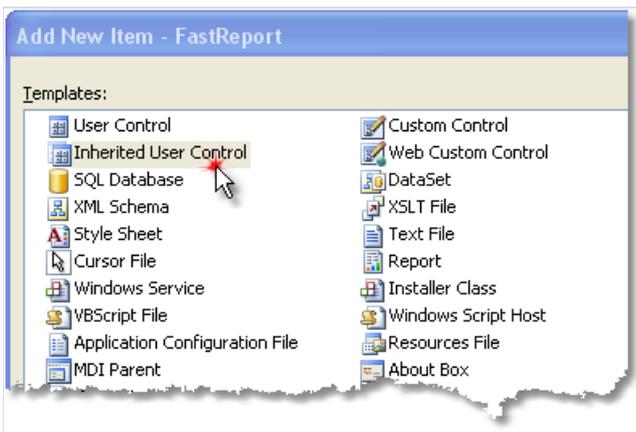
- select the ConnectionEditorBase base class type:
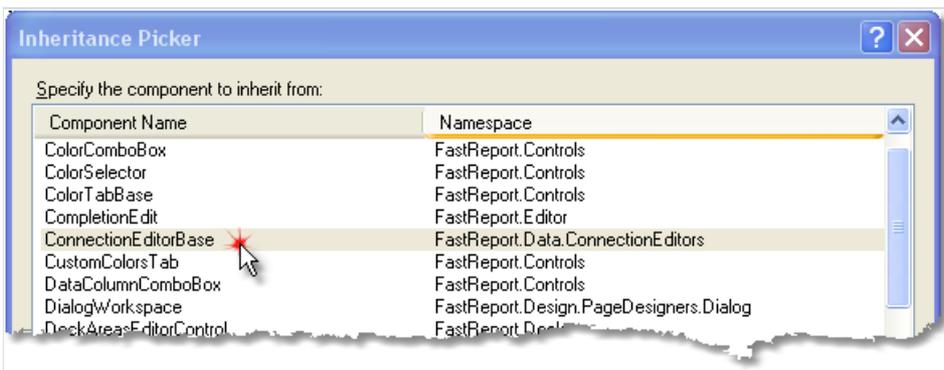
A control that looks like this will be added to the project:
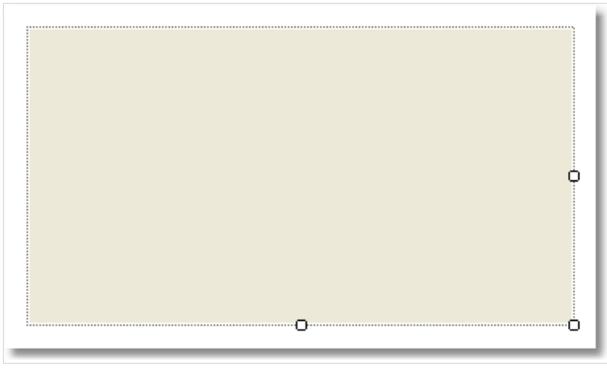
On the form of the editor, place the controls you need. Change the height of the editor to accommodate all the elements. The width of the editor is fixed and cannot be changed.

The methods of the base class allow you to fill the controls with values from the connection and, conversely, to pass values from the elements to the connection.

GetConnectionString method

This method is called when you close the editor window with the OK button. It should return a connection string that contains the values entered in the editor.

SetConnectionString method

This method is called the first time the editor is displayed. It must parse the connection string into its component parts and display their values in the editor. To parse a connection string, it is convenient to use a class of type DbConnectionStringBuilder, which is available in every connection type. For example, for MS SQL it is

`SqlConnectionStringBuilder` .

Registering a connection in FastReport

In order for your connection to be used in the FastReport designer, it must be registered. There are two ways to do this.

Method 1: Your connection is part of your project (i.e. it is not in a separate .dll plugin). Registration is done using the following code:

```
FastReport.Utils.RegisteredObjects.AddConnection(typeof(MyDataConnection));
```

This code must be executed once in the entire lifetime of the application, before you start the designer.

Method 2. Your connection is contained in a separate .dll plugin. In this case you can connect the plugin to FastReport, and it will be loaded every time you work with the designer. To do this, you need to declare a public class like `FastReport.Utils.AssemblyInializerBase` in the plugin and register your connection in its constructor:

```
public class AssemblyInitializer : FastReport.Utils.AssemblyInitializerBase
{
  public AssemblyInitializer()
  {
    FastReport.Utils.RegisteredObjects.AddConnection(typeof(MyDataConnection));
  }
}
```

When you load your plugin, FastReport will find classes like AssemblyInitializerBase and initialize them.

To attach a plugin to FastReport, bring up the designer and select the `Settings...` item in the View menu. In the window that opens, select the "Plugins" tab and add your .dll plugin.

This can also be done by adding the module name to the FastReport configuration file. This file is created in the user directory by default:

```
C:\Documents and Settings\User\Local Settings\Application Data\FastReport\FastReport.config
```

Add a plugin inside the `Plugins` tag:

```
<?xml version="1.0" encoding="utf-8"?>
<Config>
  ...
  <Plugins>
    <Plugin Name="c:\Program Files\MyProgram\MyPlugin.dll"/>
  </Plugins>
</Config>
```

Release Notes

FastReport .NET 2023.2

New opportunities

Blazor WebAssembly support

Added FastReport.Blazor.Wasm package with Blazor WebAssembly support for owners of FastReport .NET Enterprise and higher editions. Now you can use Razor components to display a report in your WebAssembly application. Attention! Blazor WebAssembly support is currently in beta.

```
<WebReportContainer WebReport="WebReport" />
```

[Read more in this article.](#)

Ability to open another report page

The designer now allows you to open and add pages and dialog forms of another report to the developing report. To do this, go to the "File" menu and select "Open Page...".
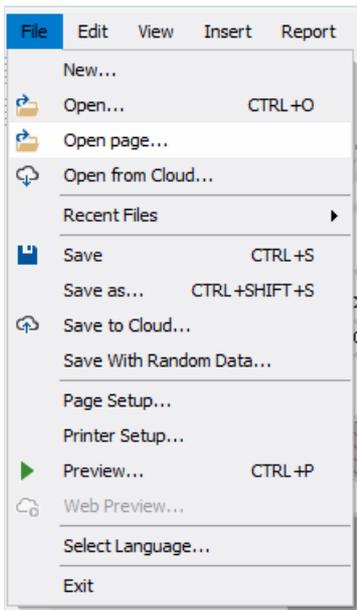
Next, the standard file selection dialog box will open where you can select a report. After that a window will appear with a list of pages and a preview of the selected page.
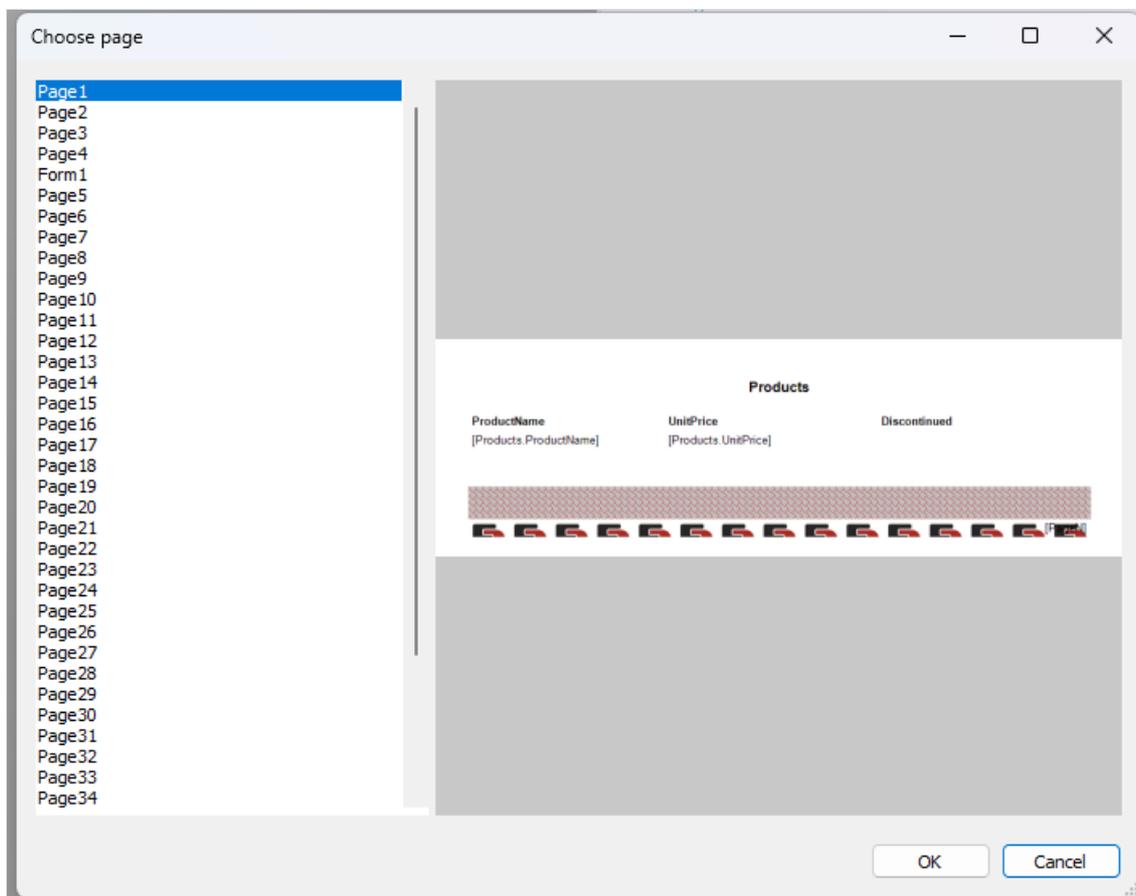
Here you can select one or more pages to be added to the current report. The names of pages and all objects contained in them, will be changed to unique, if the report already has them. This is necessary to avoid errors, as identical names are not allowed.

Read more in this article.

New icons for the Ribbon interface

New Visual Studio-style icons have been added to the Ribbon interface in the designer.
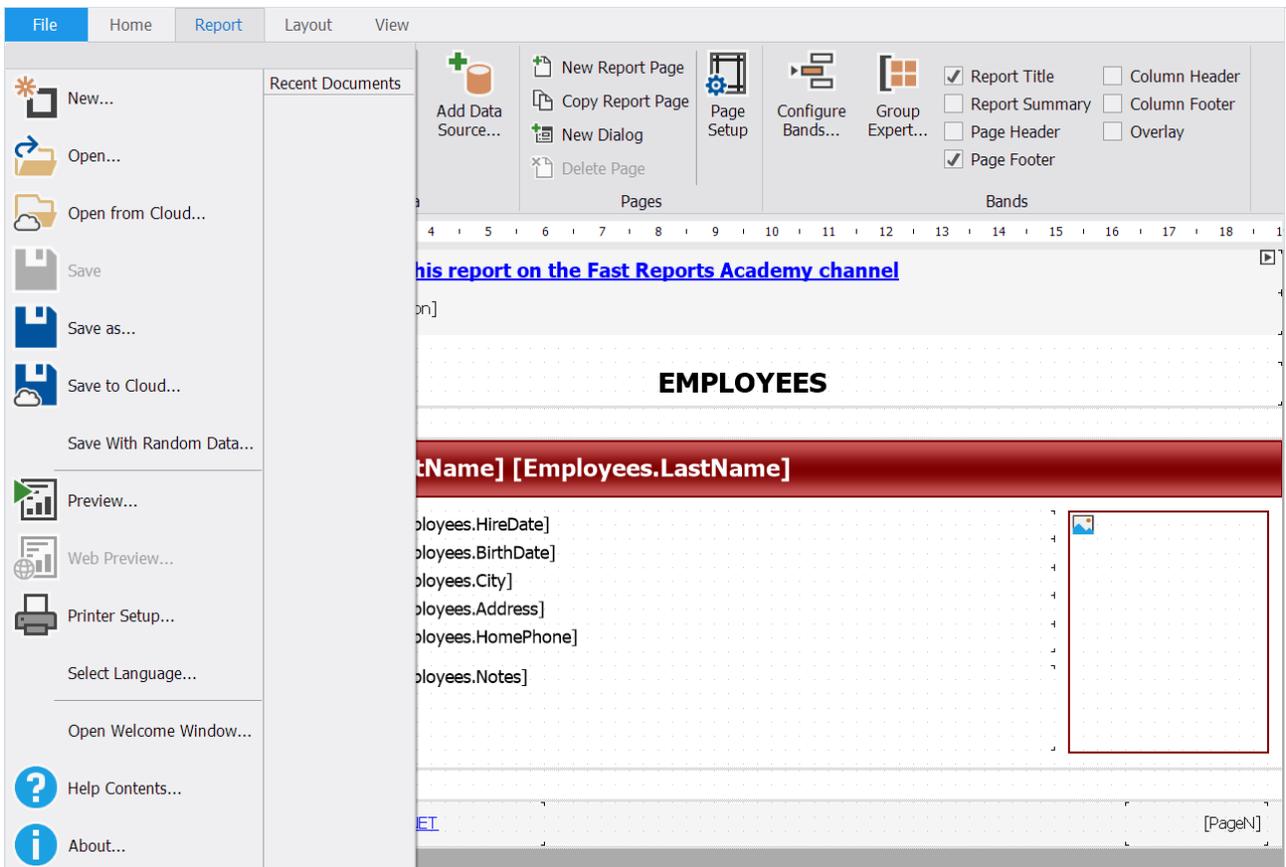
You can select them in the user interface options.
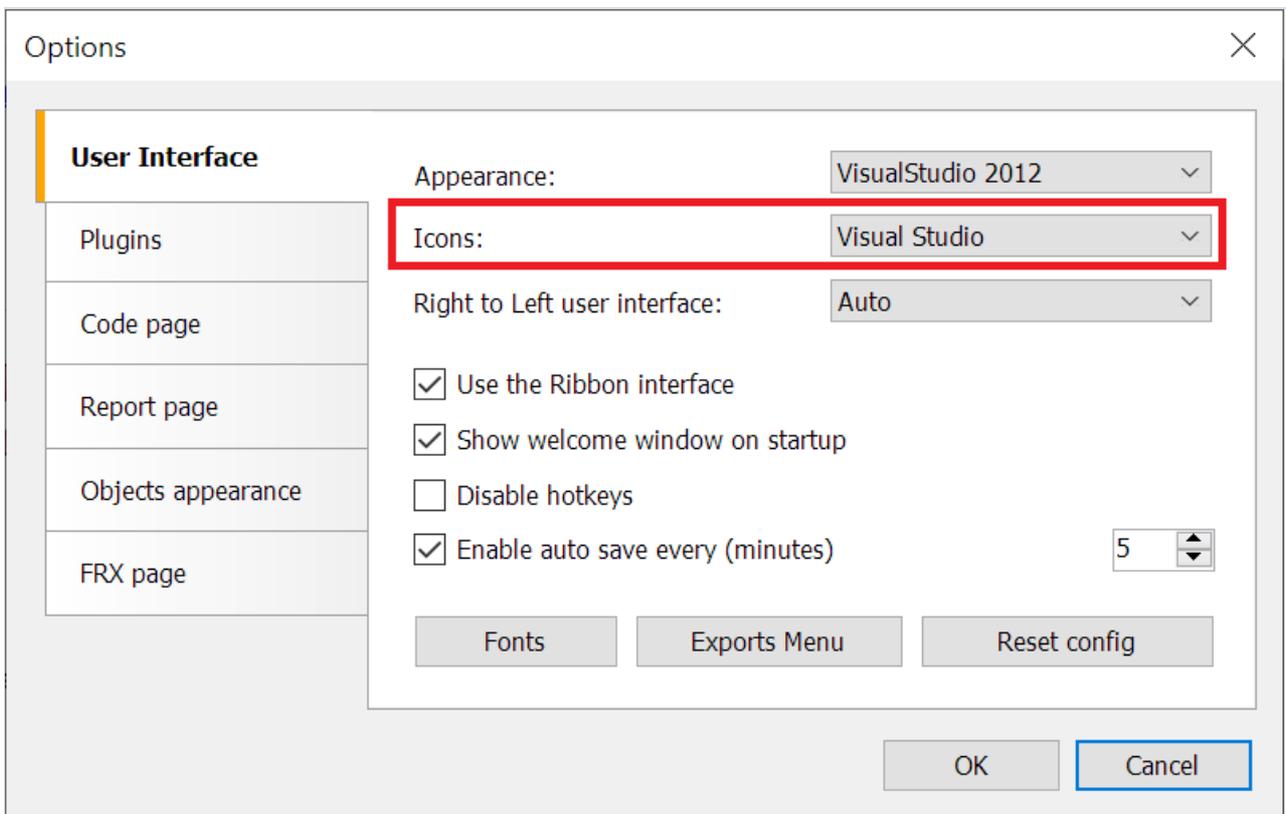
Will need to restart the designer for the changes to take effect.

Filter in the properties window

A new button has been added to the properties window that allows you to enable the display of object-specific properties. For example, for a text object, this mode displays the Text, Font properties. Common object properties such as Top, Left, Height and Width are not displayed.

Report validator changes

The report validator now doesn't run in the background, but runs with a separate "Validate Report" button in the "Report" menu. In addition, the validator window has been removed, and its messages are displayed in the window "Messages".

Ability to hide connection string

Added a new property Config.ConnectionStringVisible, which gives the ability to hide the connection string in the designer. Can be used to differentiate permissions between the application developer and the report user. When set to false, the user will not be able to see and edit connection strings in the designer.

WebReport changes

Added support for MemoryCache. By default, at the moment, the current WebReportCache is used. You can enable MemoryCache when registering FastReport services:

```
services.AddFastReport(options =>
{
  options.CacheOptions.UseLegacyWebReportCache = false;
});
```

Unlike the built-in cache in WebReport, MemoryCache unloads WebReport instances more aggressively after a certain time CacheOptions.CacheDuration of WebReport instance inactivity, which can help in cases where the old cache for some reason does not clear memory.

Added the ability to fix the toolbar on the screen. Now you can configure the toolbar to always stay in place, even when scrolling through the page. This is convenient when working with large reports - the toolbar will always be visible.

To pin the toolbar on the screen, you need to set the following property:

```
webReport.Toolbar.Sticky = true;
```

Now the toolbar will always be in view
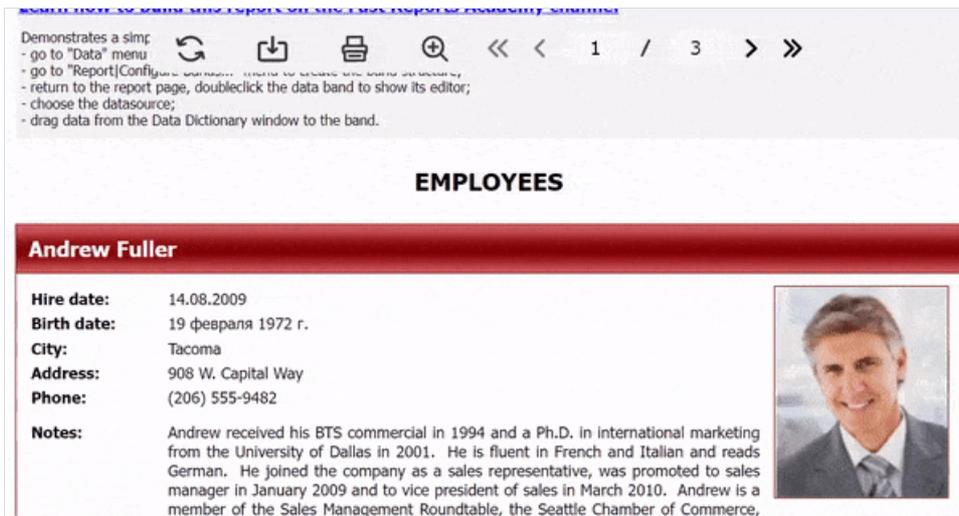
Also, the ability to customize the export settings window has been added. Now it can be made fixed on the screen and displayed in the foreground. To do this, you need to set the following property:

```
webReport.Toolbar.Exports.PinnedSettingsPosition = true;
```
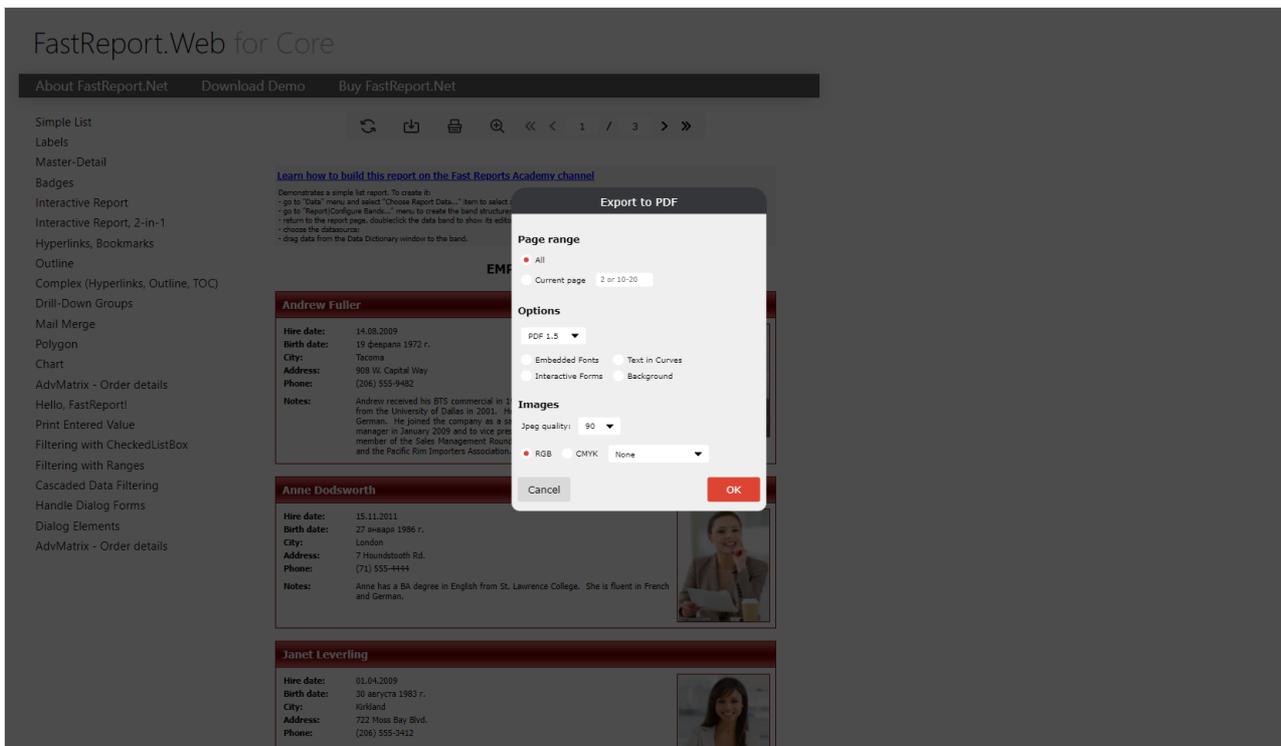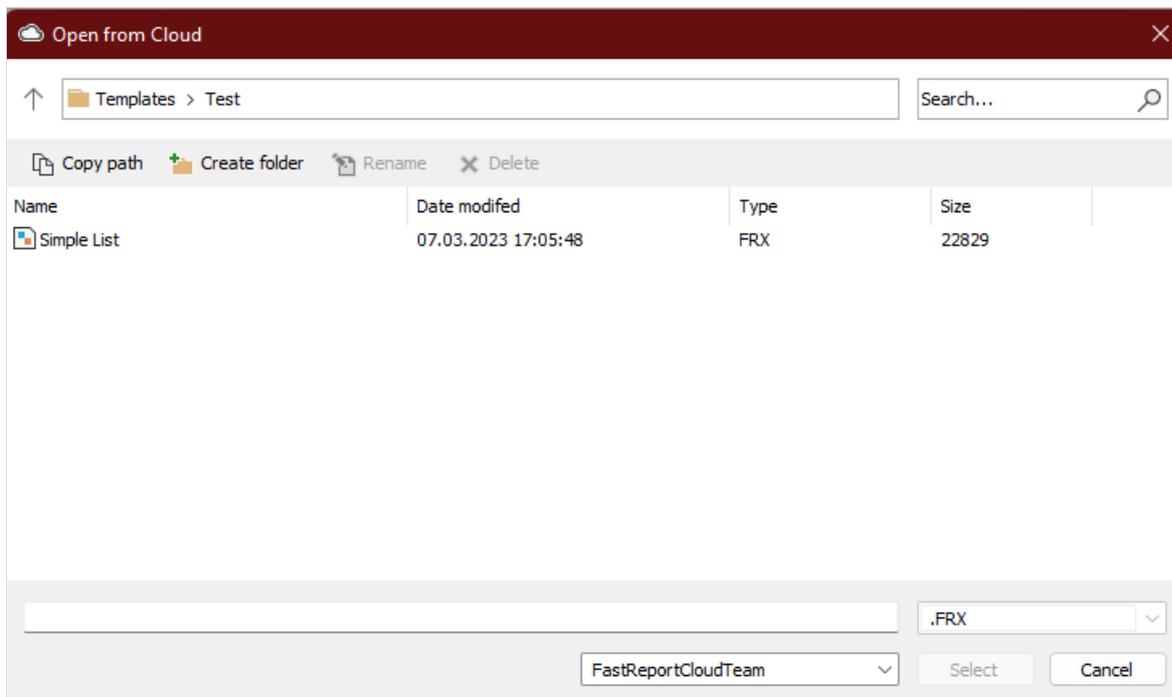
Validation for entering a range of pages has been added to the export settings window. Now, in case of incorrect input, the field will look like this.
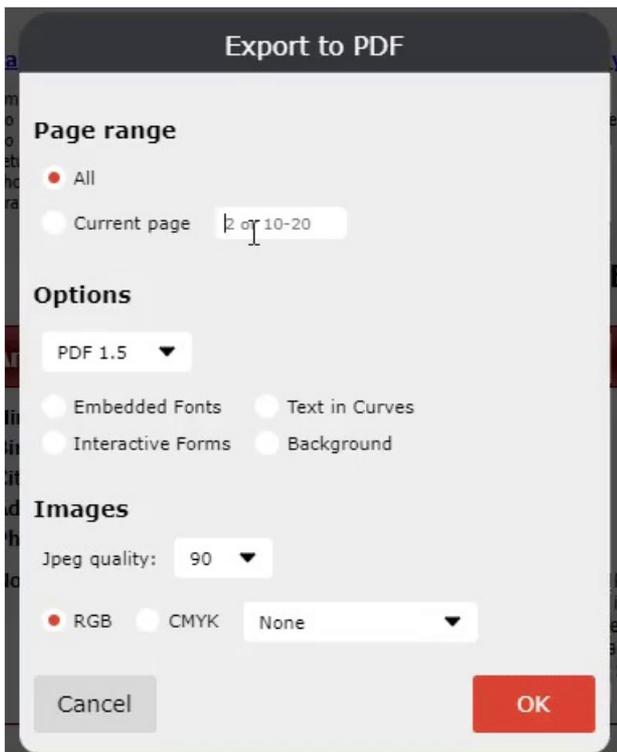
FastReport.Core.Skia improvements

Improved the performance of the FastReport.Core.Skia package. Export errors have been fixed, examples are listed below:

Fixed the rendering of objects with CanShrink = true:

Before	After
This is TextObject with CanShrink = true. This is TextObject with CanShrink = true. This is TextObject with CanShrink = true. This is TextObject with CanShrink = true. This is TextObject with CanShrink = true. This is TextObject with CanShrink = true. This is TextObiect with	This is TextObject with CanShrink = true. This is TextObject with CanShrink = true. This is TextObject with CanShrink = true. This is TextObject with CanShrink = true. This is TextObject with CanShrink = true. This is TextObject with CanShrink = true.

Fixed the background rendering of objects with transparent backgrounds:

Before	After
TEST LOGO	TEST LOGO

Added a standard font that depends on the operating system. Now, if the font from the report is not detected in the system, the export will not produce an error, but will render the report with the standard font.

For other fixes, please refer to the full list of changes.

Updated design of FastReport Cloud file manager

Updated the design of the file manager window for more convenient and intuitive use of the service. Changes have been made to the layout of interface elements and color scheme, which will improve the overall visual perception of users.

Full list of changes

```
   [Engine]
+ added property Config.ConnectionStringVisible, which indicates whether the connection strings of data
sources will be displayed in the designer;
- fixed a bug with extraction of procedures in connection that cannot contain procedures;
- fixed a bug where the first column of the page was always displayed in the leftmost position;
- fixed a bug when GaugeObject.Value property was set equal GaugeObject.Minimum, if new value was more
than GaugeObject.Maximum. Now it will be set equal GaugeObject.Maximum;

   [Designer]
+ added the ability to open report from FastReport Cloud using recent files list;
+ added a context menu to the page panel elements;
+ a context menu for creating new pages and dialog forms has been added for the panel with report pages;
+ added new Visual Studio style icons for the Ribbon interface;
+ added "Sync" button in the "Report Tree" window;
+ added Filter button in the Properties window;
+ added HiDPI icons for Ribbon-interface;
+ added support of DBNull and Guid types for parameters;
* now the name of the attached file when exporting to mail, can be set from the code when creating the
export form;
* report validator now runs from "Report|Validate report" menu. "Messages" window is used to display
validation messages;
* changed interface of QR code editor;
- fixed a bug on right clicking Data Sources menu item;
- fixed a bug when checkbox "Select all" was not visible in Data wizard;
- fixed a bug causing System.NullReferenceException when deleting dialog form;
- fixed the absence of the Api key when re-opening the Account->Server window if it was entered in the
standard server item;
- fixed incorrect web address when trying to preview webview for custom server;
- fixed the problem of collapsing panels and incorrect change of the language of tabs and bars when
changing the localization in the Ribbon interface;
- fixed issue with adding tables that were not selected in the connection wizard;
- fixed a bug causing System.NullReferenceException when creating connection to stored procedure;
- fixed exception when manually entering an invalid parameter type;
- fixed a bug where it was impossible to set an object to a transparent color;
- fixed reopening of the query wizard;

   [Preview]
+ added a message about sending a report to the mail in the status bar;

   [Exports]
```

+ added word wrapping in cells when exporting to Excel 2007;
- fixed a bug that made MSChart text blurred after HTML export;
- fixed incorrect margins when exporting the report to HTML;
- fixed an error that made the transparent background become white with Skia;
- fixed a bug with an extra empty page when exporting if there are bands with the Exportable property equal false;
- fixed a bug when padding top was not taken into account when exporting to layered HTML;
- fixed an error that made the text go beyond the table when the page was zoomed out in HTML export;

  [WebReport]
+ added Blazor WebAssembly support;
+ added support for DI in WebReport.Core/Blazor. To use, call services.AddFastReport();
+ added support for Microsoft.Extensions.Caching.Memory.MemoryCache instead of the standard WebReportLegacyCache. To use, when registering a DI container, use services.AddFastReport(options => options.CacheOptions.UseLegacyWebReportCache = false);
+ implemented the ItemCheck event in CheckedListBox;
+ added an option to enable the toolbar to display regardless of the screen position in WebReport using WebReport.Toolbar.Sticky property;
+ added asynchronous version of method WebReport.Designer.SaveMethod - WebReport.Designer.SaveMethodAsync;
+ added validation of page range in WebReport export settings window;
+ added WebReport.Toolbar.Exports.PinnedSettingsPosition property. If enabled, the container of export settings will be fixed on the screen and displayed in the foreground;
- fixed a bug when selection mode in ListBox was multiple, but it was able to select only one item;
- fixed a bug of non-refreshing dialog when CheckedBox was the initiator of the event. In this case, add at least one dependent object of the CheckedBox to the DetailControl property;
- fixed a bug when in .NET Framework MVC application the report with dialog form on clicking "OK" would not hide dialog form and not show loading of the report;
- fixed an error that caused extra pages to appear when printing;
- fixed incorrect work of report 'Interactive Report' on WebReport.Core;
- fixed rare NullReferenceException in WebReportLegacyCache;

  [Online Designer]
- fixed a bug where First Page Source, Other Page Source, Last Page Source and Duplex properties was not saved when changing ReportPage;
- fixed an error that made the report preview not refresh before pressing "Refresh" button;

  [.Net Core]
+ the script compiler will now display errors depending on the selected locale set with FastReport.Utils.Res.LoadLocale() or FastReport.Utils.Config.CompilerSettings.CultureInfo;
- fixed an issue where text with CanShrink = True was incorrectly rendered after export with Skia;
- fixed a bug with incorrect indent width between characters with TextRenderType = HtmlTags in Skia;
- fixed a bug that caused the watermark with transparency to have a gray background when exporting with Skia;
- fixed an error that caused incorrect calculation of table row height;

  [CoreWin]
- fixed error when trying to add new data connection;

  [Mono]
+ added zoom control in preview and designer windows;
- fixed problem of scaling PreviewControl;

  [Demos]
+ added demo-app ASP.NET Core (Razor pages) under .NET 6.0;
* updated demo applications for FastReport Core;

  [Extras]
- fixed a situation in which the host during logout could not match the one during authorization;
- fixed a bug when updating an expired session in the Account window, a browser opens and requests re-authorization.
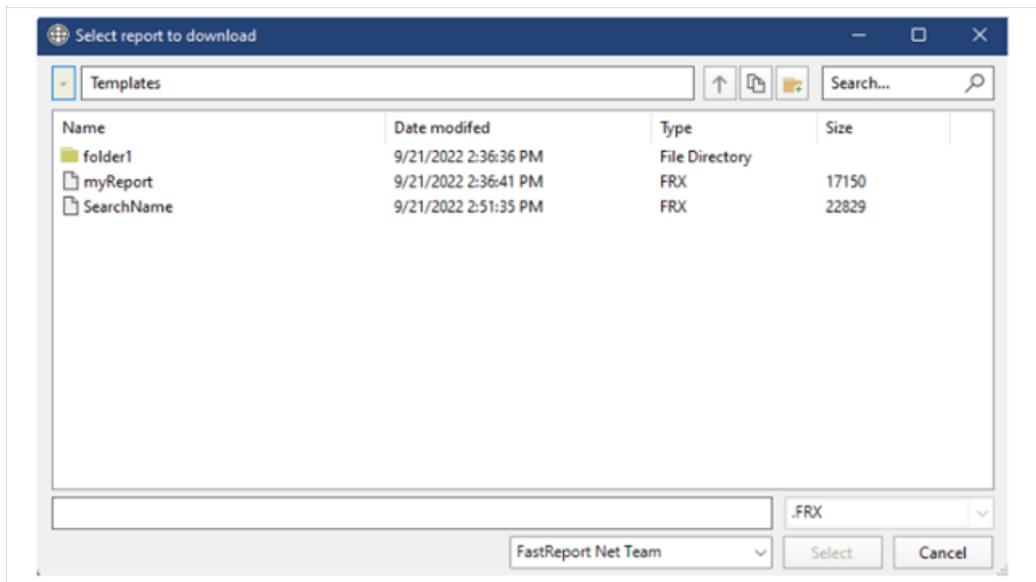
FastReport .NET 2023.1

New opportunities

Integration with FastReport Cloud

FastReport .NET, FastReport CoreWin, and FastReport Mono now support some interaction experience with FastReport Cloud.

Downloading and uploading reports

Now you can download the report from Cloud and work on it in the designer, or vice versa — upload your files to Cloud.
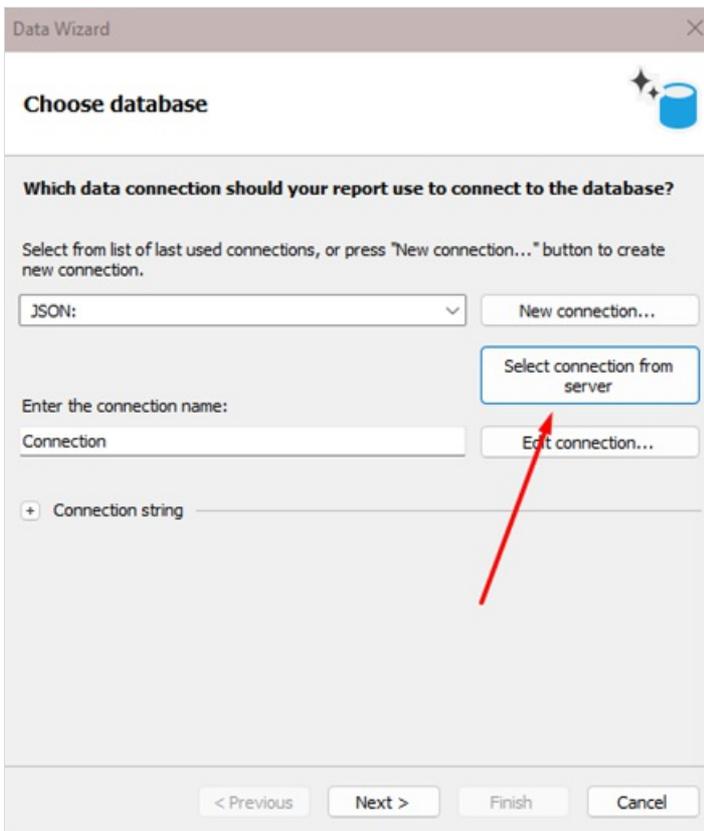

Web Preview

A web preview function has also appeared in addition to the standard preview. The report can only be viewed this way if it was opened from Cloud.

Connecting to FastReport Cloud data sources

FastReport Cloud can store connections to data sources. From now on, you have the option to add these data sources to your report.

It also became possible to upload the connection to Cloud.
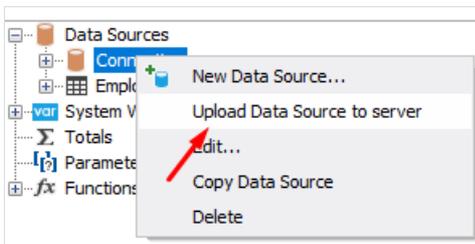
Read this article to learn more about the new features

.NET 7 support

We have added .NET 7. support for FastReport.Core and FastReport.CoreWin. This platform improves application performance and adds many new features to your projects.

Report validator improvements

Increased work speed

Now the report validator runs in a single thread. The speed of its work is significantly optimized. You can notice the changes in processing reports with a large number of errors. While the validator is checking the report, the check window shows a respective message.
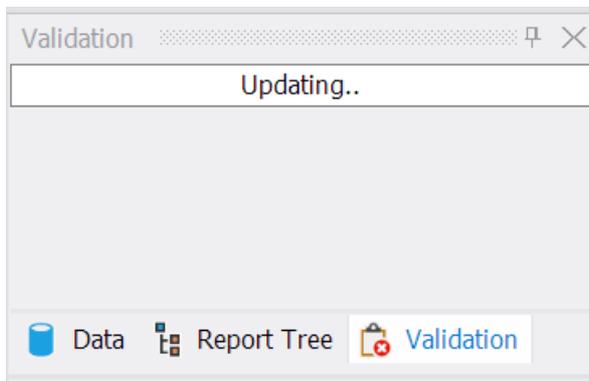
In this case, you can edit the report. A table with errors will appear upon completion of the validator.

Validator table setup

For convenience, we have added a new column with error numbers. Its display can be enabled or disabled via the table context menu. In the same way, you can customize the display of the error type column.
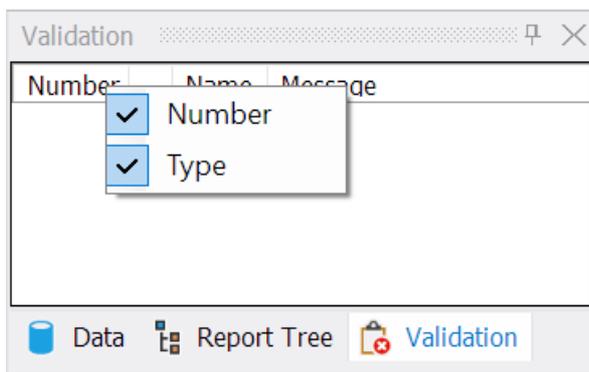
JasperReports Template Converter

We have added the option to convert report templates from JasperReports to FastReport .NET templates. JasperReports reports may contain objects that are not supported by the FastReport designer. These objects will not be converted or will be replaced to make the generated report as similar as possible to the one created in JasperReports.

Read more in the article.

MSChartObject improvements and fixes

The MSChartObject object has many properties and settings. The most frequently used ones are moved to the object editor. Properties that are not available in the editor can be modified using the Object Inspector. However, there was a problem with these properties — when they were changed, the report was not seen as modified. As a result, saving was not available.

To save the report, it was necessary to change its other property or object. In addition, the values of the specified properties were reset to their default values when preparing a report and after closing the preview window.

This bug has been fixed in the new version.

Connection to Stored Procedures in MsSQL

We have added the option to connect to procedures stored in MsSQL. This was previously available via a database query. Now you can connect to procedures much more conveniently using the interface of database table connection. They will be displayed in the selection window along with the tables.
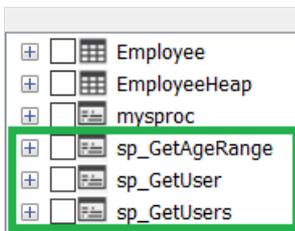
Once you select a procedure, a window with parameter settings, if any, will appear.

[Read more in the article.](#)

Export Improvements

"Print Optimized" option in RTF export

We have added a new PrintOptimized property and a corresponding option in the export window. Enabling this option will greatly increase the quality of the exported image. However, the size of the output file will be bigger.
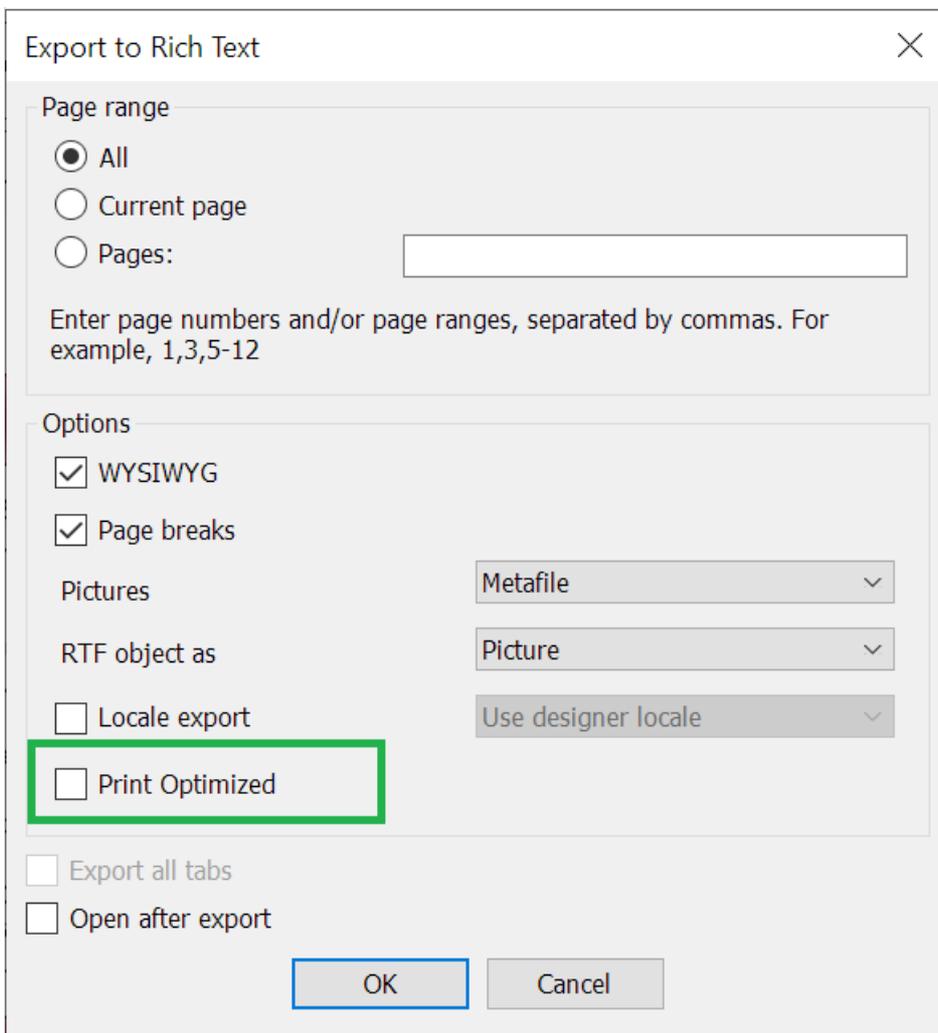
UseFileStream property in Excel 2007 export

We have added a new option, UseFileStream, for exporting to Excel 2007. It can only be used when exporting from code to a file. This option is useful when exporting reports with a large number of pages (several tens of thousands) in multiple threads. This will help you avoid memory shortage errors. In other cases, it does not make much sense to use it and it is not recommended to do it. Example:

```
Report report = new Report();
Excel2007Export export = new Excel2007Export();
export.UseFileStream = true;
report.Export(export, "report.xlsx");
```

Accounting format when exporting to Excel 2007

You can now export the currency data format as an accounting format. To do this, a corresponding option has been added in the export window and the CurrencyToAccounting property.

Update of WebReport Core/Blazor design

The appearance of the toolbar has been changed. The color palette has been changed to lighter, with the toolbar now centered by default.

The forms of export settings have also been changed. Previously, some properties were set using a slider. For greater convenience, these properties are now set as a drop-down list.
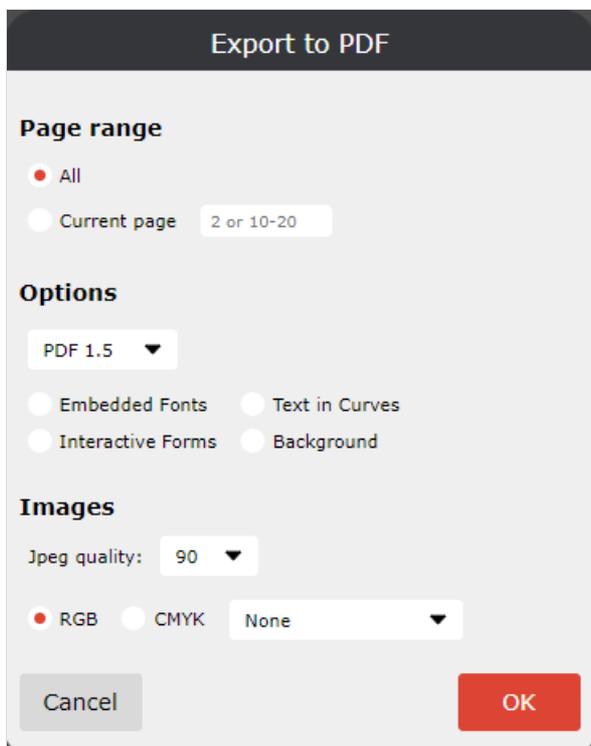
We changed the toolbar design, but we kept the possibility of its customization, so it is possible to partially restore the previous design. To do this, use the following settings:

```
var toolbarSettings = new ToolbarSettings
{
    Color = Color.LightGray,
    Roundness = RoundnessEnum.None,
    ContentPosition = ContentPositions.Left,
    IconTransperency = IconTransparencyEnum.None
};


webReport.Toolbar = toolbarSettings;
```

Upgrading the minimum .NET Framework version from 4.0 to 4.6.2

We are upgrading the minimum supported version of FastReport .NET to .NET Framework 4.6.2 due to the following:

- support for .NET Framework 4.0 has long been ended;
- there are problems with building the FastReport source code in the latest versions of Microsoft Visual Studio;
- the need to implement new APIs.

Full list of changes

```
    [Engine]
+ added property Report.IsPrepared;
+ added TextRenderType.Inline;
+ implemented converter of JasperReports templates;
+ implemented connection to stored procedures in MsSQL;
* increased minimum version of .NET Framework from 4.0 to 4.6.2;
* receiving JSON in the data source is exposed to the interface part;
- fixed a bug leading to System.ArgumentException when TextObject.FontWidthRatio property equal zero;
- fixed highlight of text in RTF parser;
- fixed multiple requests to get image when using URL in ImageLocation;
- fixed IsNull function;
- fixed a bug with drawing RichObject with aligned pictures;
```

- fixed an issue where the calculation of vertical distances was incorrect when converting RichObject to text;
- fixed AdvMatrix object bug with report refresh;
- fixed a bug with getting JSON row of JsonTableDataSource;
- fixed a bug leading to infinite loop when building table if there is not enough space on page for one row;

  [Designer]
+ added ability to create calculated column for IEnumerable data sources;
+ added window with message about loading a report when opening a file;
+ added column with error numbers in table of report validation;
+ added the ability to hide and show columns with the number and type of error in the report validation table;
+ added notification form when trying to resave report that has already been modified;
+ added ability to show web preview of report that was opened from FastReport Cloud;
+ added ability to interact with data source from Cloud - downloading, uploading, updating;
* increased the speed of the report validator;
* the delete band button is now disabled in situations where the band cannot be deleted;
* changed root folder name on FastReport Cloud form, it depends now on localization;
* now there is not possible to create a table in the query wizard if another table with the same name already exists;
- fixed data tree view with IEnumerable data source, which column was not adding, if it consists of value type;
- fixed a bug with localization of the "Remove" button in the report properties on the "Script" tab;
- fixed a bug with selection object after click on row in "Validation" window;
- fixed a bug due to which selected object did not change when changing the height of the band with mouse;
- fixed a problem with System.OverflowException when editing text object without editor;
- fixed a bug causing System.StackOverflowException when copying formatting;
- fixed selection of object located on inactive page when clicking on row in "Validation" window;
- fixed showing progress of updating list of errors in "Validation" window when changing report;
- fixed an error with an invalid value when changing the line color in the MSChartObject editor;
- fixed order of switching by "Tab" key in connection forms;
- fixed a bug where the border properties of the chart axes were not saved when they were changed in the editor;
- fixed incorrect values when changing the interval in the stripes on the axes in MSChartObject;
- fixed an error that occurred when deleting a band through the band configurator if the classic mode for displaying bands is selected;
- fixed an error that occurs when clicking the "Delete" button on the configure bands form if there are no bands in the report;
- fixed an error that occurs when removing bands from the workspace with holding left mouse button;
- fixed restoring state of GridControl when closing column editor form;
- fixed an error that occurs when clicking on the "Cancel" button in the Grid object column editor;
- fixed displaying label about report change when changing MSChartObject;
- fixed moving columns of GridControl in column editor form;
- fixed bugs when dragging objects from the report tree to pages and the "Code" tab;
- fixed errors in the query constructor window when adding a table to the workspace and when creating relationships between tables;

  [Preview]
+ added tooltip for the "Copy" field in the "Send by E-mail" form;
- fixed display of the print form when increasing the display scaling;
- fixed a bug when new exports did not appear in the menu;
- fixed order of switching by "Tab" key in export forms;
- fixed left indent of RichObject;

  [Exports]
+ added option "Print optimized" in RTF export;
+ added the ability to export currency data format as accounting in Excel 2007 export;
+ added UseFileStream option for Excel 2007 export;
* increased export forms for correct display of inscriptions in different localizations;
- fixed a bug with exporting lines drawn from right to left or from bottom to top when exporting to layered HTML;
- fixed a bug with exporting Tahoma italic font to PDF;
- fixed a bug that resulted in a System.ArgumentException when exporting to a stream with the ImageExport.SeparateFiles property enabled;
- fixed a bug in SVG export where some shapes were drawn twice;
- fixed a bug with the export of the accounting format in Excel 2007, which did not take into account the number of decimal places;

number of decimal places;
- fixed memory leaks in tabular-type exports;
- fixed a bug with temporary file deletion in case of emergency program shutdown during export to PDF;
- fixed a bug with exporting italic and bold fonts to PDF;
- fixed a bug due to which the background of objects with a Solid fill was not printed from the browser;
- fixed a bug with set method of HtmlTemplates.IndexTemplate property;
- fixed export of 4-byte symbols to PDF;
- fixed row height multiplier in export to RTF;
- fixed row height multiplier in table export to Word 2007;
- fixed position of first object on page with non-zero value in export to Word 2007;
- fixed a bug of access to temporary file when exporting to Excel 2007 using the UseFileStream and SplitPages properties;
- fixed a bug with localization of CurrencyToAccounting property in Excel 2007 export;
- fixed navigation buttons and page numbering display in HTML export;
- fixed ascent and descent of font in PDF-export;

  [WebReport]
* reworked WebReport.ReportPrepared property, now this property is bound to the same report's property;
* updated WebReport design for FastReport.Core.Web and FastReport.Web.Blazor;
- fixed a rare crash when trying to add an empty data source to WebReport;
- fixed a bug due to which Outline did not work in WebReport.LoadPrepared();
- removed refresh button when loading prepared report (.fpx);
- fixed an issue due to which tabs of RichObject were incorrectly calculated in WebReport;
- removed page selection in export settings for single-page reports;

  [.Net Core]
+ added support for .NET 7;
+ added LoadReport method with stream instead of filename string for Stimulsoft Import;

  [CoreWin]
- removed extra components from the Visual Studio toolbar;

  [Demos]
* updated design of demo reports;
- fixed a bug with scaling child windows in new demo application;

  [Extras]
+ plugin FastReportBGObjects was updated, added support for Bubble chart;
* connection to ElasticSearch moved to a separate plugin.

FastReport .NET 2022.3

New features

Skia support:

FastReport.Core now supports graphics and text rendering using the SkiaSharp library which is used instead of System.Drawing.Common + libgdiplus on Linux systems (but also works on other operating systems).

For this, packages with the .Skia suffix are used:

- FastReport.Core.Skia
- FastReport.Web.Skia

This version has limited support for the .NET Framework and is mainly targeted at .NET Core/.NET projects. To use it in your application, just change the package name FastReport.Core -> FastReport.Core.Skia, and add the following packages to work on Linux (on Windows and macOS, the necessary packages are added automatically):

- SkiaSharp.NativeAssets.Linux
- HarfBuzzSharp.NativeAssets.Linux

Read more about Skia support in the next article.

Report validator

A "Validation" tab has been added to the report designer (on the right, next to the "Data" and "Report Tree" tabs). Here you can check the report template and get a list of errors and warnings.

All this is displayed in a table with the object name (if there is one) and error description. If you select a row in the table, the corresponding object will be highlighted in the designer.
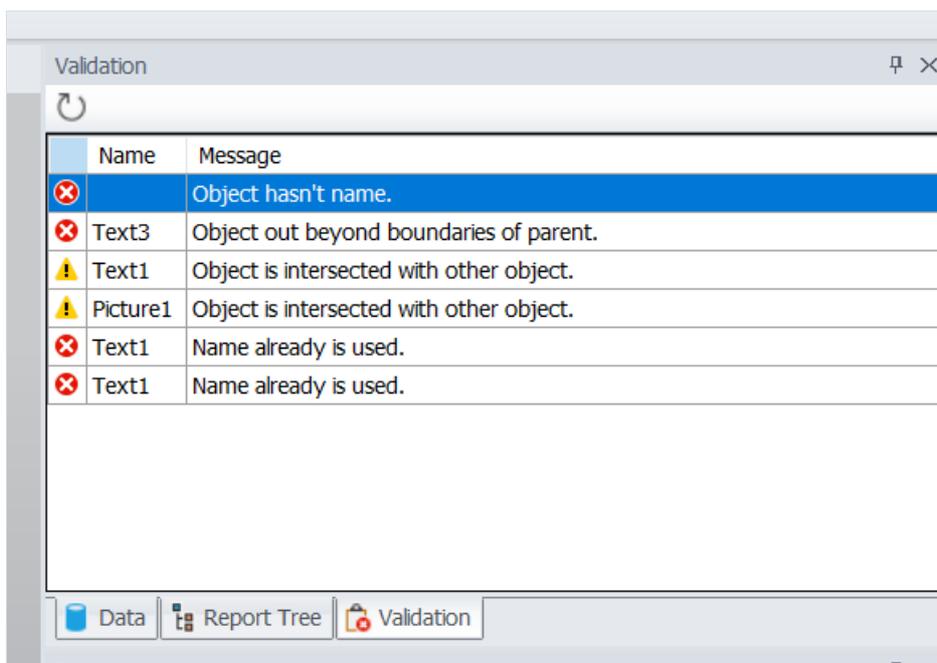
Errors and warnings can be of the following types: unnamed objects, objects with the same name, overlapping objects, objects with zero height or width, and objects that are partially or completely outside the parent object.

Objects without names and objects with the same name are critical errors. They can lead to various errors and even

crash the application while preparing a report. Besides, without a validator, these errors are very hard to find.

Intersecting objects is not a serious error. In some cases, they can be useful and used purposefully (e.g., lines or rectangles). Intersecting text objects, in most cases, can lead to incorrect exports. Especially in table exports, such as Excel. The export will result in a lot of extra cells, etc. It is necessary to be careful with such objects.

Objects partially exceeding parent object boundaries (e.g. band or page) can also be useful in rare situations. But in most cases, it causes errors in the preparation and export of the report.

Objects that are completely outside the parent one is a serious error. Finding such objects without a validator is also very hard.

Intersecting objects and objects outside the parent can now be highlighted in color (which you can choose) if the corresponding setting in designer options is enabled.
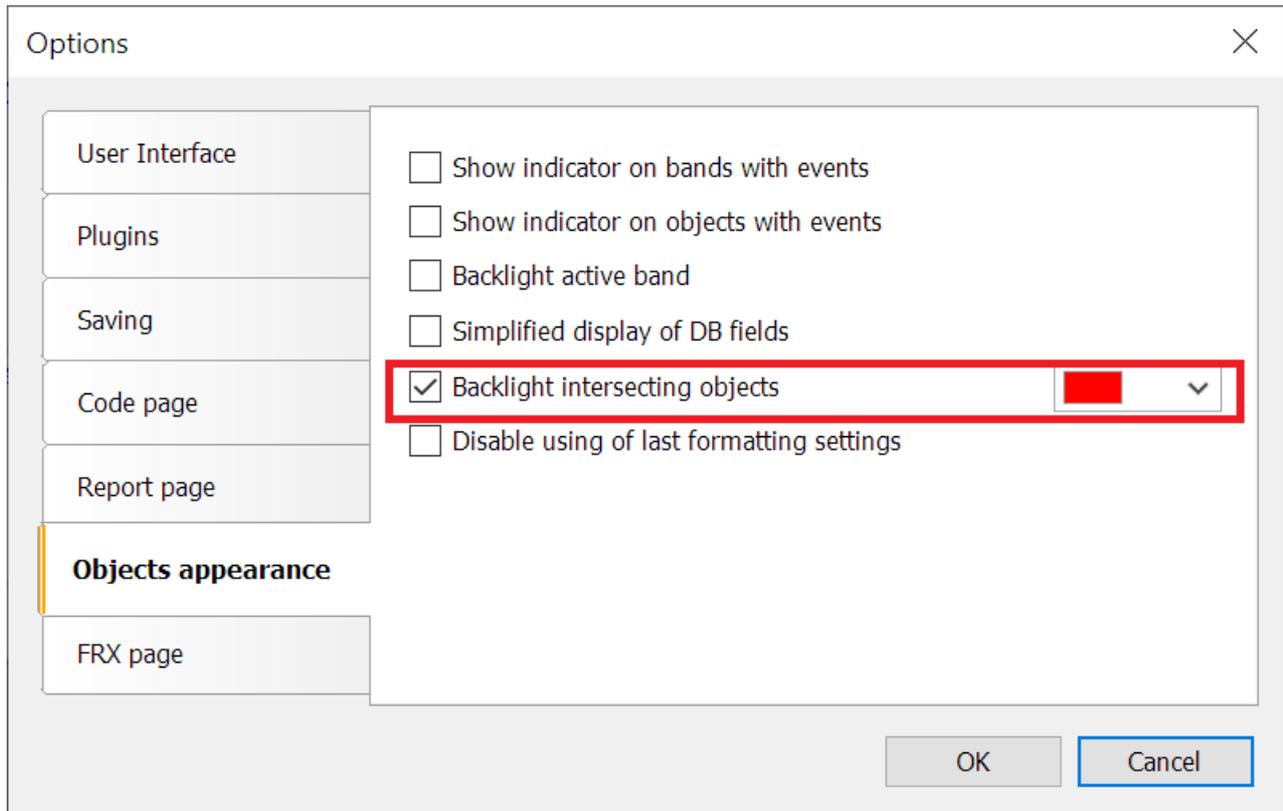
It is not necessary to use report validation. But it can be useful when your report doesn't work or look the way you want it to.

Read more about report validator in the next article.

FRX Editor

Sometimes it is necessary to edit the contents of the FRX file using third-party text editors. Now you can do this more conveniently, directly in the report designer. The FRX editor is added for this purpose. By default, it is disabled. You can enable it in the designer options.
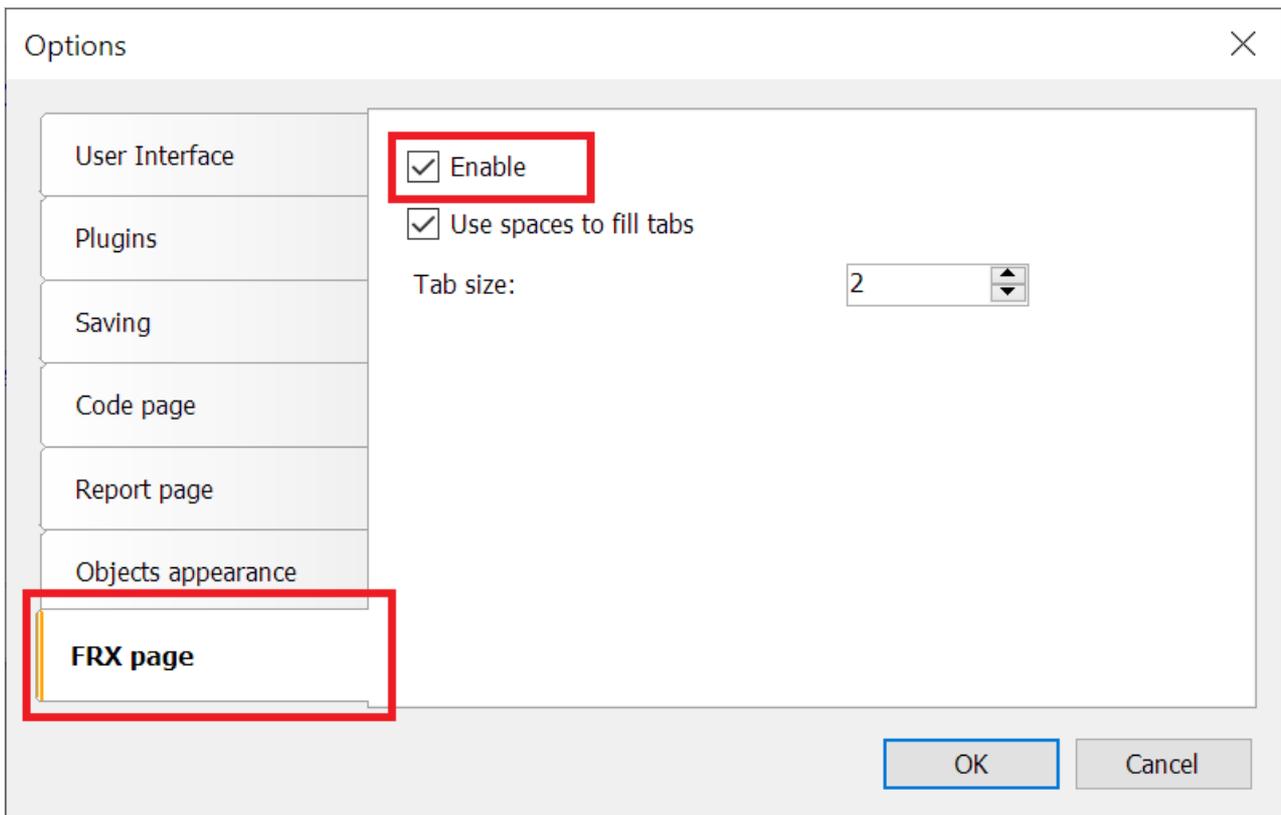
In the report designer, the FRX tab will appear to the left of the Code tab.
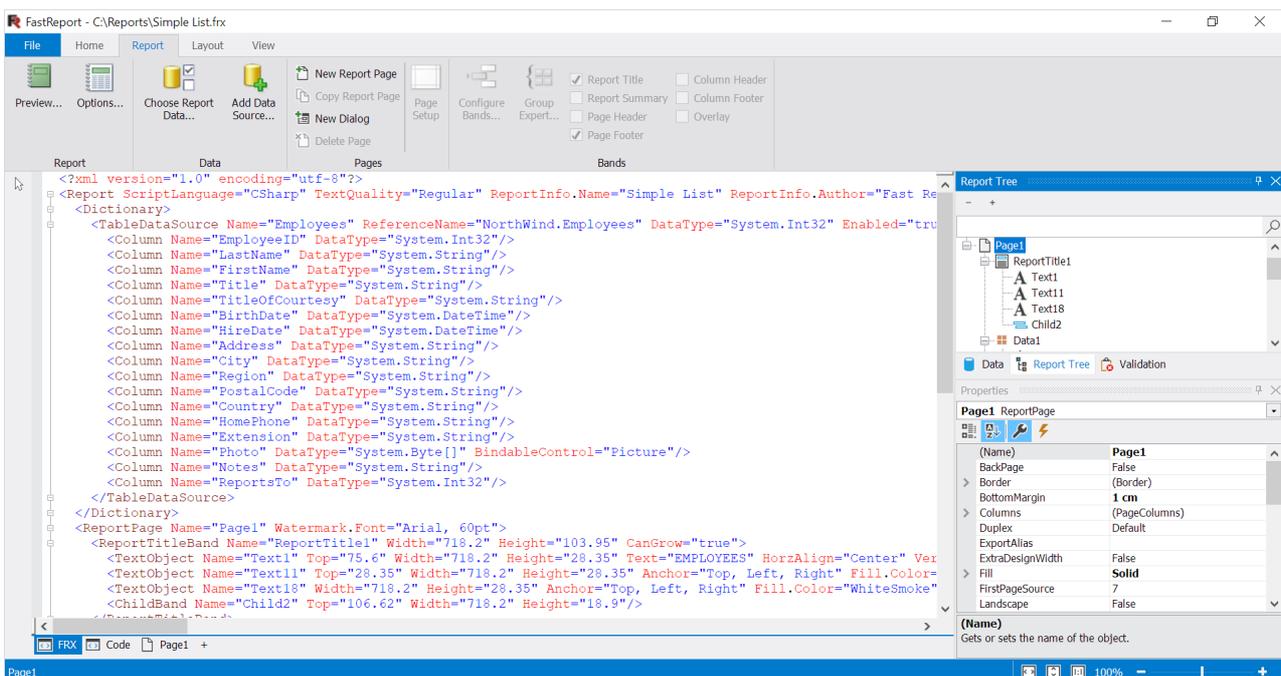
The changes made here, will be immediately applied to the report and displayed on its pages.

Read more about the FRX editor in the following article.

StimulSoft report converter

Added the ability to convert report templates from StimulSoft to FastReport .NET templates.

StimulSoft reports may contain implementation objects that are not supported by the FastReport designer. These objects will not be exported or will be replaced by others in such a way that the generated report is as similar as possible to the one created in StimulSoft. It is important to note that the import of cross-bands is implemented by moving their contents to the parent band.

Copying dialog pages

Added the ability to copy dialog pages. Both using the context menu of the dialog page and using the «Report -> Copy Report Page» button.
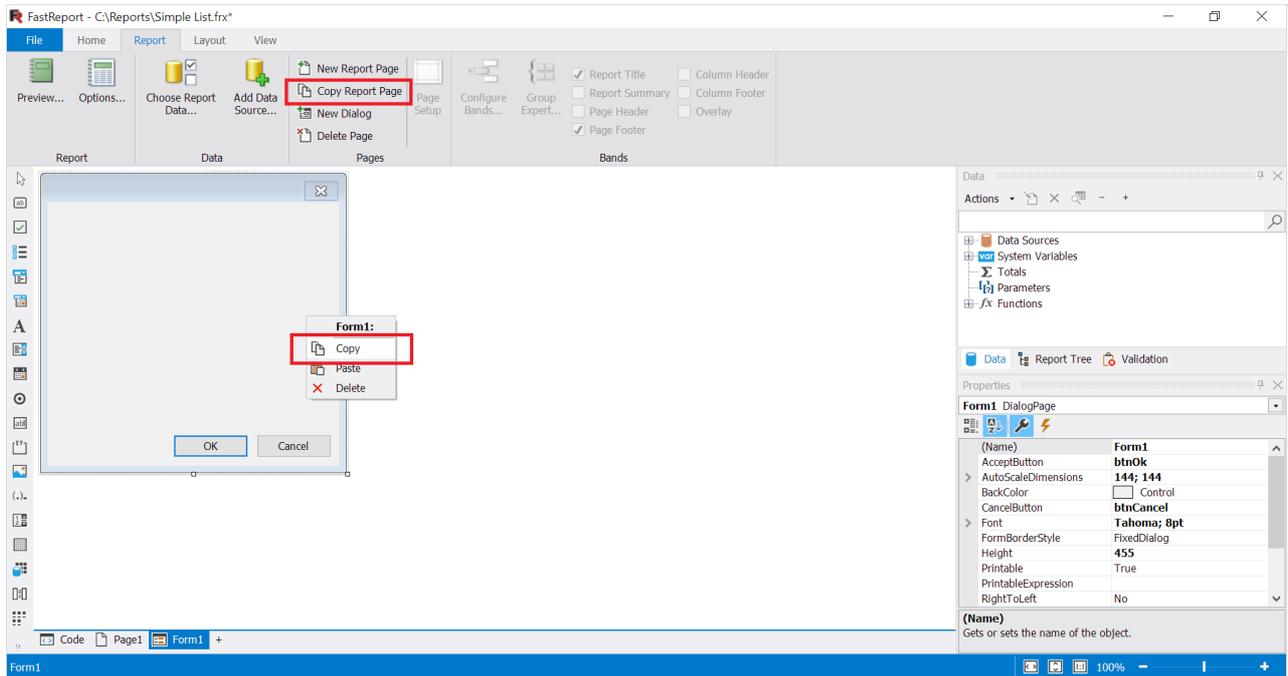
Copying creates a copy of the dialog page with a unique name. All child objects will also have unique names. However, the event handlers of the objects will be the same as those of the original page. If necessary, you must create new handlers.

Also now dialog pages can be deleted not only with the «Report -> Delete Page» button, but also via the context menu in the form editor and report tree.

Disabling last formatting settings

When creating an object in the designer, its settings will be applied to the next created object of the same type.

For example, if you create a text object, set its font size, borders, fill color, the next text object will be created with the same settings.

This is useful when you need to create several objects with the same or similar settings.

In situations when you don't need this designer behavior you can disable it in options.
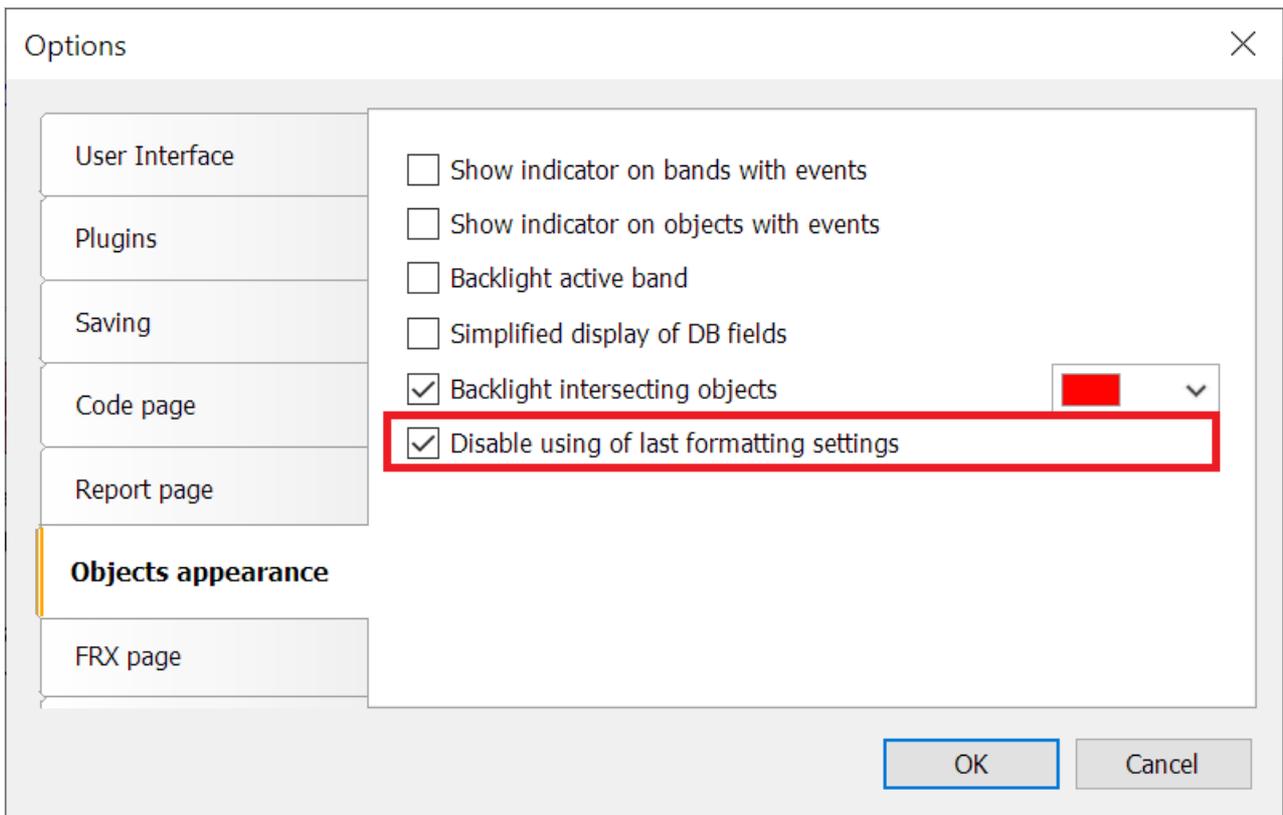
This will create objects with default settings.

Export all tabs

When viewing interactive reports, you can open detailed reports in new tabs.
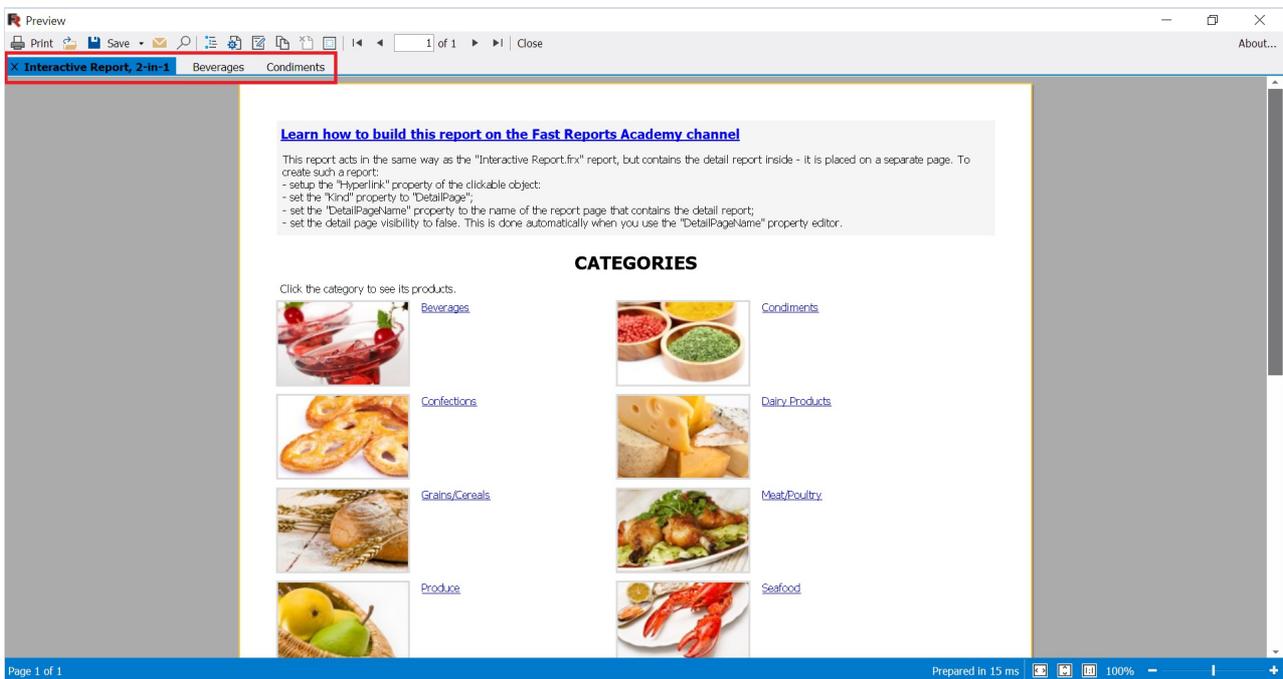
You can see three open tabs here. Previously, only the active tab was exported. Now you can export all tabs to one file using the new "Export all tabs" option.
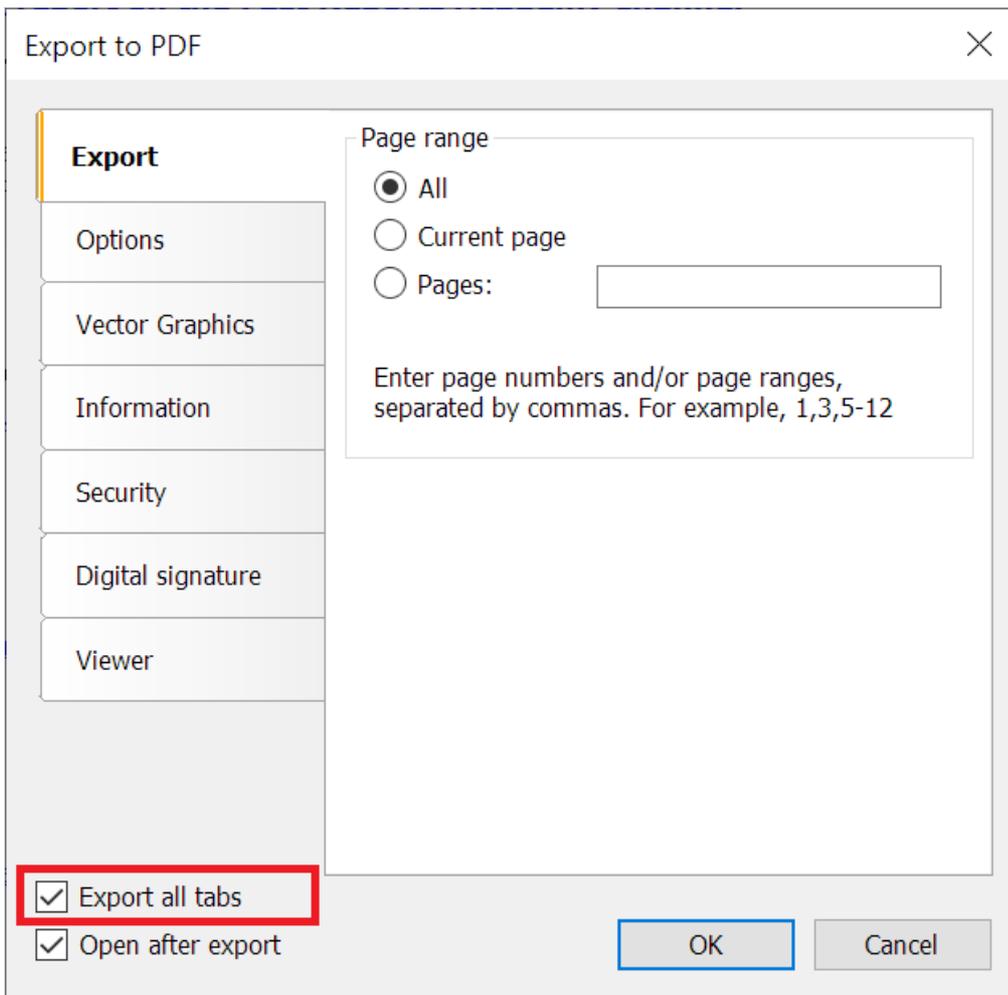
Detailed description of referenced assemblies and installed plugins

Now when you hover your mouse over a dll in the plugins list (Options -> Plugins) and in the list of links to builds (Report -> Options -> Script), detailed information with description, version, size, creation date, etc. is displayed.

Exports improvements

PDF export improvements:

Linux version:

- support for complex languages (Arabic, Hebrew etc.) in Skia version;

All version:

- support for Font Fallback (automatic font selection mechanism for displaying characters that are not supported by the current font);
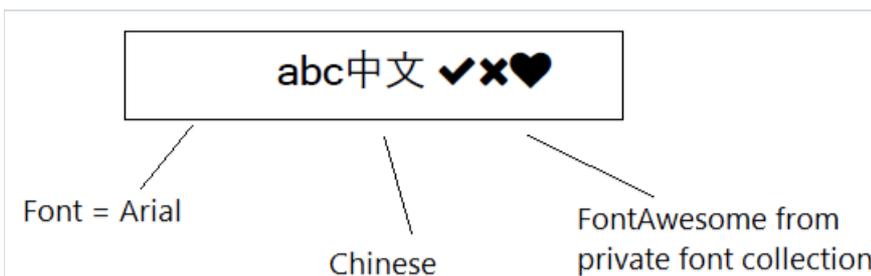
- accurate positioning of special characters such as vowels and accents;

UseFileStream property in PDF export

A new UseFileStream option has been added for PDF export. It can only be used when exporting from code to file. This option is useful when exporting reports with a large number of pages (several tens of thousands) in multiple threads. It allows you to avoid memory shortage errors. In other cases, it does not make much sense to use it. Example:

```
Report report = new Report();
PDFExport export = new PDFExport();
export.UseFileStream = true;
report.Export(export, "report.pdf");
```

Export of locale in Word, PowerPoint, Rich Text, OpenOffice Write and OpenOffice Calc exports

You can now select the language of the document in these exports. By default the language selected in the designer is used.

Also added option "Show Gridlines" when exporting to Excel 2007.

Complete list of changes

[Engine]

- implemented converter reports of StimulSoft;
- added changing name of JSON data source in expressions when it's renamed;

- added converting of PaperSize property when converting reports from StimulSoft;
- added checking existence of referenced assembly when converting reports from StimulSoft;
- added PrintOnParent property to Table and Matrix objects;
- added loading of report parameters when converting reports from RDL;
- added loading of subreports when converting reports from RDL;
- added the feature to store JSON connection data using the StoreData property;
- optimized speed in reports containing large amount of objects;

- changed exception text when calculating and formatting expression if e.InnerException is null;
- when loading RDL report, page width will be equal section width in case when there is no page width;

- fixed length calculation encoding DataMatrix C40 and text;
- handled System.ComponentModel.Win32Exception when printing with disabled Print Spooler;
- fixed hide border of picture when printing with auto size;
- fixed stack overflow error when prepare report with child band of page footer and then start new page option enabled for it;
- fixed a bug with not passing path of base report to current one in Unix OS;
- fixed a bug with creating subreport and page with the same name when converting reports from StimulSoft;
- fixed a bug with invalid names when converting reports from StimulSoft;
- fixed a bug with TotalPages in Page.VisibleExpression that causes an exception when double pass is disabled;
- fixed a bug when band can grow out of page;
- fixed a bug when objects can grow out of band or ContainerObject;
- fixed "back indent" feature in RTF translator;
- fixed RichText line spacing when RTF translated to report objects;
- fixed an error with ConnectionString property in JsonDataSourceConnectionStringBuilder class when value was without a request headers;

[Designer]

- added the report validator that helps to find invalid objects (duplicate names, negative sizes, etc.);
- added editor for RichObject.Text property;
- added FRX editor in report designer;
- added detailed description of referenced assemblies and installed plugins;
- added the ability to copy dialog pages;
- added the ability to delete dialog pages using the context menu;
- added ability to disable using of last formatting options when creating objects;
- added integration with FastReport.Id;
- added call to online-documentation in the report designer;
- added wizard for visualization of control identification signs;
- add tooltip about right and bottom indents for guides and objects in designer;
- added ability to select color of backlight intersecting objects in designer;
- added possibility to connect bases of Access 2007;

- changed the look of ElasticSearch connection editor form;
- changed the text fields in CISWizardForm with units to text fields that only support numbers;

- fixed a bug leading to System.NullRefereceException when creating calculated column for subtable JSON;
- fixed a bug leading to System.FormatException when drawing labels of maps;
- fixed a bug leading to the System.NullReferenceException when clicking the "Paste" button in the context menu of dialog pages;
- fixed a bug with scaling zoom controls of designer in HiDPI mode when run from old demo application;
- fixed opening form of save changes after save all report;
- fixed unscalable items in welcome window;
- fixed backlighting intersected charts;
- fixed exception on rename JSON table;

- fixed UpdateStatusBar in DialogWorkspace;
- fixed a bug with localization of "Account..." button in menu "File";
- fixed canceling selection of object if its properties are changed;
- fixed a bug when switching to the "Code" page did not occur after adding an event handler;

[Preview]

- implemented export of all open tabs;

- fixed a bug leading to System.NullReferenceExteption when preparing report with RichObject on system without printers;
- fixed a bug in the MSChart object in HiDPI mode;

[Exports]

- added export of locale in Word, PowerPoint, Rich Text, OpenOffice Write and OpenOffice Calc exports;
- added encryption of the password of the digital signature certificate in PDF-export when it is saved;
- added option "Show gridlines" when exporting to Excel 2007;
- added data types export to DBF;
- added a new property to the SVG export PrefixStyle, which allows you to set a prefix for all styles inside the SVG export;
- added option "Use locale formatting of data" when exporting to Excel 2007;
- added PDFExport.UseFileStream property, which allows to export huge reports on systems with low amount of RAM without System.OutOfMemoryException;

- set UTF-8 as default encoding in DBF export;

- fixed incorrect scaling pictures in layered HTML-export when enabled high quality SVG and zoom more than 1;

- fixed a bug leading to System.IndexOutOfRangeException when exporting font without kerning to PDF;

- fixed a bug with scaling picture in layered HTML-export;

- fixed a bug leading to System.NullReferenceException when exporting report with empty page to Word 2007;

- fixed memory leak in PDF export with some CJK fonts;

- fixed a bug when SVG picture was not rotated to needed angle in HTML/Blazor export;

- fixed repeated rendering of table cell in SVG export;

- fixed incorrect pageStyle when printing from browser for table HTML export;

- fixed exception when export object with negative size in HTML export;

- fixed export to pdf if Compressed = false;

- fixed incorrect record of border-collapse property in table HTML-export;

- fixed a bug in Excel-export, when the fill in the output file did not change the first time;

- fixed export of watermark to PostScript;

- fixed error of font scale when export to PDF;

- fixed a bug where a text object with HtmlTags exported to RTF was not modified by the
  ‚ ' tags;

[WebReport]

- onlineDesigner properties are moved to webReport.Designer with backwards compatibility;

- fixed a bug when event "CheckedChanged" handled by RadioButton was not performed;
- fixed incorrect scaling of Dialog components in Blazor;
- fixed a bug with incorrect font size in Excel export;
- fixed a bug in Blazor when font of text object with property TextRenderType = HtmlParagraph was always default;

[.NET Core]
- fixed incorrect search for public-methods in report script;
- fixed problem of creating a fontlist file on Azure;

[CoreWin]
- fixed behavior of WinForms components in Toolbox for Visual Studio (Design-Time);
- fixed incorrect launch of the browser when clicking on links in CoreWin;
- for FastReport.CoreWin, reports with a script that use the WinForms API have been fixed;

[Demos]
- added the ability to change the localization of a new demo application without restarting it;
- added demo on React with FastReport.Core;

- fixed position of one chart in Chart.frx;

[Plugins]
- implemented connection to Cassandra;

- updated RPTImportPlugin;

[Extras]
- added FastReprot.Web (only for .NET Framework) and FastReport.VSDesign libraries for FastReport.Net* packages;
- added an option to import reports using streams;

[Service]
- fixed incorrect version of FastReport.Compat in FastReport.Net packages.
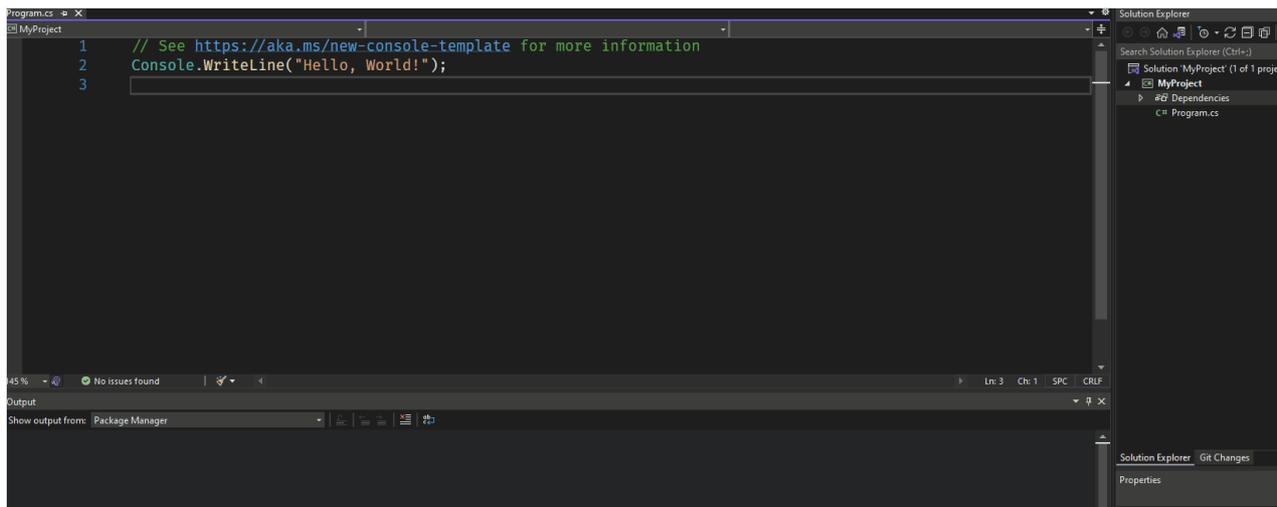
FastReport .NET 2022.2
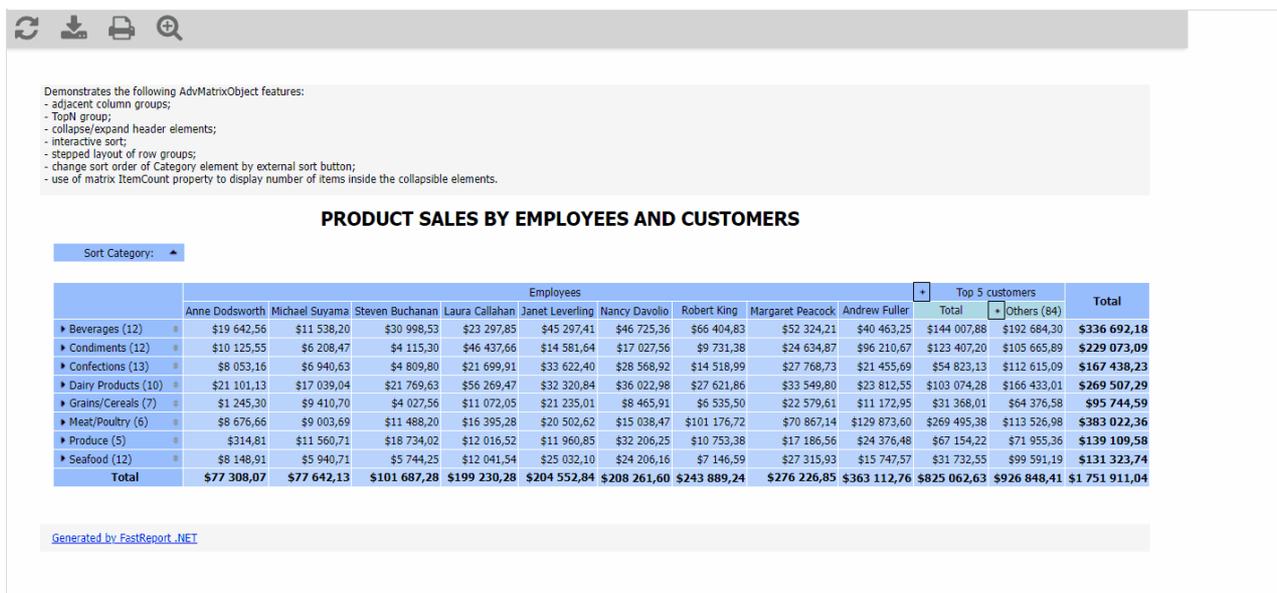
New features

Fast Reports NuGet-server

We recently launched our own NuGet-server - a repository of licensed Fast Reports products for users. Now you can conveniently download the latest versions of our components on any operating system.

More details in article.

Added support for interactivity of the "Advanced Matrix" object in WebReport:

All WebReports (.NET Framework, .NET Core, Blazor Server) now have interactivity support for the new Advanced Matrix. The collapse and sort buttons now work correctly in the browser.

Ability to save the report with random data

The file menu in the report designer has a new item "Save with random data...". When you save the report in this way, all data sources will be saved in the report and the data in them will be replaced with random values.

Backlight of intersecting objects in the designer

Now, when you place objects on the report page, the color will highlight the intersecting objects and objects

outside of bands and page. In a correct report, there should not be such objects. Ignoring to comply with this recommendation, can lead to a number of problems in the preparation and exporting of reports. By default, this option is disabled.

You can enable it in the designer settings (menu File->Options).
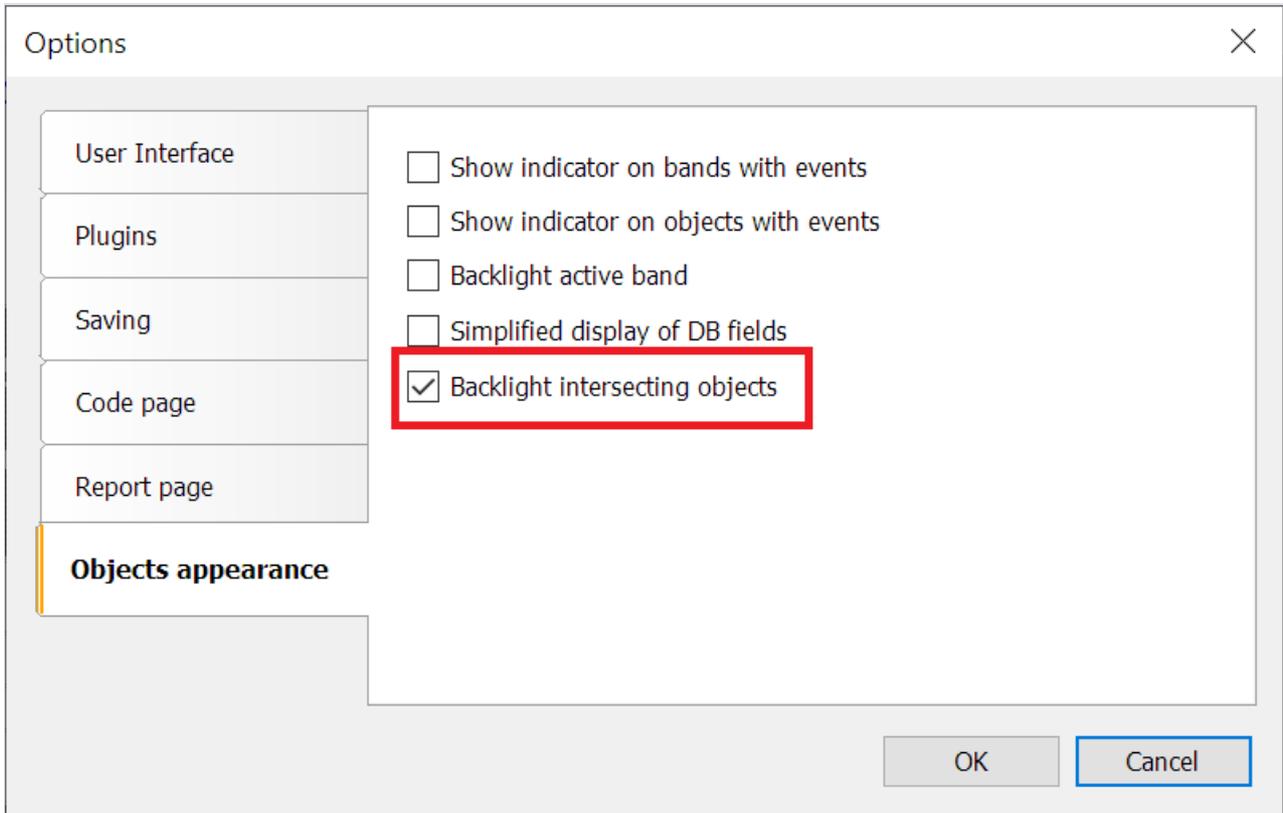
Backlight demo below:
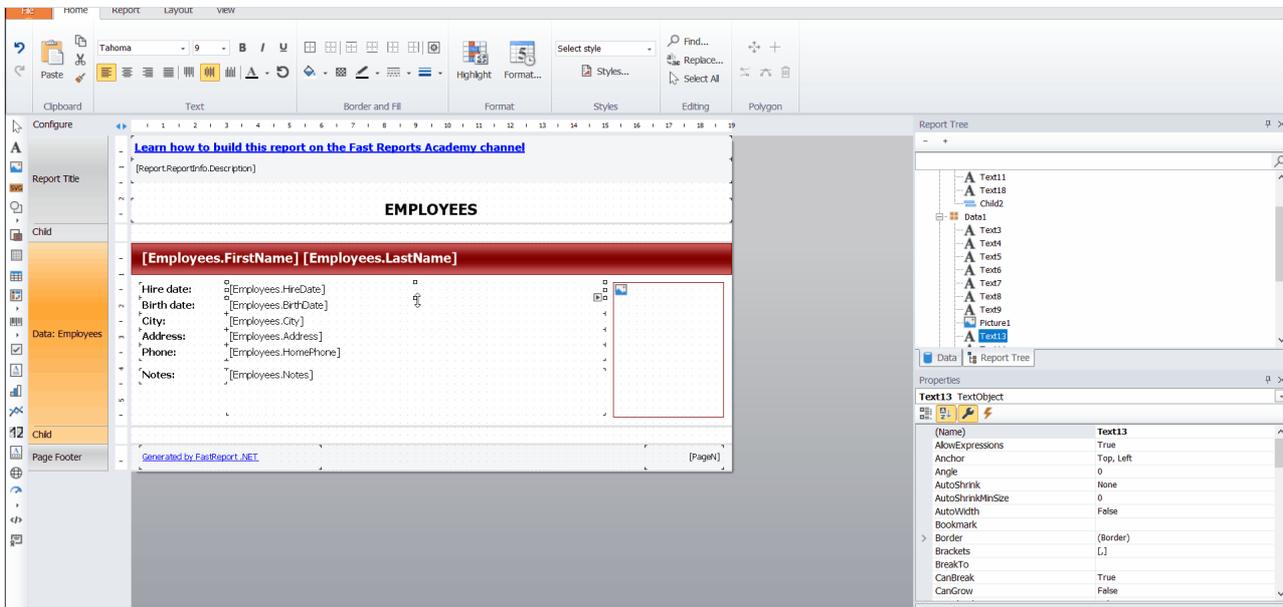
Ruler with guides in the RichObject editor

The new tool allows you to conveniently adjust indentation and tab positions when editing RichObject. Read more in article.

A console RTF conversion utility has been added

With it, you can conveniently convert RTF files into report templates.

Ability to use XLSX files as data sources

You can now retrieve data from Excel 2007 files as from a database and use it in a report.

Read more in article.

Exports Improvements

"Pinned cells" option when exporting to Excel 2007

This feature, allows you to define an area of the sheet that will always be visible when you scroll. You can lock: the first row, the first column, a certain number of rows and columns.

The settings look like this:

And the way this option works is:

ZPL II support when exporting to ZPL

"High quality SVG" option in HTML export

With this setting enabled, the quality of SVG images will be significantly higher when exporting. Note, however, that the amount of memory occupied will also be higher.

Added export of bookmarks and internal links into Word

Implemented export tab character width in PDF, Word, HTML and RTF

New PrefixStyle property in SVG export

This property allows you to set a prefix for all styles when exporting to SVG.

Added export of number and date formats in Excel 97

Export of tab character width in PDF, Word, HTML and RTF

Complete list of changes

[Engine]

- added ability to save report with random data;

- the ExportBand method now uses the BandBase argument instead of Base;

- fixed bugs with double calling events AfterData, BeforePrint and AfterPrint of ContainerObject;
- fixed a bug leading to System.NullReferenceException when running reports with dialog forms;
- fixed a bug with not working VisibleExpression property of subreports and pages;
- fixed a bug with vertical shift of non-intersecting objects when converting RTF;
- fixed a bug with right anchor on pages with unlimited width and landscape orientation;
- fixed translation of lists when converting RTF;
- fixed a bug with not working RichObject.AllowExpressions property;
- fixed a bug leading to System.OverflowException when drawing unlimited page without preparing;

[Designer]

- added backlight of intersecting objects;
- added ruler with guides in RichObject editor;
- replaced password symbols on dots in object inspector;
- added warning about possible stack overflow when putting Matrix or AdvMatrix on repeated bands;

- removed error message when text of barcode consist expression;
- fixed a bug with disable hot keys option;
- fixed dropdown menu when click on LineStyle and LineWidth button;
- fixed a bug with viewing data in designer;
- fixed bugs leading to System.NullReferenceException when dragging objects into AdvMatrix;
- fixed a bug with incorrect showing settings of shadow in border editor;

[Preview]

- fixed a bug leading to System.NullReferenceException when clicking on editable TextObject;
- fixed a bug with not working hyperlinks in report with multi-column databands;
- fixed a bug when exporting a report resulted to saving the prepared report;
- fixed a bug with setting lists of available exports and exports to clouds in PreviewControl;

[Exports]

- added export to ZPL II;
- added option "High Quality SVG" in export to HTML;
- added option "Pinned cells" in export to Excel 2007;
- added ability to scale print in export to Excel 2007;
- added export of bookmarks and inner hyperlinks to Word;
- added export of numbers and dates format to Excel 97;
- added encryption of personal data in Email-export;
- added indent of RichObject in export to RTF;
- added line break of RichObject in export to RTF;
- added indent of TextObject when exporting to Word;
- added export of tab width in PDF, Word, HTML and RTF exports;

- added property PrefixStyle to SVG-export, which allows to set a prefix for all styles;

- improved export of RichObject to Excel 2007;
- removed FastReport Cloud and XMPP exports;

- fixed incorrect rotation of landscape orientation of pages when printing HTML if they used styles from previous pages;
- fixed a bug with font scale when export to PDF;
- fixed a memory lose when export SVG objects to HTML with option "High Quality SVG";
- fixed a bug with embedding fonts for which packing is prohibited in PDF-export;
- fixed a bug with exporting tab symbols to Word;
- fixed fill background picture and property of line-height in export to HTML;
- fixed a bug with exporting custom dash line of SVGObject to PDF;
- fixed a bug with exporting borders of spanned cells to SVG;

[WebReport]

- added interactivity for advanced matrix in WebReport;

- fixed closing canceling processing in OnFormClosing in Core web dialogs;

[.NET Core]

- fixed a bug with not working "open after export" option;

[WebReport Core]

- now the DatePicker icon looks the same in all browsers;

[Demos]

- added a new demo for Blazor with a demonstration of working with two reports;

- fixed a bug due to which the cursor did not change when hovering over links in the new demo;

- fixed a bug with AdvMatrix in new demo;

[Plugins]

- added connection to Excel

- fixed SQLite connector for FastReport.Core, FastReport.CoreWin and FastReport.OpenSource;

- fixed a bug with ConnectionString to Firebird;

[Extras]

- added tool for conversion of RTF documents to report templates (\Extras\Misc\rtf2frx).

FastReport .NET 2022.1

New features

Added new "Advanced Matrix" object:

	Anne Dodsworth	Michael Suyama	Steven Buchanan	Laura Callahan	Janet Leverling	Nancy Davolio	Robert King	Margaret Peacock	Andrew Fuller	Total
▼ Beverages (12)	$19 642,56	$11 538,20	$30 998,53	$23 297,85	$45 297,41	$46 725,36	$66 404,83	$52 324,21	$40 463,25	**$336 692,18**
1. Chai	3%	13%	3%	5%	4%	2%	1%	7%	4%	**$12 788,10**
2. Chang	3%	16%	3%	5%	3%	7%	2%	9%	3%	**$16 374,96**
3. Chartreuse verte	1%	39%	69%	29%	5%	4%	0%	7%	4%	**$42 732,54**
4. Côte de Blaye	74%		20%		56%	50%	30%	56%	62%	**$144 728,74**
5. Guaraná Fantástica	1%	5%	1%	4%	1%	2%	1%	0%	1%	**$4 504,37**
6. Ipoh Coffee				12%	14%	14%	3%	4%	9%	**$23 526,70**
7. Lakkalikööri	5%	7%	2%	15%	7%	8%	1%	3%	3%	**$15 760,44**
8. Laughing Lumberjack Lager				1%	0%		0%	1%	2%	**$2 396,80**
9. Outback Lager	4%	5%		6%	3%	3%	2%	6%	3%	**$10 672,65**
10. Rhönbräu Klosterbier	4%	8%	1%	6%	3%	3%	53%	2%	2%	**$43 212,49**
11. Sasquatch Ale	1%	4%	0%	1%	2%		5%	1%	2%	**$6 350,40**
12. Steeleye Stout	4%	4%	0%	17%	1%	6%	2%	3%	5%	**$13 644,00**
▶ Condiments (12)	$10 125,55	$6 208,47	$4 115,30	$46 437,66	$14 581,64	$17 027,56	$9 731,38	$24 634,87	$96 210,67	**$229 073,09**
▶ Confections (13)	$8 053,16	$6 940,63	$4 809,80	$21 699,91	$33 622,40	$28 568,92	$14 518,99	$27 768,73	$21 455,69	**$167 438,23**
▶ Dairy Products (10)	$21 101,13	$17 039,04	$21 769,63	$56 269,47	$32 320,84	$36 022,98	$27 621,86	$33 549,80	$23 812,55	**$269 507,29**
▶ Grains/Cereals (7)	$1 245,30	$9 410,70	$4 027,56	$11 072,05	$21 235,01	$8 465,91	$6 535,50	$22 579,61	$11 172,95	**$95 744,59**
▶ Meat/Poultry (6)	$8 676,66	$9 003,69	$11 488,20	$16 395,28	$20 502,62	$15 038,47	$101 176,72	$70 867,14	$129 873,60	**$383 022,36**
▶ Produce (5)	$314,81	$11 560,71	$18 734,02	$12 016,52	$11 960,85	$32 206,25	$10 753,38	$17 186,56	$24 376,48	**$139 109,58**
▶ Seafood (12)	$8 148,91	$5 940,71	$5 744,25	$12 041,54	$25 032,10	$24 206,16	$7 146,59	$27 315,93	$15 747,57	**$131 323,74**
Total	$77 308,07	$77 642,13	$101 687,28	$199 230,28	$204 552,84	$208 261,60	$243 889,24	$276 226,85	$363 112,76	**$1 751 911,04**

Here is a list of its key features:

- row and column headers can contain groups and simple elements in any order. This allows you to build asymmetric reports;
- collapse buttons allow you to interactively manage the visibility of individual elements;
- sorting buttons allow you to interactively sort the matrix by the selected values, including the total values;
- Top N grouping allows you to display N values in the header, and group the remaining values into a separate element with the ability to expand;
- output of matrix headers in a stepped form;
- sorting headers by total values;
- a wide range of aggregate functions;
- support of custom aggregate functions;
- a wide range of special functions that allow you to get the values of totals, adjacent cells, as well as functions for calculating percentages;
- support for "Sparkline" and "Gauge" objects in data cells.

Expand/collapse matrix elements

TopN group with ability to expand/collapse its elements

Block or stepped layout

| | Top 5 customers | | Total |
	Total	⊞ Others (84)	
▸ Beverages (12) ⇕	$144 007,88	$192 684,30	$336 692,18
▸ Condiments (12) ⇕	$123 407,20	$105 665,89	$229 073,09
▸ Confections (13) ⇕	$54 823,13	$112 615,09	$167 438,23
▸ Dairy Products (10) ⇕	$103 074,28	$166 433,01	$269 507,29
▸ Grains/Cereals (7) ⇕	$31 368,01	$64 376,58	$95 744,59
▾ Meat/Poultry (6) ⇕	$269 495,38	$113 526,98	$383 022,36
1. Alice Mutton	$13 428,48	$19 269,90	$32 698,38
2. Mishi Kobe Niku	$1 319,20	$5 907,30	$7 226,50
3. Pâté chinois	$7 137,60	$10 288,80	$17 426,40
4. Perth Pasties	$6 916,56	$13 657,61	$20 574,17
5. Thüringer Rostbratwurst	$239 795,40	$60 573,28	$300 368,67
6. Tourtière	$898,14	$3 830,10	$4 728,24
▸ Produce (5) ⇕	$67 154,22	$71 955,36	$139 109,58
▸ Seafood (12) ⇕	$31 732,55	$99 591,19	$131 323,74
Total	$825 062,63	$926 848,41	$1 751 911,04

Sort elements by the total value

Matrix header can contain different types of elements

	2011	2012	2013	2014	2015	2012 to 2011 change	2012 to 2011 %
Andrew Fuller	3900	2100			1800	-1800	53,85%
Janet Leverling	6100	3200				-2900	52,46%
Nancy Davolio	3300	2700	3100		1700	-600	81,82%
Steven Buchanan			3999	8100			
Total	13300	8000	7099	8100	3500	-5300	60,15%

Special functions to get other cell's value

Learn more about this object in the documentation.

Added GS1 DataBar barcodes: Limited, Omnidirectional, Stacked and Stacked Omnidirectional.

GS1 DataBar Omnidirectional
(01)12345678901231

GS1 DataBar Limited
(01)12345678901231

GS1 DataBar Stacked Omnidirectional
(01)12345678901231

GS1 DataBar Stacked
(01)12345678901231

New properties: Config.CompilerSetting.ExceptionBehaviour and Config.CompilerSetting.Placeholder.

These properties allow you to customize the behavior when exceptions with invalid database field and table names occur.

Config.CompilerSetting.Placeholder is a string variable that is used to replace expressions with nonexistent names. By default, the value of this variable is an empty string.

Config.CompilerSetting.ExceptionBehaviour can have the following values: **ExceptionBehaviour.Default** - default behavior, as it was before. If there are errors with invalid names, an error message is displayed. Report preparation is interrupted. **ExceptionBehaviour.ReplaceExpressionWithExceptionMessage** - invalid expressions are replaced by the text of the exception message. Errors are not shown at that. Report preparation is not interrupted. **ExceptionBehaviour.ShowExceptionMessage** - A message appears with the exception text, after pressing **OK**, report preparation continues. Incorrect expressions are replaced with the value of **Placeholder** variable. **ExceptionBehaviour.ReplaceExpressionWithPlaceholder** - invalid expressions are simply replaced with **Placeholder**. No error messages. Report preparation is not interrupted.

Example with variable values: **ExceptionBehaviour = ExceptionBehaviour.ReplaceExpressionWithPlaceholder Placeholder = "NO DATA!**
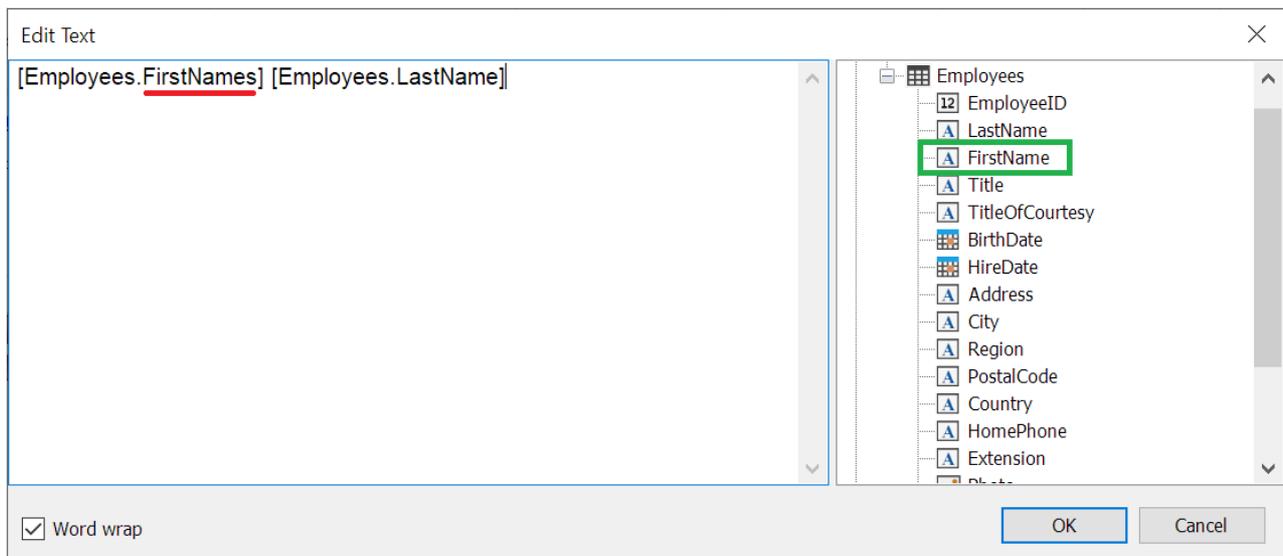

Here you can see that the table has a field named **FistName**, but it's not specified correctly in the expression.

EMPLOYEES

NO DATA! Fuller

Hire date:	14.08.2009
Birth date:	19 февраля 1972 г.
City:	Tacoma
Address:	908 W. Capital Way
Phone:	(206) 555-9482
Notes:	Andrew received his BTS commercial in 1994 and a Ph.D. in international marketing from the University of Dallas in 2001. He is fluent in French and Italian and reads German. He joined the company as a sales representative, was promoted to sales manager in January 2009 and to vice president of sales in March 2010. Andrew is a member of the Sales Management Roundtable, the Seattle Chamber of Commerce, and the Pacific Rim Importers Association.
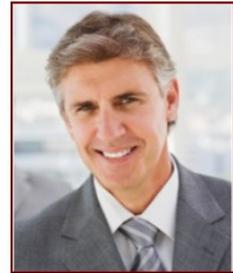
NO DATA! Dodsworth

Hire date:	15.11.2011
Birth date:	27 января 1986 г.
City:	London
Address:	7 Houndstooth Rd.
Phone:	(71) 555-4444
Notes:	Anne has a BA degree in English from St. Lawrence College. She is fluent in French and German.

And this is the result of preparing such a report. Previously it would have been impossible to prepare it due to errors.

Improved translation quality of RTF into report objects.

Conversion of RTF into report objects is optimized. RTF translation in table cells is added. Lots of bugs fixed.

Exports improvements

Implemented export of watermark to Word and RTF.

Added SVG image scaling in export matrix.

This improves the quality of exported images when exporting to Word and Excel. However, this increases the size of the output file. To use this feature, you must enable the "Print optimized" option when exporting.

Export groups to single sheets in Excel 2007 has been implemented.

Excel 2007 has added the ability to export a property that determines the size and location of the image when exporting.

Now you can define how the image will behave in a cell when its position and size are changed. In doing so, the image can:

- move and resize together with the cell
- move together with the cell, but not change its size
- don't move or resize

Implemented the ability to hide or show grid lines when exporting to Excel 97.

Added "Don't rotate landscape pages when printing" option in HTML export.

Previously, we were forcibly rotating landscape-oriented pages when printing. The reason was that browsers

cannot correctly print reports with pages both in portrait and landscape orientation. When you print such documents, pages with landscape orientation are cut off by the width of pages with portrait orientation. Now, you can adjust whether to rotate pages in landscape orientation or not. In addition, a bug where landscape-oriented pages were always rotated, even when there are no portrait-oriented pages, has been fixed.

.NET 6 support

Added .NET 6 support for FastReport.Core and FastReport.CoreWin.

## Improvements to WebReport for Core and Blazor Server

Export settings

Added the ability to configure the export of a report in the required format from WebReport. When the **webReport.Toolbar.Exports.EnableSettings** property is activated, the settings icon will appear next to the required export button. Clicking on this button will display a window with a list of possible settings (exported pages, properties, etc.).
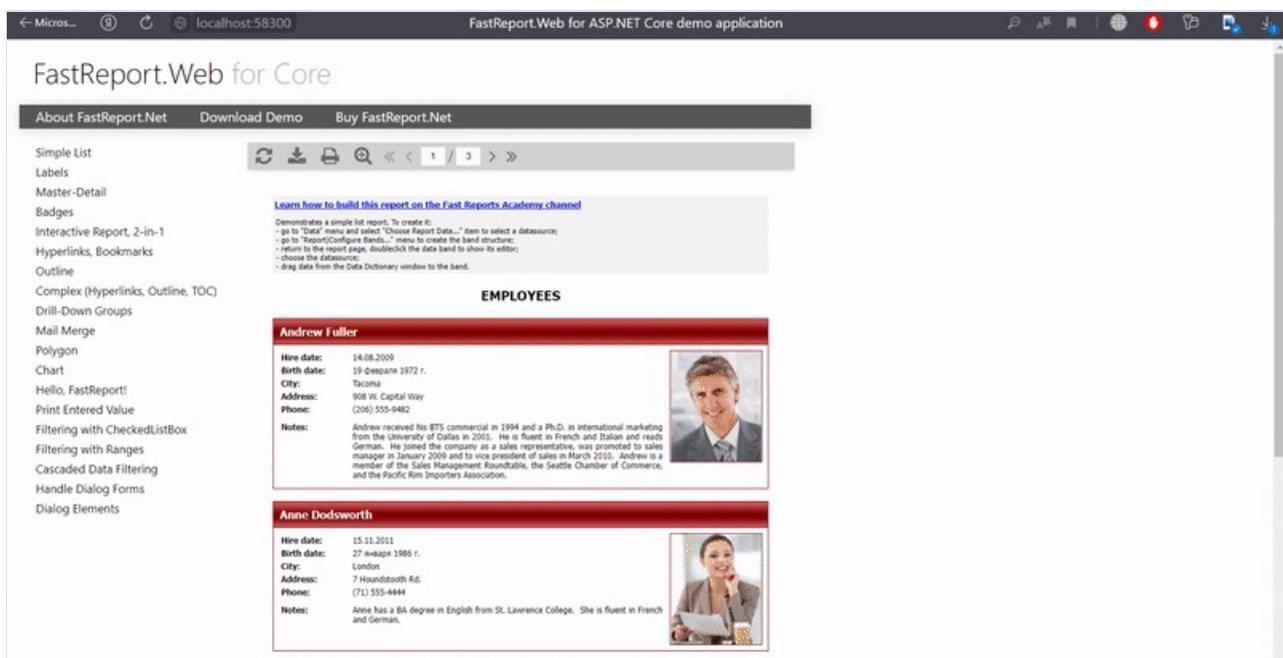
The export settings window can be customized using the **webReport.Toolbar.Exports.Color** and **webReport.Toolbar.Exports.FontSettings** properties.

Separating different report pages into bookmarks

Added the ability to open different ReportPage pages in different WebReport tabs. To activate this feature, you need to enable the **webReport.SplitReportPagesInTabs** option.
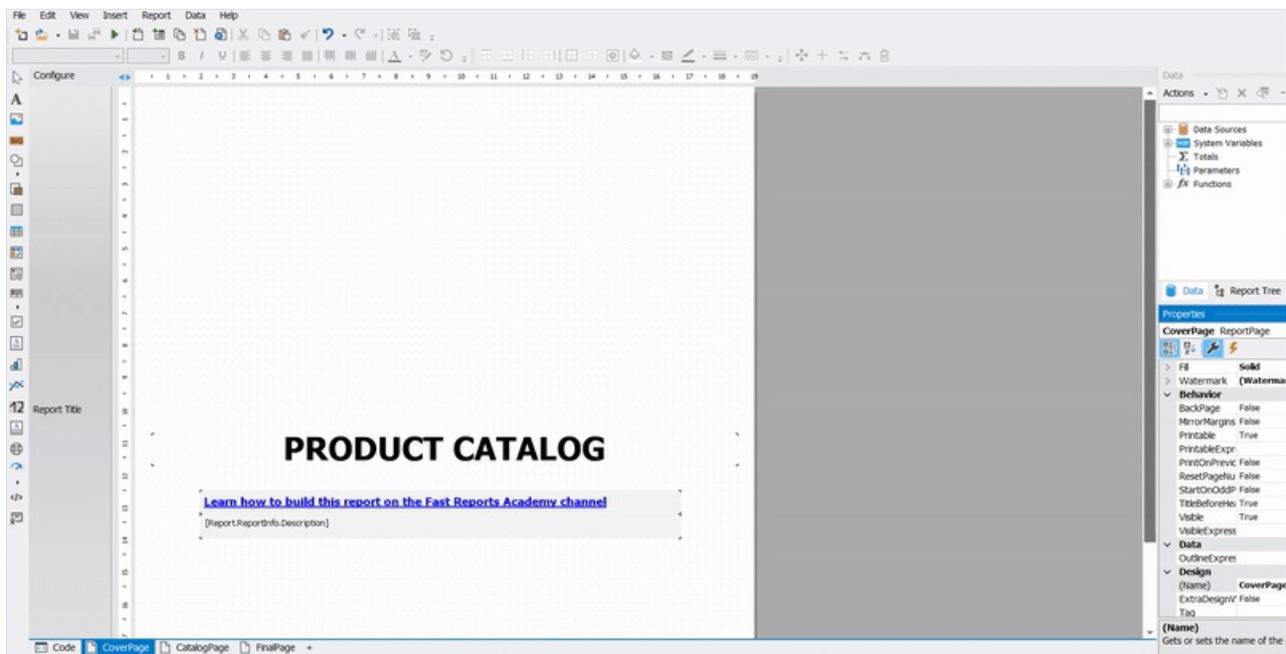
Static Styles in WebReport

To be able to override the standard styles of the toolbar, outline and other elements for your own customization, static class names have been added. These include: `fr-toolbar` , `fr-toolbar-item` , `fr-toolbar-narrow` , `fr-toolbar-dropdown-content` , `fr-toolbar-zoom-selected` , `fr-toolbar-pointer` , `fr-toolbar-notbutton` , `fr-toolbar-slash` .

Complete list of changes

[Engine]

- added a new AdvMatrixObject
- added GS1 DataBar barcodes: Limited, Omnidirectional, Stacked and Stacked Omnidirectional
- added new properties: Config.CompilerSetting.ExceptionBehaviour and Config.CompilerSetting.Placeholder. These properties give the ability to customize the behavior when exceptions are thrown with incorrect names of database tables and fields.
- added translation of RichObject inside TableCell

- reworked translation of RichObject into report objects

- fixed ShiftMode of translated RTF object
- fixed a bug with two parameters with the same name in report leading to System.ArgumentException
- fixed a bug with subreport containing multicolumn Databand
- fixed a bug with wrong band height calculation
- fixed a bug with displaying of hyperlinks when converting RTF to report objects
- fixed translation of RichObject if it set from a report script
- fixed a bug with private fonts added to Config.PrivateFontCollection

[.Net Core]

- added support for .NET 6

- fixed incorrect search for Bold-Italic fonts

[Designer]

- added verification of entered data in editing window of the QR code of SberBank

- fixed a bug with line break in text object editor
- fixed a bug when converting rdl reports containing matrices inside table cells

- fixed a bug with guide lines in the designer
- fixed a bug with Report tree window
- fixed a bug leading to System.NullReferenceException and crash of the designer during its launch when the Auto Guides option is enabled

[Preview]

- fixed a bug with shifting the position of objects when switching the view of bands while editing a prepared page

[Exports]

- implemented export of watermark to Word
- implemented export of watermark to RTF
- added "Don't rotate landscape pages when printing" option in export to HTML
- added the ability to change the name of the attached file when sending by Email
- add zooming of SVG images in export matrix
- added the ability to export a property that determines the size and position of the image when exporting to Excel 2007
- implemented ability to hide or show gridlines when exporting to Excel 97
- implemented export of groups on separate sheets to Excel
- implemented export of transparency level watermark images to Word
- implemented export image size of watermark to RTF

- fixed a bug leading to System.NullReferenceException when exporting to text, tables with rows count less then one
- fixed incorrect left padding of tables in export to Word
- fixed a bug with Wingdings font in HTML tags when exporting to HTML
- fixed a bug with export Wingdings and Webdings fonts to HTML
- fixed a bug with width of frame in export to PowerPoint
- fixed a bug with exporting objects with transparent fill to RTF
- fixed a bug with exporting objects with transparent fill to Word
- fixed a bug leading to System.OutOfMemoryException when exporting to PDF
- fixed incorrect line break display when exporting to HTML
- fix out of memory when export to PDF
- fixed bugs in export to PDF in non-Windows systems
- fixed a bug with exporting tables with more than 63 columns to Word 2007
- fixed a bug leading to a memory leak and System.OutOfMemoryException in PDF-export when the "Text in curves" option is enabled
- fixed a bug with line break in HTML-export

[WebReport]

- fixed a bug with new line character when using Wingdings font

[WebReport Core / Blazor Server]

- added the ability to configure the properties of exporting a report from WebReport. When the webReport.Toolbar.Exports.EnableSettings property is activated, the settings icon will appear next to the required export button
- added property webReport.SplitReportPagesInTabs, which allows you to split different ReportPage-s in different tabs of WebReport
- added static class names for the ability to override the standard styles of toolbar, outline and other elements

- fixed updating WebReport when entering a value from the keyboard in the DateTimePicker field
- fixed the width of tabs with non-standard sizes of the report page

[Demos]

- added demo of using WebReport Core for .NET 5
- added demo of using WebReport Core for Angular
- added demo of using WebReport Blazor for Blazor Server

FastReport .NET 2021.4

Editions changes

The WinForms and Win+WebForms editions are gone. Instead, a new Standard edition has been added that includes Windows Forms components, ASP.NET components, and .NET Core / .NET 5 / Blazor support.

New features

- Added new Visual Studio-styled icons. You may switch between icon packs in the "View/Options/User Interface" window (or, "File/Options/User Interface" if you use ribbon UI):
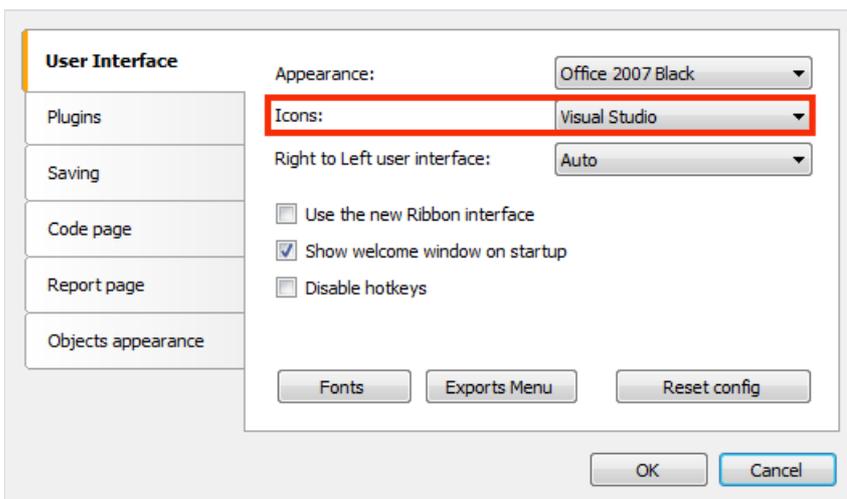
The new icons are well suited for hiDPI screens.

- Added ability to reset the FastReport configuration stored in the `FastReport.Config` file in the user's profile. It can be done in the "View/Options/User Interface" window, the "Reset config" button. Restart of the designer is required in order to apply changes.

- Added ability for the Text object to display DB filed names in a simplified form when designing a report. You can activate this option in the "View/Options/Objects appearance" window. This option is off by default. When you turn it on the Text object with a single DB field will display the field name part only, with no datasource name:

It makes the report look cleaner especially when it contains a lot of small objects. You still can see a full text of object in the status bar.

- Added ability to set up each cell in the Matrix object's corner area. To do this use the cell's context menu and its commands "Split cell", "Merge cells":
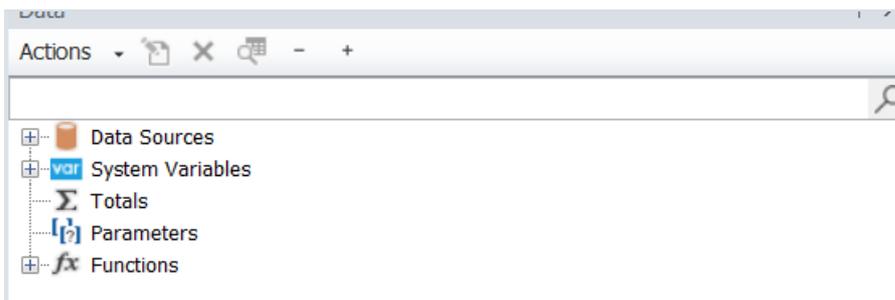
- Added ability to connect to ElasticSearch. Connection available in data wizard and from code.

- Added barcode Japanese Post 4 - State Code.

- Added the collapse all/expand all buttons and, a search field for the report tree and data tree in the designer. When clicking on `+` tree will expand. On `-` tree will collapse.

These changes should simplify working with reports containing many objects and/or data sources.

- The RicthText to report objects converter has been significantly improved and optimized.

- The number of available exports in WebReport Core/Blazor Server has increased significantly.

- Added FastReport Business Graphics integration objects (\Extras\Objects\FastReportBGObjects).

Universal plugins FastReport.Data

Packages with FastReport.Data plugins have been updated. Now they include plugins for different FastReport editions (.NET, Core, CoreWin, OpenSource) and automatically connect the necessary library, depending on the product you use. The FastReport edition 2021.3.0 or higher is required for correct work.

The FastReport.Core.Data, FastReport.CoreWin.Data, and FastReport.OpenSource.Data plugins are declared obsolete and are no longer supported.

Improvements in publishing user applications using FastReport

For user applications on .NET Core 3.0+ and .NET 5+ using FastReport.Core, FastReport.CoreWin, FastReport.OpenSource has been added Single File Applications (SFA) support.

Also, added support for publishing application with trimmed unused libraries - MSBuild property - PublishTrimmed*.

Warning! In some cases you may need to explicitly specify the list of builds that .NET should not trim. This may be useful if your report script uses these libraries, but your application's code does not make use of them.

This is done using the TrimmerRootAssembly property. In this case, for example, it's explicitly stated that the System.Security library doesn't need to be trimmed:

```
<ItemGroup>
  <TrimmerRootAssembly Include="System.Security" />
</ItemGroup>
```

Localizations

In the logic of the localization change small changes were made.

1. Added package **FastReport.Localization**. This package contains localization files for FastReport.NET, FastReport.Core, FastReport.CoreWin, FastReport.Mono, FastReport.OpenSource products and creates the Localization directory in the output directory of the user project when adding this package.

2. Added new API for changing the localization using the CultureInfo type - *FastReport.Utils.Res.LoadLocale(CultureInfo culture)*.

   When this method is called, FastReport searches for the appropriate localization for the selected culture. Loaded locales are cached. For this method to work correctly, you must install the FastReport.Localization package from step 1 in your project or set the path to the folder with the localization files in the *FastReport.Utils.Res.LocaleFolder* property.

Changes and improvements in the WebReport Core/Blazor toolbar

- Toolbar settings were moved from WebReport class to WebReport.Toolbar property of ToolbarSettings class.
- Added toolbar settings: Position, color of dropdown menu, font, transparency of icons, changing icon color, changing content position. These properties are available in the webReport.Toolbar.
- During report loading, the toolbar is no longer displayed.
- Added the ShowOnDialogPage property to the Toolbar object (true by default), which will allow to turn off the toolbar rendering if a dialog window is currently open
- Added more exports to the toolbar dropdown menu. These properties are available in webReport.Toolbar.Exports.ExportTypes. List of added exports: HTML, Hpgl, Dxf, Json, LaTeX, Ppml, PS, Xaml, Zpl, Excel97, Svg.

```
ToolbarSettings toolbar = new ToolbarSettings()
{
    Color = Color.LightBlue,
    DropDownMenuColor = Color.LightBlue,
    ShowOnDialogPage = false,
    DropDownMenuTextColor = Color.Black,
    IconColor = IconColors.Black,
    Position = Positions.Right,
    FontSettings = new Font("Arial", 14, FontStyle.Bold),
    Exports = new ExportMenuSettings()
    {
        ExportTypes = Exports.Pdf | Exports.Excel97 | Exports.Rtf
    }
    // or
    //Exports = ExportMenuSettings.All
};
webReport.Toolbar = toolbar;
```

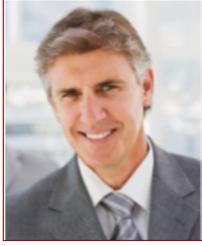Learn how to build this report on the Fast Reports Academy channel

Demonstrates a simple list report. To create it:
- go to "Data" menu and select "Choose Report Data..." item to select a datasource;
- go to "Report|Configure Bands..." menu to create the band structure;
- return to the report page, doubleclick the data band to show its editor;
- choose the datasource;
- drag data from the Data Dictionary window to the band.

WebReport Core/Blazor Server improvements

Added support for Blazor Server components for FastReport.Core3.Web package (CoreWin).

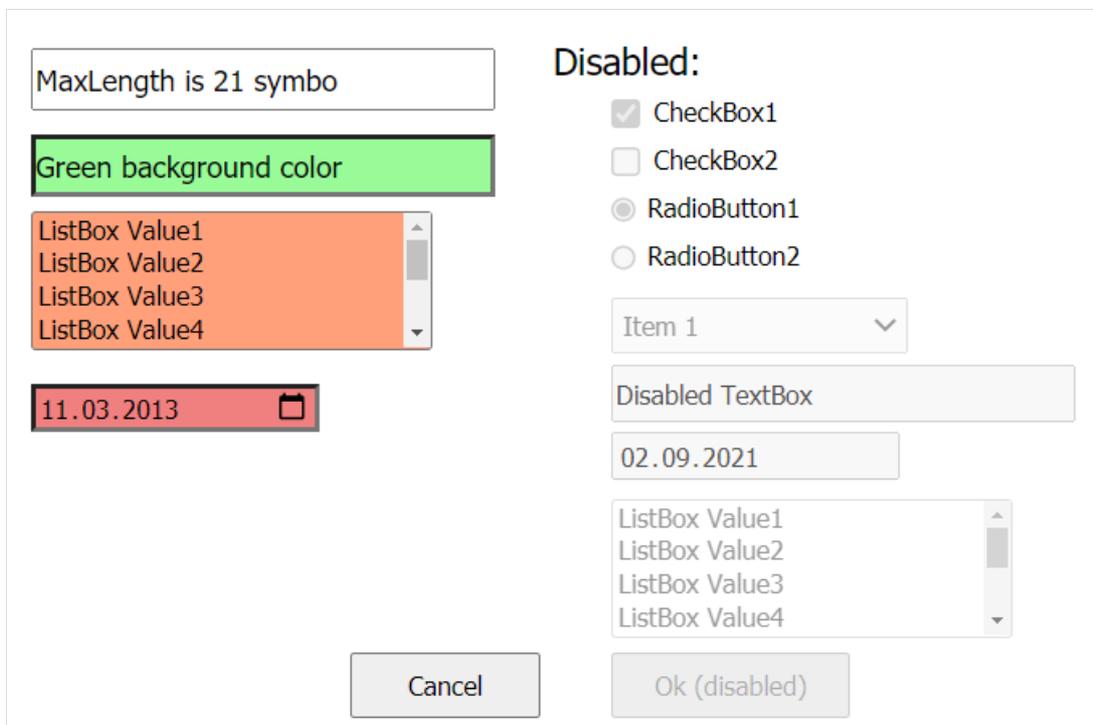Improved support for dialog report components:

- DateTimePicker has been improved. In DateTimePicker.Format.Time mode it displays only time, in DateTimePicker.Format.Short mode - only date, DateTimePicker.Format.Long - both date and time

- Added support of MaxLength property for TextBox
- Added support for the Enabled property
- Added support for background-color

Fixes

- Fixed bug with `Dock` and `Anchor` properties of the objects that are inside the Table/Matrix cell.
- Fixed stack overflow bug when you add `Subreport` object to the page footer band.
- Fixed bug with SVG export if the system's DPI setting is greater than 96DPI.

Complete list of changes

[Engine]

- added connection to ElasticSearch
- added a new barcode - Japanese PostNet
- added the Res.LoadLocale (CultureInfo) method, which changes the selected locale by the CultureInfo argument. Loaded locales are cached. For correct operation, the added FastReport.Localization package is required

- optimized and unified converter RichText to report objects

- fixed a bug with incorrect tab width when TextObject.TextRenderType = TextRenderType.HtmlTextRenderer
- fixed a bug with SubreportObject on a page footer band which caused StackOverflow exception
- fixed a bug with Dock and Anchor properties of objects inside table/matrix cells
- fixed a bug leading to System.ArgumentException when drawing PictureObject located outside the band
- fixed a bug with incorrect work of right anchor (Anchor = AnchorStyles.Right) when page has unlimited width
- fixed a bug with replacing a custom font with a default font when preparing a report
- fixed a bug with vertical alignment when converting RTF (by default, now Top instead of Center)
- fixed a bug with converting RTF tables to report objects

[Designer]

- added simplified display of DB field names in the designer
- added collapse all/expand all button and search field for Report tree and Data tree
- new icons added. Use the designer's "View|Options|User interface" dialog to switch between icon packs.

- fixed a bug leading to the crash of the report designer with an incorrect table in the data source.

[Preview]

- fixed a bug with saving prepared reports containing converted RichObject

[Exports]

- added option when export to Word 2007 "Do not add section breaks on page breaks". By default, both page breaks and section breaks are added.

- fixed page-break in Html export (PageBreaks property)
- fixed SVG export with "Multiply export" parameter
- fixed SVG export bug on hidpi monitor
- fixed the names of files saved in the zip archive
- fixed tab symbols width when export RichObject
- fixed XPS export bug where documents exported on Linux would not open on Windows
- fixed bugs with incorrect work of Anchor and Dock properties when exporting pages with unlimited width
- fixed a bug in Excel 2007 export of text objects with enabled HtmlParagraph render type. Disable WYSIWYG export option to export text instead of images.

[WebReport]

- added background-color support for dialogs in WebReport
- added support for the Enabled property for dialogs in WebReport
- added support for the MaxLength property for the TextBox dialog component in WebReport

- optimized loading of localization for Toolbar

- fixed incorrect page background-color for HTML/Blazor export on Safari browsers
- fixed a bug with hanging of online designer save call-back in WebReport with sessions
- fixed bugs with incorrect work Anchor and Dock properties on pages with unlimited width

[Online Designer]

- fixed save/preview from OnlineDesigner with page in Landscape orientation

[.Net Core]

- added support for Single File Application

- updated dependencies for FastReport.Compat and FastReport.DataVisualization. FastReport.Compat now correctly detects the possibility of using the WinForms API. FastReport.DataVisualization now has no dependency on System.Data.SqlClient and System.Drawing.Common

- fixed a bug where the report did not work with data from the custom library, although it was registered with ReferencedAssemblies in CoreWin
- fixed application crash when loading a report with unknown Font in multiple threads on Linux
- fixed a bug "Could not load type 'System.Drawing.Design.UITypeEditor'"
- fixed loading of table names in XmlDataConnection
- fixed a bug due to which the report and resources were not loaded when publishing/debugging using IIS/IIS Express. For correct work, you need to call the `UseFastReport()` method before 'UseMvc/UseEndpoints'

[WebReport Core/Blazor Server]

- added support for Blazor components for FastReport.Core3.Web package

- fixed incorrect output of multiline text in Blazor (Interactive Forms & TextBox)

- added xml comments (DocumentationFile) to Web libraries
- added a property to disable showing of the toolbar on the dialog page of the report: webReport.Toolbar.ShowOnDialogPage
- added more exports to the toolbar dropdown menu. These properties are available in webReport.Toolbar.Exports
- added the ability to customize the toolbar: Position, color of the drop-down menu, font, transparency of icons, change the color of icons, change the position of content. These properties are available in webReport.Toolbar
- dialog DateTimePicker for WebReport has been improved. In DateTimePicker.Format.Time mode, it displays

only time, in DateTimePicker.Format.Short mode - only date, DateTimePicker.Format.Long - both date and time.

- fixed missing line breaks for the Label dialog component in WebReport

[Extras]

- added package 'FastReport.Localization', which includes FastReport localization files in your project for working with different languages
- added FastReport Business Graphics integration objects (\Extras\Objects\FastReportBGObjects)

[Demos]

- implemented transition to list of reports, when clicking on the arrow on folder in new demo

- changed Target Framework for new demo to 4.7.2
- changed color of inactive buttons in the thumbnail view mode of new demo
- changed background color of the zoom slider in new demo
- changed background color when displaying dialog forms in the new demo
- changed background color of interactive reports tabs in new demo
- changed the location of the folder with report thumbnails for the demo application. Now this folder is located not in Program Files but in AppData\Local

- fixed problems with displaying interface elements of the new demo application
- fixed a bug that caused saving a prepared report when clicking on the drop-down items in the save menu of the new demo application
- fixed a bug with alignment of reports in preview window of the new demo
- fixed a bug with double launching dialog forma when selecting a report in new demo
- fixed a bug with stretching thumbnails in the new demo
- fixed a bug leading to a lag when moving the window of new demo
- fixed a bug in the new demo with simultaneous displaying of thumbnails in folder and report bars

[Plugins]

- packages with plugins-connectors FastReport.Data have been updated. Now they include plugins for different FastReport editions (.NET, Core, CoreWin, OpenSource) and automatically include the necessary library, depending on the product used
- postgres npgsql version downgrade from 4.0.3 to 3.2.7

Contacts and technical support

You can always ask questions about using the product by email, or by using the form on the website.

We also welcome your suggestions on how to improve our product.