Table of Contents

1 Introduction

We are glad to welcome you to the online version of the documentation of **Stimulsoft Reports** products. The documentation describes the basics of using the **API** of our software. Here we will review how to pass data from code in a report, export reports to various file formats, inherit reports, work with the components and more:

Welcome to **Stimulsoft**:

- Technical Support
- Trial Limitations
- Web Links and Online Resources
- Product Evaluation
- Engine

- Reports and Dashboards for ASP.NET WebForms
- Reports and Dashboards for ASP.NET MVC
- Reports and Dashboards for ASP.NET Core MVC
- Reports and Dashboards for ASP.NET Core Razor
- Reports and Dashboards for Blazor
- Reports and Dashboards for WinForms
- Reports for Angular

- Reports and Dashboards for JS
- Reports and Dashboards for PHP
- Reports and Dashboards for Python
- Reports for Java
- Reports for WPF
- PDF Forms

The first part of the documentation contains the description of work with visual parts of **Stimulsoft** products.

1.1 Technical Support

Registered users and users who are evaluating the software may get technical support.

For technical questions, use the Email address: support@stimulsoft.com
For subscription, payment questions, use Email address: sales@stimulsoft.com
For other questions, use Email address: info@stimulsoft.com
If you have problems with our products, you may contact us through our feedback form at https://www.stimulsoft.com/en/support

If you are a registered user and you contact us for technical support, use the same Email address you used when you purchased our product. Otherwise, it will be difficult to identify you as a registered user. This can slow down our response. Please let us know when your Email address changes.

To solve your problem quickly, we need the following information:

- Product name and its version;
- A detailed description of the problem and how to reproduce it;
- Your operating system (98, ME, 2000, XP, Vista, Window 7 etc.), its version, and the localization of established service packs;

- Version of Microsoft .NET Framework or other development environment and installed service packs;
- A name of your development environment and its version;
- Additional information that can help us solve the problem.

1.2 Trial Limitations

Trial Versions

The free trial of Stimulsoft software is a full-featured version. It has a few limitations, which are as follows:

- The evaluation period is limited to 60 days for Stimulsoft components;
- The Trial watermark is printed on each report page or the dashboard panel.

Registered Versions

If your reports are displaying a **TRIAL** watermark, this means that you are using a trial version of the product. Log in to your account and activate the product using the license key.

Check out how to activate

- ASP.NET HTML5 Viewer
- ASP.NET HTML5 Designer
- ASP.NET MVC HTML5 Viewer
- ASP.NET MVC HTML5 Designer
- ASP.NET Core HTML5 Viewer
- ASP.NET Core HTML5 Designer
- Angular Components
- JavaScript Components
- WinForms Components
- Java Components
- WPF Components

1.3 Web Links and Online Resources

This section describes how to get information about the latest news and announcements of software products, and information about known issues and questions that users are interested in.

- The official website of the company is available at https://stimulsoft.com
  - Description of products you may find at https://stimulsoft.com/en/products
  - Download the latest version of the product by the link https://stimulsoft.com/en/downloads
  - You can read the latest news of the company at https://stimulsoft.com/en/blog/news

- In addition, you can download packages of products Stimulsoft from other resources:
  - Reports.Web, Reports.Blazor, Reports.Angular Reports.Net, Reports.Wpf,

Reports.Web.NetCore, Dashboards.Blazor, Dashboards.Win, Dashboards.Web, Dashboards.Web.NetCore from NuGet at https://www.nuget.org/profiles/Stimulsoft
- Reports.Java from Maven at http://central.maven.org/maven2/com/stimulsoft
- Reports.JS and Dashboards.JS from npm at https://www.npmjs.com/search?q=stimulsoft
- Reports.PHP and Dashboards.PHP from composer at https://packagist.org/?query=stimulsoft

› You can evaluate our reporting and dashboard tools online - https://demo.stimulsoft.com

› To create, store and then deploy reports in your applications, use the cloud service of Stimulsoft https://cloud.stimulsoft.com

› A huge number of video lessons are available on our YouTube channel https://www.youtube.com/user/StimulsoftVideos

› You can use samples for various platforms:
- on GitHub at https://github.com/stimulsoft
- on our website at https://www.stimulsoft.com/en/samples

› Find us in social networks and messengers:


https://twitter.com/stimulsoft


https://www.linkedin.com/company/stimulsoft


https://www.facebook.com/Stimulsoft


https://www.stimulsoft.com/en/rss


https://t.me/stimulsoft


+48690104472


WhatsApp - Stimulsoft


Skype - Stimulsoft

Also, visit our Forum to communicate with other users of Stimulsoft Reports - http://forum.stimulsoft.com/index.php
Here you can read and discuss various topics related to tools for creating reports. For more information about the product in other Internet resources, please use the search engines.

2 Deployment

Information

Stimulsoft Ultimate package contains libraries, scripts, and other files necessary for working with reports and dashboards in WinForms, ASP.NET, ASP.NET MVC, .NET Core, WPF, JavaScript, PHP, and Java applications. The product includes a report and data analysis engine, reports and dashboards designers and viewers for all supported platforms.

This section describes the various distribution options for reports, libraries, and files supplied with Stimulsoft software.

Name	Description
Reports.WEB	The package contains libraries for working with reports in ASP.NET, ASP.NET MVC, and .NET Core applications.
Reports.NET	The package contains libraries for working with reports in WinForms applications.
Reports.JS	The package contains scripts for working with reports in JavaScript applications.
Reports.WPF	The package contains libraries for working with reports in WPF applications.
Reports.PHP	The package contains scripts and styles for working with reports in client-server applications using PHP.
Reports.JAVA	The package contains JAR files for working with reports in Java applications.
Dashboards.WEB	The package contains libraries for working with dashboards and reports in ASP.NET, ASP.NET MVC, and .NET Core applications.

Dashboards.WIN	The package contains libraries for working with dashboards and reports in WinForms and WPF applications.
Dashboards.JS	The package contains scripts for working with dashboards and reports in JavaScript applications.
Stimulsoft Designer	The application contains files, libraries to run the standalone report designer.
Stimulsoft Designer.JS	The application contains files, scripts to run the standalone JavaScript report designer.
Stimulsoft Demo	The application contains files, libraries to run the reports and dashboards demonstration.

Information

A dmg is supplied in which **Stimulsoft Designer for Mac** is presented as a single application. After installation, run the **Stimulsoft Designer.app** file to start the report designer.

2.1 Libraries of Report.WEB Package

Reports.WEB package contains the following libraries:

Libraries	Description
Stimulsoft.Base.dll	The library contains common base interfaces and classes for all products.
Stimulsoft.Data.dll	The library contains classes and methods for data analysis, transformation, and filtering.
Stimulsoft.Map.dll	The library contains resources for working with region maps.
Stimulsoft.Report.dll	The library contains all the functionality of the report generator - rendering, exporting, working with data.
Stimulsoft.Report.Check.dll	The library contains all methods for working with the Report Checker.
Stimulsoft.Report.Design.dll	The library contains resources and control classes of the report designer.

Stimulsoft.Report.Helper.dll	The library contains resources extended products localization – description of properties and actions, error messages.
Stimulsoft.Report.Mvc.dll	The library contains resources and control classes of the viewer and designer for ASP.NET MVC.
Stimulsoft.Report.Web.dll	The library contains resources and control classes of the viewer for ASP.NET.
Stimulsoft.Report.WebDesign.dll	The library contains resources and control classes of the designer for ASP.NET.
Stimulsoft.Report.Mvc.NetCore.dll	The library contains resources and control classes of the viewer and designer for .NET Core MVC.
Stimulsoft.System.dll	The library contains system interfaces and classes required for the report generator with .NET Standard compatibility.
Stimulsoft.System.Web.dll	The library contains system interfaces and classes required for Web controls with .NET Standard compatibility.
Svg	The library contains functionality for using SVG images in reports and dashboards.
LibExcel	The library is applicable for using XLS files as a data source in reports and dashboards.

2.2 Libraries of Reports.NET Package

Reports.NET package contains the following libraries:

Libraries	Description
Stimulsoft.Base.dll	The main library contains common base interfaces and classes for all products.
Stimulsoft.Controls.dll	The library contains additional controls that are used in the designer and viewer for desktop applications.
Stimulsoft.Controls.Win.dll	The library contains main controls that are used in the designer and viewer for desktop applications.
Stimulsoft.Data.dll	The library contains classes and methods for data analysis, transformation, and filtering.
Stimulsoft.Datab	The library is used for creating and editing database

ase.dll	connections. The library contains the SQL Query Builder.
Stimulsoft.Design.dll	The library contains classes for working with reports in Design Time for Visual Studio.
Stimulsoft.Editor.dll	The library contains a text editor, which is used in the report designer.
Stimulsoft.Map.dll	The library contains resources for working with region maps.
Stimulsoft.Report.dll	The main library contains all the functionality of the report generator - rendering, exporting, working with data.
Stimulsoft.Report.Check.dll	The library contains all methods for working with the Report Checker.
Stimulsoft.Report.Design.dll	The library contains resources and control classes of the WinForms report designer.
Stimulsoft.Report.Helper.dll	The library contains resources extended products localization – description of properties and actions, error messages.
Stimulsoft.Report.Web.dll	The library contains resources and control classes of the viewer for ASP.NET.
Stimulsoft.Report.Win.dll	The library contains resources and control classes of the WinForms viewer.
Svg	The library contains functionality for using SVG images in reports and dashboards.

2.3 Scripts of Reports.JS Package

Reports.JS package contains the following scripts:

Scripts	Description
Common Scripts *:	
stimulsoft.blockly.editor.js	The script contains visual editor Blockly for creating event scripts in a report. The Event handler is embedded in report engine.
stimulsoft.designer.js	The script for working with the report designer.

stimulsoft.reports.js	The main script contains all the functionality of the report generator - rendering, exporting, working with data.
stimulsoft.reports.maps.js	The script contains resources for working with region maps.
stimulsoft.viewer.js	The script for working with the report viewer.
stimulosft.reports.js can be split into *:	
stimulsoft.reports.engine.js	The main script of the report generator.
stimulsoft.reports.chart.js	The script is necessary for working with charts.
stimulsoft.reports.export.js	The script is necessary for exporting reports.
stimulsoft.reports.import.xlsx.js	The script is necessary for importing data from XLSX files.
Other Files:	
stimulsoft.reports.d.ts	The header file describes the syntax and structure of functions and properties.

* - Also, you can use packed scripts instead of common scripts or parts of the stimulsoft.reports.js. Packed scripts allow you to reduce the script file size by several times. The names of packed scripts correspond with the names of scripts, but they contain the word **pack** in the name before the **js** extension. For example, a packed analog of the **stimulsoft.reports.js** script will be the **stimulsoft.reports.pack.js**.

2.4 Libraries of Reports.WPF Package

Reports.WPF package contains the following libraries:

Libraries	Description
Stimulsoft.Base.dll	The main library contains common base interfaces and classes for all products.

Stimulsoft.Client.Designer.dll	The library contains resources and classes of the WPF report designer V2.
Stimulsoft.Data.dll	The library contains classes and methods for data analysis, transformation, and filtering.
Stimulsoft.Database.dll	The library is used for creating and editing database connections. The library contains the SQL Query Builder.
Stimulsoft.Editor.dll	The library contains a text editor, which is used in the report designer.
Stimulsoft.Map.dll	The library contains resources for working with region maps.
Stimulsoft.Report.dll	The main library contains all the functionality of the report generator - rendering, exporting, working with data.
Stimulsoft.Report.Check.dll	The library contains all methods for working with the Report Checker.
Stimulsoft.Report.Helper.dll	The library contains resources extended products localization – description of properties and actions, error messages.
Stimulsoft.Report.Wpf.dll	The library contains resources and classes of the WPF report viewer.
Stimulsoft.Report.Wpf.BlackTheme.dll	This library contains the Black theme using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2003BlueTheme.dll	This library contains the Office 2003 Blue theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2003OliveGreenTheme.dll	This library contains the Office 2003 Olive Green theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2003SilverTheme.dll	This library contains the Office 2003 Silver theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2007BlackTheme.dll	This library contains the Office 2007 Black theme for using it in the WPF viewer and report designer.

Stimulsoft.Report.Wpf.Office2007BlueTheme.dll	This library contains the Office 2007 Blue theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2007SilverTheme.dll	This library contains the Office 2007 Silver theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2010BlueTheme.dll	This library contains the Office 2010 Blue theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2010WhiteTheme.dll	This library contains the Office 2010 White theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.WpfDesign.dll	The library contains resources and classes of the WPF report designer V1.

2.5 Scripts of Reports.PHP Package

Reports.PHP package contains the following libraries:

Scripts	Description
stimulsoft.blockly.editor.js	The script contains visual editor Blockly for creating event scripts in a report. The Event handler is embedded in report engine.
stimulsoft.designer.js	The script for working with the report designer.
stimulsoft.designer.pack.js	The packed script for working with the report designer. It is similar to the previous one, but a bit smaller.
stimulsoft.reports.js	The main script contains all the functionality of the report generator - rendering, exporting, working with data.
stimulsoft.reports.pack.js	The packed main script contains all the functionality of the report generator - rendering, exporting, working with data. It is similar to the previous one, but a bit smaller.
stimulsoft.reports.maps.js	The script contains resources for working with region maps.

stimulsoft.reports.maps.pack.js	The packed script contains resources for working with region maps. It is similar to the previous one but a bit smaller.
stimulsoft.viewer.js	The script for working with the report viewer.
stimulsoft.viewer.pack.js	The packed script for working with the report viewer. It is similar to the previous one but a bit smaller.
stimulsoft.reports.d.ts	The header file describes the syntax and structure of functions and properties.

2.6 JAR Files of Reports.JAVA Package

Reports.JAVA package contains the following files:

JAR Files	Description
stimulsoft-reports-base	The file contains base interfaces and classes for the product.
stimulsoft-reports-demo	The file contains a demo application.
stimulsoft-reports-lib	The file contains a utility library.
stimulsoft-reports-report	The file contains all the functionality of the report generator - rendering, exporting, working with data.
stimulsoft-reports-samples	The file contains samples.
stimulsoft-reports-viewer	The file contains resources and control classes of the java viewer.
stimulsoft-reports-web	The file contains a common part of the web designer and viewer.
stimulsoft-reports-webdesigner	The file contains resources and control classes of the web designer.
stimulsoft-reports-webviewer	The file contains resources and control classes of the web viewer.

2.7 Libraries of Dashboards.WEB Package

Dashboards.WEB package contains the following libraries:

Libraries	Description
Stimulsoft.Base.dll	The main library contains common base interfaces and classes for all products.
Stimulsoft.Dashboard	The library contains the core functionality of the Stimulsoft Dashboards - main objectives, classes, definitions.
Stimulsoft.Dashboard.Drawing	The library is used for drawing the elements of Stimulsoft Dashboards with the help of GDI technology.
Stimulsoft.Dashboard.Export	The library contains all the necessary methods for exporting dashboards.
Stimulsoft.Data.dll	The library contains classes and methods for data analysis, transformation, and filtering.
Stimulsoft.Map.dll	The library contains resources for working with region maps.
Stimulsoft.Report.dll	The main library contains all the functionality of the report generator - rendering, exporting, working with data.
Stimulsoft.Report.Check.dll	The library contains all methods for working with the Report Checker.
Stimulsoft.Report.Helper.dll	The library contains resources extended products localization – description of properties and actions, error messages.
Stimulsoft.Report.Mvc.dll	The library contains resources and control classes of the viewer and designer for ASP.NET MVC.
Stimulsoft.Report.Web.dll	The library contains resources and control classes of the viewer for ASP.NET.
Stimulsoft.Report.WebDesign.dll	The library contains resources and control classes of the designer for ASP.NET.
Svg	The library contains functionality for using SVG images in reports and dashboards.
LibExcel	The library is required for using XLS files as a data source in reports and dashboards.

2.8 Libraries of Dashboards.WIN Package

Dashboards.WIN package contains the following libraries:

Libraries	Description
Stimulsoft.Base.dll	The main library contains common base interfaces and classes for all products.
Stimulsoft.Controls.dll	The library contains additional controls that are used in the designer and viewer for desktop applications.
Stimulsoft.Controls.Win.dll	The library contains main controls that are used in the designer and viewer for desktop applications.
Stimulsoft.Dashboard.dll	The library contains the core functionality of the Stimulsoft Dashboards - main objectives, classes, definitions.
Stimulsoft.Dashboard.Design.dll	The library contains resources and control classes of the WinForms dashboard designer.
Stimulsoft.Dashboard.Drawing.dll	The library is used for drawing the elements of Stimulsoft Dashboards with the help of GDI technology.
Stimulsoft.Dashboard.Drawing.Wpf.dll	The library is used for drawing the elements of Stimulsoft Dashboards with the help of WPF technology.
Stimulsoft.Dashboard.Export.dll	The library contains all the necessary methods for exporting dashboards.
Stimulsoft.Dashboard.Viewer.dll	The library contains resources and control classes of the WinForms dashboard viewer.
Stimulsoft.Dashboard.Viewer.Wpf.dll	The library contains resources and control classes of the WPF dashboard viewer.
Stimulsoft.Data.dll	The library contains classes and methods for data analysis, transformation, and filtering.
Stimulsoft.Database.dll	The library is used for creating and editing database connections. The library contains the SQL Query Builder.
Stimulsoft.Design.dll	The library contains classes for working with reports in Design Time for Visual Studio.

Stimulsoft.Editor.dll	The library contains a text editor which is used in the report designer.
Stimulsoft.Map.dll	The library contains resources for working with region maps.
Stimulsoft.Report.dll	The main library contains all the functionality of the report generator - rendering, exporting, working with data.
Stimulsoft.Report.Check.dll	The library contains all methods for working with the Report Checker.
Stimulsoft.Report.Design.dll	The library contains resources and control classes of the WinForms report designer.
Stimulsoft.Report.Helper.dll	The library contains resources extended products localization – description of properties and actions, error messages.
Stimulsoft.Report.Web.dll	The library contains resources and control classes of the ASP.NET viewer.
Stimulsoft.Report.Win.dll	The library contains resources and control classes of the WinForms viewer.
Stimulsoft.Report.Wpf.dll	The library contains resources and control classes of the WPF viewer.
Svg	The library contains functionality for using SVG images in reports and dashboards.

2.9 Scripts of Dashboards.JS Package

Dashboards.JS package contains the following scripts:

Scripts	Description
Common Scripts *:	
stimulsoft.blockly.editor.js	The script contains visual editor Blockly for creating event scripts in a report. The Event handler is embedded in report engine.
stimulsoft.dashboards.js	The script for working with dashboards.
stimulsoft.desig	The script for working with the report designer.

ner.js	
stimulsoft.reports.js	The main script contains all the functionality of the report generator - rendering, exporting, working with data.
stimulsoft.reports.maps.js	The script contains resources for working with region maps.
stimulsoft.viewer.js	The script for working with the report viewer.
stimulosft.reports.js can be split into *:	
stimulsoft.reports.engine.js	The main script of the report generator.
stimulsoft.reports.chart.js	The script is necessary for working with charts.
stimulsoft.reports.export.js	The script is necessary for exporting reports.
stimulsoft.reports.import.xlsx.js	The script is necessary for importing data from XLSX files.
Other Files:	
stimulsoft.reports.d.ts	The header file describes the syntax and structure of functions and properties.

* - Also, you can use packed scripts instead of common scripts or parts of the stimulsoft.reports.js. Packed scripts allow you to reduce script file size by several times. The names of packed scripts correspond with the names of scripts, but they contain a **pack** word in the name before **js** extension. For example, a packed analog of the **stimulsoft.reports.js** script will be the **stimulsoft.reports.pack.js**.

2.10 Scripts of Dashboards.PHP Package

Reports.PHP package contains the following libraries:

Scripts	Description
Common Scripts *:	

stimulsoft.blockly.editor.js	The script contains visual editor Blockly for creating event scripts in a report. The Event handler is embedded in report engine.
stimulsoft.dashboards.js	The script for working with dashboards.
stimulsoft.designer.js	The script for working with the report designer.
stimulsoft.reports.js	The main script contains all the functionality of the report generator - rendering, exporting, working with data.
stimulsoft.reports.maps.js	The script contains resources for working with region maps.
stimulsoft.viewer.js	The script for working with the report viewer.
stimulosft.reports.js can be split into *:	
stimulsoft.reports.engine.js	The main script of the report generator.
stimulsoft.reports.chart.js	The script is necessary for working with charts.
stimulsoft.reports.export.js	The script is necessary for exporting reports.
stimulsoft.reports.import.xlsx.js	The script is necessary for importing data from XLSX files.
Other Files:	
stimulsoft.reports.d.ts	The header file describes the syntax and structure of functions and properties.

* - Also, you can use packed scripts instead of common scripts or parts of the stimulsoft.reports.js. Packed scripts allow you to reduce script file size by several times. The names of packed scripts correspond with the names of scripts, but they contain a **pack** word in the name before **js** extension. For example, a packed analog of the **stimulsoft.reports.js** script will be the **stimulsoft.reports.pack.js**.

2.11 Files of Stimulsoft Designer Application

Stimulsoft Designer application package contains the following files:

Name	Description
Folders:	
Localization	The folder contains a list of localization files.
Themes	The folder contains a list of default styles in reports.
.exe files	
Designer.exe	The launch file of the WinForms report designer.
Designer.Wpf.exe	The launch file of the WPF report designer V1.
DesignerV2.Wpf.exe	The launch file of the WPF report designer V2.
Stimulsoft libraries:	
Stimulsoft.Accounts.Wpf.dll	The library is used for the authorization of users in the report designer.
Stimulsoft.Base.dll	The main library contains common base interfaces and classes for all products.
Stimulsoft.Client.dll	The library contains the user configuration of the report designer and viewer.
Stimulsoft.Client.Designer.dll	The library contains resources and classes of the WPF report designer V2.
Stimulsoft.Controls.dll	The library contains additional controls that are used in the designer and viewer for desktop applications.
Stimulsoft.Controls.Win.dll	The library contains main controls that are used in the designer and viewer for desktop applications.
Stimulsoft.Dashboard	The library contains the core functionality of the Stimulsoft Dashboards - main objectives, classes, definitions.
Stimulsoft.Dashboard.Design	The library contains resources and control classes of the WinForms dashboard designer.

Stimulsoft.Dashboard.Drawing.dll	The library is used for drawing the elements of Stimulsoft Dashboards with the help of GDI technology.
Stimulsoft.Dashboard.Drawing.Wpf.dll	The library is used for drawing the elements of Stimulsoft Dashboards with the help of WPF technology.
Stimulsoft.Dashboard.Export.dll	The library contains all the necessary methods for exporting dashboards.
Stimulsoft.Dashboard.Viewer.dll	The library contains resources and control classes of the WinForms dashboard viewer.
Stimulsoft.Dashboard.Viewer.Wpf.dll	The library contains resources and control classes of the WPF dashboard viewer.
Stimulsoft.Dashboard.Viewer.Wpf.Design.dll	The library is required to integrate the WPF dashboard viewer with the Visual Studio toolbox.
Stimulsoft.Data.dll	The library contains classes and methods for data analysis, transformation, and filtering.
Stimulsoft.Database.dll	The library is used for creating and editing database connections. The library contains the SQL Query Builder.
Stimulsoft.Database.Wpf.dll	The library is used for creating and editing database connections for the WPF report engine. The library contains the SQL Query Builder.
Stimulsoft.Design.dll	The library contains classes for working with reports in Design Time for Visual Studio.
Stimulsoft.Editor.dll	The library contains a text editor which is used in the report designer.
Stimulsoft.Editor.Wpf.dll	The library contains a text editor which is used in the WPF report designer.
Stimulsoft.Map.dll	The library contains resources for working with region maps.
Stimulsoft.Report.dll	The main library contains all the functionality of the report generator - rendering, exporting, working with data.
Stimulsoft.Repor	The library contains all methods for working with the Report

t.Check.dll	Checker.
Stimulsoft.Report.Design.dll	The library contains resources and control classes of the WinForms report designer.
Stimulsoft.Report.Helper.dll	The library contains resources extended products localization – description of properties and actions, error messages.
Stimulsoft.Report.Import.dll	The library contains resources for importing reports.
Stimulsoft.Report.Mvc.dll	The library contains resources and control classes of the viewer and designer for ASP.NET MVC.
Stimulsoft.Report.Publish.dll	The library contains resources for publishing reports.
Stimulsoft.Report.Web.dll	The library contains resources and control classes of the viewer for ASP.NET.
Stimulsoft.Report.WebDesign.dll	The library contains resources and control classes of the designer for ASP.NET.
Stimulsoft.Report.Win.dll	The library contains resources and control classes of the viewer for WinForms.
Stimulsoft.Report.Wpf.dll	The library contains resources and control classes of the viewer for WPF.
Stimulsoft.Report.Wpf.BlackTheme.dll	This library contains the Black theme using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Design.dll	The library needed to integrate the WPF viewer with the Visual Studio toolbox.
Stimulsoft.Report.Wpf.Office2003BlueTheme.dll	This library contains the Office 2003 Blue theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2003OliveGreenTheme.dll	This library contains the Office 2003 Olive Green theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2003SilverTheme.dll	This library contains the Office 2003 Silver theme for using it in the WPF viewer and report designer.

Stimulsoft.Report.Wpf.Office2007BlackTheme.dll	This library contains the Office 2007 Black theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2007BlueTheme.dll	This library contains the Office 2007 Blue theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2007SilverTheme.dll	This library contains the Office 2007 Silver theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2010BlueTheme.dll	This library contains the Office 2010 Blue theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.Wpf.Office2010WhiteTheme.dll	This library contains the Office 2010 White theme for using it in the WPF viewer and report designer.
Stimulsoft.Report.WpfDesign.dll	The library contains resources and classes of the WPF report designer V1.
Stimulsoft.Report.WpfDesign.Design.dll	The library needed to integrate the WPF designer with the Visual Studio toolbox.
Stimulsoft.Server.Connect.dll	The library contains classes and methods for connecting to the server.
Stimulsoft.Server.Objects.dll	The library contains a description of objects that can be transferred to the server.
Stimulsoft.Wizard.Wpf.dll	The library contains the resources needed to create reports using the report wizards.
External libraries:	
adodb.dll	The library contains the necessary classes for accessing various types of databases.
LibExcel.dll	The library is used for XLS files as a data source in reports and dashboards.
Microsoft.Web.Infrastructure.dll	The library is used to register HTTP modules at run time dynamically.

Microsoft.Web.XmlTransform.dll	The library is used to convert XML files.
Newtonsoft.Json.dll	The library contains classes for converting objects to JSON objects.
NuGet libraries	All NuGet libraries are required to download Stimulsoft packages from NuGet.
Svg.dll	The library contains functionality for using SVG images in reports and dashboards.
System.Web	All the necessary libraries for publishing Web packages from the report designer.

2.12 Files of Stimulsoft Designer.JS Application

Stimulsoft Designer.JS application package contains the following files:

Name	Description
locales folder	The folder contains a list of localization files.
scripts folder:	The folder contains a list of Stimulsoft scripts.
stimulsoft.dashboards.js	The script for working with dashboards.
stimulsoft.designer.js	The script for working with the report designer.
stimulsoft.designer.runtime.js	The additional script for the standalone report designer.
stimulsoft.reports.js	The main script contains all the functionality of the report generator - rendering, exporting, working with data.
stimulsoft.reports.maps.js	The script contains resources for working with region maps.
stimulsoft.viewer.js	The script for working with the report viewer.
styles folder	The folder contains a list of designer and viewer themes.
Designer.exe file	The launch file of the report designer.

Information

Also, the **Stimulsoft Designer.JS** directory contains third-party folders, scripts, and files necessary for working with the auxiliary framework.

2.13 Files of Stimulsoft Demo Application

Stimulsoft Demo application package contains the following files:

Name	Description
Folders:	
Localization	The folder contains a list of localization files.
Reports	The folder contains a list of reports and dashboards templates.
.exe files	
Designer.exe	The launch file of the WinForms report designer.
Demo.exe	The launch file of the demo application.
Stimulsoft Libraries:	
Stimulsoft.Accounts.Wpf.dll	The library is used for the authorization of users in the report designer.
Stimulsoft.Base.dll	The main library contains common base interfaces and classes for all products.
Stimulsoft.Client.dll	The library contains the user configuration of the report designer and viewer.
Stimulsoft.Controls.dll	The library contains additional controls that are used in the designer and viewer for desktop applications.
Stimulsoft.Controls.Win.dll	The library contains main controls that are used in the designer and viewer for desktop applications.
Stimulsoft.Dashboard	The library contains the core functionality of the Stimulsoft Dashboards - main objectives, classes, definitions.

Stimulsoft.Dashboard.Design	The library contains resources and control classes of the WinForms dashboard designer.
Stimulsoft.Dashboard.Drawing.dll	The library is used for drawing the elements of Stimulsoft Dashboards with the help of GDI technology.
Stimulsoft.Dashboard.Drawing.Wpf.dll	The library is used for drawing the elements of Stimulsoft Dashboards with the help of WPF technology.
Stimulsoft.Dashboard.Export.dll	The library contains all the necessary methods for exporting dashboards.
Stimulsoft.Dashboard.Viewer.dll	The library contains resources and control classes of the WinForms dashboard viewer.
Stimulsoft.Dashboard.Viewer.Wpf.dll	The library contains resources and control classes of the WPF dashboard viewer.
Stimulsoft.Data.dll	The library contains classes and methods for data analysis, transformation, and filtering.
Stimulsoft.Database.dll	The library is used for creating and editing database connections. The library contains the SQL Query Builder.
Stimulsoft.Database.Wpf.dll	The library is used for creating and editing database connections for the WPF report engine. The library contains the SQL Query Builder.
Stimulsoft.Design.dll	The library contains classes for working with reports in Design Time for Visual Studio.
Stimulsoft.Editor.dll	The library contains a text editor which is used in the report designer.
Stimulsoft.Editor.Wpf.dll	The library contains a text editor which is used in the WPF report designer.
Stimulsoft.Map.dll	The library contains resources for working with region maps.
Stimulsoft.Report.dll	The main library contains all the functionality of the report generator - rendering, exporting, working with data.
Stimulsoft.Report.Check.dll	The library contains all methods for working with the Report Checker.

Stimulsoft.Report.Design.dll	The library contains resources and control classes of the WinForms report designer.
Stimulsoft.Report.Design.WebViewer.dll	The library contains resources and control classes of the web viewer.
Stimulsoft.Report.Helper.dll	The library contains resources extended products localization – description of properties and actions, error messages.
Stimulsoft.Report.Import.dll	The library contains resources for importing reports.
Stimulsoft.Report.Publish.dll	The library contains resources for publishing reports.
Stimulsoft.Report.Win.dll	The library contains resources and control classes of the viewer for WinForms.
Stimulsoft.Report.Wpf.dll	The library contains resources and control classes of the viewer for WPF.
Stimulsoft.Server.Connect.dll	The library contains classes and methods for connecting to the server.
Stimulsoft.Server.Objects.dll	The library contains a description of objects that can be transferred to the server.
Stimulsoft.Wizard.Wpf.dll	The library contains the resources needed to create reports using the report wizards.
External libraries:	
adodb.dll	The library contains the necessary classes for accessing various types of databases.
LibExcel.dll	The library is required for using XLS files as a data source in reports and dashboards.
Newtonsoft.Json.dll	The library contains classes for converting objects to JSON objects.
NuGet libraries	All NuGet libraries are required to download Stimulsoft packages from NuGet.
Svg.dll	The library contains functionality for using SVG images in reports and dashboards.

3 Reports and Dashboards for ASP.NET WebForms

ASP.NET is the technology for creating web applications and web services. Stimulsoft provides tools for creating and displaying reports and dashboards on any device using this technology.

Tools for creating and editing reports:	Tools for viewing and converting reports:	Tools for creating and editing dashboards:	Tools for viewing and converting dashboards:
HTML5 designer	HTML5 viewer	HTML5 designer	HTML5 viewer

3.1 HTML5 Viewer

YouTube

Watch videos for the ASP.NET HTML5 Viewer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

Samples

See on GitHub examples for working with the ASP.NET HTML5 Viewer component. All examples are separate projects grouped into one solution for Visual Studio.

The **HTML5 Viewer** (**StiWebViewer**) component is designed for viewing reports in the web browser. You do not need to install the .NET Framework, ActiveX components, or any special plug-ins on the client-side. All you need is any modern Web browser.

With the help of the **HTML5 Viewer**, you can view, print, and export reports on any computer with any operating system installed. Since the viewer only uses HTML and JavaScript technologies, it can be run on devices with no Flash or Silverlight support - tablets, smartphones. Also, the viewer supports Mobile and Touch interfaces, which automatically enable when using mobile devices and touch screen monitors.

The **HTML5 Viewer** component uses the AJAX technology to perform all actions (uploading a report, paging, scaling, interactivity in reports, etc.), allowing you to get rid of reloading the entire page and save Web traffic, and speed up work.

The **HTML5 Viewer** supports many themes, animated interface, bookmarks, interactive reports, editing of report elements on the page, full-screen mode, search panel, and other necessary features for viewing reports.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use the **HTML5 Viewer** in a Web project, you need to install the NuGet package of Stimulsoft.Reports.Web:

- Select "Manage NuGet Packages ..." in the context menu of the project;
- Specify Stimulsoft.Reports.Web in the search bar on the Browse tab;
- Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

If this is not possible, you should add the following assemblies to the project:

- Stimulsoft.Base.dll
- Stimulsoft.Report.dll
- Stimulsoft.Report.Check.dll
- Stimulsoft.Report.Helper.dll
- Stimulsoft.Report.Web.dll

To add the ability to view and export dashboards in a Web project, install the NuGet package Stimulsoft.Dashboards.Web (this package is associated with the package Stimulsoft.Reports.Web. If it is missed, it will be installed automatically):

- Select "Manage NuGet Packages ..." in the context menu of the project;
- Specify Stimulsoft.Dashboards.Web in the search bar on the Browse tab;
- Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

If for any reason this is not possible, you should additionally add the following assemblies to the project:

- Stimulsoft.Dashboard.dll
- Stimulsoft.Dashboard.Drawing.dll
- Stimulsoft.Dashboard.Export.dll

3.1.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the viewer, you need to place the **StiWebViewer** component on the ASPX page, set the necessary properties, and, if necessary, set the necessary event handlers. When the report viewer runs, the following actions occur:

› The .NET component generates HTML and JavaScript code that is necessary for displaying and running the viewer;

› When the component is output, the JavaScript method is launched. It requests the first page of the report on the server-side or the entire report (depending on the selected mode) and the required report parameters;

› Each action in the viewer (for example, paging, printing or exporting a report, etc.) calls a certain action on the server-side, in which you can perform the necessary manipulations with the report by subscribing to the corresponding viewer events;

› The viewer saves the report in cache or server session to speed up the work, which makes it impossible to re-build the report.

3.1.2 Activation

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiWebViewer** component.

Default.aspx.cs

```
...
public partial class _Default : Page
{
   static _Default()
   {
      //Activation with using license code
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";

      //Activation with using license file
      var path = HttpContext.Current.Server.MapPath("license.key");
      Stimulsoft.Base.StiLicense.LoadFromFile(path);
   }
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

3.1.3 Showing Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `Report.Render()` method if you want to perform specific actions with the rendered report before it is displayed in the viewer. Likewise, when using the compilation mode, you need to call the `Report.Compile()` method only if you have actions to perform with the compiled report before it is built and shown in the viewer.

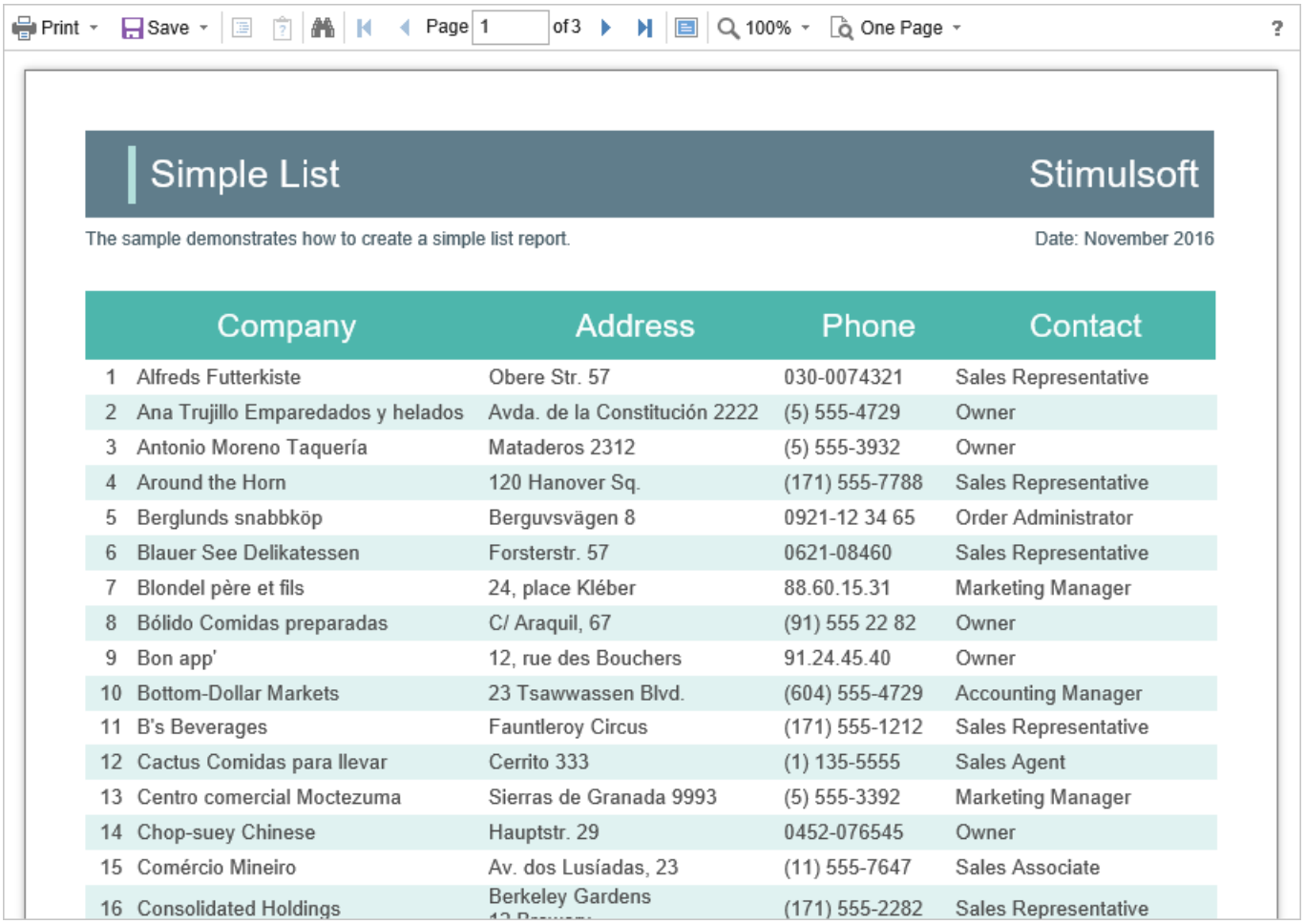
To show a report, you should add the **StiWebViewer** component to the ASPX page and assign a loaded report to it.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Reports/SimpleList.mrt"));
   // report.Load(Server.MapPath("Reports/Dashboard.mrt"));

   StiWebViewer1.Report = report;
}
...
```



	Company	Address	Phone	Contact
1	Alfreds Futterkiste	Obere Str. 57	030-0074321	Sales Representative
2	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	(5) 555-4729	Owner
3	Antonio Moreno Taquería	Mataderos 2312	(5) 555-3932	Owner
4	Around the Horn	120 Hanover Sq.	(171) 555-7788	Sales Representative
5	Berglunds snabbköp	Berguvsvägen 8	0921-12 34 65	Order Administrator
6	Blauer See Delikatessen	Forsterstr. 57	0621-08460	Sales Representative
7	Blondel père et fils	24, place Kléber	88.60.15.31	Marketing Manager
8	Bólido Comidas preparadas	C/ Araquil, 67	(91) 555 22 82	Owner
9	Bon app'	12, rue des Bouchers	91.24.45.40	Owner
10	Bottom-Dollar Markets	23 Tsawwassen Blvd.	(604) 555-4729	Accounting Manager
11	B's Beverages	Fauntleroy Circus	(171) 555-1212	Sales Representative
12	Cactus Comidas para llevar	Cerrito 333	(1) 135-5555	Sales Agent
13	Centro comercial Moctezuma	Sierras de Granada 9993	(5) 555-3392	Marketing Manager
14	Chop-suey Chinese	Hauptstr. 29	0452-076545	Owner
15	Comércio Mineiro	Av. dos Lusíadas, 23	(11) 555-7647	Sales Associate
16	Consolidated Holdings	Berkeley Gardens	(171) 555-2282	Sales Representative

Also, the **HTML5 Viewer** has a special **OnGetReport** event that you can use to assign a report to the viewer. In this case, you need to load the report in the event handler.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   OnGetReport="StiWebViewer1_GetReport">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_GetReport(object sender,
StiReportDataEventArgs e)
{
   StiReport report = new StiReport();
```

```
   report.Load(Server.MapPath("Reports/SimpleList.mrt"));

   e.Report = report;
}
...
```

Information

To assign a report, you should use the specified OnGetReport event. In this case, if the report object is lost for any reason in the cache of the server or session, the client part of the viewer initiates this event, and the report preview will be continued.

If the report is not rendered before showing, the **HTML5 Viewer** component will automatically generate it. So, you can use various types of reports to display the report - report templates, rendered reports, and reports as classes.

Default.aspx.cs

```
...
protected void StiWebViewer1_GetReport(object sender,
StiReportDataEventArgs e)
{
   StiReport report = new StiReport();
   report.LoadDocument(Server.MapPath("Reports/SimpleList.mdc"));

   e.Report = report;
}
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_GetReport(object sender,
StiReportDataEventArgs e)
{
   e.Report = new StiReportCompiledClass();
}
...
```

Since the dashboard is not a static document and requires data to work, the format of the rendered MDC document is not available for it. Instead, it is possible to use a snapshot of the report in the MRT format, which contains all the data necessary for

the dashboard to work and can be correctly displayed in the viewer.

Default.aspx.cs

```
...
protected void StiWebViewer1_GetReport(object sender,
StiReportDataEventArgs e)
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Reports/Snapshot.mrt"));

   e.Report = report;
}
...
```

Loading custom fonts

You can load custom fonts using the **StiFontCollection** class, having specified the file that contains a font. To do this, you should call the static method in the constructor to load a font.

Default.aspx.cs

```
...
public partial class _Default : Page
{
   static _Default()
   {
      Stimulsoft.Base.StiFontCollection.AddFontFile(Server.MapPath("fonts/
      my-font/font-name.ttf"));
   }
}
...
```

3.1.4 Connecting Data

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Data for rendering a report can be connected in various ways. The easiest way is to store connection settings in the report template. Also, data can be connected from the code. You can do this before assigning the report to the viewer.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
   DataSet ds = new DataSet();
   ds.ReadXml(Server.MapPath("Reports/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(Server.MapPath("Reports/TwoSimpleLists.mrt"));
   report.Dictionary.Databases.Clear();
   report.RegData("Demo", ds);

   StiWebViewer1.Report = report;
}
...
```

To connect report data, you can use the special **OnGetReportData** event, which will be called before the report is rendered.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   OnGetReportData="StiWebViewer1_GetReportData">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_GetReportData(object sender,
StiReportDataEventArgs e)
{
   DataSet dataSet = new DataSet();
   dataSet.ReadXml(Server.MapPath("Reports/Demo.xml"));
   e.Report.RegData(dataSet);
}
...
```

SQL data sources

The connection parameters to the SQL data source and any other ones can be stored in the report template. Suppose you want to set the connection parameters from the code before rendering the report (for example, for security reasons or depending on the authorized user). In that case, you can use the example below.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   OnGetReportData="StiWebViewer1_GetReportData">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_GetReportData(object sender,
StiReportDataEventArgs e)
{
   OracleConnection connection = new OracleConnection("Data
   Source=Oracle8i;Integrated Security=yes");
   connection.Open();
   OracleDataAdapter adapter = new OracleDataAdapter();
   adapter.SelectCommand = new OracleCommand("SELECT * FROM Products",
   connection);

   DataSet dataSet = new DataSet("productsDataSet");
   adapter.Fill(dataSet, "Products");

   e.Report.RegData("Products", dataSet);
}
...
```

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to change the connection string for MS SQL, adjust the query, set the query timeout for the already created connection and data sources in the report.

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
```

```
    StiReport report = new StiReport();
    report.Load(Server.MapPath("Report.mrt"));
    ((StiSqlDatabase)
    report.Dictionary.Databases["Connection"]).ConnectionString = @"Data
    Source=server;Integrated Security=True;Initial Catalog=DataBase";
    ((StiSqlSource)
    report.Dictionary.DataSources["DataSourceName"]).SqlCommand = "select *
    from Table where Column = 100";
    ((StiSqlSource)
    report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

    StiWebViewer1.Report = report;
}
...
```

Information

For SQL data sources of other types, the connection is created similarly, and an adapter corresponding to the data source type is connected. For example, for the MS SQL data source, you should connect SqlDataAdapter. For OLE DB - OleDbDataAdapter is required. Also, you should specify a connection string that matches the connection type.

You can also use data for designing reports and dashboards obtained from OData storage. In this case, you can do the authorization using a username, user password, or token. Authorization parameters are specified in the connection string to the OData storage using the ";" separator.

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{

    var report = new StiReport();

    //Authorization using a user account
    var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
    services.odata.org/V4/Northwind/
    Northwind.svc;AddressBearer=adress;UserName=UserName;Password=Password;C
    lient_Id=Your Client ID", false, null);

    //Authorization using a user token
    var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
    services.odata.org/V4/Northwind/Northwind.svc;Token=Enter your token",
    false, null);
```

```
    report.Dictionary.Databases.Add(oDataDatabase);
    oDataDatabase.Synchronize(report);

    //Query with data filter
    ((StiSqlSource)report.Dictionary.DataSources["Products"]).SqlCommand =
    "Products?$filter=ProductID eq 2";

    StiWebViewer1.Report = report;
}
...
```

The table below shows the connection string templates for different types of data sources.

Data Source	Samples of Connection Strings
MS SQL	Integrated Security=False; Data Source=myServerAddress;Initial Catalog=myDataBase; User ID=myUsername; Password=myPassword;
MySQL	Server=myServerAddress; Database=myDataBase;UserId=myUsername; Pwd=myPassword;
ODBC	Driver={SQL Server}; Server=myServerAddress;Database=myDataBase ; Uid=myUsername; Pwd=myPassword;
OLE DB	Provider=SQLOLEDB.1; Integrated Security=SSPI;Persist Security Info=False; Initial Catalog=myDataBase;Data Source=myServerAddress
Oracle	Data Source=TORCL;User Id=myUsername;Password=myPassword;
MS Access	Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Password=pass;Data Source=C:\ \myAccessFile.accdb;
PostgreSQL	Server=myServerAddress; Port=5432; Database=myDataBase;User Id=myUsername; Password=myPassword;

Firebird	User=SYSDBA; Password=masterkey; Database=SampleDatabase.fdb;DataSource=myServerAddress; Port=3050; Dialect=3; Charset=NONE;Role=; Connection lifetime=15; Pooling=true; MinPoolSize=0;MaxPoolSize=50; Packet Size=8192; ServerType=0;
SQL CE	Data Source=c:\MyData.sdf; Persist Security Info=False;
SQLite	Data Source=c:\mydb.db; Version=3;
DB2	Server=myAddress:myPortNumber;Database=myDataBase;UID=myUsername;PWD=myPassword; Max Pool Size=100;Min Pool Size=10;
Infomix	Database=myDataBase;Host=192.168.10.10;Server=db_engine_tcp;Service=1492;Protocol=onsoctcp;UID=myUsername;Password=myPassword;
Sybase	Data Source=myASEserver;Port=5000;Database=myDataBase;Uid=myUsername;Pwd=myPassword;
Teradata	Data Source=myServerAddress;User ID=myUsername;Password=myPassword;
VistaDB	Data Source=D:\folder \myVistaDatabaseFile.vdb4;Open Mode=ExclusiveReadWrite;
Universal(dotConnect)	Provider=Oracle;direct=true;data source=192.168.0.1;port=1521;sid=sid;user=user; password=pass
MongoDB	mongodb://<user>:<password>@localhost/test
OData	http://services.odata.org/v3/odata/OData.svc/
Other...	The table shows the most commonly used templates for the connection string. You can view various connection string options at the special website.

Data from XML, JSON, Excel files

Connecting to XML and JSON data sources can be stored in the report template. If you want to specify data files from the code, you can use the example below.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
  OnGetReportData="StiWebViewer1_GetReportData">
</cc1:StiWebViewer>
...
```

Default.aspx.cs (XML file)

```
...
protected void StiWebViewer1_GetReportData(object sender,
StiReportDataEventArgs e)
{
  DataSet data = new DataSet();
  data.ReadXml(Server.MapPath("Data/Demo.xml"));

  e.Report.RegData(data);
}
...
```

Default.aspx.cs (JSON file)

```
...
protected void StiWebViewer1_GetReportData(object sender,
StiReportDataEventArgs e)
{
  DataSet data
  = StiJsonToDataSetConverterV2.GetDataSetFromFile(Server.MapPath("Data/
  Demo.json"));

  e.Report.RegData(data);
}
...
```

Information

The viewer has the possibility of obtaining data from an Excel file. To do this, you can use the following method.

```
DataSet dataSet = StiExcelConnector.Get().GetDataSet(new StiExcelOptions(ar
```

3.1.5 Localization

The **HTML5 Viewer** component supports the complete localization of its interface. To localize the report viewer interface, use the special **Localization** property. As a value of this property, you should specify the path to the localization XML file (relative or absolute).

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
  Localization="Localization/en.xml">
</cc1:StiWebViewer>
...
```

When you load the report viewer, the localization file will be loaded automatically.

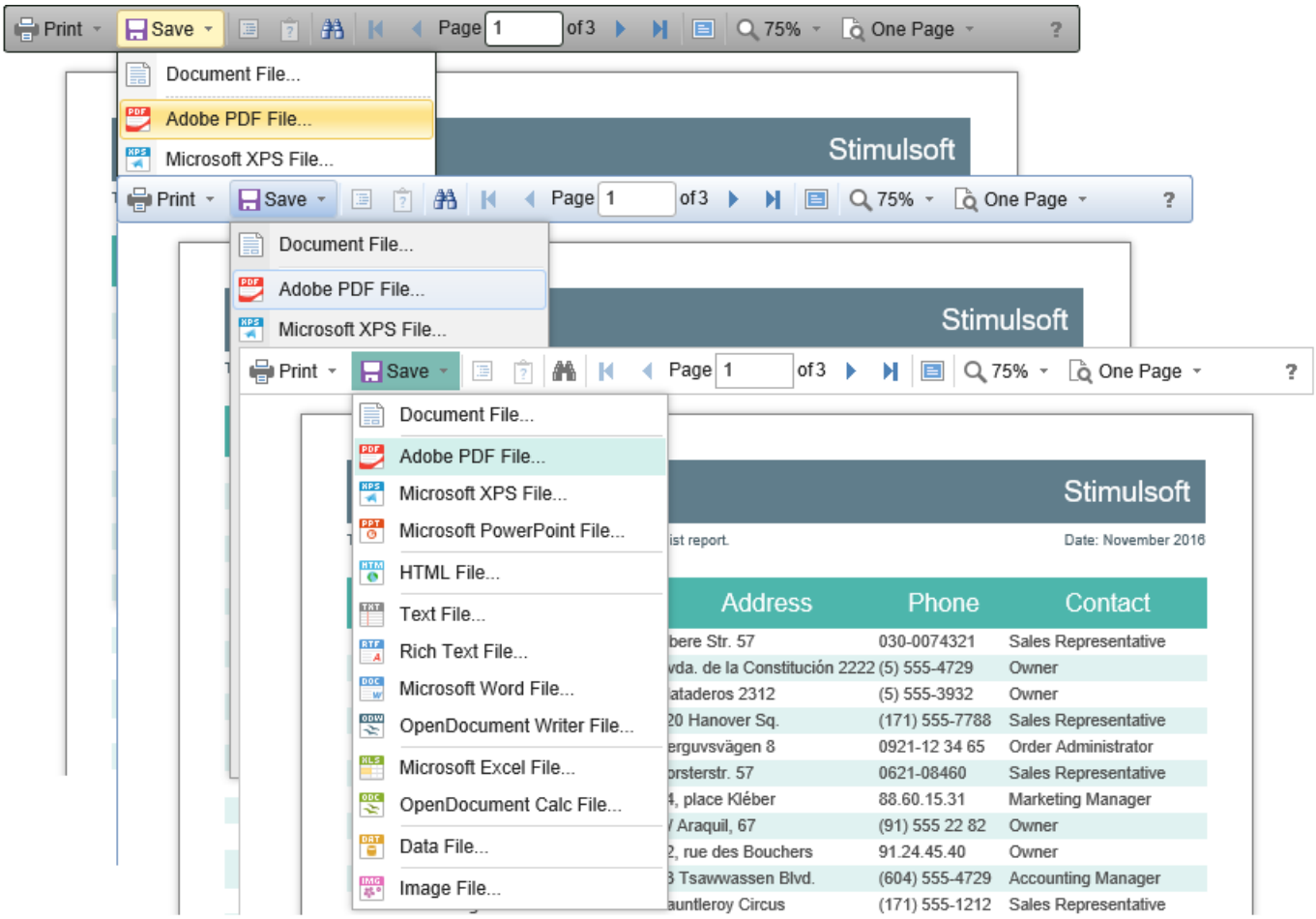
3.1.6 Using Themes

In the **HTML5 Viewer** component, you can change the appearance of visual controls. To change the theme, you should use the **Theme** property.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
  Theme="Office2022WhiteTeal">
</cc1:StiWebViewer>
...
```

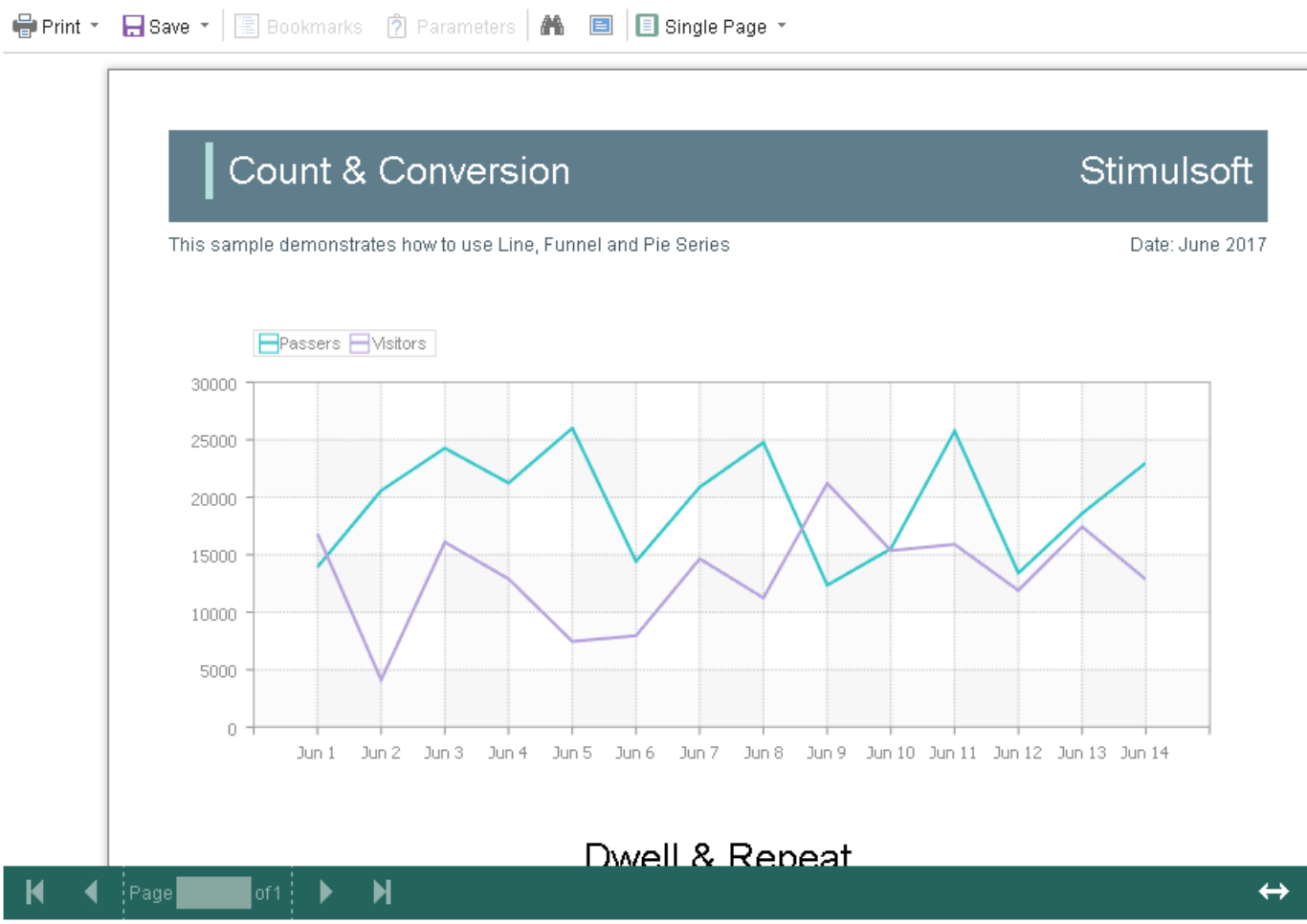
There are currently **8 themes** available with different color accents. As a result, **more than 60** variants of the appearance are available. This allows you to customize the appearance of the viewer for almost any design of the Web project.



By default, the viewer has only the top toolbar on which all the report controls are located. If necessary, the toolbar can be split into top and bottom parts. The top panel will contain the menu for printing and exporting the report and the buttons for working with parameters and bookmarks. The bottom panel will contain controls to switch between the report pages and setting the zoom of pages. To enable this mode, enable the **ToolbarDisplayMode** property. It has values **Simple** and **Separated**.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   ToolbarDisplayMode="Separated"
   ScrollbarsMode="true">
</cc1:StiWebViewer>
...
```



In addition, it is possible to set the appearance parameters for the main elements of the viewer. For example, you can change the font and color of the control panel inscriptions of the viewer, set the background of the viewer, set the color of page borders, etc. Below is a list of available properties that change the appearance of the viewer and their default values.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   BackgroundColor="White"
   ShowPageShadow="true"
   PageBorderColor="Gray"
   ToolbarBackgroundColor="Empty"
   ToolbarBorderColor="Empty"
   ToolbarFontColor="Empty"
   ToolbarFontFamily="Arial">
</cc1:StiWebViewer>
...
```

3.1.7 Basic Features

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The main features of the viewer include the following operations - showing reports, switching between report pages, changing the scale, and displaying the preview mode. All specified operations are performed in the AJAX mode without restarting the browser page. No special options or events are required to execute them.

The report viewer has a special **OnViewerEvent** event that will be called when the viewer is running. In this event, you can find out the type of action that the viewer is currently executing and get all viewer parameters passed to the server-side.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   OnViewerEvent="StiWebViewer1_ViewerEvent">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_ViewerEvent(object sender, StiViewerEventArgs
e)
{
   StiAction action = e.Action;
   StiRequestParams parameters = e.RequestParams;
}
...
```

3.1.8 Printing Reports

Information

Please note that the print option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component provides several options for printing a report. Each has its advantages and disadvantages.

Print to PDF
Printing will be done by exporting the report to **PDF**. The advantages are greater accuracy of positioning and printing of the report elements compared to other printing options. Among the drawbacks is the mandatory presence of a plug-in installed in a web browser for viewing PDF files (modern browsers have embedded PDF viewer and printer).

Print with Preview
The report will be printed in a separate pop-up browser window in **HTML**. The report can be previewed and then sent to the printer or copied to another location as text or HTML code. Advantages - cross-browser compatibility when printing, no need to install special plug-ins. The disadvantage is the relatively low accuracy of the position of the report elements due to the peculiarities of the implementation of HTML formatting.

Print without Preview
The report will be printed directly to the printer without preview. After selecting this menu item, the system print dialog is displayed. Since printing in this mode is carried out in the **HTML** format, the print quality is similar to printing a report with a preview.

Information

When printing to the **HTML format**, you should check the compliance of report page settings to printer parameters (paper size, orientation, margins, indents), and check your browser print settings, such as margins, headers, footers, watermarks printing, color printing.

Report printing events

To perform any actions, a special **OnPrintReport** event is assigned before the report is printed. In this event, you can get the type of report printing, the report itself, and

the export report settings in case of printing to **PDF**.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
    OnPrintReport="StiWebViewer1_PrintReport">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_PrintReport(object sender,
StiPrintReportEventArgs e)
{
   StiPrintAction action = e.PrintAction;
   StiReport report = e.Report;
   StiExportSettings settings = e.Settings;
}
...
```

Print setup

If you choose to print a report in the viewer panel, a menu with printing options is displayed. The **HTML5 Viewer** component can force the required printing mode. To do this, set the **PrintDestination** property to one of the following values.

- **Default** – the menu will be displayed (the default property value);
- **Pdf** – print to the PDF format;
- **Direct** – printing to the HTML format directly to the printer, the system print dialog will be displayed;
- **WithPreview** – print to HTML format with preview in a pop-up window.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   PrintDestination="Default">
</cc1:StiWebViewer>
...
```

The **HTML5 Viewer** component can completely disable report printing. To do this, set the value of the **ShowPrintButton** property to **false**.

Default.aspx

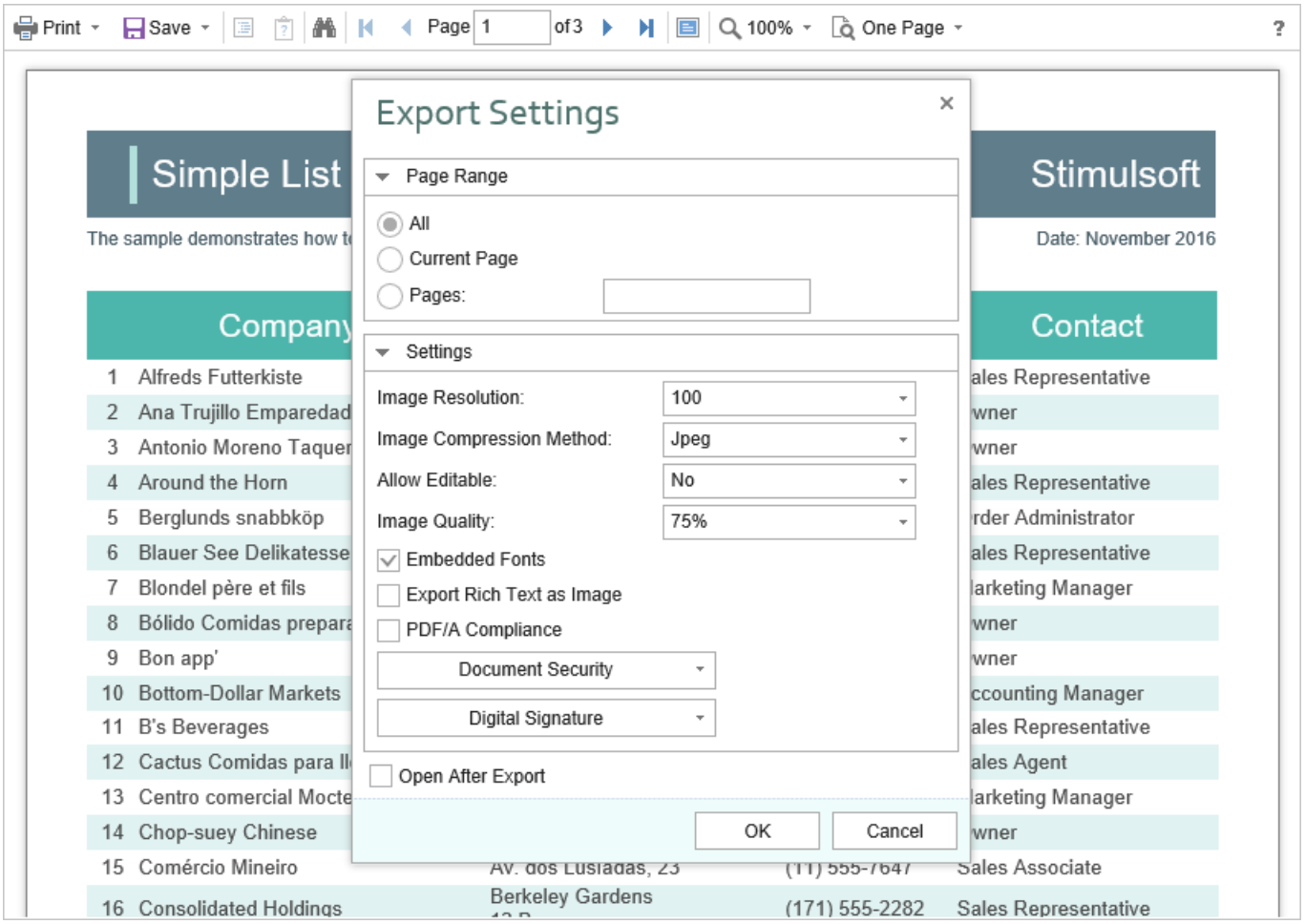
```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
    ShowPrintButton="false">
</cc1:StiWebViewer>
...
```

3.1.9 Exporting Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component allows you to export the displayed report to three dozen various formats, such as **PDF**, **HTML**, **Word**, **Excel**, **XPS**, **RTF**, **images**, **text**, and others. Export functions do not require additional settings for the viewer. You may export the dashboard to PDF, Excel, image files.



Export Events

To perform any actions, a special **OnExportReport** event is assigned before the report is exported. In this event, you can get the report export type, get the report itself, get the report export settings, and change them if necessary.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
  OnExportReport="StiWebViewer1_ExportReport">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_ExportReport(object sender,
```

```
StiExportReportEventArgs e)
{
   StiExportFormat format = e.Format;
   StiReport report = e.Report;
   StiExportSettings settings = e.Settings;
}
...
```

To perform any actions with an already exported report, the **OnExportReportResponse** event is used. This event will be called immediately before the file is saved. In this event, you can get the export format, the type of the Web content, and the name of the file to save. You can also get and, if necessary, change the byte stream of the final export file.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   OnExportReportResponse="StiWebViewer1_ExportReportResponse">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_ExportReportResponse(object sender,
StiExportReportResponseEventArgs e)
{
   StiExportFormat format = e.Format;
   string contentType = e.ContentType;
   string fileName = e.FileName;
   Stream stream = e.Stream;
}
...
```

Export Settings

Each report export format of the **HTML5 Viewer** component has a lot of settings, and each setting has its default values. Sometimes you need to set other default values. For this purpose, a special **DefaultExportSettings** property of the viewer is used. It is a container for all the default export settings.

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
```

```
{
   StiWebViewer1.DefaultExportSettings.ExportToPdf.ImageQuality = 0.75f;
   StiWebViewer1.DefaultExportSettings.ExportToPdf.ImageFormat =
   StiImageFormat.Color;
   StiWebViewer1.DefaultExportSettings.ExportToHtml.ExportMode =
   StiHtmlExportMode.Div;
}
...
```

When calling the required export through the Viewer menu, the Export Settings dialog box will be displayed. The values of the dialog box items will match the default export settings. If you change the settings in the dialog box and confirm the export of the report, the settings will be saved in the browser cookies, and upon the subsequent call, the export will be restored. The viewer allows you to disable saving the export settings if you always want to restore the default settings. To do this, just set the value of the **StoreExportSettings** property to **false**.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   StoreExportSettings="false">
</cc1:StiWebViewer>
...
```

Also, if required, you can completely hide export dialogs. Exporting will always be done with default settings. For this, it is enough to set the value of the **ShowExportDialog** property to **false**.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   ShowExportDialog="false">
</cc1:StiWebViewer>
...
```

The **HTML5 Viewer** component contains 30+ export formats, and sometimes you need to disable unwanted formats. This allows you to simplify UI and the use of the viewer. To disable unused export formats, it is enough to set the values for the corresponding properties of the viewer listed in the list below to **false**.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   ShowExportToDocument="true"
   ShowExportToPdf="true"
   ShowExportToXps="true"
   ShowExportToPowerPoint="true"
   ShowExportToHtml="true"
   ShowExportToHtml5="true"
   ShowExportToMht="true"
   ShowExportToText="true"
   ShowExportToRtf="true"
   ShowExportToWord2007="true"
   ShowExportToOpenDocumentWriter="true"
   ShowExportToExcel="true"
   ShowExportToExcelXml="true"
   ShowExportToExcel2007="true"
   ShowExportToOpenDocumentCalc="true"
   ShowExportToCsv="true"
   ShowExportToDbf="true"
   ShowExportToXml="true"
   ShowExportToDif="true"
   ShowExportToSylk="true"
   ShowExportToImageBmp="true"
   ShowExportToImageGif="true"
   ShowExportToImageJpeg="true"
   ShowExportToImagePcx="true"
   ShowExportToImagePng="true"
   ShowExportToImageTiff="true"
   ShowExportToImageMetafile="true"
   ShowExportToImageSvg="true"
   ShowExportToImageSvgz="true">
</cc1:StiWebViewer>
...
```

The **HTML5 Viewer** component can completely disable the export menu. To do this, set the value of the **ShowSaveButton** property to **false**.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   ShowSaveButton="false">
</cc1:StiWebViewer>
...
```

3.1.10 Viewing Modes

The **HTML5 Viewer** component has two modes for displaying reports - with and without scrollbars. By default, the view mode without scrollbars is set. To enable the

scrollbar view mode, set the value of the **ScrollbarsMode** property to **true**.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   ScrollbarsMode="false">
</cc1:StiWebViewer>
...
```

In the first mode (without scrollbars), the viewer displays a page or report as a whole, automatically stretching the preview space. If the width and height are specified, the viewer will truncate the page that is out of bounds. In the second mode, unlike the first one, when the page is out of bounds of the viewer's size, no truncation will be performed. Scrollbars will appear, using which you can view the entire page or report.

Information

In the report mode with scrollbars, you should set the height of the viewer. Otherwise, the default height will be set to **650 pixels**.

The **HTML5 Viewer** component provides the full-screen report or dashboard mode. By default, the standard view mode is enabled, the viewer has the specified dimensions in the settings. To enable the full-screen mode, set the **FullScreenMode** property to **true**.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   FullScreenMode="false">
</cc1:StiWebViewer>
...
```

Also, to enable or disable the full-screen mode, you can use the corresponding button on the control panel of the viewer.

The **HTML5 Viewer** component has three modes to display reports - page-by-page,

entire report, and tabular display of report pages. To control the modes, the **ViewMode** property is used. It can have one of the specified values - **SinglePage**, **Continuous**, **MultiplePages**.

Default.aspx

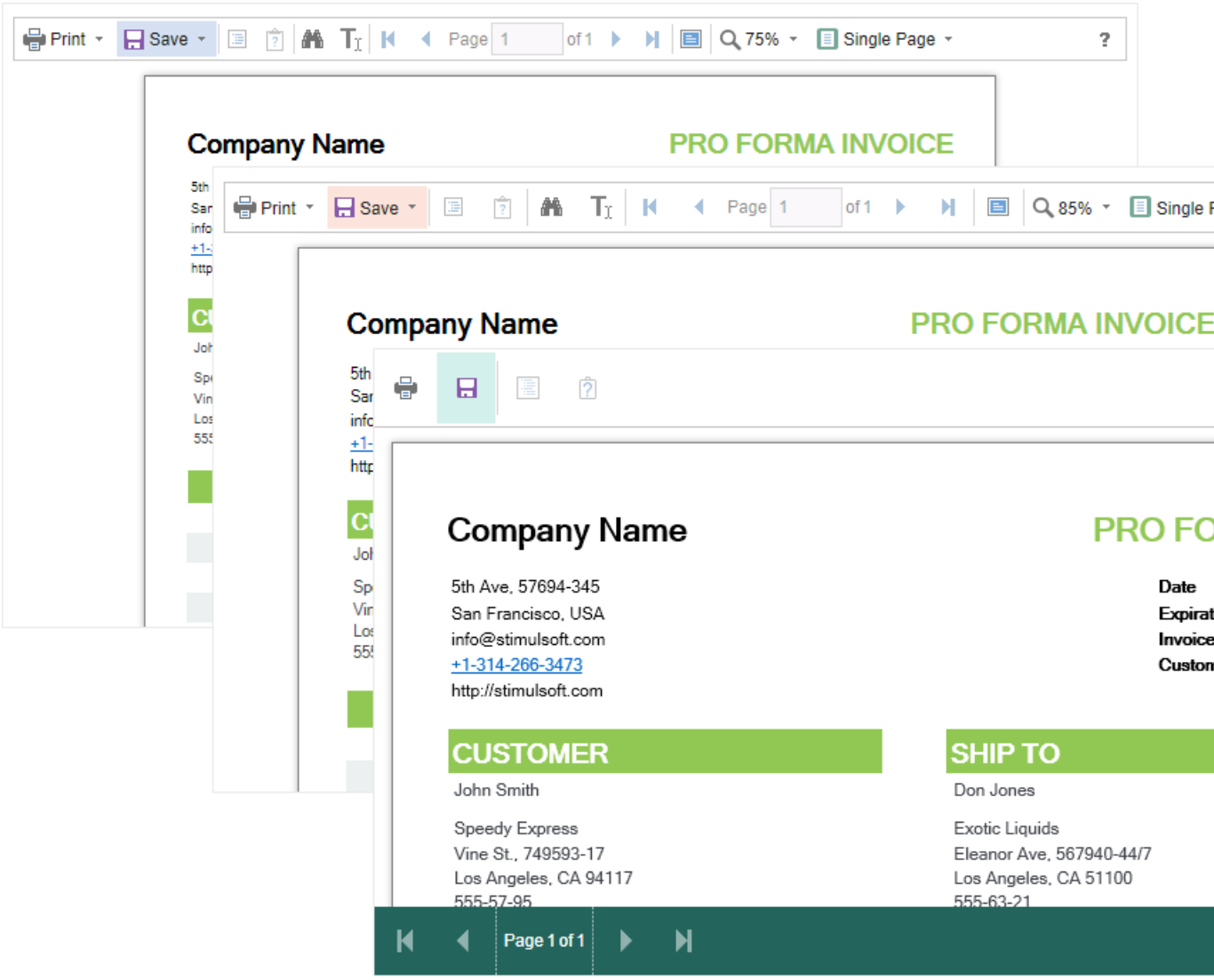
```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
    ViewMode="OnePage">
</cc1:StiWebViewer>
...
```

HTML5 Viewer component supports interaction on a regular PC display and works with a touchscreen of screens and mobile devices. **InterfaceType** property allows controlling the interface modes. The property can have one of the following values:

- **Auto** – the viewer's interface is determined automatically depending on the device the report is displayed on. That is the default value.
- **Mouse** – the standard interface with a mouse control will be used for all the screen types.
- **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability.
- **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with smartphones.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
  InterfaceType="Auto">
</cc1:StiWebViewer>
...
```



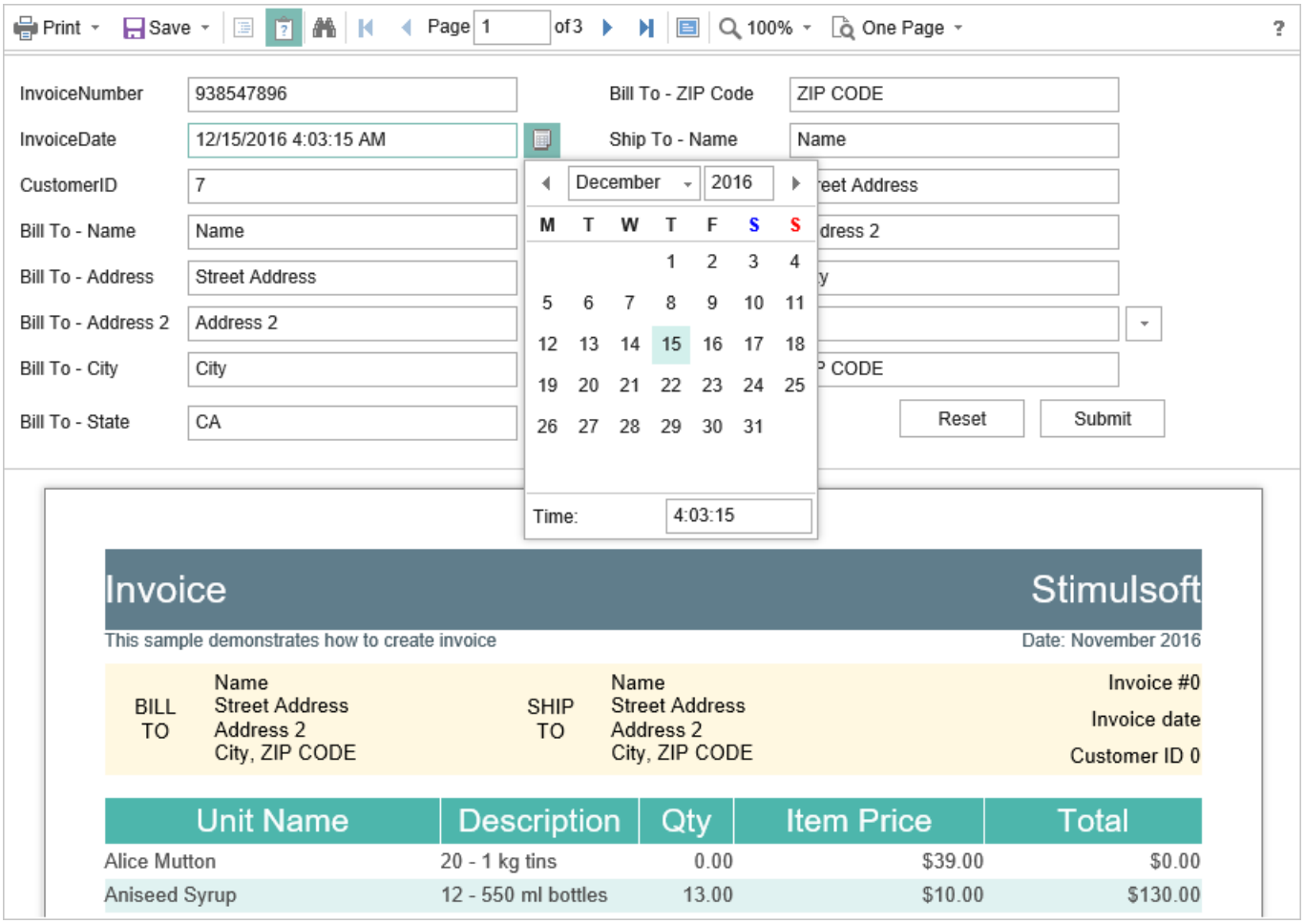
3.1.11 Work with Parameters

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To work with report parameters in the **HTML5 Viewer**, there is a special settings panel. To add a parameter to the panel, you need to define a variable in a report requested by the user. When viewing a report in the viewer, such a variable will be

automatically added to the settings panel. It supports all types of report variables (normal variables, date and time, borders, lists, etc.).



To perform any action before applying parameters, a special **OnInteraction** event will be called when there are some interactive activities in the viewer. When you use the options panel, the type of action will have the **Variables** value.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
  OnInteraction="StiWebViewer1_Interaction">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_Interaction(object sender,
StiReportDataEventArgs e)
{
   if (e.Action == StiAction.Variables)
   {
      // The values of the variables passed from the client
      Hashtable variables = e.RequestParams.Interaction.Variables;
   }
}
...
```

When working with parameters is not required, you can disable this feature. For this, you can use the **ShowParametersButton** property. Set this property to **false** in this case.

Default.aspx

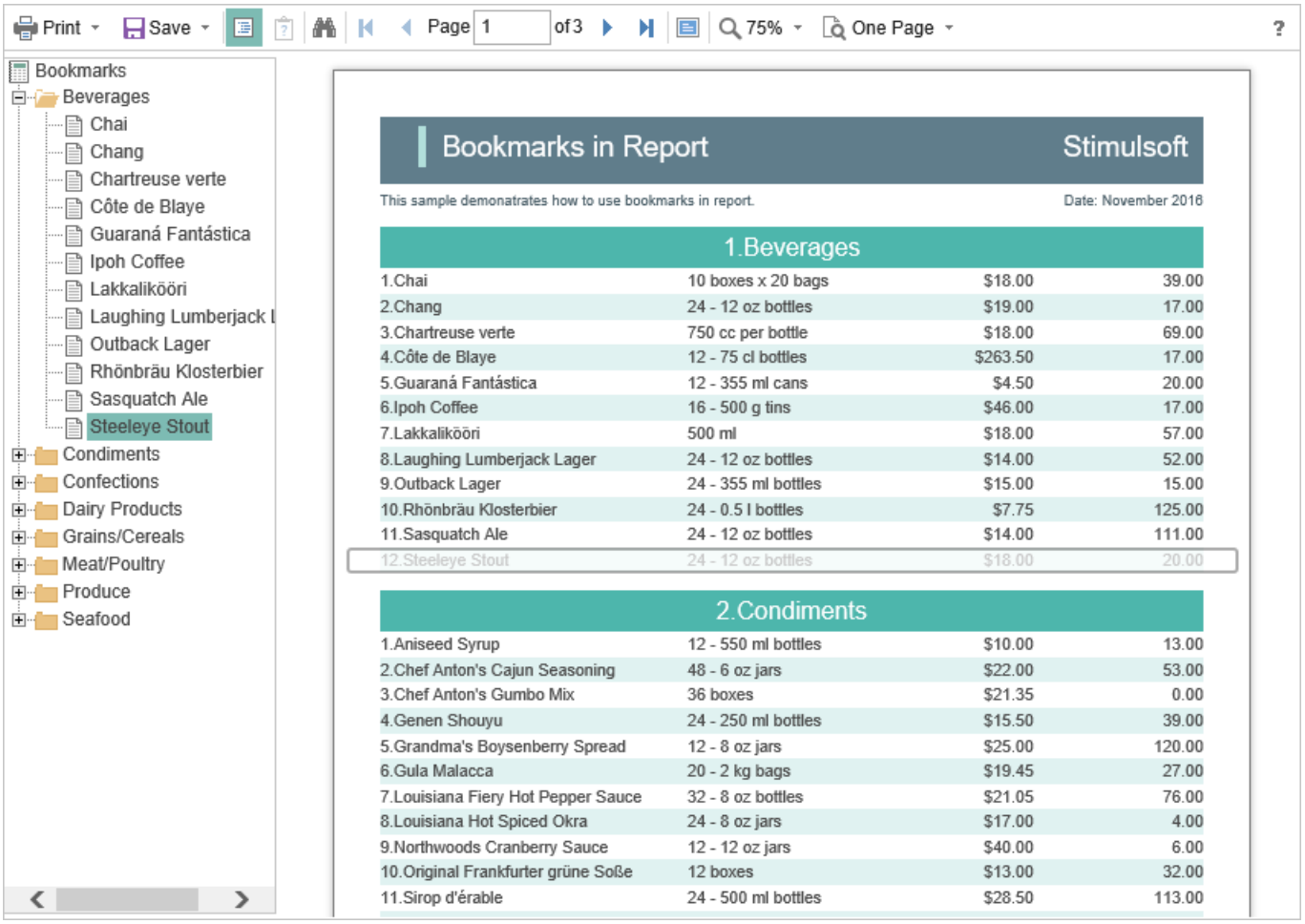
```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   ShowParametersButton="false">
</cc1:StiWebViewer>
...
```

Information

With this viewer configuration, the parameters panel will not be displayed, even if the parameters are present in the report.

3.1.12 Work with Bookmarks

The **HTML5 Viewer** component supports report bookmarks. A panel with bookmarks will be displayed when displaying such a report on the left side of the page. When you select a bookmark of the report, the viewer will carry out an automatic transition to the specified page, and the report item with a bookmark is highlighted.



By default, the bookmarks bar width is 180 pixels. The **HTML5 Viewer** component allows you to change this value. For this, the **BookmarksTreeWidth** property, which value is specified in pixels, is used.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
    BookmarksTreeWidth="200">
</cc1:StiWebViewer>
...
```

If you do not need to work with report bookmarks, you can disable this option. For this, set the **ShowBookmarksButton** property to **false**.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
    ShowBookmarksButton="false">
</cc1:StiWebViewer>
...
```

Information

In this case, report bookmarks will not be displayed, even if they are present in the displayed report. This property does not affect printing and exporting reports.

When printing a report with bookmarks, the bookmark tree will be hidden. If you want to print bookmarks with the report, it is necessary to set the **BookmarksPrint** property to **true**.

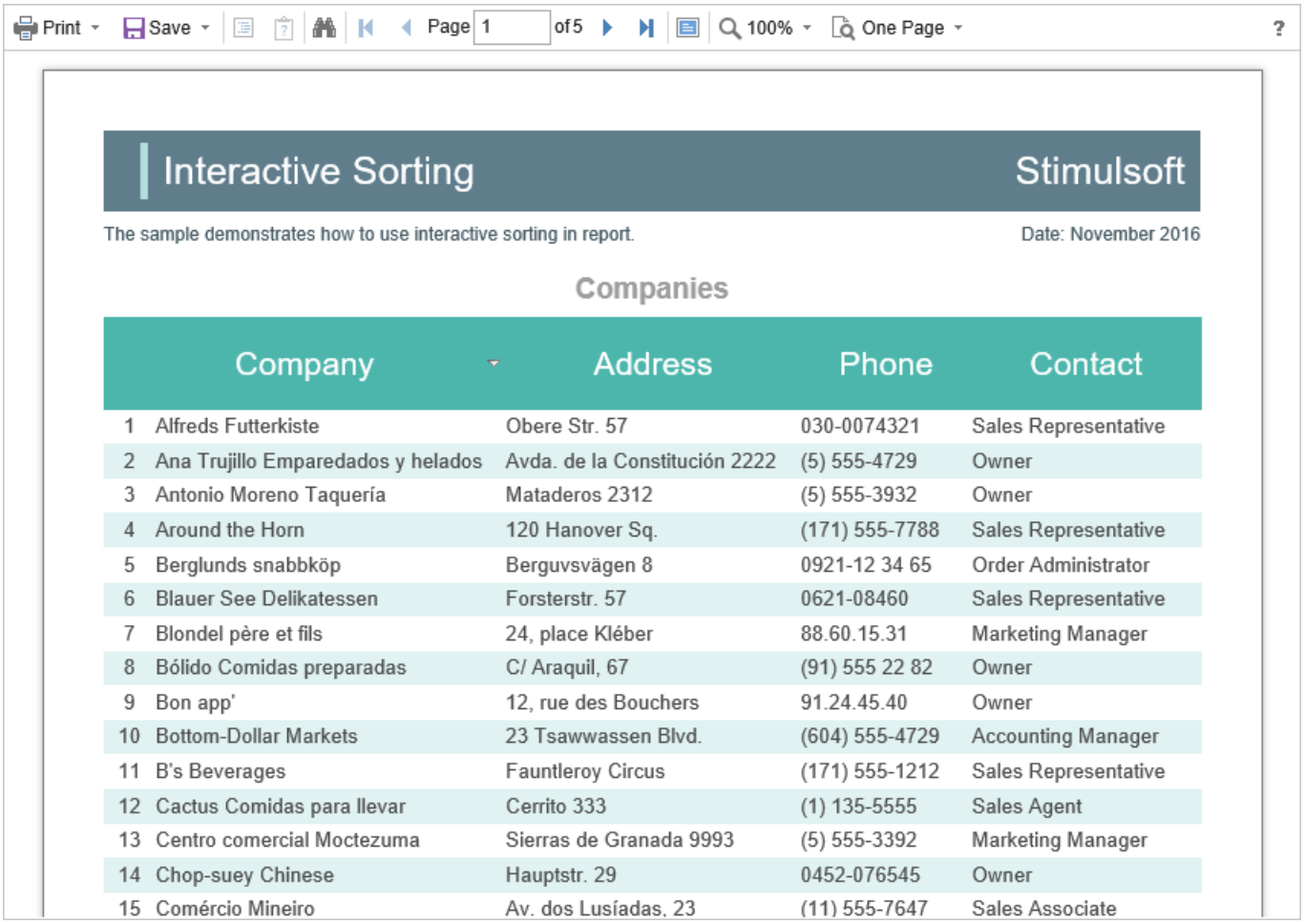
Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
    BookmarksPrint="true">
</cc1:StiWebViewer>
...
```

3.1.13 Dynamic Sorting, Collapsing, and Drill-Down

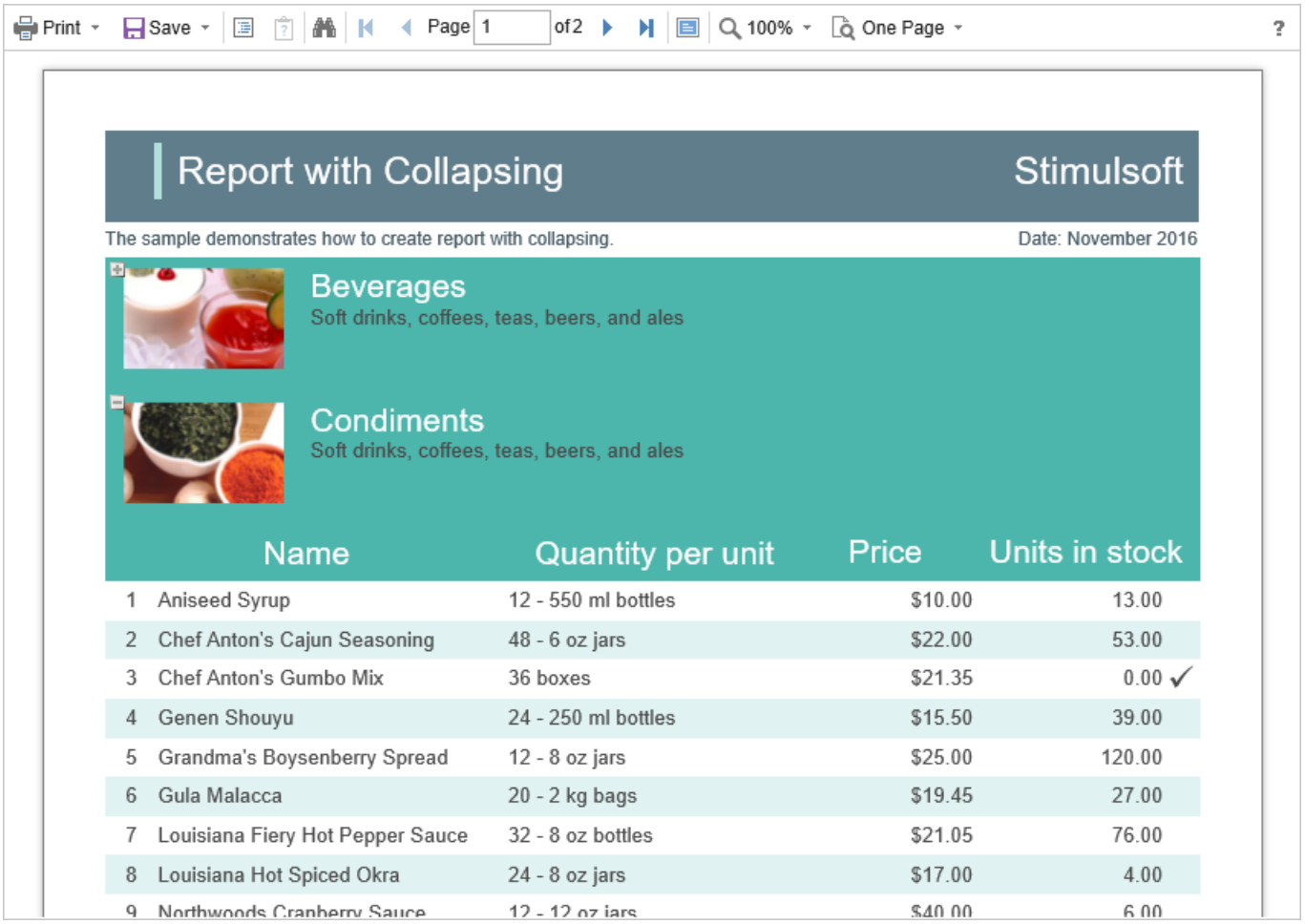
The **HTML5 Viewer** component supports dynamic sorting, collapsing, and drill-down of reports. Dynamic sorting provides the ability to change the direction of sorting in a rendered report. To do this, click on the component that has dynamic sorting enabled. Dynamic sorting is carried out in the following directions - **Ascending** and **Descending**. Each time when you click the component, the sorting direction is reversed.

Multi-level sorting is allowed in the report. To do this, hold down the **Ctrl** key and sequentially click on the sorted components in the report. To reset sorting, you can click on any sorted component without holding down the **Ctrl** key.

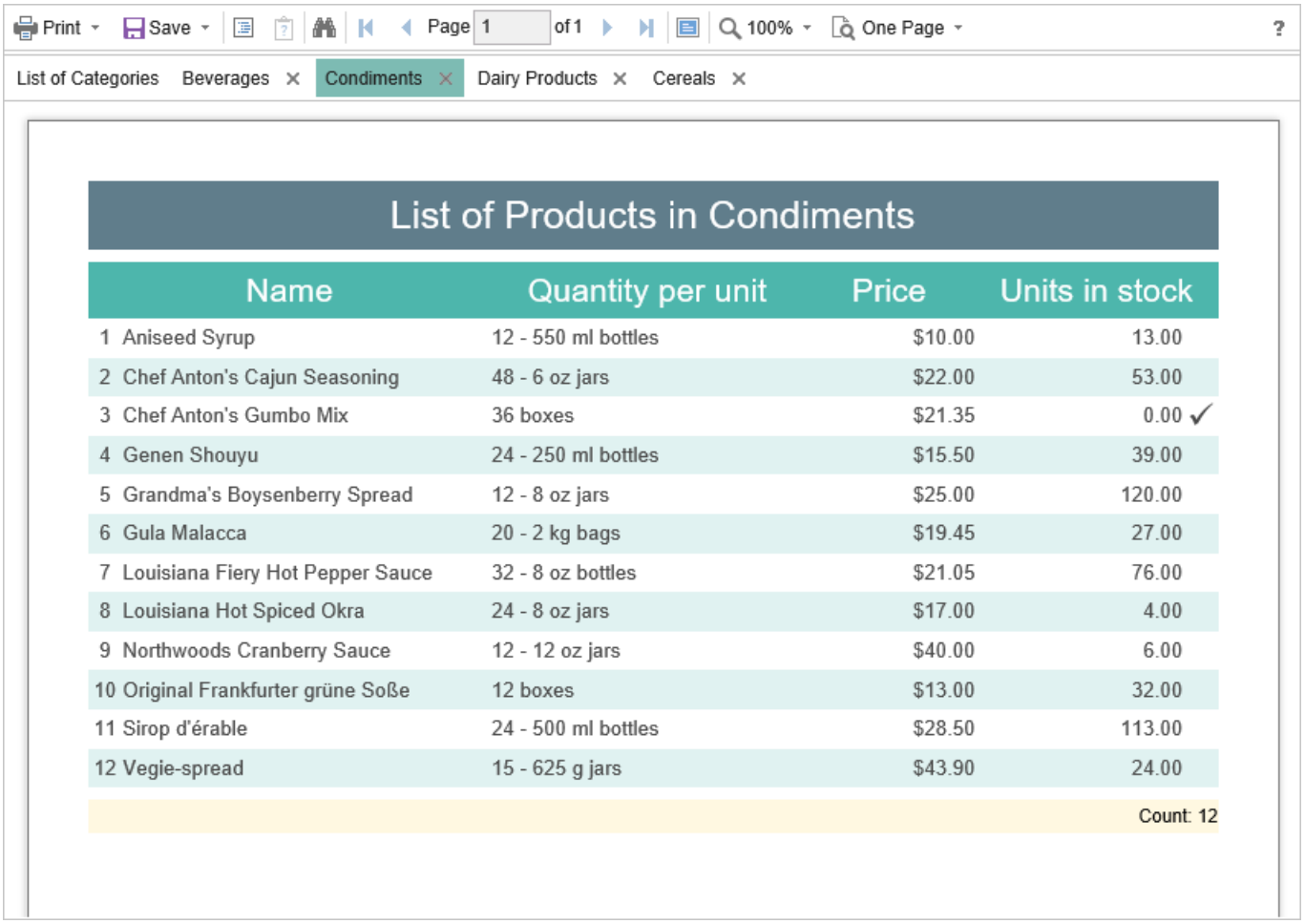


	Company	Address	Phone	Contact
1	Alfreds Futterkiste	Obere Str. 57	030-0074321	Sales Representative
2	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	(5) 555-4729	Owner
3	Antonio Moreno Taquería	Mataderos 2312	(5) 555-3932	Owner
4	Around the Horn	120 Hanover Sq.	(171) 555-7788	Sales Representative
5	Berglunds snabbköp	Berguvsvägen 8	0921-12 34 65	Order Administrator
6	Blauer See Delikatessen	Forsterstr. 57	0621-08460	Sales Representative
7	Blondel père et fils	24, place Kléber	88.60.15.31	Marketing Manager
8	Bólido Comidas preparadas	C/ Araquil, 67	(91) 555 22 82	Owner
9	Bon app'	12, rue des Bouchers	91.24.45.40	Owner
10	Bottom-Dollar Markets	23 Tsawwassen Blvd.	(604) 555-4729	Accounting Manager
11	B's Beverages	Fauntleroy Circus	(171) 555-1212	Sales Representative
12	Cactus Comidas para llevar	Cerrito 333	(1) 135-5555	Sales Agent
13	Centro comercial Moctezuma	Sierras de Granada 9993	(5) 555-3392	Marketing Manager
14	Chop-suey Chinese	Hauptstr. 29	0452-076545	Owner
15	Comércio Mineiro	Av. dos Lusíadas, 23	(11) 555-7647	Sales Associate

A report with dynamic collapsing is an interactive report in which blocks can collapse/expand their content when you click on the block title. Report elements, which can be collapsed/expanded, are indicated by special icons - **[-]** or **[+]**.



When using drill-down, under the main panel of the viewer, the drill-down panel with tabs for drill-down reports will be displayed. The currently displayed report will be highlighted.



To work with dynamic sorting, collapsing, and drill-down reports, no additional viewer settings are required. To perform any actions before the sorting, collapsing, or drill-down of the report, a special **OnInteraction** event is used. It will be called when interacting with the viewer. For each type of interactivity, the viewer has a certain type of action.

- **Sorting** – when using column sorting;
- **DrillDown** – when using drill-down;
- **Collapsing** – when using collapsing.

Default.aspx

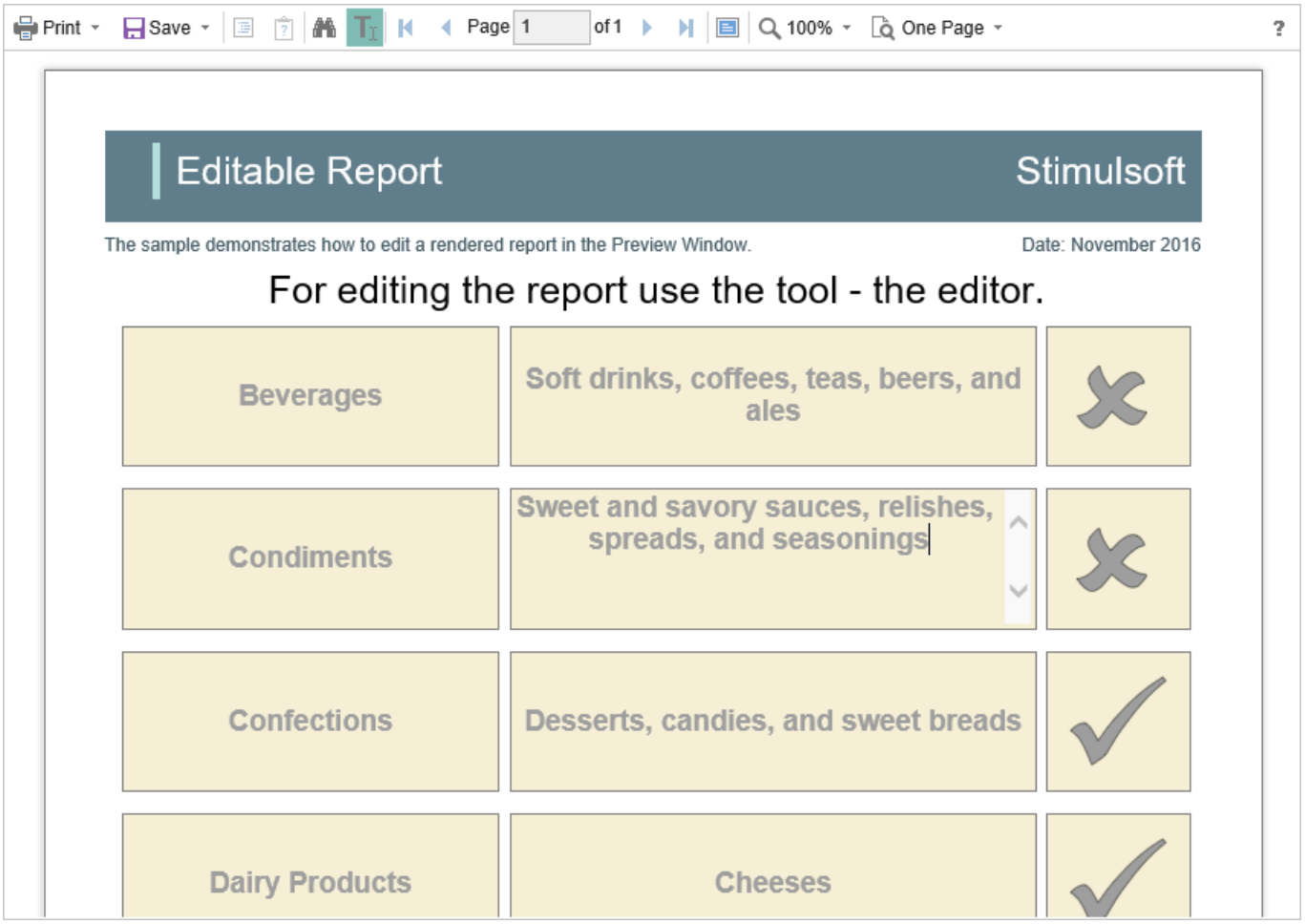
```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   OnInteraction="StiWebViewer1_Interaction">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_Interaction(object sender,
StiReportDataEventArgs e)
{
   switch (e.Action)
   {
      case StiAction.Sorting:
        break;

      case StiAction.DrillDown:
        break;

      case StiAction.Collapsing:
        break;
   }
}
...
```

3.1.14 Editing Report

The **HTML5 Viewer** component has the ability to edit report items, such as text boxes and checkboxes. You should mark the required components as editable in the report template for the editing to be possible. After displaying a report in the viewer, you need to click the corresponding button on the viewer panel to start editing. After editing, it is necessary to click the button once more, and all changes will be applied to the report.



For the report edit mode, no special settings of the viewer are required.

Information

The edited settings will be applied when you print or export a report, and the original report remains unchanged. After restarting the viewer, all the values will be returned to the initial ones.

3.1.15 Sending Report by Email

Information

Please note that the **Send Report by Email** option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component provides the ability to send reports by email. To activate this feature, you should set the **ShowSendEmailButton** property of the viewer to **true** and define the **OnEmailReport** event handler.

Default.aspx

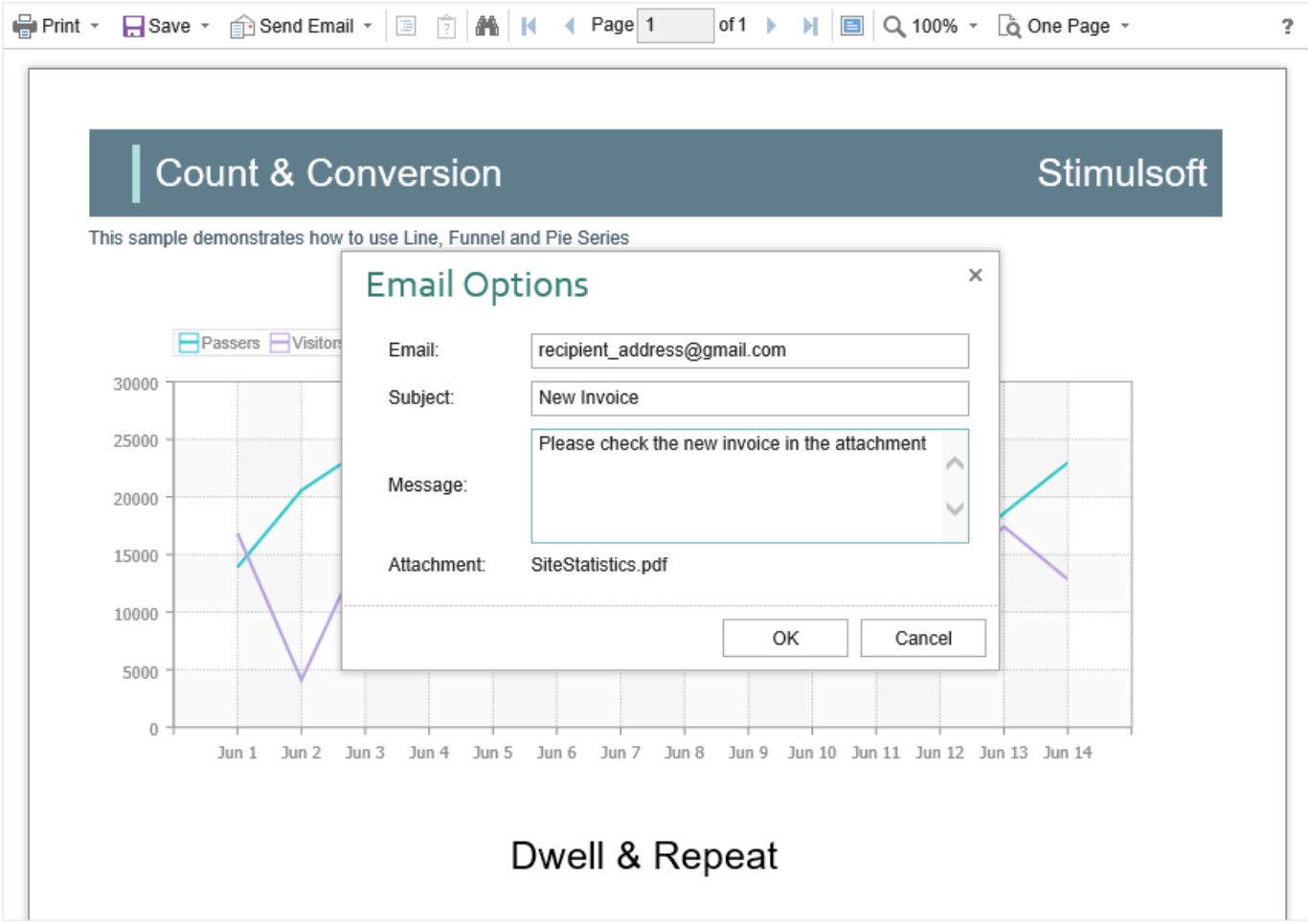
```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   ShowSendEmailButton="true"
   OnEmailReport="StiWebViewer1_EmailReport">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_EmailReport(object sender,
StiEmailReportEventArgs e)
{
   StiExportFormat format = e.Format;
   StiReport report = e.Report;
   StiExportSettings settings = e.Settings;
   StiEmailOptions options = e.Options;

   // Passed from the viewer, can be checked and changed
   // options.AddressTo = "";
   // options.Subject = "";
   // options.Body = "";

   // Should be filled here
   options.AddressFrom = "admin_address@test.com";
   options.Host = "smtp.test.com";
   options.Port = 465;
   options.UserName = "admin_address@test.com";
   options.Password = "admin_password";

   // options.CC.Add("email@test.com");
   // options.BCC.Add("email@test.com");
   // options.EnableSsl = true;
}
...
```

In the **OnEmailReport** event, you can find the export type, get the report, and get the report export settings and change them, if necessary. Also, in this event, you should set email parameters, such as sender address, server name, port number, user name, and password - all these parameters will be used to send an email.

When you send a report by email, the menu to select the attachment format is displayed. This matches the menu to select an export format. After choosing the format, the dialog to put send email parameters such as email recipient, subject, and message.



After confirmation of sending the email, the above described **OnEmailReport** event will be called. You can check and correct the data entered in this form. The exported report file will be attached to the email automatically.

The **HTML5 Viewer** component allows you to set default values for the send email form. The **DefaultEmailAddress**, **DefaultEmailSubject**, and **DefaultEmailMessage** properties can be used for this. By default, these properties are empty.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   DefaultEmailAddress="recipient_address@gmail.com"
   DefaultEmailSubject="New Invoice"
   DefaultEmailMessage="Please check the new invoice in the attachment"

   ShowSendEmailButton="true"
   OnEmailReport="StiWebViewer1_EmailReport">
</cc1:StiWebViewer>
...
```

3.1.16 Calling Designer from Viewer

The **HTML5 Viewer** component has the ability to call the report designer. The special **Design** button in the toolbar of the viewer (the button is disabled by default) should be used. To use this feature, you should set the **ShowDesignButton** property to **true** and define the **OnDesignReport** event handler.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   ShowDesignButton="true"
   OnDesignReport="StiWebViewer1_DesignReport">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_DesignReport(object sender,
StiReportDataEventArgs e)
{
   StiReport report = e.Report;
   this.Response.Redirect("Designer.aspx?report=" + report.ReportName);
}
...
```

Information

The viewer does not run the designer itself. It only calls the specified event and passes the previewed report as arguments. In the event, you can set the redirect to the ASPX page, on which the report designer is placed.

3.1.17 Caching

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component allows you to use the server cache to store rendered reports. If you do not use caching, you should load the report, connect data, and render it again every time you request a page. If you use caching, the previously rendered report will be loaded from the cache every time you refresh the page.

When using caching, it should be taken into account that every report saved in the cache takes up server memory and, with a large number of requests to reports, this can become a critical issue. Therefore, you need to choose between two options: low memory requirements but high in performance or low-performance requirements but high in memory.

You can manage caching with the following properties.

The CacheMode property

This property of the viewer enables caching and sets its type. It can take one of the following values, specified in the **StiServerCacheMode** enumeration:

- **None** – caching is disabled. Each action of the viewer requires loading the report from the file and, if it is a report template, then render it;
- **ObjectCache** – for caching, the server cache is used. The report object is saved in this cache (set by default);
- **StringCache** – for caching, the server cache is used. The report is saved as a packed string in this cache;
- **ObjectSession** – the current session, in which the report object is saved, is used for caching;
- **StringSession** – for caching, the current session is used. The report is saved as a packed string in this cache.

The CacheItemPriority property

This property sets the priority of the report stored in the server's cache. It affects the automatic clearing of the server memory in case of a lack of memory. The lower the priority is, the greater is the chance of removing information from memory.

The CacheTimeout property

This property specifies the amount of time in minutes for which you want to save the report in the server cache. If you use caching and the requested report is not found in the cache (the objects storage time has expired), then it will be requested again using a special **OnGetReport** event, then connect the report data and render it.

StiCacheHelper

The **HTML5 Viewer** component provides the ability to define your methods of working with report caching. For this purpose, a special class **StiCacheHelper** is used. It contains methods for obtaining a report from the cache and saving the report to the cache. It is necessary to create a new class inherited from **StiCacheHelper** and reload the above methods, which respectively have the names - **GetReport** and **SaveReport**.

Default.aspx.cs

```
...
public partial class _Default : Page
{
   public class StiMyCacheHelper : StiCacheHelper
   {
      public override StiReport GetReport(string guid)
      {
         string path =
         Path.Combine(HttpContext.Current.Server.MapPath(string.Empty),
         "CacheFiles", guid);
         if (File.Exists(path))
         {
            StiReport report = new StiReport();
            string packedReport = File.ReadAllText(path);
            if (guid.EndsWith("template"))
            report.LoadPackedReportFromString(packedReport);
            else report.LoadPackedDocumentFromString(packedReport);

            return report;
```

```
        }
        return null;

        //return base.GetReport(guid);
      }

      public override void SaveReport(StiReport report, string guid)
      {
        string packedReport = guid.EndsWith("template") ?
        report.SavePackedReportToString() :
        report.SavePackedDocumentToString();
        string path =
        Path.Combine(HttpContext.Current.Server.MapPath(string.Empty),
        "CacheFiles", guid);
        File.WriteAllText(path, packedReport);

        //base.SaveReport(report, guid);
      }
    }

    static _Default()
    {
      StiWebViewer.CacheHelper = new StiMyCacheHelper();
    }
}
...
```

To initialize the work with report caching using the created class, it is enough to set it as a value of the static **StiWebViewer.CacheHelper** property in the ASPX page constructor.

Information

If report caching is disabled (the **CacheMode** property of the viewer is set to **None**), the specified class will not be used.

3.1.18 Export and Printing from Code

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text. Please note that the print option is available only for reports and not for dashboards.

HTML5 Viewer is used to print reports in various ways and export reports to various formats. These actions are performed using the viewer menu. If you want to print or export a report using the code, for example, in the event of pressing the button, you can use the special **StiReportResponse** class. This class contains a set of static methods that allow you to print or export a report from the code, and the report viewer is not required.

Default.aspx

```
...
<asp:Button ID="Button1" runat="server" onclick="Button1_Click"
Text="Print Report" />
<asp:Button ID="Button2" runat="server" onclick="Button2_Click"
Text="Export Report" />
...
```

Default.aspx.cs

```
...
private StiReport LoadSimpleList()
{
   DataSet dataSet = new DataSet();
   dataSet.ReadXml(Server.MapPath("Reports/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(Server.MapPath("Reports/SimpleList.mrt"));
   report.RegData(dataSet);

   return report;
}

protected void Button1_Click(object sender, EventArgs e)
{
   StiReport report = LoadSimpleList();

   StiReportResponse.PrintAsPdf(report);
   //StiReportResponse.PrintAsHtml(report);
}

protected void Button2_Click(object sender, EventArgs e)
{
   StiReport report = LoadSimpleList();

   StiReportResponse.ResponseAsPdf(report);
   //StiReportResponse.ResponseAsExcel2007(report);
   //StiReportResponse.ResponseAsText(report);
   //StiReportResponse.ResponseAsJson(report);
}
...
```

The **StiReportResponse** class contains methods for printing in PDF and HTML formats, as well as methods to export the report in any of the supported formats. As arguments, methods can take various export settings, displaying modes and options for saving received files.

3.1.19 Viewer Events

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component supports events that allow you to execute necessary operations before specific actions, such as printing and exporting, sending reports by email, interactivity, etc. Below is a sample of processing viewer events.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
  OnExportReport="StiWebViewer1_ExportReport">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void StiWebViewer1_ExportReport(object sender,
StiExportReportEventArgs e)
{
  if (e.Format == StiExportFormat.Pdf)
  {
    StiPdfExportSettings pdfSettings = e.Settings as
    StiPdfExportSettings;
    pdfSettings.ImageQuality = 50;
    pdfSettings.ImageResolution = 50;
    pdfSettings.ImageCompressionMethod =
    StiPdfImageCompressionMethod.Jpeg;
  }
}
...
```

List of events

Name	Description
OnGetReport	The event occurs when requesting a report for preview.
OnGetReportData	The event occurs when connecting data of a report before it is rendered.
OnPrintReport	The event occurs when printing reports. This is not relevant when viewing dashboards.
OnExportReport	The event occurs when exporting reports.
OnExportReportResponse	The event occurs after exporting reports before saving the exported file.
OnEmailReport	The event occurs when sending a report by Email. This is not relevant when viewing dashboards.
OnInteraction	The event occurs when interactive actions of the viewer, such as using report variables, dynamic collapsing, and sorting.
OnDesignReport	The event occurs when pressing the Design button on the toolbar of the viewer.
OnViewerEvent	The event occurs when any action of the viewer.

3.1.20 Timeout

When working with the **StiWebViewer** component, you can set the timeout for various operations — storing the report in the cache, server response, and query execution. The timeout setting is done using the component properties and report options.

CacheTimeout Property

Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
  CacheTimeout="10">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
   StiWebViewer1.CacheTimeout = 10;
}
...
```

Using the cache will increase the speed of the report viewer. See the chapter Caching for more information

RequestTimeout Property

Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   RequestTimeout="10">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
   StiWebViewer1.RequestTimeout = 30;
}
...
```

CommandTimeout Option

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to set the query timeout for the already created connection and data sources in the report.

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{

   StiReport report = new StiReport();
   report.Load(Server.MapPath("Report.mrt"));
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   StiWebViewer1.Report = report;
}
...
```

3.1.21 Viewer Settings

HTML5 Viewer can be setup using the component properties which can be set on the ASPX page using C#/VB code. Below are samples of using the viewer properties.

Default.aspx

```
...
<cc1:StiWebViewer ID="StiWebViewer1" runat="server"
   ShowPrintButton="false"
   ScrollbarsMode="true"
   ShowTooltips="false"
   ShowExportToDbf="false"
   ShowExportToDif="false"
   Zoom="75"
   Theme="Office2022WhiteTeal">
</cc1:StiWebViewer>
...
```

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
   StiWebViewer1.ShowPrintButton = false;
   StiWebViewer1.ScrollbarsMode = true;
   StiWebViewer1.ShowTooltips = false;
   StiWebViewer1.ShowExportToDbf = false;
   StiWebViewer1.ShowExportToDif = false;
   StiWebViewer1.Zoom = 75;
   StiWebViewer1.Theme = StiTheme.Office2022WhiteTeal;
   StiWebViewer1.ReportDisplayMode = StiReportDisplayMode.Auto;
}
...
```

Please note that all dashboard elements have their own save options and full-screen

buttons for preview. There are no special options to control displaying them, but they can be disabled through the properties of the element. The code below should be added after loading the report before passing it to the viewer.

Default.aspx.cs

```
...
var dbsElementInteraction = (report.GetComponentByName("RegionMap1") as
Stimulsoft.Report.Dashboard.IStiElementInteraction).DashboardInteraction;
(dbsElementInteraction as
Stimulsoft.Report.Dashboard.IStiInteractionLayout).ShowFullScreenButton =
false;
(dbsElementInteraction as
Stimulsoft.Report.Dashboard.IStiInteractionLayout).ShowSaveButton = false;
...
```

Basic settings

Name	Description
Width	Sets the width of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. The default width is 100%.
Height	Sets the height of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the automatic height is set depending on the size of the report page, or 650 pixels in the view mode of the viewer with scrollbars.

Work with server

Name	Description
RequestTimeout	Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For

	big reports, it is recommended to increase this value.
CacheTimeout	Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.
CacheMode	Sets the report caching mode. It can take one of the following values of the **StiServerCacheMode** enumeration: › **None** – caching is disabled; the report will be reloaded each time using the **OnGetReport** event; › **ObjectCache** – the cache is used as the storage, the report is stored as an object (default value); › **ObjectSession** – the session is used as the storage, the report is stored as an object; › **StringCache** – the server cache is used as the storage, the report is serialized to a packed string; › **StringSession** – the session is used as a repository, the report is serialized into a packed string.
CacheItemPriority	Sets the priority of the report stored in the server cache. This property affects the automatic clearing of the server memory in case of a shortage. The lower the priority is, the greater is the chance of removing information from memory.
UseRelativeUrls	Sets the viewer mode in which relative references are used for AJAX requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value that specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-1** removes the port number.
AllowAutoUpdateCache	Sets the mode for automatic cache update. The

	report stored in the cache or server session will be automatically re-saved after a certain period of time if the designer is idle (every 3 minutes). By default, the property is set to **true**.
PassQueryParametersForResources	Enables transferring of all parameters of the URL request when generating links to the resources of the viewer. If it is set to **false**, only the necessary parameters are used to query the resources of the viewer, which contributes to the correct work of the browser cache. By default, the property is set to **true**.
PassQueryParametersToReport	Enables using all the URL parameters of the request as the variable values. The variables' names must match the parameters. The default value of the property is **false**.
PassFormValues	Enables passing the values of the POST form to the client-side if these values are required to be used in the events of the viewer. When the property is enabled, the helper method - **GetFormValues()** - returns a collection of parameters of a form. By default, the property is set to **false**.
ShowServerErrorPage	Enables displaying an HTML page with the details of the error that occurred on the server-side. When the property is enabled, the details of the error will be displayed in the viewer window. If the property is disabled, only the numeric error code and a short error text in the dialog box will be displayed. By default, the property is set to **true**.
UseCompression	Enables compression of the viewer requests into the GZip stream. That allows to decrease the amount of internet traffic but slows down the viewer slightly. The default value of the property is **false**.
UseCacheForResources	Enables caching of the component resources on the server-side. The following resources are

	supported: scripts, styles, and images. This option improves the load speed of the component and also reduces the server load in multi-client environments. The default value is **true**.
UseLocalizedCache	Sets a value that enables the use of a different cache depending on the selected localization. The default value of the property is **false**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
Theme	Specifies the theme of the viewers' layout. The list of available themes can be found in the **StiTheme** enumeration. The default value is **Office2022WhiteBlue**.
CustomCss	Sets the path to the CSS file of the viewer's styles. The standard styles of the chosen theme will not be loaded if this property has got a value. The default value of the property is an empty string.
Localization	Specifies the path to the XML localization file. The path can be absolute or relative. By default, the English localization is used, which is built into the viewer and does not require additional XML files.
BackgroundColor	Sets the background color of the viewer. By default, it is set to **White**.
PageBorderColor	Sets the border color of the viewer. By default, it is set to **Gray**.
RightToLeft	Sets the **Right to Left** mode for viewer controls.

	By default, the property is set to **false**.
FullScreenMode	Sets the full-screen display mode of the viewer. By default, the property is set to **false**.
ScrollbarsMode	Sets the preview mode with scrollbars. By default, the property is set to **false**.
OpenLinksWindow	Sets the target window for opening links contained in the report. By default, it is set to **Blank** (new window).
OpenExportedReportWindow	Sets the target window for opening the export file from the viewer. By default, it is set to **Blank** (new window).
DesignWindow	Sets the target window for opening the report designer. By default, it is set to **Self** (current window).
ShowTooltips	Enables displaying tips for the viewer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enables displaying links to online documentation for the viewer controls. By default, the property is set to **true**.
ShowDialogsHelp	Sets a value which indicates that show or hide the help button in dialogs. By default, the property is set to **true**.
PageAlignment	Sets the position of the report page in the viewer window. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – the page will be aligned left; › **Center** – the page will be centered (default value); › **Right** – the page will be aligned right.
ShowPageShadow	Enables displaying shadow for report pages. By default, the property is set to **true**.
BookmarksPrint	Enables printing of report bookmarks (besides the report itself). By default, the property is set

	to **false**.
BookmarksTreeWidth	Sets the width of the bookmarks panel in pixels. By default, the width is 180 pixels.
ParametersPanelPosition	Specifies the position of the report parameters panel. It can take one of the following **StiParametersPanelPosition** enumeration values: › **Top** - the panel will be docked to the top margin (default value); › **Left** – the panel will be docked to the left margin.
ParametersPanelMaxHeight	Sets the maximum height of the parameters bar in pixels. By default, the maximum height is 300 pixels.
ParametersPanelColumnsCount	Sets the number of columns to display report parameters. By default, there are 2 columns.
ParametersPanelSortDataItems	Gets or sets a value which indicates that variable items will be sorted. By default, the property is set to **true**.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
InterfaceType	Sets the type of interface used for the viewer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the viewer's interface is determined automatically depending of the device that is report is displayed on. That is the default value. › **Mouse** – the standard interface with a mouse control will be used for all the screen types. › **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been

	increased in size to simplify the control of the viewer and to improve its usability. › **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with the smartphones.
AllowMobileMode	Enables or disables displaying a report or dashboard in the mobile mode. If the option is set to **false**, then the mobile view will not be used. If the option is set to **true**, the mobile view mode will be used when opening the viewer on mobile devices. By default, the option is set to **true**.
ChartRenderType	Sets the chart displaying mode on the report page. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages. It can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN

	markup.
DatePickerFirstDayOfWeek	Sets the first day of the week for the date picker. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Monday** – the first day of the week is Monday (default value); › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default, the property is set to **false**.
AllowTouchZoom	Sets ability to change the scale of the report page by using the two-fingers gesture (Pinch to Zoom) for the touch-screens. The default value of the property is **true**.
ShowReportIsNotSpecifiedMessage	Sets a value which indicates that 'The report is not specified' message will be shown. The default value of the property is **true**.
ImagesQuality	Gets or sets the image quality that will be used on the viewer page. It has the following values: › **StiImagesQuality.Low** - low quality, used to speed up loading reports and saves memory; › **StiImagesQuality.Normal** - normal quality, suitable for most cases (default value); › **StiImagesQuality.High** - high quality, used for ultra high-definition displays, but may slow down the loading of pages.
PrintToPdfMode	Sets the Print to PDF mode. It has the following values: › **StiPrintToPdfMode.Hidden** - hidden print mode (default value); › **StiPrintToPdfMode.Popup** - the PDF document will be displayed before printing in a pop-up window.
CombineReportPages	Sets a value which indicates that if a report contains several pages, then they will be combined in preview. By default, the property is

	set to **false**.
SaveMenuImageSize	Sets a value which indicates images size of the save menu in the viewer. The default value of the property is **true**.

Toolbar

Name	Description
ShowToolbar	Enables displaying the viewer toolbar. By default, the property is set to **true**.
ToolbarDisplayMode	Specifies the display mode of the toolbar of the viewer. It can take one of the following values of the **StiToolbarDisplayMode** enumeration values: › **Simple** - all controls are located on the same control panel (default value); › **Separated** - the control panel is split into top and bottom panels.
ToolbarBackgroundColor	Sets the color of the viewer toolbar. By default, color of the selected theme is used.
ToolbarBorderColor	Sets the color of the borders of the Viewer toolbar. By default, color of the selected theme is used.
ToolbarFontColor	Sets the text color for the toolbar and the viewer menu. By default, color of the selected theme is used.
ToolbarFontFamily	Sets the font for the toolbar and the viewer menu. The default font of the selected theme is used.
ToolbarAlignment	Sets the alignment mode for the controls on the viewer toolbar. It can take one of the following values of the **StiContentAlignment** enumeration:

	› **Left** – elements will be aligned left; › **Center** – elements will be centered; › **Right** – elements will be aligned right; › **Default** – the alignment depends on the RightToLeft property (default value).
ShowButtonCaptions	Enables text of the buttons on the toolbar of the viewer. By default, the property is set to **true**.
ShowPrintButton	Enables showing the button - **Print** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowOpenButton	Enables displaying the **Open** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the viewer when viewing reports or dashboards.. By default, the property is set to **true**.
ShowSendEmailButton	Enables showing the button - **Send Email** - on the toolbar of the viewer. By default, the property is set to **false**. Also, you should add the OnEmailReport event handler.
ShowFindButton	Enables showing the button - **Find** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowBookmarksButton	Enables showing the button - **Bookmarks** - on the toolbar of the viewer. By default, the property is set to **true**. If the button is hidden, the bookmarks panel will not be displayed even if there are bookmarks in the report.
ShowParametersButton	Enables showing the button - **Parameters** - on the toolbar of the viewer. By default, the property is set to **true**. If the button is hidden, the parameters panel will not be displayed even if there are parameters in the report.
ShowResourcesButton	Enables showing the button - **Resources** - on

	the toolbar of the viewer. By default, the property is set to **true**. If the button is hidden, the resources panel will not be displayed even if there are resources in the report.
ShowEditorButton	Enables showing the button - **Editor** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowFullScreenButton	Enables displaying the **Full Screen** button on the toolbar of the viewer when viewing reports or dashboards.. By default, the property is set to **true**.
ShowFirstPageButton	Enables showing the button - **First Page** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowPreviousPageButton	Enables showing the button - **Previous Page** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowCurrentPageControl	Enables showing the current report page indicator. By default, the property is set to **true**.
ShowNextPageButton	Enables showing the button - **Next Page** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowLastPageButton	Enables showing the button - **Last Page** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowZoomButton	Enables showing the report zoom button. By default, the property is set to **true**.
ShowViewModeButton	Enables showing the button for selecting the display mode of report pages. By default, the property is set to **true**.
ShowDesignButton	Enables displaying the **Design** button on the toolbar of the viewer when viewing reports or dashboards.. By default, the property is set to **false**.
ShowAboutButton	Enables showing the button - **About** - on the

	toolbar of the viewer. By default, the property is set to **true**.
ShowRefreshButton	Sets a visibility of the **Refresh** button in the toolbar of the viewer. By default, the property is set to **true**.
ShowPinToolbarButton	Enables displaying of the **Pin Toolbar** button on the viewer's toolbar. The button is available only in the Mobile mode of the viewer's interface. The default value of the property is **true**.
PrintDestination	Sets the report printing mode. It can take one of the following values of the **StiPrintDestination** enumeration: › **Default** – a menu with a choice of printing modes will be displayed (default value); › **Pdf** – printing will be done in the PDF format; › **Direct** – printing will be done to the HTML format directly to the printer, the system print dialog will be displayed; › **PopupWindow** – printing will be done in the HTML format via the preview window of the report.
ViewMode	Sets the mode for displaying report pages. It can take one of the following values of the **StiWebViewMode** enumeration values: › **SinglePage** - displays one page of the report selected in the toolbar of the viewer (default value); › **Continuous** - displays all pages of the report; › **MultiplePages** - displays all report pages as a table.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. The values are from 10 to 500 percent. You can also set one of the following values:

	› **StiZoomMode.PageWidth** – when the viewer runs, the zoom, necessary to display the report by the page width, will be set; › **StiZoomMode.PageHeight** – when the viewer runs, the zoom, required to display the page height of the report, will be set.
MenuAnimation	Enables animation when the viewer menu shows/hides. By default, the property is set to **true**.
ShowMenuMode	Sets the display mode of the viewer menu. It can take one of the following values of the **StiShowMenuMode** enumeration: › **Click** – shows menu by mouse click (default value); › **Hover** – shows menu by hovering the mouse cursor.
AutoHideToolbar	Enables auto-hiding of the viewer's toolbar. The property will work only for the Mobile mode of the viewer's interface. The default value of the property is **false**.

Samples

See samples how to add a custom button on the toolbar of the viewer.

Export report

Name	Description
DefaultExportSettings	This group of properties is used to specify the default export settings for each export type. These settings will be applied to the export dialogs when the viewer runs or to the report, if export dialogs are disabled.

StoreExportSettings	Enables saving selected settings in the export dialogs. Settings will be stored in browser cookies. By default, the property is set to **true**.
ShowExportDialog	Enables showing export options dialog box. If the property is set to **false**, the export will be done with the default settings. By default, the property is set to **true**.
ShowExportToDocument	Enables the export menu item - **Document File**. By default, the property is set to **true**.
ShowExportToPdf	Enables displaying the **Adobe PDF file** export menu item when viewing reports, and the **Adobe PDF** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToXps	Enables the export menu item - **Microsoft XPS File**. By default, the property is set to **false**.
ShowExportToPowerPoint	Enables the export menu item - **Microsoft PowerPoint 2007/2010 File**. By default, the property is set to **true**.
ShowExportToHtml	Enables the export menu item - **File**. By default, the property is set to **true**.
ShowExportToHtml5	Enables the export menu item - **HTML5 File**. By default, the property is set to **true**.
ShowExportToMht	Enables the export menu item - **MHT Web Archive**. By default, the property is set to **true**.
ShowExportToText	Enables the export menu item - **Text File**. By default, the property is set to **true**.
ShowExportToRtf	Enables the export menu item - **Rich Text File**. By default, the property is set to **true**.
ShowExportToWord2007	Enables the export menu item - **Microsoft Word 2007/2010 File**. By default, the property is set to **true**.
ShowExportToOpenDocumentWriter	Enables the export menu item - **OpenDocument Writer File**. By default, the property is set to **true**.

ShowExportToExcel	Enables the export menu item **Microsoft Excel File**. By default, the property is set to **true**.
ShowExportToExcelXml	Enables the export menu item - **Microsoft Excel Xml File**. By default, the property is set to **true**.
ShowExportToExcel2007	Enables displaying the **Microsoft Excel 2007/2010 File** export menu item when viewing reports, and the **Microsoft Excel** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToOpenDocumentCalc	Enables the export menu item - **OpenDocument Calc File**. By default, the property is set to **true**.
ShowExportToCsv	Enables the export menu item - **CSV File**. By default, the property is set to **true**.
ShowExportToDbf	Enables the export menu item - **DBF File**. By default, the property is set to **true**.
ShowExportToXml	Enables the export menu item - **XML File**. By default, the property is set to **true**.
ShowExportToDif	Enables the export menu item - **Data Interchange Format (DIF) File**. By default, the property is set to **true**.
ShowExportToSylk	Enables the export menu item - **Symbolic Link (SYLK) File**. By default, the property is set to **true**.
ShowExportToJson	Enables the export menu item - **JSON File**. By default, the property is set to **true**.
ShowExportToImageBmp	Enables displaying the **BMP Image** export menu item when viewing reports, and the **BMP Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageGif	Enables displaying the **GIF Image** export menu item when viewing reports, and the **GIF Image** item when viewing dashboards. By default, the property is set to **true**.

ShowExportToImageJpeg	Enables displaying the **JPEG Image** export menu item when viewing reports, and the **JPEG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePcx	Enables displaying the **PCX Image** export menu item when viewing reports, and the **PCX Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePng	Enables displaying the **PNG Image** export menu item when viewing reports, and the **PNG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageTiff	Enables displaying the **TIFF Image** export menu item when viewing reports, and the **TIFF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageMetafile	Enables displaying the **Windows Metafile** export menu item when viewing reports, and the **Windows Metafile** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvg	Enables displaying the **Scalable Vector Graphics (SVG) File** export menu item when viewing reports, and the **Scalable Vector Graphics (SVG) File** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvgz	Enables displaying the **Compressed SVG (SVGZ) File** export menu item when viewing reports, and the **Compressed SVG (SVGZ) File** item when viewing dashboards. By default, the property is set to **true**.
ShowOpenAfterExport	Enables displaying the **Open After Export** parameter in export settings menu. By default, the property is set to **true**.

Email

Name	Description
ShowEmailDialog	Enables displaying settings for sending the report via email. If the dialog box is disabled, the email will be sent with the settings set on the server-side in the **OnEmailReport** event. By default, the property is set to **true**.
ShowEmailExportDialog	Enables displaying export options dialog box when sending email. If the property is set to **false**, the export will be done with the default settings. By default, the property is set to **true**.
DefaultEmailAddress	Sets the default recipient email, i.e. the address to which the email with the attached report will be sent.
DefaultEmailSubject	Sets the default email subject (header).
DefaultEmailMessage	Sets the default email message (text).

3.2 HTML5 Designer

YouTube

Watch videos for working with ASP.NET HTML5 Designer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

Samples

See on GitHub examples of working with the ASP.NET HTML5 Designer component. All samples are separate projects grouped into one solution for Visual Studio.

The **HTML5 Designer (StiWebDesigner)** component is designed to create reports in the web browser. You do not need to install the .NET Framework, ActiveX components, or any special plug-ins on the client-side. All you need is any modern

Web browser.

With the help of **HTML5 Designer**, you can create, edit, save and preview reports on any computer with any operating system installed. Since the designer only uses HTML and JavaScript technologies, it can be run on devices without Flash or Silverlight support - tablets, smartphones. Also, the designer supports the Touch interface, which is automatically enabled when using devices with a touch screen.

The **HTML5 Designer** component uses the AJAX technology to perform all actions on reports, which allows you to get rid of reloading the entire page, save Web traffic, and speed up work.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use **HTML5 Designer** in a Web-project, the installation of the Stimulsoft.Reports.Web NuGet package is required:
- Select 'Manage NuGet Packages...' menu item in the project's pop-up menu;
- In the 'Browse' tab, type 'Stimulsoft.Reports.Web' in the search textbox;
- Click the Stimulsoft.Report.Web package, select the version of the package, and click **Install**. If the package should be updated, use the **Update** button.

If for some reason, that is not possible, the following assemblies should be added to the project:
- Stimulsoft.Base.dll
- Stimulsoft.Report.dll
- Stimulsoft.Report.Check.dll
- Stimulsoft.Report.Helper.dll
- Stimulsoft.Report.Web.dll
- Stimulsoft.Report.WebDesign.dll

To add the ability to create and edit dashboards in a Web project, install the NuGet package Stimulsoft.Dashboards.Web (this package is associated with the package

Stimulsoft.Reports.Web. If it is missed, it will be installed automatically):
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Dashboards.Web in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

If for some reason this is not possible, you should additionally add the following assemblies to the project:
› Stimulsoft.Dashboard.dll
› Stimulsoft.Dashboard.Drawing.dll
› Stimulsoft.Dashboard.Export.dll

3.2.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the designer, you need to place the **StiWebDesigner** component on the ASPX page, set the necessary properties, and, if necessary, set the necessary event handlers. When the report designer runs, the following actions occur:

› The .NET component generates HTML and JavaScript code that is necessary for displaying and running the designer;
› When the component is output, the JavaScript method is launched. It requests the report template on the server side displays it in the designer window;
› Various actions in the designer (for example, report preview, saving the report template, export reports, sorting, drill-down, etc.) call a particular action on the server-side. You can perform the necessary manipulations with the report.

3.2.2 Activation

YouTube

Watch videos that show how to activate the ASP.NET HTML5 Designer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiWebDesigner** component.

Default.aspx.cs

```
...
public partial class _Default : Page
{
   static _Default()
   {
      //Activation with using license code
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";

      //Activation with using license file
      var path = HttpContext.Current.Server.MapPath("license.key");
      Stimulsoft.Base.StiLicense.LoadFromFile(path);
   }
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

3.2.3 Editing Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

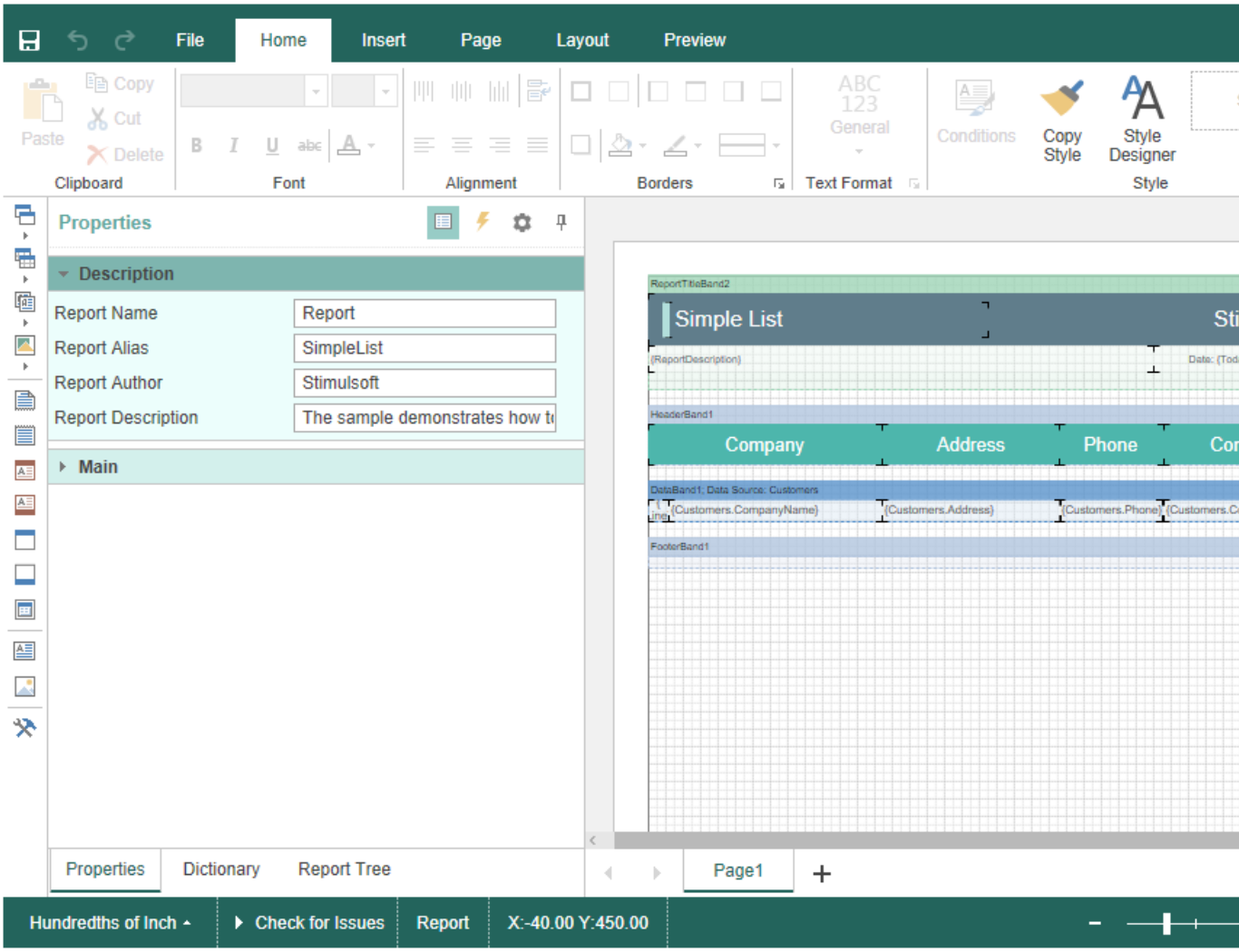
To edit a report template, you need to add the **StiWebDesigner** component to the ASPX page and assign a loaded report template to it.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Reports/SimpleList.mrt"));
   //report.Load(Server.MapPath("Reports/Dashboard.mrt"));

   StiWebDesigner1.Report = report;
}
...
```



Also, **HTML5 Designer** has a special **OnGetReport** event that you can use to assign a report template. In this case, you need to load the report in the event handler.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
  OnGetReport="StiWebDesigner1_GetReport">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void StiWebDesigner1_GetReport(object sender,
StiReportDataEventArgs e)
```

```
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Reports/SimpleList.mrt"));

   e.Report = report;
}
...
```

Information

The **OnGetReport** event will be called regardless of whether the report was previously assigned or not. If the report is already assigned to the designer, then, in the event arguments, the **e.Report** property will contain the loaded report object. You can change it or assign a new report.

By default, **HTML5 Designer** uses the entire area of the browser window to edit the report. To display a component in a specific HTML page with the specific position and dimensions, it is enough to set its width and height using the **Width** and **Height** properties.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   Width="1000px" Height="800px">
</cc1:StiWebDesigner>
...
```

3.2.4 Creating New Reports and New Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the designer with a new (empty) report, no action is required. When the component is loaded, the new report will be created automatically. If necessary, you can create a new report object, preload the data, or perform other necessary actions.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   OnGetReport="StiWebDesigner1_GetReport">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void StiWebDesigner1_GetReport(object sender,
StiReportDataEventArgs e)
{
   e.Report = new StiReport();
   //var newDashboard = StiReport.CreateNewDashboard();
}
...
```

You can also create a new report using the main menu of the designer. The **OnCreateReport** event is used to load data for a new report or perform other necessary actions. This event will be called when you create a new empty report or a report using the wizard.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   OnCreateReport="StiWebDesigner1_CreateReport">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void StiWebDesigner1_CreateReport(object sender,
StiReportDataEventArgs e)
{
   StiReport report = new StiReport();
   //var newDashboard = StiReport.CreateNewDashboard();

   // Register data for the new report, if necessary
   DataSet data = new DataSet("Demo");
   data.ReadXml(Server.MapPath("Data/Demo.xml"));
   report.RegData(data);
   //newDashboard.RegData(data);
   report.Dictionary.Synchronize();
   //newDashboard.Dictionary.Synchronize();
```

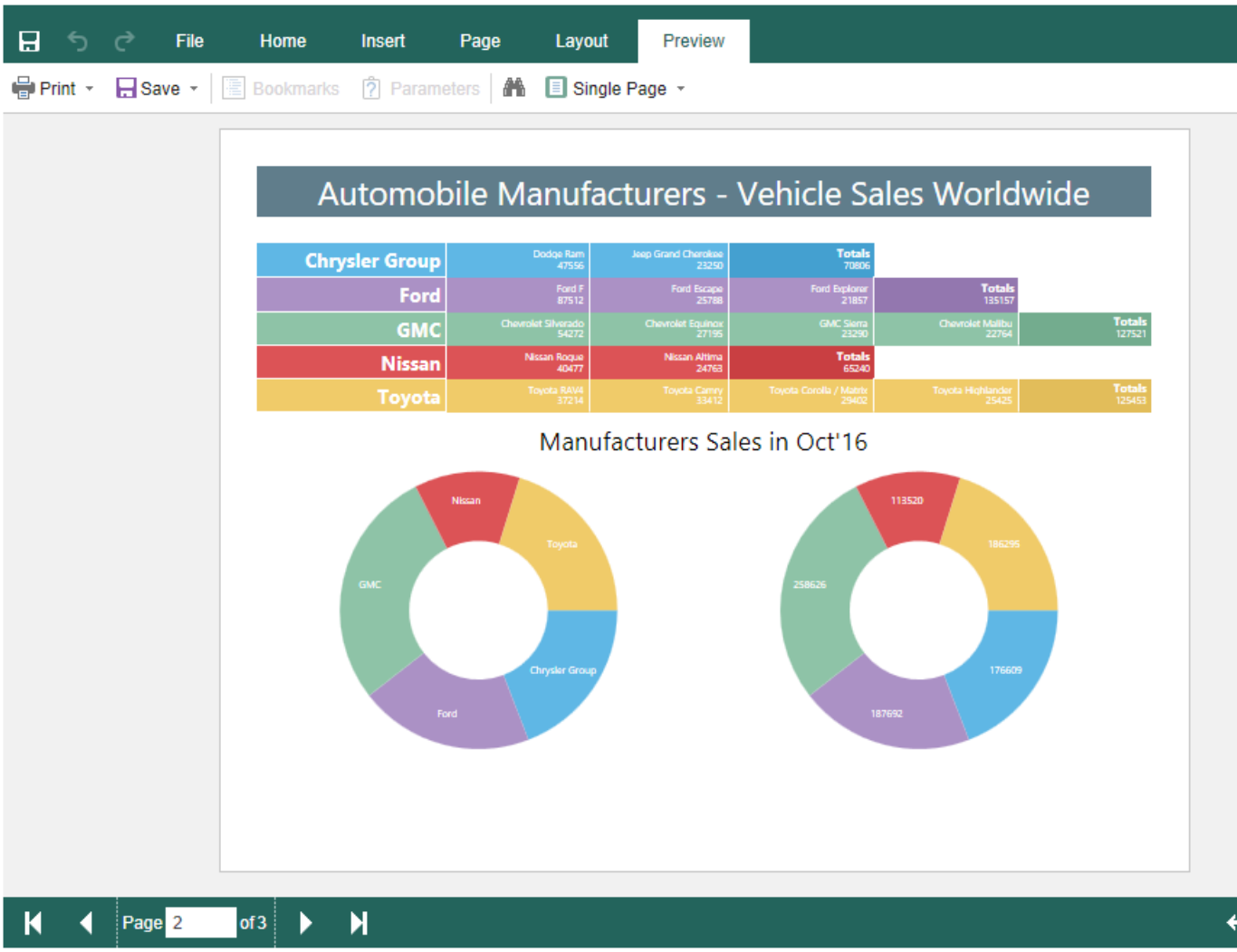
```
    e.Report = report;
    //e.Report = newDashboard;
}
...
```

3.2.5 Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component provides the ability to preview reports. To preview the report, just go to the appropriate tab in the designer window. The report template will be transferred to the server-side, rendered, and displayed in the embedded viewer.



Before previewing the report, it is possible to perform any necessary actions, for example, connect data for the report. To do this, you can use a special **OnPreviewReport** event which will be called before previewing the report. In the arguments of the event, there will be a report to be previewed. The **OnPreviewReport** event is called before preparing and rendering a report for viewing till its saving to the cash.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
  OnPreviewReport="StiWebDesigner1_PreviewReport">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void StiWebDesigner1_PreviewReport(object sender,
StiReportDataEventArgs e)
{
   DataSet data = new DataSet("Demo");
   data.ReadXml(Server.MapPath("Data/Demo.xml"));
   e.Report.RegData(data);
}
...
```

If you need to take actions on your report immediately before displaying the report, you can use the **OnGetPreviewReport** event, which is called after the request of the prepared report from the cash.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   OnGetPreviewReport="StiWebDesigner1_GetPreviewReport">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void StiWebDesigner1_GetPreviewReport(object sender,
StiReportDataEventArgs e)
{
   DataSet data = new DataSet("Demo");
   data.ReadXml(Server.MapPath("Data/Demo.xml"));
   e.Report.RegData(data);

   //report.IsRendered = false;
}
...
```

Information

So as in this event, a prepared report for viewing is transferred. If you need to render again, you should set the **report.IsRendered = false** flag.

3.2.6 Additional Features of Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The preview window of the **HTML5 Designer** component has a fully functional interactive HTML5 Viewer that can print and export reports, supports working with report parameters, dynamic sorting, interactive reports, collapsing, etc. To use these features, you do not need any additional settings for the report designer.

In any of the above actions, you can work with the report template, such as changing its properties and parameters and connecting new data for rendering.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   OnExportReport="StiWebDesigner1_ExportReport">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void StiWebDesigner1_ExportReport(object sender,
StiReportDataEventArgs e)
{
   e.Report.ReportName = "MyReportName";
   e.Report.ReportAlias = "MyReportAlias";
}
...
```

Information

Suppose you do not need any additional options to preview the report (for example, exporting or printing a report). In that case, you can disable them using the appropriate properties of the **HTML5 Designer** component.

3.2.7 Saving Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component provides two ways of saving the report, which are available in the main menu and the main panel of the designer - **Save Report** and **Save As**. In turn, each of these ways has its modes and settings.

Saving a report and dashboard on the server-side

To save the edited report on the server-side, you need to set the **OnSaveReport** special event, which will be called when you select the **Save Report** menu item or click the Save button on the main panel of the designer.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   OnSaveReport="StiWebDesigner1_SaveReport">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void StiWebDesigner1_SaveReport(object sender,
StiReportDataEventArgs e)
{
   StiReport report = e.Report;

   // Save the report template
   // ...
}
...
```

By default, after saving the report, the designer continues working without displaying any messages. If necessary, after saving the report, it is possible to display a dialog box with an error or a text message. For this purpose, the special properties - **e.ErrorCode** and **e.ErrorString** in the arguments of the event are used.

Default.aspx.cs

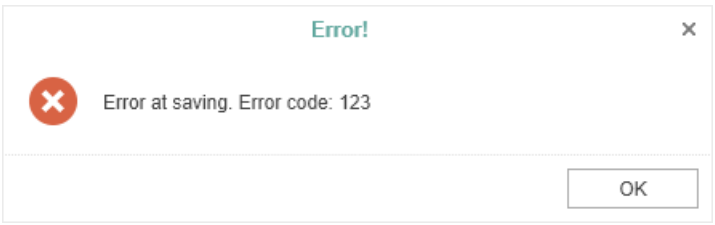
```
...
protected void StiWebDesigner1_SaveReport(object sender,
StiReportDataEventArgs e)
{
   StiReport report = e.Report;

   // Save the report template
   // ...

   e.ErrorCode = 123;
   //e.ErrorString = "Some message after saving";
}
...
```

You can get a report name from the designer save dialog or an original report name.

Default.aspx.cs

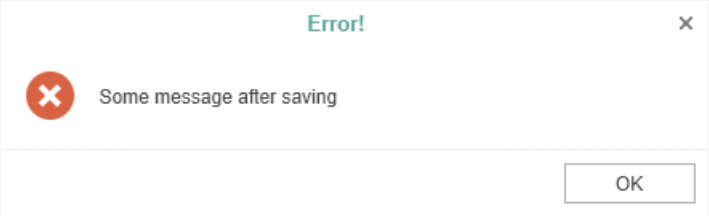
```
...
protected void StiWebDesigner1_SaveReport(object sender,
StiSaveReportEventArgs e)
{
   //Report name from the designer save dialog
   var reportName = e.FileName;

   //Original report name from properties
   var reportName = e.Report.ReportName;
}
...
```

If you set a positive integer value for the **e.ErrorCode** property, the user will see the error message of saving the report and the error code, where the error code is the integer value.



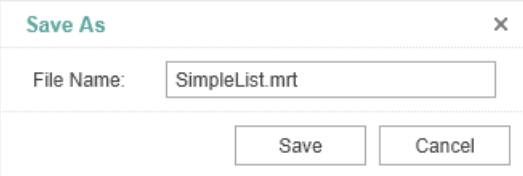
If you set a string value for the **e.ErrorString** property, a dialog with the specified

text will be displayed. The text can contain both a save error message or a warning, or any other message.



Saving reports and dashboards on the client side

To save the edited report on the client-side as a file, no additional designer settings are required. It is enough to click the **Save As** main menu item. The dialog box will be displayed. In this dialog, you can change the name of the report file. The file will be saved to the local disk of the computer.



The **HTML5 Designer** component provides the ability to change the behavior of the specified save option. For this purpose, the special **OnSaveReportAs** event is used in the designer. If you use this event, the report will be saved on the server-side. The work of this event will be similar to the **OnSaveReport** event.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   OnSaveReportAs="StiWebDesigner1_SaveReportAs">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void StiWebDesigner1_SaveReportAs(object sender,
StiReportDataEventArgs e)
{
   StiReport report = e.Report;
```

```
    // Save the report template
    // ...
}
...
```

Saving settings

The report is saved in the background mode without reloading the page in the web browser window. Suppose you need to control the process of saving the report visually. In that case, you should change the value of the **SaveReportMode** (or **SaveReportAsMode**) property of the designer to one of the three specified values - **Hidden** (default value), **Visible**, or **NewWindow**.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   OnSaveReportAs="StiWebDesigner1_SaveReportAs"
   SaveReportAsMode="Visible">
</cc1:StiWebDesigner>
...
```

If the **SaveReportMode** property is set to **Visible**, the report save event will be called in the current browser window in the normal (visible) mode using the POST request. If the **SaveReportMode** property is set to **NewWindow**, the report save event will be called in a new window of the web browser. By default, this property is set to **Hidden** - the report save event is called in the background using the AJAX request and is not displayed in the browser window. The same values and behavior are applicable to the **SaveReportAsMode** property.

3.2.8 Localization

The **HTML5 Designer** component supports the complete localization of its interface. Use the special **Localization** property to localize the report designer interface. As a value of this property, you should specify the path to the localization XML file (relative or absolute).

Default.aspx

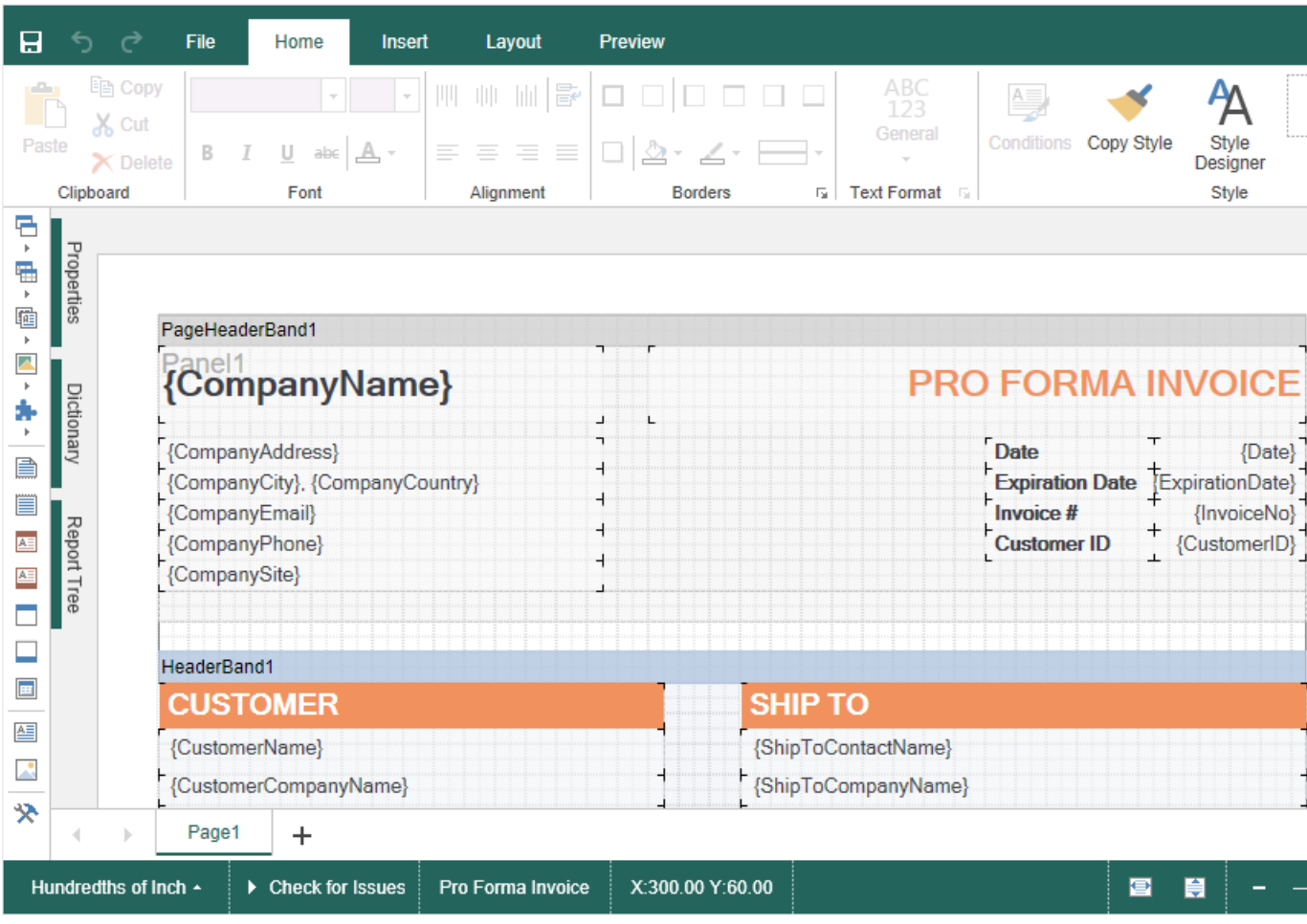
```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   Localization="Localization/en.xml">
</cc1:StiWebDesigner>
```

```
...
```

The interface of the report designer allows you to select the necessary localization from an accessible list. To do this, the value of the **LocalizationDirectory** property must be the folder in which the localization XML files are stored.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   Localization="Localization/en.xml"
   LocalizationDirectory="Localization">
</cc1:StiWebDesigner>
...
```



Information

If the value for the **Localization** property is set, then when you run the report designer, the localization specified in this property will always be applied. If the property value is not set, the localization selected from the list of available localizations in the report designer panel will be automatically loaded.

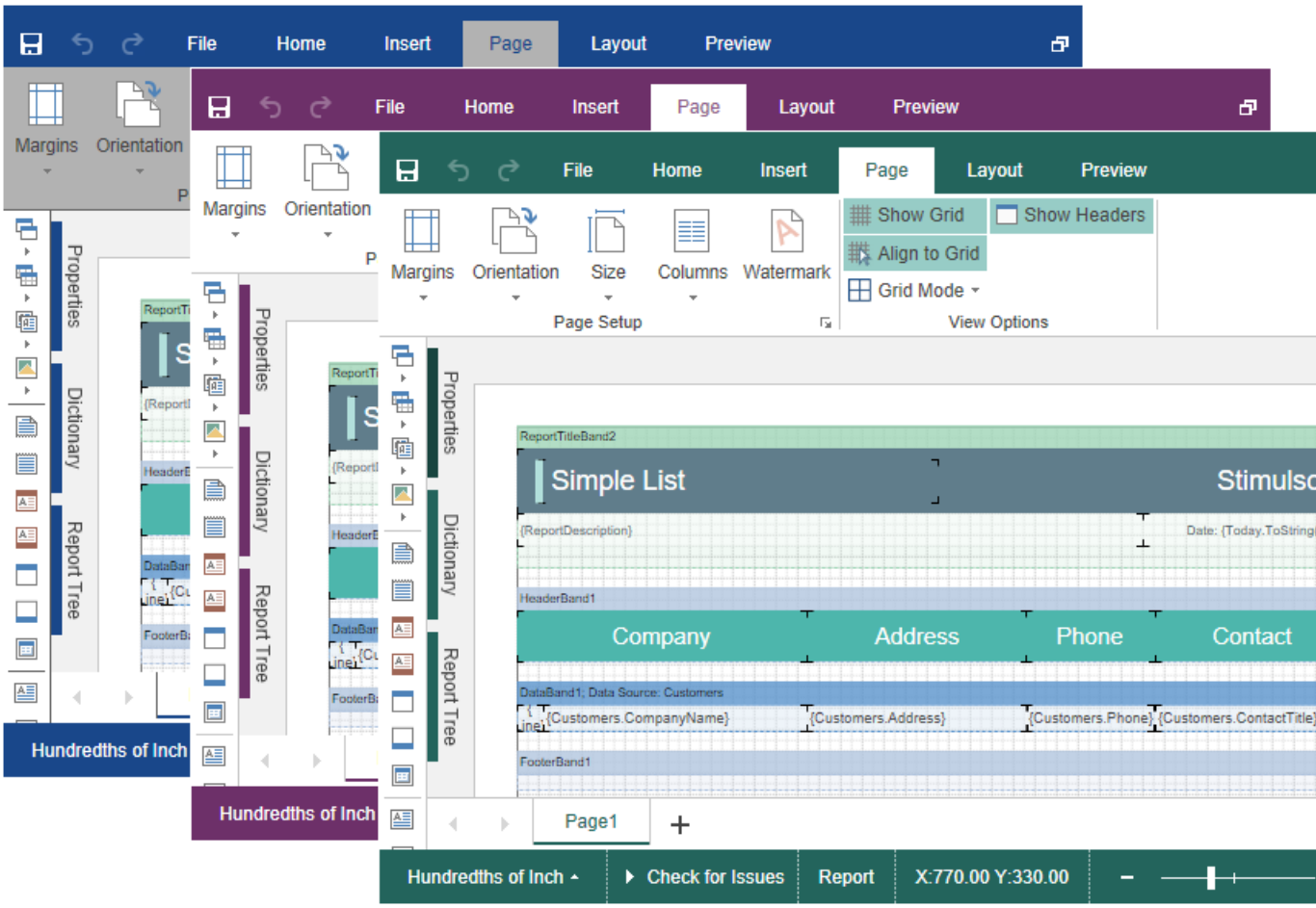
3.2.9 Using Themes

In the **HTML5 Designer** component, you can change the appearance of visual controls. To change the theme, you should use the **Theme** property.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
  Theme="Office2022WhiteTeal">
</cc1:StiWebDesigner>
...
```

There are currently **2 themes** available with different color accents. As a result, **more than 50** variants of the appearance are available. This allows you to customize the appearance of the designer for almost any design of the Web project.



3.2.10 Caching

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

HTM5 Designer uses the server cache to store the editable report template. It is necessary because the client part of the designer contains only a visual representation of components of a report template. The report object itself with all the parameters and properties is stored on the server-side.

You can manage caching with the following properties.

The CacheMode property

This property of the designer enables caching and sets its type. It can take one of the following values, specified in the **StiServerCacheMode** enumeration:

- **None** – caching is disabled;
- **ObjectCache** – for caching, the server cache is used. The report object is saved in this (set by default);
- **StringCache** – for caching, the server cache is used. The report is saved as a packed string in this cache;
- **ObjectSession** – the current session, in which the report object is saved, is used for caching;
- **StringSession** - for caching, the current session is used, in which the report is saved as a packed string.

The CacheItemPriority property

This property sets the priority of the report stored in the cache of the server. It affects the automatic clearing of the server memory in case of memory shortage. The lower the priority is, the greater is the chance of removing information from memory.

The CacheTimeout property

This property specifies the amount of time in minutes you want to store the report in the server cache. If, when using caching, the requested report is not found in the cache (time of storing this report expired), then it will be requested again using the special **OnGetReport** event. In this case, the unsaved changes may be lost.

The **HTML5 Designer** component provides the ability to specify its methods for working with report caching. For this purpose, a special **StiCacheHelper** class is used. It contains methods for obtaining a report from the cache and saving the report to the cache. It is necessary to create a new class inherited from **StiCacheHelper** and reload the above methods, which respectively have the names - **GetReport** and **SaveReport**.

Default.aspx.cs

```
...
public partial class _Default : Page
{
   public class StiMyCacheHelper : StiCacheHelper
   {
      public override object GetObject(string guid)
      {
         string path =
         Path.Combine(HttpContext.Current.Server.MapPath(string.Empty),
         "CacheFiles", guid);
         if (File.Exists(path))
         {
            byte[] cacheData = File.ReadAllBytes(path);
            return StiCacheHelper.GetObjectFromCacheData(cacheData);
         }
         return null;

         //return base.GetObject(guid);
      }

      public override void SaveObject(object obj, string guid)
      {
         byte[] cacheData = StiCacheHelper.GetCacheDataFromObject(obj);
         string path =
         Path.Combine(HttpContext.Current.Server.MapPath(string.Empty),
         "CacheFiles", guid);
         File.WriteAllBytes(path, cacheData);

         //base.SaveObject(obj, guid);
      }

      public override void RemoveReport(string guid)
      {
         string path =
         Path.Combine(HttpContext.Current.Server.MapPath(string.Empty),
         "CacheFiles", guid);
         if (File.Exists(path))
         {
            File.Delete(path);
         }
      }
   }

   static _Default()
   {
      StiWebDesigner.CacheHelper = new StiMyCacheHelper();
   }
}
...
```

To initialize the work with report caching using the created class, it is enough to set it as the value of the **StiWebDesigner.CacheHelper** static property in the ASPX

page constructor.

3.2.11 Designer Events

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component supports events that allow you to execute necessary operations before specific actions, such as creating and editing report templates, previewing, printing and exporting, interactivity, etc. Below is a sample for processing designer events.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   OnGetReport="StiWebDesigner1_GetReport"
   OnCreateReport="StiWebDesigner1_CreateReport"
   OnSaveReport="StiWebDesigner1_SaveReport">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void StiWebDesigner1_GetReport(object sender,
StiReportDataEventArgs e)
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Reports/SimpleList.mrt"));

   e.Report = report;
}

protected void StiWebDesigner1_CreateReport(object sender,
StiReportDataEventArgs e)
{
   DataSet data = new DataSet();
   data.ReadXmlSchema(Server.MapPath("Data/Demo.xsd"));
   data.ReadXml(Server.MapPath("Data/Demo.xml"));

   e.Report.RegData(data);
   e.Report.Dictionary.Synchronize();
}
```

```
protected void StiWebDesigner1_SaveReport(object sender,
StiReportDataEventArgs e)
{
   try
   {
      e.Report.Save(Server.MapPath("Reports/" + e.Report.ReportName +
      ".mrt"));
   }
   catch (Exception ex)
   {
      e.ErrorString = ex.Message;
   }
}
...
```

Events

Name	Description
OnGetReport	The event occurs when requesting a report for editing.
OnCreateReport	The event occurs when creating new reports from the designer menu.
OnOpenReport	The event occurs when you open a report from the designer menu. In the arguments of the event, the loaded report will be sent.
OnPreviewReport	The event occurs when going to the preview tab, and when interactive activities such as using report variables, dynamic collapsing, drill-down, and sorting a report when previewing it.
OnSaveReport	The event occurs when clicking the Save button on the panel or from the main menu of the designer.
OnSaveReportAs	The event occurs when clicking the Save As button from the main menu of the designer. If the event is not specified, the report will be saved to the local disk.

OnExportReport	The event occurs when exporting reports.
OnExportReportResponse	The event occurs when after exporting reports before saving the exported report file.
OnExit	The event occurs when clicking the Exit button in the main menu of the designer.

3.2.12 Timeout

When working with the **StiWebDesigner** component, you can set the timeout for various operations — storing the report in the cache, server response, and query execution. The timeout setting is done using the component properties and report options.

CacheTimeout Property
Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
  CacheTimeout="10">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
  StiWebDesigner1.CacheTimeout = 10;
}
...
```

Using cache will increase the speed of the report designer. See the chapter Caching for more information

RequestTimeout Property

Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   RequestTimeout="10">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
   StiWebDesigner1.RequestTimeout = 30;
}
...
```

CommandTimeout Option

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to set the query timeout for the already created connection and data sources in the report.

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{

   StiReport report = new StiReport();
   report.Load(Server.MapPath("Report.mrt"));
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   StiWebDesigner1.Report = report;
}
...
```

3.2.13 Add custom functions

Information

See on GitHub example of adding a custom function in the ASP.NET HTML5 Designer component.

You can add a custom function to the **Dictionary** in the report designer when integrating it into your application. After adding the custom function, you can use this in creating reports and dashboards. Below is the example of adding a function for calculating the sum total.

Default.aspx.cs

```
...
public static decimal MySum(object value)
{
   if (!ListExt.IsList(value))
   return Stimulsoft.Base.Helpers.StiValueHelper.TryToDecimal(value);

   return Stimulsoft.Data.Functions.Funcs.SkipNulls(ListExt.ToList(value))
   .TryCastToDecimal()
   .Sum();
}
...
static _Default()
{
   StiFunctions.AddFunction("MyCategory", "MySum",
   "description", typeof(_Default),
   typeof(decimal), "Calculates a sum of the specified set of values.",
   new[] { typeof(object) },
   new[] { "values" },
   new[] { "A set of values" }).UseFullPath = false;
}
...
```

3.2.14 Settings

The **HTML5 Designer** can be configured using the component properties that can be set on the ASPX page or using the C#/VB code. Below are examples of setting designer properties.

Default.aspx

```
...
<cc1:StiWebDesigner ID="StiWebDesigner1" runat="server"
   ShowClone="false"
   ShowCrossDataBand="false"
   ShowFileMenuOpen="false"
   PermissionBusinessObjects="None"
   PermissionDataConnections="View"
   DefaultUnit="Centimeters"
   Localization="Localization/de.xml"
   Theme="Office2022WhiteTeal">
</cc1:StiWebDesigner>
...
```

Default.aspx.cs

```
...
protected void Page_Load(object sender, EventArgs e)
{
   StiWebDesigner1.ShowClone = false;
   StiWebDesigner1.ShowCrossDataBand = false;
   StiWebDesigner1.ShowFileMenuOpen = false;
   StiWebDesigner1.PermissionBusinessObjects = StiDesignerPermissions.None;
   StiWebDesigner1.PermissionDataConnections = StiDesignerPermissions.View;
   StiWebDesigner1.DefaultUnit = StiReportUnitType.Centimeters;
   StiWebDesigner1.Localization = "Localization/de.xml";
   StiWebDesigner1.Theme = StiDesignerTheme.Office2022WhiteTeal;
   StiWebDesigner1.ReportDisplayMode = StiReportDisplayMode.Auto;
}
...
```

Basic Settings

Name	Description
Width	Sets the width of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. The default width is 100%. By default, the component is expanded to the entire area of the browser window.
Height	Sets the height of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. The default height is 800

	pixels. By default, the component is expanded to the entire area of the browser window.

Server

Name	Description
RequestTimeout	Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.
CacheTimeout	Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.
CacheMode	Sets the report caching mode. It can take one of the following values of the **StiServerCacheMode** enumeration: › **None** – caching is disabled in **HTML5 Designer**; › **ObjectCache** – the cache is used as the storage, the report is stored as an object (default value); › **ObjectSession** – the session is used as the storage, the report is stored as an object; › **StringCache** – the server cache is used as the storage, the report is serialized to a packed string; › **StringSession** – the session is used as a repository, the report is serialized into a packed string.
CacheItemPriority	Sets the priority of the report stored in the server cache. This property affects the automatic clearing of the server memory in case of a lack of memory. The lower the priority is, the greater is the chance of removing information from

	memory.
AllowAutoUpdateCache	Sets the mode for automatic cache update. The report stored in the cache or server session will be automatically re-saved after a certain period of time if the designer is idle (about every 3 minutes). By default, the property is set to **true**.
UseRelativeUrls	Sets the designer mode in which relative URLs are used for requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value that specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-1** removes the port number.
PassQueryParametersForResources	Enables transferring all request URL parameters when generating links to the resources of the designer. If **false**, only the necessary parameters are used to request the resources of the designer. This corresponds to the correct operation of the browser cache. By default, the property is set to **true**.
ShowServerErrorPage	Sets a value that enables or disables the display of the detailed server error in the preview. The default value is **true**.
UseCompression	Enables compression of designer requests in the GZip stream. This allows you to reduce the amount of Internet traffic but slows down the designer. By default, the property is **false**.
UseCacheForResources	Enables caching of the component resources on the server-side. The following resources are supported: scripts, styles, and images. This option improves the load speed of the component and also reduces the server load in multi-client environments. The default value is **true**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the

	correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
Theme	Specifies the theme of the designer layout. The list of available themes can be found in the **StiDesignerTheme** enumeration. The default value is **Office2022WhiteBlue**.
CustomCss	Specifies the path to the CSS file of styles for the report designer. If this property is set, the standard styles of the selected theme will not be loaded. The default value is an empty value.
Localization	Specifies the path to the XML localization file. The path can be absolute or relative. By default, the English localization is used, which is built into the viewer and does not require additional XML files.
LocalizationDirectory	Specifies the path to the directory with XML localization files. The localization files located in the specified folder will be loaded to the localization list in the designer panel.
DefaultUnit	Sets the units for the size of the report and all its components. By default, centimeters are used.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. It can take one of the following values of the **StiZoomMode** enumeration: › **PageWidth** – when the designer runs, the zoom, necessary to display the report by the page width, will be set; › **PageHeight** – when the designer runs, the zoom, required to display the page height of

	the report, will be set.
ShowAnimation	Enables animation for various elements of the designer interface. By default, the property is set to **true**.
ShowTooltips	Enables displaying tooltips for designer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enable displaying links to online documentation in tooltips for designer controls. By default, the property is set to **true**.
ShowDialogsHelp	Enables displaying links to online documentation on the titles of dialog forms of the designer. By default, the property is set to **true**.
InterfaceType	Sets the type of interface used for the designer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the interface type of the designer will be selected automatically depending on the device used (default value); › **Mouse** – forced use of the interface to control the designer with the mouse; › **Touch** – forced use of the Touch interface to control the designer via the touch screen (mobile devices), also in this mode, the interface elements are increased.
DatePickerFirstDayOfWeek	Sets the first day of the week for the select date item. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Monday** – the first day of the week is Monday (default value); › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default, the property is set to **false**.

FormatForDateControls	This feature allows you to customize the format for date controls. By default, the current option does not have a specified value, and the date format is determined based on the browser's locale.
ShowReportTree	Enables displaying the tree of report components. By default, the property is set to **true**.
ChartRenderType	Gets or sets the type of the chart in the preview. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages in the preview tab. Can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN markup.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.

ParametersPanelSortDataItems	Sets a value that indicates that variable items will be sorted. By default, the property is set to **true**.
CloseDesignerWithoutAsking	Sets a value that indicates that the designer will be closed without asking. By default, the property is set to **true**.
ShowSystemFonts	Sets visibility of the system fonts in the fonts list. By default, the property is set to **true**.
ShowNewPageButton	Sets visibility of the **New Page** button in the designer. By default, the property is set to **true**.
ShowNewDashboardButton	Sets visibility of the **New Dashboard** button in the designer. By default, the property is set to **true**.
WizardTypeRunningAfterLoad	Calls the Report wizard after starting the report designer. It may have one of the following **StiWizardType** enumeration values: › **None** - runs the report designer without running the report wizard; › **StandardReport** - runs the Standard wizard; › **MasterDetailReport** - runs the Master-Detail wizard; › **LabelReport** - runs the Label wizard; › **InvoicesReport** - runs the Invoice wizard; › **OrdersReport** - runs the Order wizard; › **QuotationReport** - runs the Quote wizard.

Behavior

Name	Description
ShowSaveDialog	Enables displaying the dialog to insert a report name when it is saved. The name of the report will be transferred in the parameters of the report designer. By default, the property is set to **true**.

UndoMaxLevel	Sets the maximum number to cancel actions with the report (the Undo/Redo function). A big value of this property will consume memory on the server side to store the undo parameters. The default value is **6**.
AllowChangeWindowTitle	Allows using the title of the browser window to display the file name of the edited report. By default, the property is set to **true**.
SaveReportMode	Sets the mode to save the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - saving of the report is called in the current web browser window in the visible mode using the POST request; › **NewWindow** - saving of the report is called in a new window (tab) of the web browser.
SaveReportAsMode	Sets the mode for saving the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - saving of the report is called in the current web browser window in the visible mode using the POST request; › **NewWindow** - saving of the report is called in a new window (tab) of the web browser.
CheckReportBeforePreview	Sets the value that allows running the report checker before preview.

FileMenu

Name	Description
Visible	Enables displaying the main menu of the report designer. By default, the property is set to **true**.
ShowNew	Enables showing the main menu item - **New**. By default, the property is set to **true**.
ShowFileMenuNewReport	Sets a visibility of the new report button in the designer. By default, the property is set to **true**.
ShowFileMenuNewDashboard	Sets a visibility of the new dashboard button in the designer. By default, the property is set to **true**.
ShowOpen	Enables showing the main menu item - **Open**. By default, the property is set to **true**.
ShowSave	Enables showing the main menu item - **Save**. By default, the property is set to **true**.
ShowSaveAs	Enables showing the main menu item - **Save As**. By default, the property is set to **true**.
ShowClose	Enables showing the main menu item - **Close**. By default, the property is set to **true**.
ShowExit	Enables showing the main menu item - **Exit**. By default, the property is set to **false**.
ShowReportSetup	Enables showing the main menu item - **Report Setup**. By default, the property is set to **true**.
ShowOptions	Enables showing the main menu item - **Options**. By default, the property is set to **true**.
ShowInfo	Enables showing the main menu item - **Info**. By default, the property is set to **true**.
ShowAbout	Enables showing the main menu item - **About**. By default, the property is set to **true**.
ShowHelp	Enables showing the main menu item - **Help**. By default, the property is set to **true**.

Dictionary

Name	Description
Visible	Enables showing the data dictionary of the report. By default, the property is set to **true**.
UseAliases	Allows you to use aliases in the data dictionary. It has the three values of the **StiUseAliases** enumeration: › **Auto** - defines the mode of using aliases from a saved value in cookies (default value); › **True** - sets the mode of using aliases in the data dictionary; › **False** - disables the mode of using aliases in the data dictionary.
NewReportDictionary	It allows you to create a new data dictionary or join the existing one when creating a new report in the designer. It has the three values of the **StiNewReportDictionary** enumeration: › **Auto** - defines the mode to create or join the data dictionary from a saved value in cookies (default value); › **DictionaryNew** - sets the mode to create a new data dictionary when creating a new report; › **DictionaryMerge** - sets the mode to join the existing data dictionary with a new one when creating a new report in the designer.
ShowDictionaryContextMenuProperties	Sets a visibility of the **Properties** item in the dictionary context menu. By default, the property is set to **true**.
ShowDictionaryActions	Sets a visibility of the **Actions** menu on the dictionary toolbar. By default, the property is set to **true**.
PermissionDataConnections	Sets the available actions to connect data to the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataSources	Sets available actions on report data sources. It can take one or more values from the

	StiDesignerPermissions enumeration.
PermissionDataColumns	Sets the available actions on data columns in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionBusinessObjects	Sets the available actions on the business objects in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataRelations	Sets the available actions to linking data in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionVariables	Sets available actions on report variables. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionResources	Sets the available actions for the resources in the Report Dictionary. Takes one or several values from the **StiDesignerPermissions** enumeration.
PermissionSqlParameters	Sets the available actions for the parameters of the SQL queries for the Report DataSources. Takes one or several values from **StiDesignerPermissions** enumeration.
DataTransformationsPermissions	Sets the available actions on data transformation. It can take one or more values from the **StiDesignerPermissions** enumeration.

The table below shows all available values for the **StiDesignerPermissions** enumeration, which can be set for the dictionary elements of the report.

Value	Description
None	Disables any action on the item of the data dictionary.
All	Allows any action on the item of the data dictionary.

Create	Allows creating a specific data dictionary item.
Delete	Allows deleting a specific data dictionary item.
Modify	Allows modifying a specific data dictionary item.
View	Allows viewing a specific data dictionary item.
ModifyView	Allows modifying and viewing a specific data dictionary item.

Toolbar

Name	Description
ShowToolbar	Enables displaying the toolbar of the designer. By default, the property is set to **true**.
ShowSetupToolboxButton	Enables displaying the button to call the dialog box with settings for the side toolbar. By default, the property is set to **true**.
ShowInsertButton	Enables displaying the **Insert** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowLayoutButton	Enables displaying the tab **Layout** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowPageButton	Enables displaying the tab **Page** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowPreviewButton	Enables displaying the tab **Preview** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the designer. By default, the property is set to **true**.
ShowAboutButton	Enables displaying the **About** on the toolbar of the designer. By default, the property is set to **false**.

PropertiesGrid

Name	Description
Visible	Enables displaying the property panel. By default, the property is set to **true**.
Width	Sets the width of the property panel. By default, the width is set to **370 px**.
LabelWidth	Specifies the width of the labels on the properties panel. By default, the width is set to **160 px**.
PropertiesGridPosition	Sets **Left** or **Right** position of the properties grid in the designer. It has the three values of the **StiPropertiesGridPosition** enumeration: › **Left**; › **Right**.
ShowPropertiesWhichUsedFromStyles	Sets a visibility of the properties which used from styles in the designer. By default, the property is set to **false**.

Components

Name	Description
ShowText	Enables displaying the **Text** component in the insert menu for report components. By default, the property is set to **true**.
ShowTextInCells	Enables displaying the **Text in Cells** component in the insert menu for report components. By default, the property is set to **true**.
ShowRichText	Enables displaying the **Rich Text** component in the insert menu for report components. By default, the property is set to **true**.
ShowImage	Enables displaying the **Image** component in the

	insert menu for report components. By default, the property is set to **true**.
ShowBarCode	Enables displaying the **Bar Code** component in the insert menu for report components. By default, the property is set to **true**.
ShowShape	Enables displaying the **Shape** component in the insert menu for report components. By default, the property is set to **true**.
ShowHorizontalLinePrimitive	Enables displaying the **Horizontal Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowVerticalLinePrimitive	Enables displaying the **Vertical Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowRectanglePrimitive	Enables displaying the **Rectangle** component in the insert menu for report components. By default, the property is set to **true**.
ShowRoundedRectanglePrimitive	Enables displaying the **Rounded Rectangle** component in the insert menu for report components. By default, the property is set to **true**.
ShowPanel	Enables displaying the **Panel** component in the insert menu for report components. By default, the property is set to **true**.
ShowClone	Enables displaying the **Clone** component in the insert menu for report components. By default, the property is set to **true**.
ShowCheckBox	Enables displaying the **Check Box** component in the insert menu for report components. By default, the property is set to **true**.
ShowSubReport	Enables displaying the **Sub Report** component in the insert menu for report components. By default, the property is set to **true**.
ShowZipCode	Enables displaying the **Zip Code** component in

	the insert menu for report components. By default, the property is set to **true**.
ShowTable	Enables displaying the **Table** component in the insert menu for report components. By default, the property is set to **true**.
ShowCrossTab	Enables displaying the **Cross-Tab** component in the insert menu for report components. By default, the property is set to **true**.
ShowChart	Enables displaying the **Chart** component in the insert menu for report components. It affects on all chart types. By default, the property is set to **true**.
ShowMap	Enables displaying the **Map** component in the insert menu for report components. By default, the property is set to **false**.
ShowGauge	Enables displaying the **Gauge** component in the insert menu for report components. By default, the property is set to **false**.
ShowSparkline	Enables displaying the **Sparkline** component in the insert menu for report components. By default, the property is set to **true**.
ShowMathFormula	Enables displaying the **Math Formula** component in the insert menu for report components. By default, the property is set to **true**.
ShowElectronicSignature	Enables displaying the **Electronic Signature** component in the insert menu for report components. By default, the property is set to **true**.
ShowPdfDigitalSignature	Enables displaying the **PDF Digital Signature** component in the insert menu for report components. By default, the property is set to **true**.

Bands

Name	Description
ShowReportTitleBands	Enables displaying the **Report Title** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowReportSummaryBand	Enables displaying the **Report Summary** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageHeaderBand	Enables displaying the **Page Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageFooterBand	Enables displaying the **Page Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowGroupHeaderBand	Enables displaying the **Group Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowGroupFooterBand	Enables displaying the **Group Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowHeaderBand	Enables displaying the **Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowFooterBand	Enables displaying the **Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowColumnHeaderBand	Enables displaying the **Column Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowColumnFooterBand	Enables displaying the **Column Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowDataBand	Enables displaying the **Data** item in the **Bands** menu of the designer. By default, the property is set to **true**.

ShowHierarchicalBand	Enables displaying the **Hierarchical** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowChildBand	Enables displaying the **Child** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowEmptyBand	Enables displaying the **Empty** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowOverlayBand	Enables displaying the **Overlay** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowTableOfContents	Enables displaying the **Table of Contents** item in the **Bands** menu of the designer. By default, the property is set to **true**.

DashboardElements

Name	Description
ShowTableElement	Enables displaying the **Table** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowCardsElement	Enables displaying the **Cards** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowChartElement	Enables displaying the **Chart** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowGaugeElement	Enables displaying the **Gauge** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowPivotTableElement	Enables displaying the **Pivot** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.

ShowIndicatorElement	Enables displaying the **Indicator** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowProgressElement	Enables displaying the **Progress** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowRegionMapElement	Enables displaying the **Region Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowOnlineMapElement	Enables displaying the **Online Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowImageElement	Enables displaying the **Image** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTextElement	Enables displaying the **Text** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowPanelElement	Enables displaying the **Panel** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowShapeElement	Enables displaying the **Shape** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowButtonElement	Enables displaying the **Button** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowListBoxElement	Enables displaying the **List Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowComboBoxElement	Enables displaying the **Combo Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTreeViewElement	Enables displaying the **Tree View** element in the Dashboard Elements menu of the designer.

	By default, the property is set to **true**.
ShowTreeViewBoxElement	Enables displaying the **Tree View Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowDatePickerElement	Enables displaying the **Date Picker** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.

CrossBands

Name	Description
ShowCrossGroupHeaderBand	Enables displaying the **Cross Group Header** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossGroupFooterBand	Enables displaying the **Cross Group Footer** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossHeaderBand	Enables displaying the **Cross Header** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossFooterBand	Enables displaying the **Cross Footer** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossDataBand	Enables displaying the **Cross Data** section in the insert menu for report components. By default, the property is set to **true**.

Dashboards

Name	Description
ShowNewDashboardButton	Sets a visibility of the **New Dashboard** button

	in the designer. By default, the property is set to **true**.

Pages

Name	Description
ShowNewPageButton	Sets a visibility of the **New Page** button in the designer. By default, the property is set to **true**.

When designing a report or dashboard in the report designer, you can also define **ExportOptions**, **EmailOptions**, and **PreviewToolbarOptions** on the **Preview** tab. These options are similar to the report viewer options.

4 Reports and Dashboards for ASP.NET MVC

ASP.NET MVC is a technology for creating web applications and web services using the **Model-View-Controller** template. Stimulsoft provides tools for creating and displaying reports and dashboards on any device using this technology.

Tools for creating and editing reports:	Tools for viewing and converting reports:	Tools for creating and editing dashboards:	Tools for viewing and converting dashboards:
HTML5 designer	HTML5 viewer	HTML5 designer	HTML5 viewer

4.1 HTML5 Viewer

YouTube

Watch videos for the ASP.NET MVC HTML5 Viewer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions

and suggestions in the comments to the video.

Samples

See on GitHub examples for working with the ASP.NET MVC HTML5 Viewer component. All samples are separate projects grouped into one solution for Visual Studio.

The **HTML5 Viewer** (**StiMvcViewer**) component is designed for viewing reports in the web browser. You do not need to install the .NET Framework, ActiveX components, or any special plug-ins on the client-side. All you need is any modern Web browser.

With the help of the **HTML5 Viewer**, you can view, print, and export reports on any computer with any operating system installed. Since the viewer only uses HTML and JavaScript technologies, it can be run on devices with no Flash or Silverlight support - tablets, smartphones. Also, Viewer supports the following interfaces: Mobile Interface and Touch Interface. These interfaces will be switched on automatically when the mobile devices are used or for the touchscreen displays.

The **HTML5 Viewer** component uses the AJAX technology to perform all actions (uploading a report, paging, scaling, interactivity in reports, etc.), allowing you to get rid of reloading the entire page and save Web traffic, and speed up work.

The **HTML5 Viewer** supports many themes, animated interface, bookmarks, interactive reports, editing of report elements on the page, full-screen mode, search panel, and other necessary features for viewing reports.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use the **HTML5 Viewer** in a Web project, you should install the NuGet package of Stimulsoft.Reports.Web:

- Select "Manage NuGet Packages ..." in the context menu of the project;
- Specify Stimulsoft.Reports.Web in the search bar on the Browse tab;
- Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

If this is not possible, you should add the following assemblies to the project:

- Stimulsoft.Base.dll
- Stimulsoft.Report.dll
- Stimulsoft.Report.Check.dll
- Stimulsoft.Report.Helper.dll
- Stimulsoft.Report.Mvc.dll
- Stimulsoft.Report.Web.dll
- Stimulsoft.Report.WebDesign.dll

To add the ability to view and export dashboards in a Web project, install the NuGet package Stimulsoft.Dashboards.Web (this package is associated with the package Stimulsoft.Reports.Web. If it is missed, it will be installed automatically):

- Select "Manage NuGet Packages ..." in the context menu of the project;
- Specify Stimulsoft.Dashboards.Web in the search bar on the Browse tab;
- Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

If for any reason, this is not possible, you should additionally add the following assemblies to the project:

- Stimulsoft.Dashboard.dll
- Stimulsoft.Dashboard.Drawing.dll
- Stimulsoft.Dashboard.Export.dll

- How this works?
- Activation
- Showing Reports and Dashboards
- Connecting data
- Interactive Reports
- Timeout
- Editing Rendered Reports
- Sending Reports by Email

- Localization
- Printing Reports
- Exporting Reports and Dashboards
- Viewing Modes
- Work with Parameters
- Work with Bookmarks
- Viewer Settings
- Calling Designer from Viewer
- Caching
- Using Themes
- Basic Features
- Additional Methods
- Export and Printing from Code

4.1.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the viewer, you need to place the **StiMvcViewer** component on the page, set the necessary settings, and set the necessary actions in the view controller. When the report viewer runs, the following actions occur:

› The .NET component generates HTML and JavaScript code that is necessary for displaying and running the viewer;

› When the component is output, the JavaScript method is launched. It requests the first page of the report on the server-side or the entire report (depending on the selected mode) and the required report parameters;

› Each action in the viewer (for example, paging, printing or exporting a report, etc.) calls a certain action on the server-side, in which you can perform the necessary manipulations with the report;

› To speed up the work, the viewer saves the report in cache or server session, which makes it impossible to re-build the report.

4.1.2 Activation

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by

downloading a file with the license key. Below is an example of activating the **StiMvcViewer** component.

HomeController.cs

```
...
public class HomeController : Controller
{
   static HomeController()
   {
      //Activation with using license code
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";

      //Activation with using license file
      var path = System.Web.HttpContext.Current.Server.MapPath("~/Content/
      license.key");
      Stimulsoft.Base.StiLicense.LoadFromFile(path);
   }
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

4.1.3 Showing Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `Report.Render()` method if you want to perform specific actions with the rendered report before it is displayed in the viewer. Likewise, when using the compilation mode, you need to call the `Report.Compile()` method only if you have actions to perform with the compiled report before it is built and shown in the viewer.

To show the report, you need to add the **StiMvcViewer** component to the page, set it to the minimum required properties, and specify the necessary actions in the view controller.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Actions =
      {
         GetReport = "GetReport",
         ViewerEvent = "ViewerEvent"
      }
})
...
```

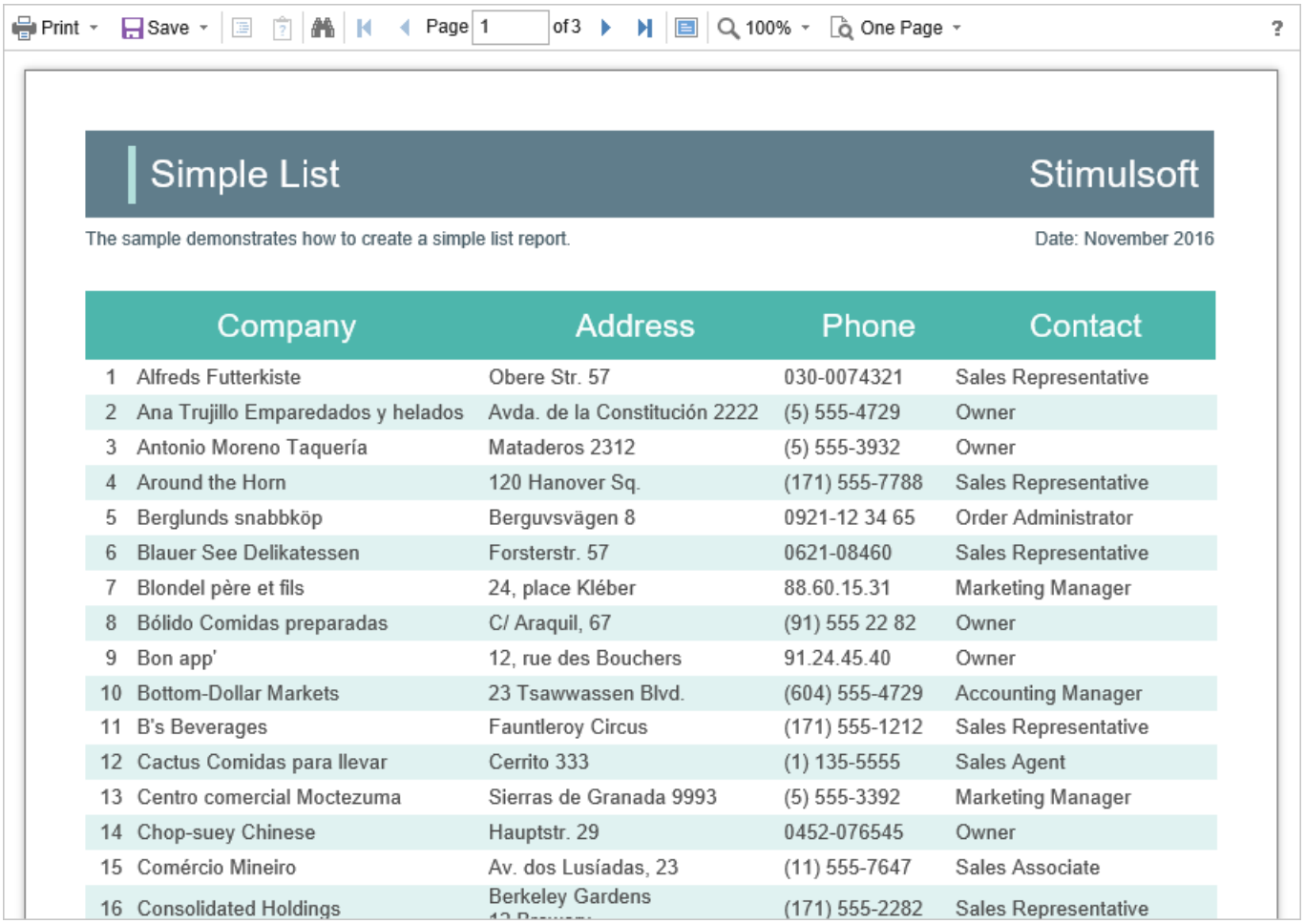
HomeController.cs

```
...
public ActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("~/Content/SimpleList.mrt"));
   //report.Load(Server.MapPath("~/Content/Dashboard.mrt"));

   return StiMvcViewer.GetReportResult(report);
}

public ActionResult ViewerEvent()
{
   return StiMvcViewer.ViewerEventResult();
}
...
```

In the above example, the processing of two actions of the viewer is added. The **GetReport** action returns the report prepared for preview. The **ViewerEvent** action handles the viewer events.

Information

The **ViewerEvent** action is mandatory. Without it, the correct operation of the viewer is not possible.



	Company	Address	Phone	Contact
1	Alfreds Futterkiste	Obere Str. 57	030-0074321	Sales Representative
2	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	(5) 555-4729	Owner
3	Antonio Moreno Taquería	Mataderos 2312	(5) 555-3932	Owner
4	Around the Horn	120 Hanover Sq.	(171) 555-7788	Sales Representative
5	Berglunds snabbköp	Berguvsvägen 8	0921-12 34 65	Order Administrator
6	Blauer See Delikatessen	Forsterstr. 57	0621-08460	Sales Representative
7	Blondel père et fils	24, place Kléber	88.60.15.31	Marketing Manager
8	Bólido Comidas preparadas	C/ Araquil, 67	(91) 555 22 82	Owner
9	Bon app'	12, rue des Bouchers	91.24.45.40	Owner
10	Bottom-Dollar Markets	23 Tsawwassen Blvd.	(604) 555-4729	Accounting Manager
11	B's Beverages	Fauntleroy Circus	(171) 555-1212	Sales Representative
12	Cactus Comidas para llevar	Cerrito 333	(1) 135-5555	Sales Agent
13	Centro comercial Moctezuma	Sierras de Granada 9993	(5) 555-3392	Marketing Manager
14	Chop-suey Chinese	Hauptstr. 29	0452-076545	Owner
15	Comércio Mineiro	Av. dos Lusíadas, 23	(11) 555-7647	Sales Associate
16	Consolidated Holdings	Berkeley Gardens	(171) 555-2282	Sales Representative

If the report is not rendered before showing, the **HTML5 Viewer** component will automatically render it. Thus, you can use report templates, rendered reports, and reports in the form of classes to show the report.

HomeController.cs

```
...
public ActionResult GetReport()
{
   StiReport report = new StiReport();
   report.LoadDocument(Server.MapPath("~/Content/SimpleList.mdc"));

   return StiMvcViewer.GetReportResult(report);
}
...
```

HomeController.cs

```
...
public ActionResult GetReport()
{
   StiReport report = new StiReportCompiledClass();

   return StiMvcViewer.GetReportResult(report);
}
...
```

Since the dashboard is not a static document and requires data to work, the format of the rendered MDC document is not available for it. Instead, it is possible to use a snapshot of the report in the MRT format, which contains all the data necessary for the dashboard to work and can be correctly displayed in the viewer.

Default.aspx.cs

```
...
public ActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("~/Reports/Snapshot.mrt"));

   return StiMvcViewer.GetReportResult(report);
}
...
```

Loading custom fonts

You can load custom fonts using the **StiFontCollection** class, having specified the file that contains a font. To do this, you should call the static method in the controller constructor to load a font.

ViewerController.cs

```
...
public class ViewerController : Controller
{
   static ViewerController()
   {
      Stimulsoft.Base.StiFontCollection.AddFontFile(Server.MapPath("~/
      fonts/my-font/font-name.ttf"));
   }
}
...
```

4.1.4 Connecting Data

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Data to a report can be connected in various ways. The easiest way is to store connection settings in the report template. You can also connect the data from the code. You can do this when the report is loaded in the **GetReport** action.

HomeController.cs

```
...
public ActionResult GetReport()
{
   DataSet ds = new DataSet();
   ds.ReadXml(Server.MapPath("~/Content/Data/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(Server.MapPath("~/Content/TwoSimpleLists.mrt"));
   report.Dictionary.Databases.Clear();
   report.RegData("Demo", ds);

   return StiMvcViewer.GetReportResult(report);
}
...
```

Data for the report can be connected not only when the report is loaded. For example, you can connect new data at the moment of interactive actions in the viewer (applying report parameters, sorting, drill-down, collapsing). To do this, you should set the **Interaction** action for the **HTML5 Viewer** component and, in the action handler, connect the data for the current report. In the same way, you can connect data in other actions of the viewer.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Actions =
      {
```

```
        GetReport = "GetReport",
        ViewerEvent = "ViewerEvent",
        Interaction = "ViewerInteraction"
    }
})
...
```

HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   DataSet data = new DataSet();
   data.ReadXml(Server.MapPath("~/Content/Data/Demo.xml"));

   StiReport report = StiMvcViewer.GetReportObject();
   report.RegData("Demo", data);

   return StiMvcViewer.InteractionResult(report);
}
...
```

If you want to connect new data only for a certain interactive action of the viewer, for example, only when you apply report parameters you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class. They are passed to any server-side on any request and contain all necessary information and states of the client part of the viewer. To determine the type of action of the viewer, it is enough to check the **Action** property of the viewer parameters.

HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   StiRequestParams requestParams = StiMvcViewer.GetRequestParams();
   if (requestParams.Action == StiAction.Variables)
   {
      DataSet data = new DataSet();
      data.ReadXml(Server.MapPath("~/Content/Data/Demo.xml"));

      StiReport report = StiMvcViewer.GetReportObject();
      report.RegData("Demo", data);

      return StiMvcViewer.InteractionResult(report);
   }

   return StiMvcViewer.InteractionResult();
}
...
```

SQL data sources

The connection parameters to the SQL data source and any other ones can be stored in the report template. Suppose you want to set the connection parameters from the code before rendering the report (for example, for security reasons or depending on the authorized user). In that case, you can use the example below.

HomeController.cs

```
...
public ActionResult GetReport()
{
   OracleConnection connection = new OracleConnection("Data
   Source=Oracle8i;Integrated Security=yes");
   connection.Open();
   OracleDataAdapter adapter = new OracleDataAdapter();
   adapter.SelectCommand = new OracleCommand("SELECT * FROM Products",
   connection);

   DataSet dataSet = new DataSet("productsDataSet");
   adapter.Fill(dataSet, "Products");

   StiReport report = new StiReport();
   report.Load(Server.MapPath("~/Content/SqlSampleReport.mrt"));
   report.RegData("Products", dataSet);

   return StiMvcViewer.GetReportResult(report);
}
...
```

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to change the connection string for MS SQL, adjust the query, set the query timeout for the already created connection and data sources in the report.

HomeController.cs

```
...
public ActionResult GetReport()
{

   StiReport report = new StiReport();
```

```
    report.Load(Server.MapPath("Report.mrt"));
    ((StiSqlDatabase)
    report.Dictionary.Databases["Connection"]).ConnectionString = @"Data
    Source=server;Integrated Security=True;Initial Catalog=DataBase";
    ((StiSqlSource)
    report.Dictionary.DataSources["DataSourceName"]).SqlCommand = "select *
    from Table where Column = 100";
    ((StiSqlSource)
    report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

    return StiMvcViewer.GetReportResult(report);
}
...
```

Information

For SQL data sources of other types, the connection is created similarly, and an adapter corresponding to the data source type is connected. For example, for the MS SQL data source, you should connect SqlDataAdapter. For OLE DB - OleDbDataAdapter is required. Also, you should specify a connection string that matches the connection type.

You can also use data for designing reports and dashboards obtained from OData storage. In this case, you can do the authorization using a username, user password, or token. Authorization parameters are specified in the connection string to the OData storage using the ";" separator.

HomeController.cs

```
...
public ActionResult GetReport()
{

    var report = new StiReport();

    //Authorization using a user account
    var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
    services.odata.org/V4/Northwind/
    Northwind.svc;AddressBearer=adress;UserName=UserName;Password=Password;C
    lient_Id=Your Client ID", false, null);

    //Authorization using a user token
    var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
    services.odata.org/V4/Northwind/Northwind.svc;Token=Enter your token",
    false, null);
```

```
    report.Dictionary.Databases.Add(oDataDatabase);
    oDataDatabase.Synchronize(report);

    //Query with data filter
    ((StiSqlSource)report.Dictionary.DataSources["Products"]).SqlCommand =
    "Products?$filter=ProductID eq 2";

    return StiMvcViewer.GetReportResult(report);
}
...
```

The table below shows the connection string templates for different types of data sources.

Data Source	Connection String Template
MS SQL	Integrated Security=False; Data Source=myServerAddress;Initial Catalog=myDataBase; User ID=myUsername; Password=myPassword;
MySQL	Server=myServerAddress; Database=myDataBase;UserId=myUsername; Pwd=myPassword;
ODBC	Driver={SQL Server}; Server=myServerAddress;Database=myDataBase ; Uid=myUsername; Pwd=myPassword;
OLE DB	Provider=SQLOLEDB.1; Integrated Security=SSPI;Persist Security Info=False; Initial Catalog=myDataBase;Data Source=myServerAddress
Oracle	Data Source=TORCL;User Id=myUsername;Password=myPassword;
MS Access	Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Password=pass;Data Source=C:\ \myAccessFile.accdb;
PostgreSQL	Server=myServerAddress; Port=5432; Database=myDataBase;User Id=myUsername; Password=myPassword;
Firebird	User=SYSDBA; Password=masterkey;

	Database=SampleDatabase.fdb;DataSource=myServerAddress; Port=3050; Dialect=3; Charset=NONE;Role=; Connection lifetime=15; Pooling=true; MinPoolSize=0;MaxPoolSize=50; Packet Size=8192; ServerType=0;
SQL CE	Data Source=c:\MyData.sdf; Persist Security Info=False;
SQLite	Data Source=c:\mydb.db; Version=3;
DB2	Server=myAddress:myPortNumber;Database=myDataBase;UID=myUsername;PWD=myPassword; Max Pool Size=100;Min Pool Size=10;
Infomix	Database=myDataBase;Host=192.168.10.10;Server=db_engine_tcp;Service=1492;Protocol=onsoctcp;UID=myUsername;Password=myPassword;
Sybase	Data Source=myASEserver;Port=5000;Database=myDataBase;Uid=myUsername;Pwd=myPassword;
Teradata	Data Source=myServerAddress;User ID=myUsername;Password=myPassword;
VistaDB	Data Source=D:\folder \myVistaDatabaseFile.vdb4;Open Mode=ExclusiveReadWrite;
Universal(dotConnect)	Provider=Oracle;direct=true;data source=192.168.0.1;port=1521;sid=sid;user=user; password=pass
MongoDB	mongodb://<user>:<password>@localhost/test
OData	http://services.odata.org/v3/odata/OData.svc/
Other...	The table shows the most commonly used templates for the connection string. You can view various connection string options at the special website.

Data from XML, JSON, Excel files

Connecting to XML and JSON data sources can be stored in the report template. If you want to specify data files from the code, you can use the example below.

HomeController.cs

```
...
public ActionResult GetReport()
{
   DataSet data = new DataSet();
   data.ReadXml(Server.MapPath("~/Content/Data/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(Server.MapPath("~/Content/SimpleList.mrt"));
   report.RegData(data);

   return StiMvcViewer.GetReportResult(report);
}
...
```

HomeController.cs

```
...
public ActionResult GetReport()
{
   DataSet data
   = StiJsonToDataSetConverterV2.GetDataSetFromFile(Server.MapPath("~/
   Content/Data/Demo.json"));

   StiReport report = new StiReport();
   report.Load(Server.MapPath("~/Content/SimpleList.mrt"));
   report.RegData(data);

   return StiMvcViewer.GetReportResult(report);
}
...
```

Information

The viewer has the possibility of obtaining data from an Excel file. To do this, you can use the following method.

```
DataSet dataSet = StiExcelConnector.Get().GetDataSet(new StiExcelOptions(ar
```

4.1.5 Localization

The **HTML5 Viewer** component supports the complete localization of its interface. To localize the report viewer interface, use the special **Localization** property. The value of this property should specify the path to the localization XML file (relative or

absolute).

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Localization = "~/Content/Localization/en.xml"
})
...
```

When you load the report viewer, the localization file will be loaded automatically.

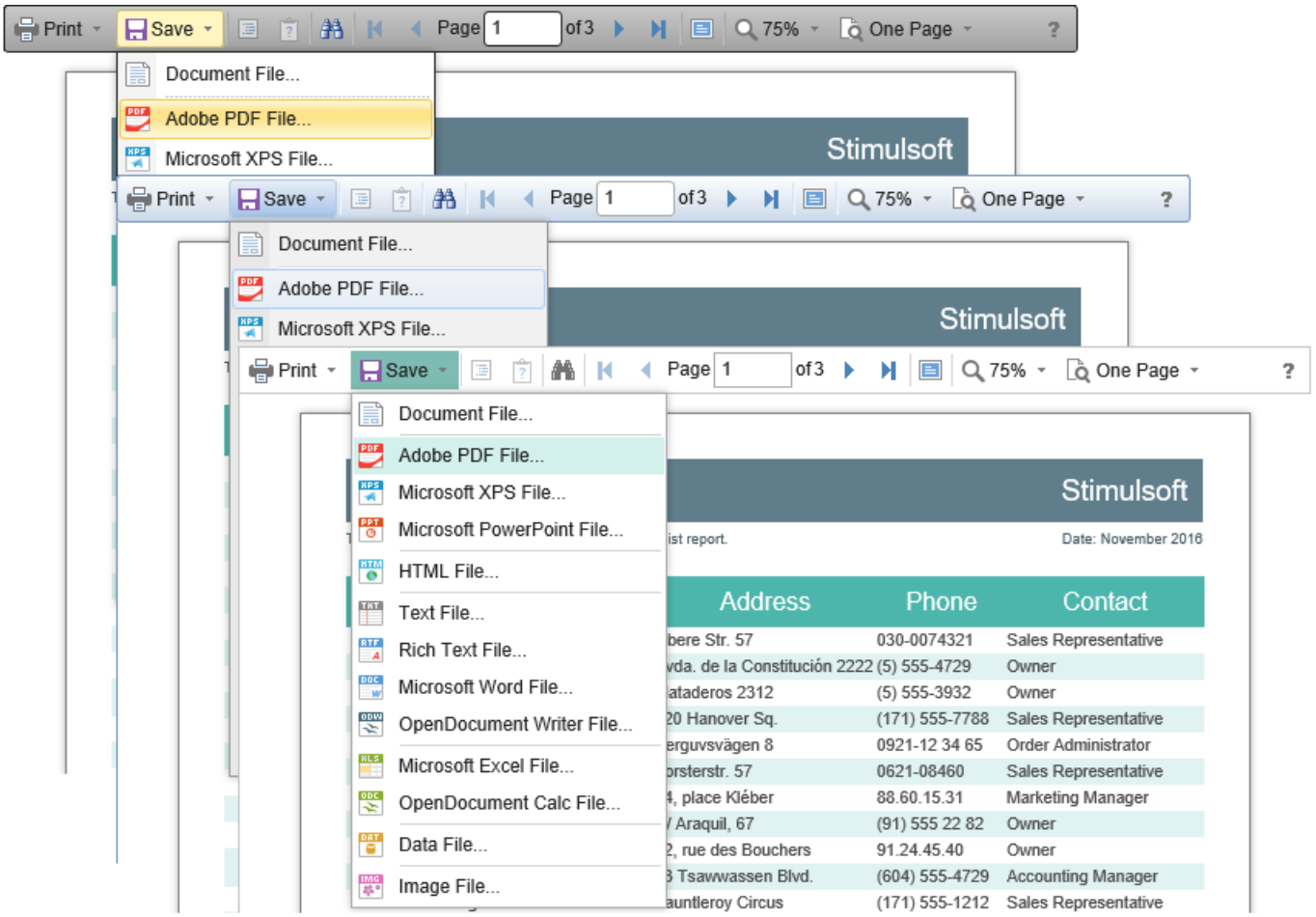
4.1.6 Using Themes

The **HTML5 Viewer** component can change the appearance of visual controls. To change the theme, use the **Theme** property, which can take one of the values of the **StiViewerTheme** enumeration.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Theme = StiTheme.Office2022WhiteTeal
})
...
```

There are currently **8 themes** available with different color accents. As a result, **more than 60** variants of the appearance are available. This allows you to customize the appearance of the viewer for almost any design of the Web project.

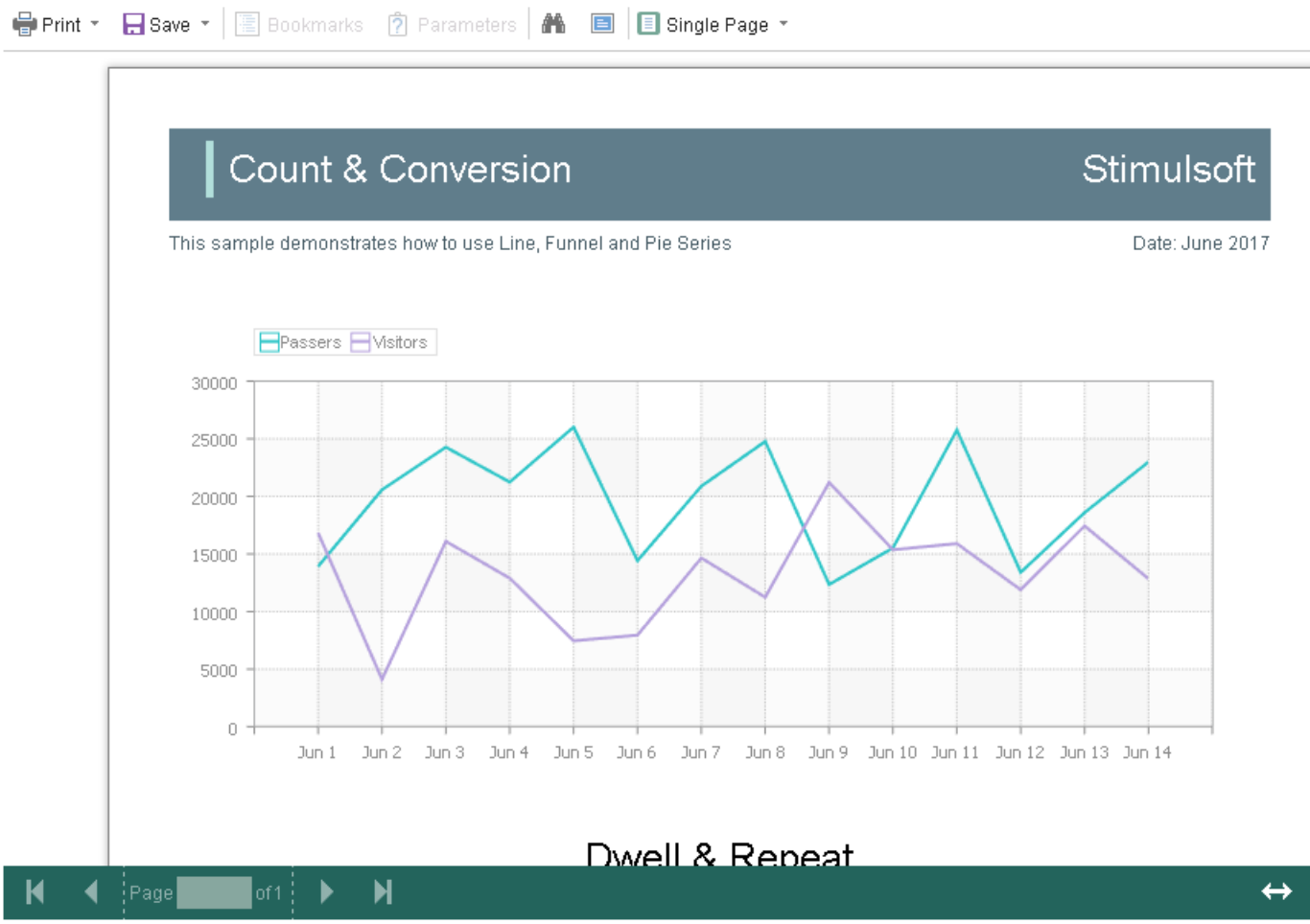


By default, the viewer has only the top toolbar on which all the report controls are located. If necessary, the toolbar can be split into top and bottom parts. The top panel will contain the menu for printing and exporting the report and the buttons for working with parameters and bookmarks. The bottom panel will contain controls to switch between the report pages and setting the zoom of pages. To enable this mode, enable the **ToolbarDisplayMode** property. It has values **Simple** and **Separated**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Appearance =
      {
         ScrollbarsMode = true
      },
      Toolbar =
      {
```

```
        DisplayMode = StiToolbarDisplayMode.Separated
    }
})
...
```



In addition, it is possible to set the appearance parameters for the main elements of the viewer. For example, you can change the font and color of the control panel inscriptions of the viewer, set the background of the viewer, set the color of page borders, etc. Below is a list of available properties that change the appearance of the viewer and their default values.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
```

```
    Appearance =
    {
      BackgroundColor = Color.White,
      PageBorderColor = Color.Blue,
      ShowPageShadow = true
    },
    Toolbar =
    {
      BackgroundColor = Color.White,
      BorderColor = Color.Gray,
      FontColor = Color.Black,
      FontFamily = "Arial"
    }
})
...
```

4.1.7 Basic Features

The main features of the viewer include the following operations:

- Displaying the report.
- Switching between the report pages.
- Changing the scale.
- Displaying the preview mode.

All specified operations are performed in the AJAX mode without restarting the browser page. For the correct work of these operations, you should define a special ViewerEvent action.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
  new StiMvcViewerOptions() {
    Actions =
    {
      ViewerEvent = "ViewerEvent"
    }
})
...
```

HomeController.cs

```
...
public ActionResult ViewerEvent()
{
  // Some code before viewer event
  // ...

  return StiMvcViewer.ViewerEventResult();
}
...
```

Information

This action is mandatory. Without it, the correct operation of the viewer is not possible.

The **ViewerEvent** action returns a prepared HTML page of the report (or set of pages), built taking into account the current state of the viewer. If necessary, you can change the parameters of the current report in the specified action and update the report data in case of interactive actions of the viewer.

4.1.8 Printing Reports

Information

Please note that the print option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component provides several options for printing a report. Each has its advantages and disadvantages.

Print to PDF
Printing will be done by exporting the report to PDF. The advantages are greater accuracy of positioning and printing of the report elements compared to other printing options. Among the drawbacks is the mandatory presence of a plug-in installed in a web browser for viewing PDF files (modern browsers have embedded PDF viewer and printer).

Print with Preview
The report will be printed in a separate pop-up browser window in HTML. The report can be previewed and then sent to the printer or copied to another location as text or HTML code. Advantages - cross-browser compatibility when printing, no need to install special plug-ins. The disadvantage is the relatively low accuracy of the position of the report elements due to the peculiarities of the implementation of HTML formatting.

Print without Preview

The report will be printed directly to the printer without preview. After selecting this menu item, the system print dialog is displayed. Since printing in this mode is carried out in the HTML format, the print quality is similar to printing a report with a preview.

Information

When printing to the **HTML format**, you should check the compliance of report page settings and printer parameters (paper size, orientation, margins, indents) and check your browser print settings, such as margins, headers, footers watermarks printing, color printing.

The print function does not require additional settings for the viewer. Before printing a report, you can define a special **PrintReport** action if you need to perform any actions.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Actions =
      {
         PrintReport = "PrintReport"
      }
})
...
```

HomeController.cs

```
...
public ActionResult PrintReport()
{
   // Some code before print
   // ...

   return StiMvcViewer.PrintReportResult();
}
...
```

Print setup

If you choose to print a report in the viewer panel, a menu with printing options is displayed. The **HTML5 Viewer** component is able to force the required printing mode. To do this, set the **PrintDestination** property to one of the following values of the **StiPrintDestination** enumeration.

- **Default** – the menu will be displayed (the default property value);
- **Pdf** – print to the PDF format;
- **Direct** – printing to the HTML format directly to the printer, the system print dialog will be displayed;
- **WithPreview** – print to the HTML format with preview in a pop-up window.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Toolbar =
      {
         PrintDestination = StiPrintDestination.Default
      }
})
...
```

The **HTML5 Viewer** component can completely disable report printing. To do this, set the value of the **ShowPrintButton** property to **false**.

Index.cshtml

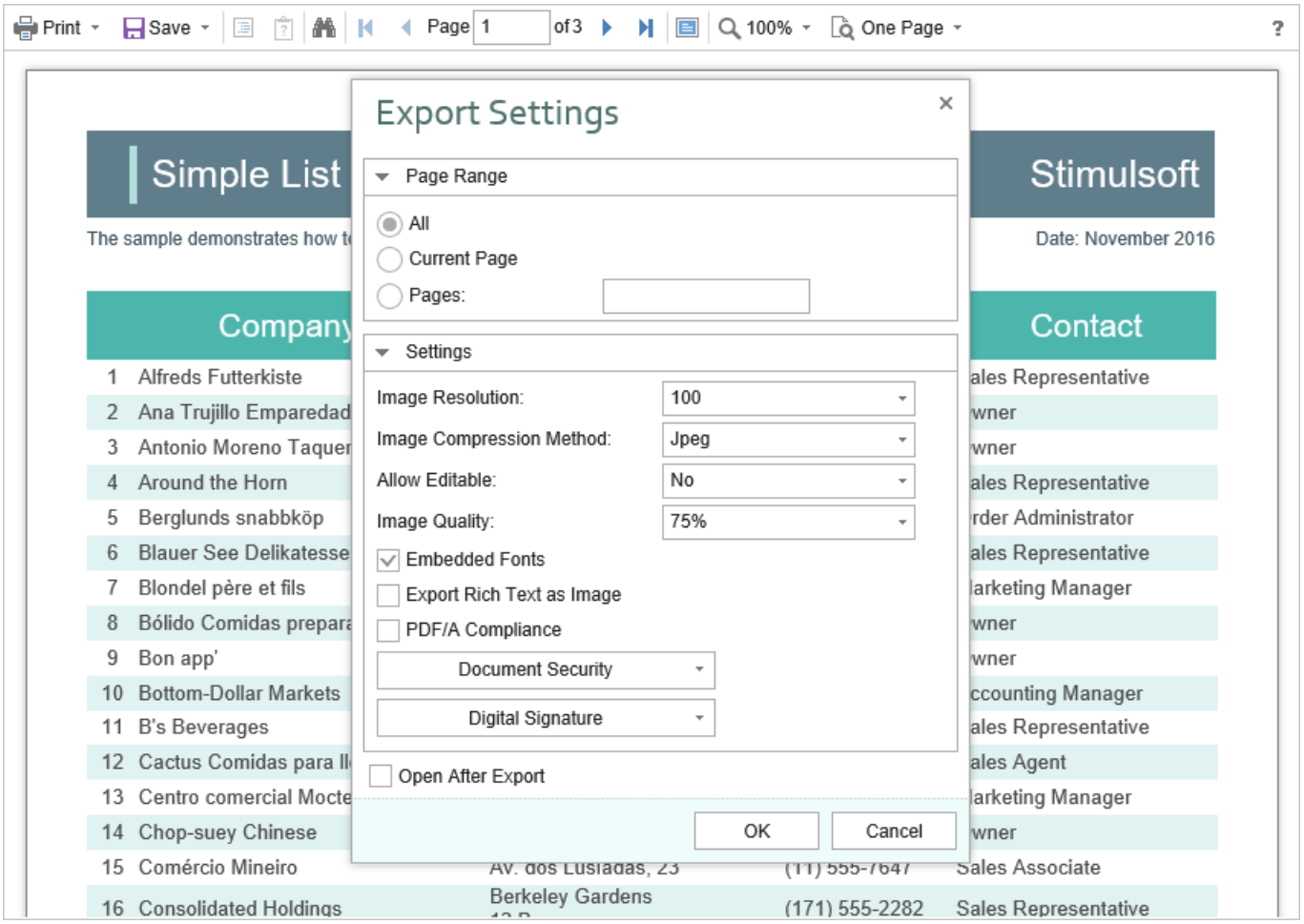
```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Toolbar =
      {
         ShowPrintButton = false
      }
})
...
```

4.1.9 Exporting Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component allows you to export the displayed report to three dozen various formats, such as **PDF**, **HTML**, **Word**, **Excel**, **XPS**, **RTF**, **images**, **text**, and others. You may export the dashboard to PDF, Excel, image files.



The export function does not require additional settings for the viewer. If you need to perform any actions before exporting the report, you can define a special **ExportReport** action.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
```

```
    Actions =
    {
      ExportReport = "ExportReport"
    }
})
...
```

HomeController.cs

```
...
public ActionResult ExportReport()
{
  // Some code before export
  // ...

  return StiMvcViewer.ExportReportResult();
}
...
```

Export settings

Each report export format of the **HTML5 Viewer** component has a lot of settings, and each setting has its default values. Sometimes you need to set other default values. For this purpose, a special **DefaultSettings** property of the viewer is used. It is a container for all the default export settings.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
  new StiMvcViewerOptions() {
    Exports =
    {
      DefaultSettings =
      {
        ExportToPdf =
        {
          ImageQuality = 0.75f,
          ImageFormat = Stimulsoft.Report.Export.StiImageFormat.Color
        },
        ExportToHtml =
        {
          ExportMode = Stimulsoft.Report.Export.StiHtmlExportMode.Div,
          UseEmbeddedImages = true
        }
      }
    }
})
...
```

If it is required, you can completely hide export dialogs. Exporting will always be done with default settings. For this, it is enough to set the value of the **ShowExportDialog** property to **false**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Exports =
      {
         ShowExportDialog = false
      }
})
...
```

The **HTML5 Viewer** component contains 30+ export formats, and sometimes you need to disable unwanted formats. This allows you to simplify UI and the use of the viewer. To disable unused export formats, it is enough to set the values for the corresponding properties of the viewer listed in the list below to **false**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Exports =
      {
         ShowExportToDocument = true,
         ShowExportToPdf = true,
         ShowExportToXps = true,
         ShowExportToPowerPoint = true,
         ShowExportToHtml = true,
         ShowExportToHtml5 = true,
         ShowExportToMht = true,
         ShowExportToText = true,
         ShowExportToRtf = true,
         ShowExportToWord2007 = true,
         ShowExportToOpenDocumentWriter = true,
         ShowExportToExcel = true,
         ShowExportToExcelXml = true,
         ShowExportToExcel2007 = true,
         ShowExportToOpenDocumentCalc = true,
         ShowExportToCsv = true,
         ShowExportToDbf = true,
         ShowExportToXml = true,
         ShowExportToDif = true,
         ShowExportToSylk = true,
         ShowExportToImageBmp = true,
```

```
        ShowExportToImageGif = true,
        ShowExportToImageJpeg = true,
        ShowExportToImagePcx = true,
        ShowExportToImagePng = true,
        ShowExportToImageTiff = true,
        ShowExportToImageMetafile = true,
        ShowExportToImageSvg = true,
        ShowExportToImageSvgz = true
      }
})
...
```

The **HTML5 Viewer** component can completely disable the export menu. To do this, set the value of the **ShowSaveButton** property to **false**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Toolbar =
      {
         ShowSaveButton = false
      }
})
...
```

4.1.10 Viewing Modes

The **HTML5 Viewer** component has two modes for displaying reports - with and without scrollbars. By default, the view mode without scrollbars is set. To enable the scrollbar view mode, set the value of the **ScrollbarsMode** property to **true**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Appearance =
      {
         ScrollbarsMode = true
      }
})
...
```

In the first mode (without scrollbars), the viewer displays a page or report as a whole, automatically stretching the preview space. If the width and height are specified, the viewer will truncate the page that is out of bounds. In the second mode, unlike the

first one, when the page is out of bounds of the viewer's size, no truncation will be performed. Scrollbars will appear, using which you can view the entire page or report.

Information

In the report mode with scrollbars, you should set the height of the viewer. Otherwise, the default height will be set to **650 pixels**.

The **HTML5 Viewer** component provides the full-screen report and dashboard mode. By default, the standard view mode is enabled, the viewer has the specified dimensions in the settings. To enable the full-screen mode, set the **FullScreenMode** property to **true**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Appearance =
      {
         FullScreenMode = true
      }
})
...
```

Also, to enable or disable the full-screen mode, you can use the corresponding button on the control panel of the viewer.

The **HTML5 Viewer** component has three modes to display reports - page-by-page, entire report, and tabular display of report pages. To control the modes, the **ViewMode** property is used. It can have one of the specified values - **SinglePage**, **Continuous**, **MultiplePages**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Toolbar =
      {
```

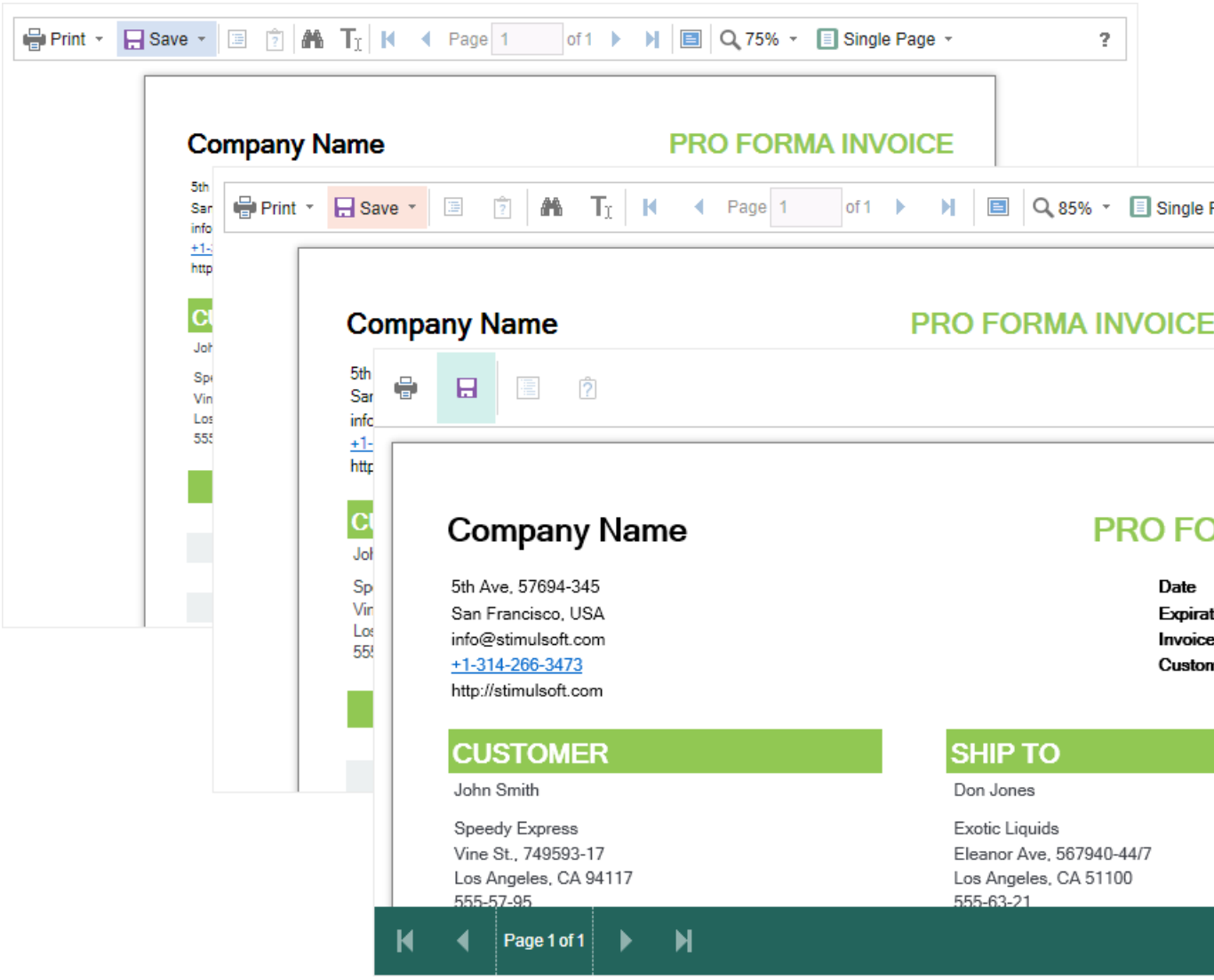
```
        ViewMode = StiWebViewMode.OnePage
      }
})
...
```

HTML5 Viewer component supports interaction on a regular PC display and works with a touchscreen of screens and mobile devices. **InterfaceType** property allows controlling the interface modes. The property can have one of the following values:

- **Auto** – the viewer's interface is determined automatically depending on the device the report is displayed on. This is the default value.
- **Mouse** – the standard interface with a mouse control will be used for all the screen types.
- **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability.
- **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with smartphones.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Appearance =
      {
         InterfaceType = StiInterfaceType.Auto
      }
})
...
```



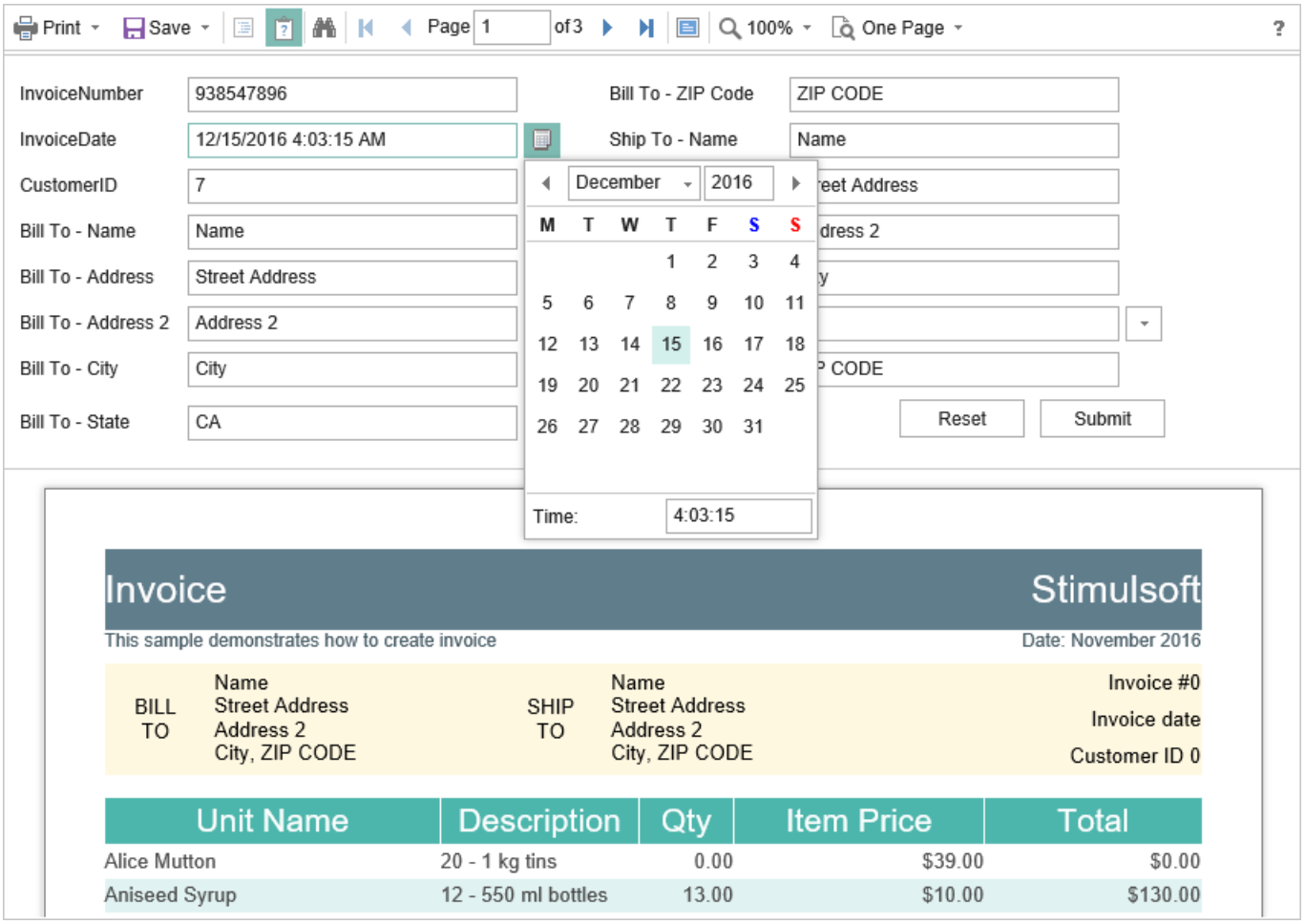
4.1.11 Work with Parameters

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To work with report parameters in the **HTML5 Viewer**, there is a special settings panel. To add a parameter to the panel, you need to define a variable in a report requested by the user. When viewing a report in the viewer, such a variable will be

automatically added to the settings panel. It supports all report variables (normal variables, date and time, borders, lists, etc.).



To work with reports with parameters, no additional viewer settings are required. If you need to perform some actions before applying the parameters, you can define a special **Interaction** action.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Actions =
      {
         Interaction = "ViewerInteraction"
      }
})
...
```

HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   // Some code before any interaction
   // ...

   return StiMvcViewer.InteractionResult();
}
...
```

This action is called during any interactive actions of the viewer. If you need to perform any actions only when applying report parameters, you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class. They are passed to any server-side on any request and contain all necessary information and states of the client part of the viewer. To determine the type of action of the viewer, it is enough to check the Action property of the viewer parameters.

HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   StiRequestParams requestParams = StiMvcViewer.GetRequestParams();
   if (requestParams.Action == StiAction.Variables)
   {
      // Some code before apply parameters
   }

   return StiMvcViewer.InteractionResult();
}
...
```

If you do not need to work with parameters, you can completely disable this feature. To do this, use the **ShowParametersButton** property in the **Toolbar** section of properties, which should be set to **false**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Toolbar =
      {
```

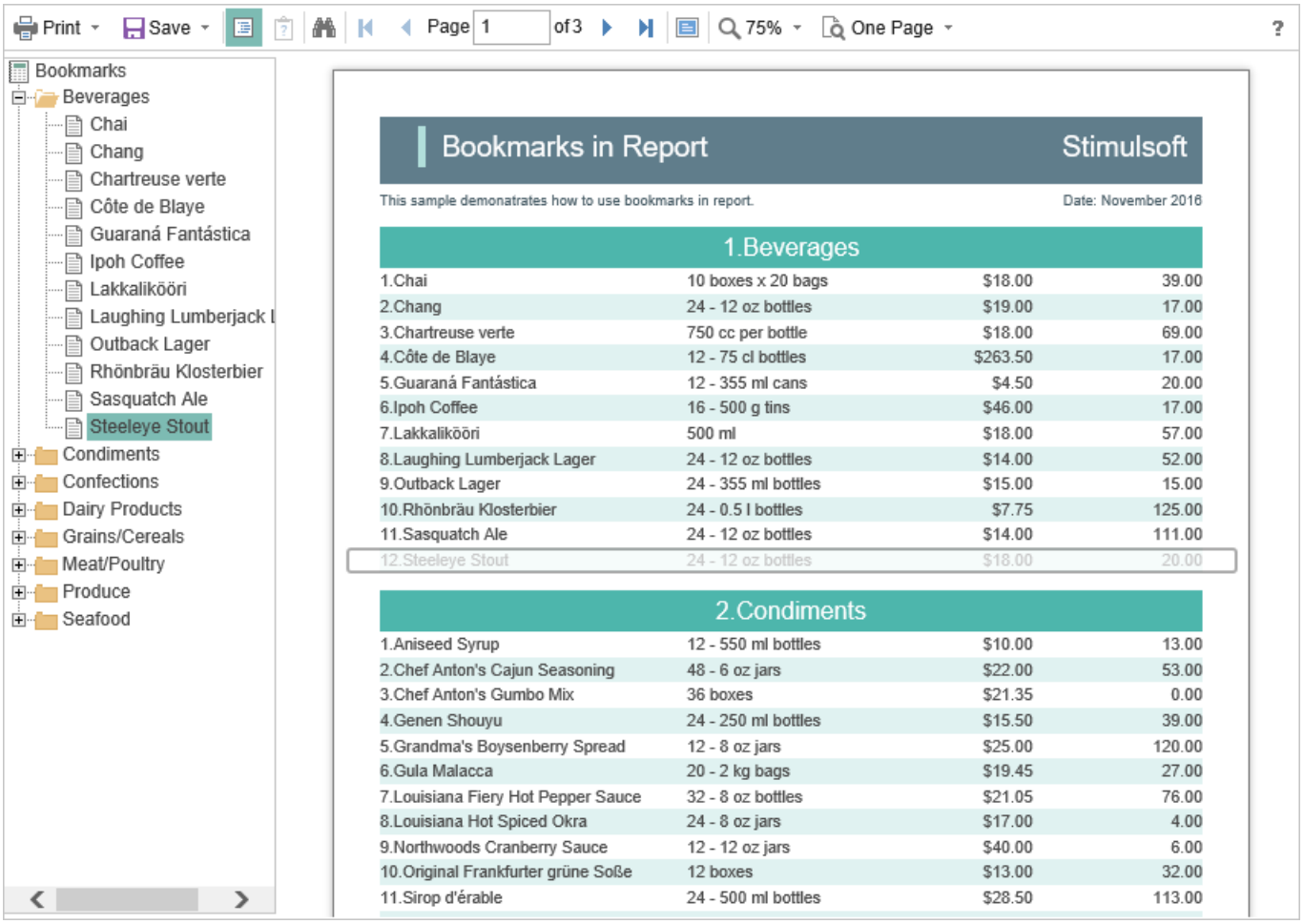
```
        ShowParametersButton = false
    }
})
...
```

Information

The options panel will not be displayed with such a viewer configuration, even if the parameters are present in the displayed report.

4.1.12 Work with Bookmarks

The **HTML5 Viewer** component supports report bookmarks. A panel with bookmarks will be displayed when displaying such a report on the left side of the page. When you select a bookmark of the report, the viewer will carry out an automatic transition to the specified page, and the report item with a bookmark is highlighted.



By default, the bookmarks bar width is 180 pixels. The **HTML5 Viewer** component allows you to change this value. For this, the **BookmarksTreeWidth** property, which value is specified in pixels, is used.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Appearance =
      {
         BookmarksTreeWidth = 200
      }
})
...
```

If work with report bookmarks is not required, you can disable this feature. For this, set the **ShowBookmarksButton** property to **false**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Toolbar =
      {
         ShowBookmarksButton = false
      }
})
...
```

Information

In this case, report bookmarks will not be displayed, even if they are present in the displayed report. This property has no effect on printing and exporting reports.

When printing a report with bookmarks, the bookmark tree will be hidden. If you want to print bookmarks with the report, it is necessary to set the **BookmarksPrint** property to **true**.

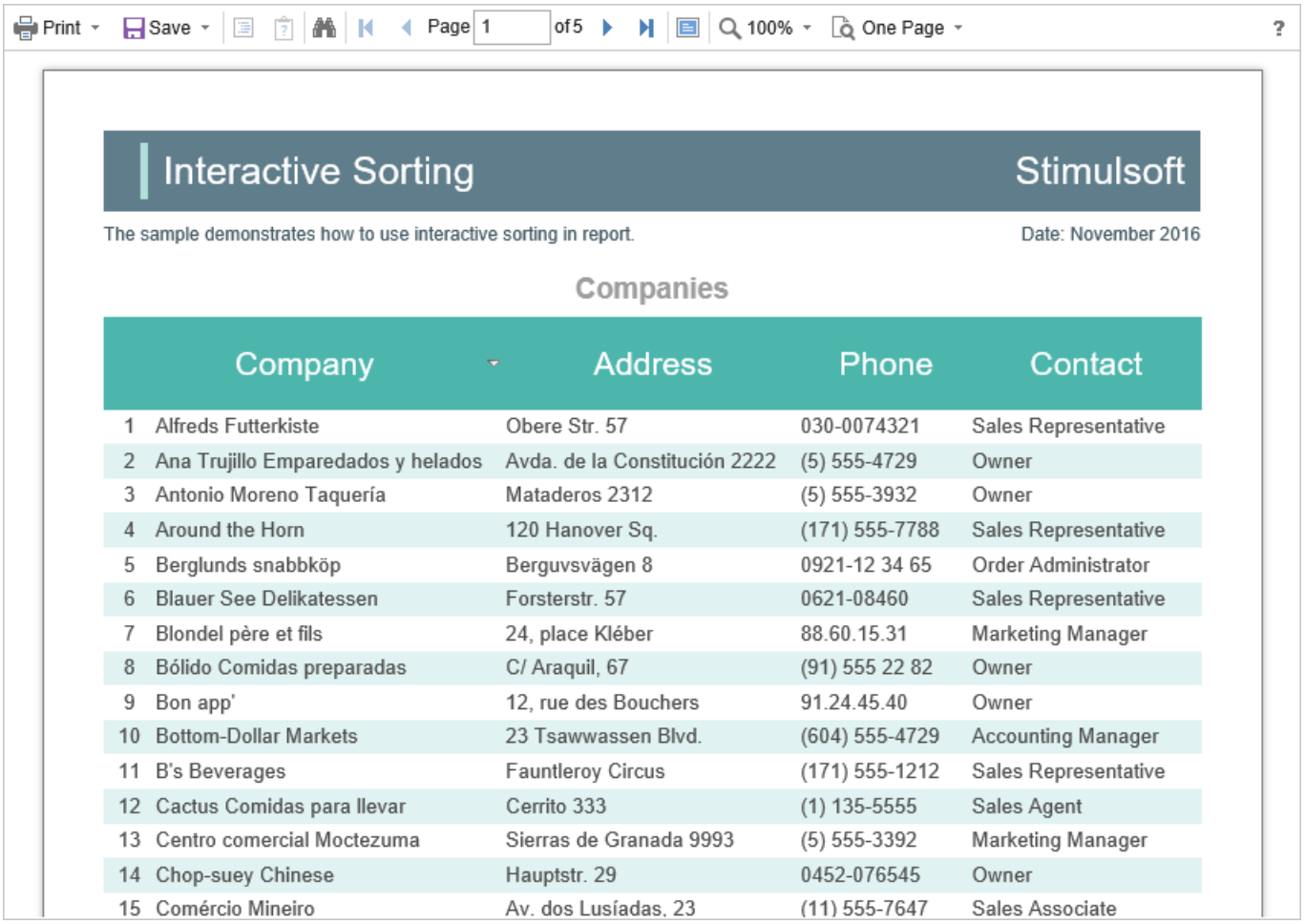
Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Appearance =
      {
         BookmarksPrint = true
      }
})
...
```

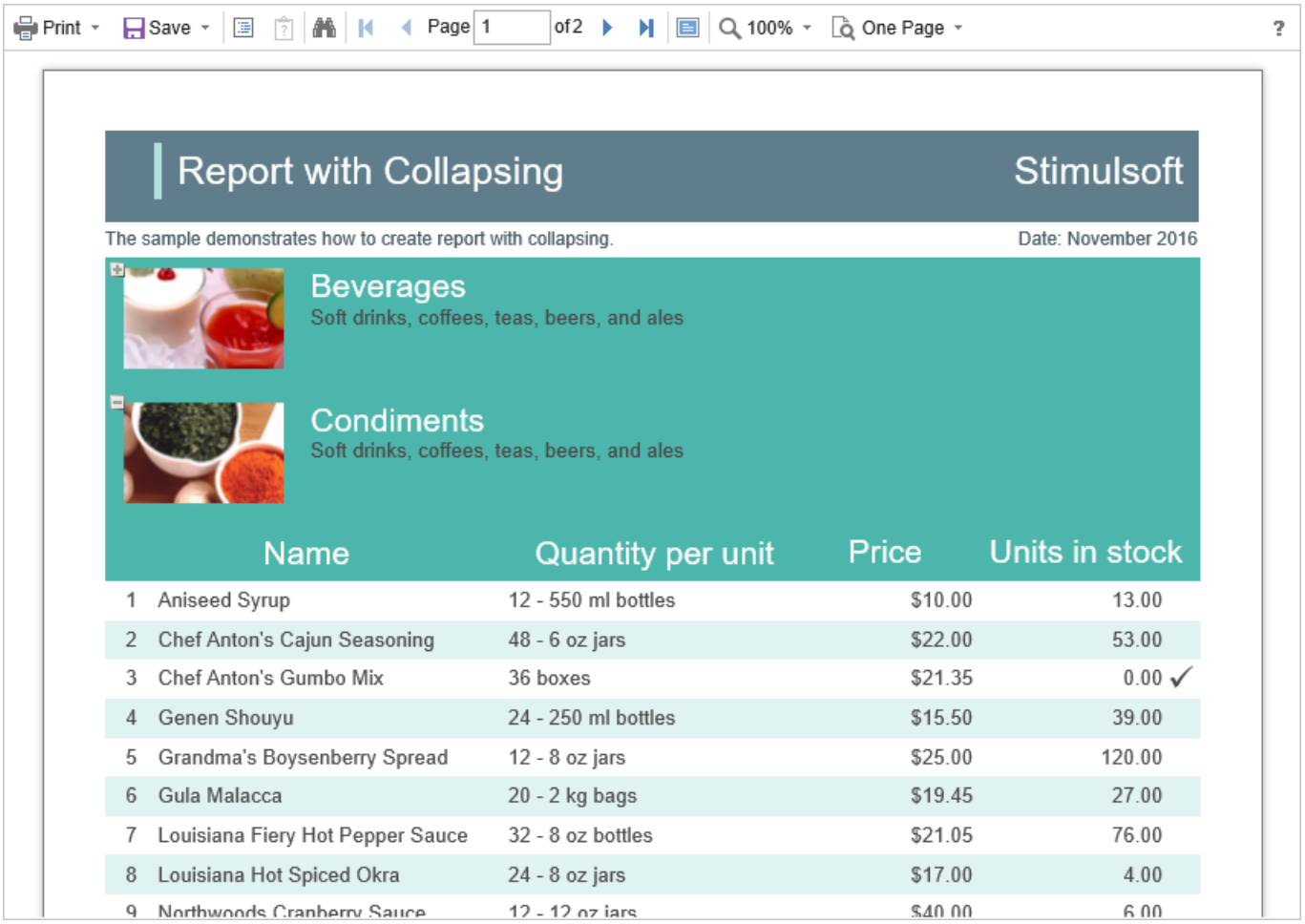
4.1.13 Dynamic Sorting, Collapsing, and Drill-Down

The **HTML5 Viewer** component supports dynamic sorting, collapsing, and drill-down of reports. Dynamic sorting provides the ability to change the direction of sorting in a rendered report. To do this, click on the component that has the dynamic sorting enabled. Dynamic sorting is carried out in the following directions - **Ascending** and **Descending**. Each time the component is clicked, the sorting direction is reversed.

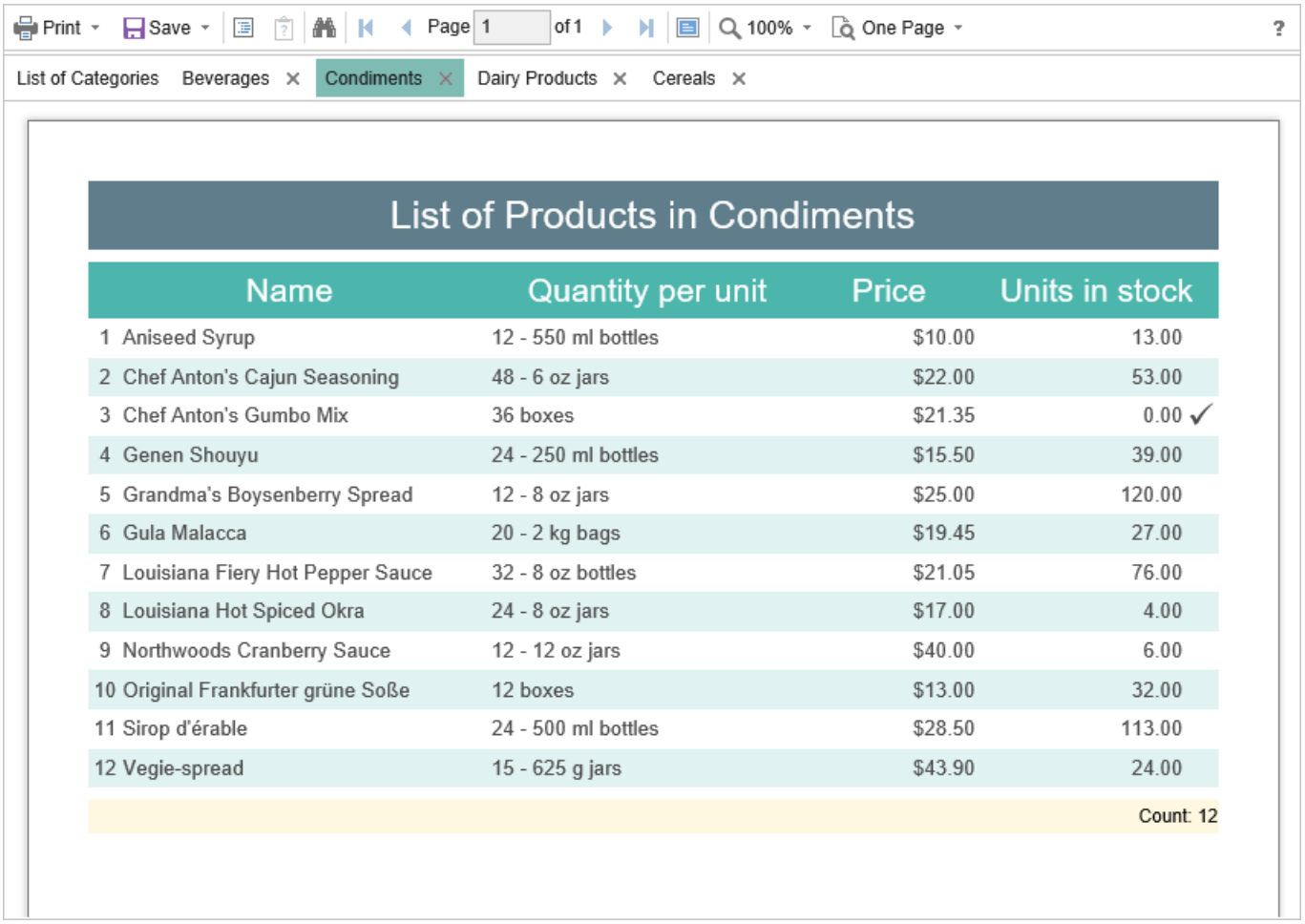
Multi-level sorting is allowed in the report. To do this, hold down the **Ctrl** key and sequentially click on the sorted components in the report. To reset sorting, you can click on any sorted component without holding down the **Ctrl** key.



A report with dynamic collapsing is an interactive report in which blocks can collapse/expand their content when you click on the block title. Report elements, which can be collapsed/expanded, are indicated by special icons - **[-]** or **[+]**.



When using drill-down, under the main panel of the viewer, the drill-down panel with tabs for drill-down reports will be displayed. The currently displayed report will be highlighted.



To work with dynamic sorting, collapsing, and drill-down reports, no additional viewer settings are required. A special interaction action is used to perform any actions before sorting, collapsing, or drill-down. It will be called when interactive action of the viewer.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Actions =
      {
         Interaction = "ViewerInteraction"
      }
})
...
```

HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   // Some code before any interaction
   // ...

   return StiMvcViewer.InteractionResult();
}
...
```

To get the type of action, you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class. They are passed to any server-side by any request and contain all necessary information and states of the client part of the viewer. For each type of interactivity, the viewer has a certain type of action:

- **Sorting** – when using column sorting;
- **DrillDown** – when using drill-down in reports;
- **Collapsing** – when using collapsing report blocks.

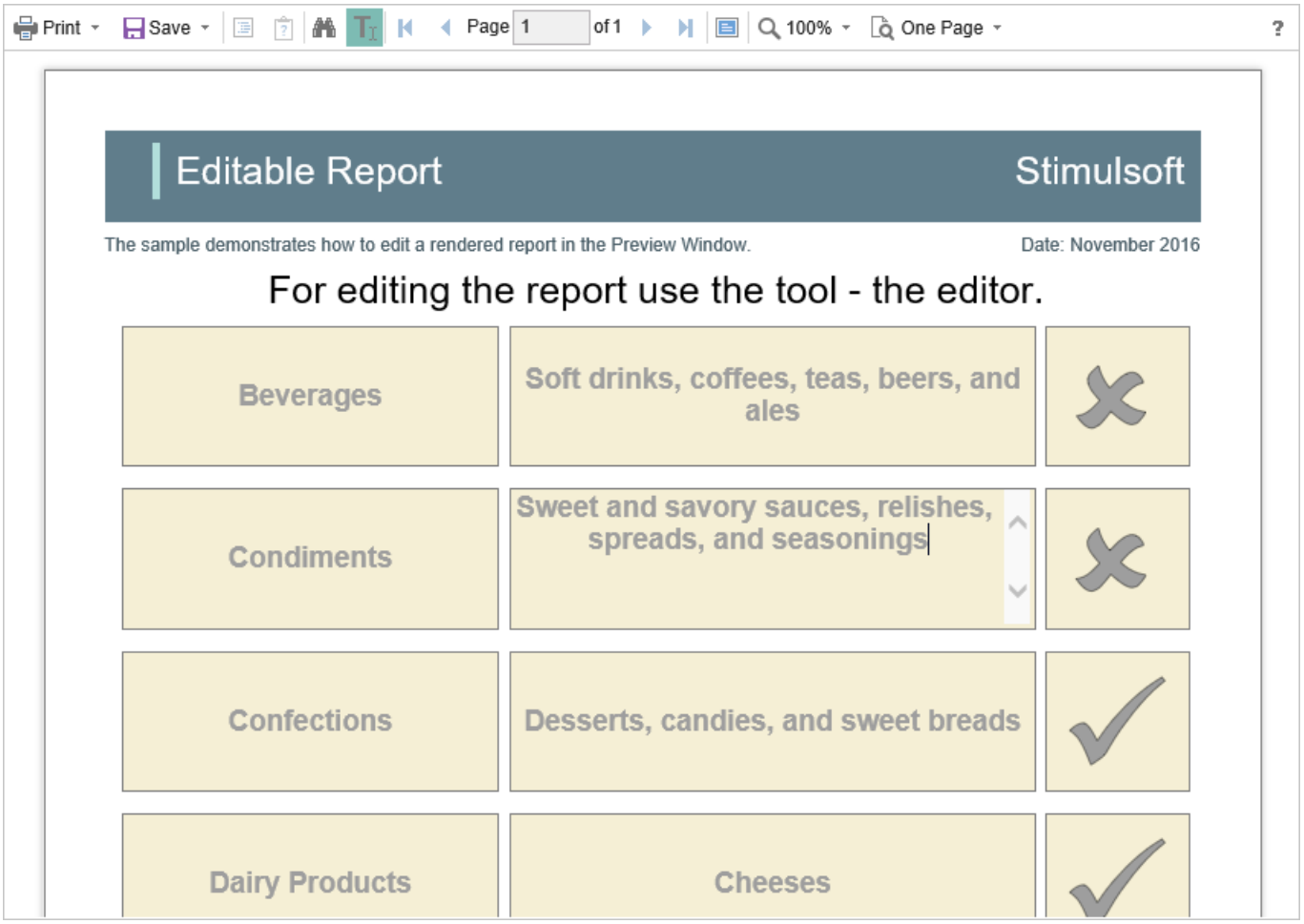
HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   StiRequestParams requestParams = StiMvcViewer.GetRequestParams();
   switch (requestParams.Action)
   {
      case StiAction.Sorting:
         break;

      case StiAction.DrillDown:
         break;

      case StiAction.Collapsing:
         break;
   }

   return StiMvcViewer.InteractionResult();
}
...
```

4.1.14 Editing Report

The **HTML5 Viewer** component has the ability to edit report items, such as text boxes and checkboxes. You should mark the required components as editable in the report template for the editing to be possible. After displaying a report in the viewer, you need to click the corresponding button on the viewer panel to start editing. After editing, it is necessary to click the button once more, and all changes will be

applied to the report.



For the report edit mode, no special settings of the viewer required.

Information

The edited settings will be applied when you print or export a report, and the original report remains unchanged. After restarting the viewer, all the values will be returned to the initial ones.

4.1.15 Sending Report by Email

Information

Please note that the Send Report by Email option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component provides the ability to send reports by email. To activate this feature, you should set the **ShowSendEmailButton** property of the viewer to **true** and define the **EmailReport** action.

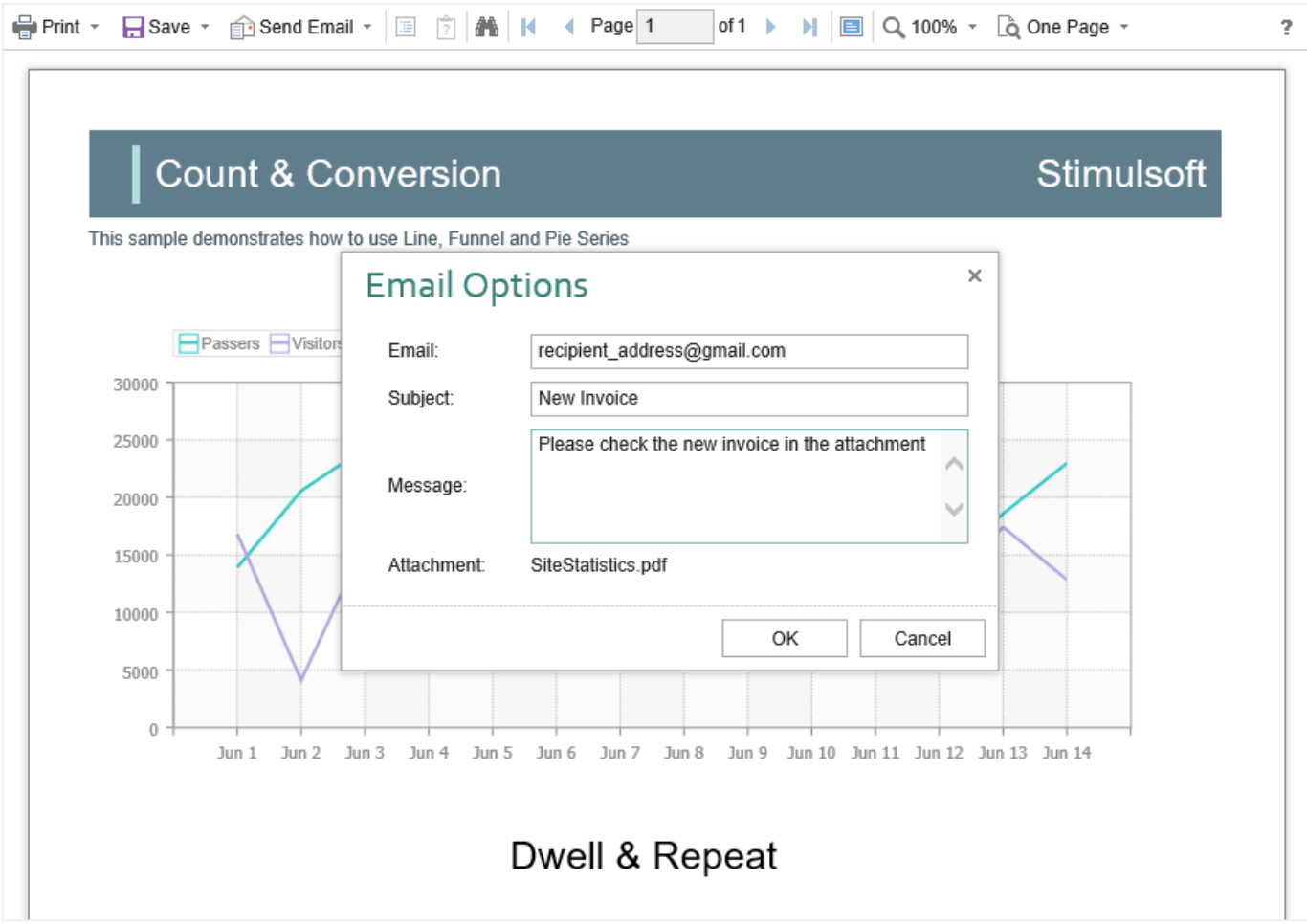
Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Actions =
      {
         EmailReport = "EmailReport"
      },
      Toolbar =
      {
         ShowSendEmailButton = true
      }
})
...
```

HomeController.cs

```
...
public ActionResult EmailReport()
{
   StiEmailOptions options = StiMvcViewer.GetEmailOptions();

   // Passed from the viewer, can be checked and changed
   // options.AddressTo = "";
   // options.Subject = "";
   // options.Body = "";

   // Should be filled here
   options.AddressFrom = "admin_address@test.com";
   options.Host = "smtp.test.com";
   options.Port = 465;
   options.UserName = "admin_address@test.com";
   options.Password = "admin_password";

   // options.CC.Add("email@test.com");
   // options.BCC.Add("email@test.com");
   // options.EnableSsl = true;

   return StiMvcViewer.EmailReportResult(options);
}
...
```

When sending a report by email, the menu to select the attachment format is displayed. It corresponds to the menu for selecting the format for exporting the report. After selecting the format, the dialog to send email parameters, such as the recipient's email, subject, and text of the message, is displayed.



After confirmation of sending the email, the above described **EmailReport** event will be called. You can check and correct the data entered in this form. The exported report file will be attached to the email automatically.

The **HTML5 Viewer** component allows you to set default values for the send email form. The **DefaultEmailAddress**, **DefaultEmailSubject**, and **DefaultEmailMessage** properties can be used for this. By default, these properties are empty.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Email =
      {
         DefaultEmailAddress = "recipient_address@gmail.com",
         DefaultEmailSubject = "New Invoice",
         DefaultEmailMessage = "Please check the new invoice in the
         attachment"
      }
})
...
```

4.1.16 Calling Designer from Viewer

The **HTML5 Viewer** component has the ability to call the report designer. The special **Design** button in the toolbar of the viewer (the button is disabled by default) should be used. To use this feature, you should set the **ShowDesignButton** property to true and define the **DesignReport** event handler.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Actions =
      {
         DesignReport = "DesignReport"
      },
      Toolbar =
      {
         ShowDesignButton = true
      }
})
...
```

HomeController.cs

```
...
public ActionResult DesignReport()
{
   StiReport report = StiMvcViewer.GetReportObject();
   ViewBag.ReportName = report.ReportName;

   return View("Designer");
}
...
```

Information

The viewer does not run the designer. It only calls the specified action, in which you can get all the necessary parameters. Then, in action, you can implement a redirection to another View, which contains the report designer.

4.1.17 Caching

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component allows you to use the server cache to store rendered reports. If you do not use caching, you should load the report, connect data, and render it again every time you request a page. If you use caching, the previously rendered report will be loaded from the cache every time you refresh the page.

When using caching, it should be taken into account that every report saved in the cache takes up server memory and, with a large number of requests to reports, this can become a critical issue. Therefore, you need to choose between two options: low memory requirements but high in performance or low-performance requirements but high in memory.

You can manage caching with the following properties.

The CacheMode property

This property of the viewer enables caching and sets its type. It can take one of the following values, specified in the **StiServerCacheMode** enumeration:

- **None** – caching is disabled. Each action of the viewer requires loading the report from the file and, if it is a report template, then render it;
- **ObjectCache** – for caching, the server cache is used. The report object is saved in this cache (set by default);
- **StringCache** – for caching, the server cache is used. The report is saved as a

packed string in this cache;
- **ObjectSession** – the current session, in which the report object is saved, is used for caching;
- **StringSession** – for caching, the current session is used. The report is saved as a packed string in this cache.

The CacheItemPriority property

This property sets the priority of the report stored in the server's cache. It affects the automatic clearing of the server memory in case of a lack of memory. The lower the priority is, the greater is the chance of removing information from memory.

The CacheTimeout property

This property specifies the amount of time in minutes for which you want to save the report in the server cache. If you use caching and the requested report is not found in the cache (the objects storage time has expired), it will be requested again using a special **GetReport** event, then connect the report data and render it.

StiCacheHelper

The **HTML5 Viewer** component provides the ability to define your methods of working with report caching. For this purpose, a special class **StiCacheHelper** is used. It contains methods for obtaining a report from the cache and saving the report to the cache. It is necessary to create a new class inherited from **StiCacheHelper** and reload the above methods, which respectively have the names - **GetReport** and **SaveReport**.

HomeController.cs

```
...
public class ViewerController : Controller
{
   public class StiMyCacheHelper : StiCacheHelper
   {
      public override StiReport GetReport(string guid)
      {
```

```
        string path =
        System.IO.Path.Combine(System.Web.HttpContext.Current.Server.MapPat
        h("~/"), "CacheFiles", guid);
        if (System.IO.File.Exists(path))
        {
          StiReport report = new StiReport();
          string packedReport = System.IO.File.ReadAllText(path);
          if (guid.EndsWith("template"))
          report.LoadPackedReportFromString(packedReport);
          else report.LoadPackedDocumentFromString(packedReport);

          return report;
        }
        return null;

        //return base.GetReport(guid);
      }

      public override void SaveReport(StiReport report, string guid)
      {
        string packedReport = guid.EndsWith("template") ?
        report.SavePackedReportToString() :
        report.SavePackedDocumentToString();
        string path =
        System.IO.Path.Combine(System.Web.HttpContext.Current.Server.MapPat
        h("~/"), "CacheFiles", guid);
        System.IO.File.WriteAllText(path, packedReport);

        //base.SaveReport(report, guid);
      }
    }

    static ViewerController()
    {
      StiMvcViewer.CacheHelper = new StiMyCacheHelper();
    }
}
...
```

To initialize the work with report caching using the created class, it is enough to set it as a value of the static **StiMvcViewer.CacheHelper** property in the controller constructor.

Information

If report caching is disabled (the **CacheMode** property of the viewer is set to **None**), the specified class will not be used.

4.1.18 Additional Methods

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

For **HTML5 Viewer**, several additional methods are used to get the object of the currently viewed report, parameters of the viewer's current state, and other useful data. These methods can be used in any actions of the viewer.

The GetReportObject() method

Returns the report object with which the viewer is currently working. It is possible to perform the necessary actions with it - register new data sets, change report properties, assign parameters or load another report to the object. Then, the report can be returned to the viewer, specifying it as a parameter in the resulting action method.

HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   StiReport report = StiMvcViewer.GetReportObject();
   report.ReportName = "MyReportName";

   return StiMvcViewer.InteractionResult(report);
}
...
```

The GetRouteValues() method

Returns values for URLs with which the viewer page was opened. Thus, it is possible to get the initial collection of run page parameters in any viewer action and use these values for any checks and conditions.

HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   RouteValueDictionary routeValues = StiMvcViewer.GetRouteValues();

   return StiMvcViewer.InteractionResult();
}
...
```

You can also get values of URL parameters by parameter name, specifying it as the parameter of the called action of the viewer.

HomeController.cs

```
...
public ActionResult ViewerInteraction(string id)
{
   return StiMvcViewer.InteractionResult();
}
...
```

The GetFormValues() method

Returns the values of the form that initiated (opened by the POST request) a page of the viewer. Thus, it is possible to get a collection of form parameters in any action of the viewer.

HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   NameValueCollection formValues = StiMvcViewer.GetFormValues();

   return StiMvcViewer.InteractionResult();
}
...
```

By default, this feature is disabled to optimize requests of the client-side of the viewer to the server. To enable it, set the **PassFormValues** property to **true**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
```

```
   new StiMvcViewerOptions() {
      Server =
      {
         PassFormValues = true
      }
})
...
```

The GetRequestParams() method

Returns all parameters of the current state of the viewer passed to the server-side. They can be useful for determining the type of action that the viewer is currently executing - for example, to determine the type of export, and all action parameters.

HomeController.cs

```
...
public ActionResult ExportReport()
{
   StiRequestParams requestParams = StiMvcViewer.GetRequestParams();
   if (requestParams.ExportFormat == StiExportFormat.Pdf)
   {
      StiReport report = StiMvcViewer.GetReportObject();

      // Some action with report for the PDF export
      // ...

      return StiMvcViewer.ExportReportResult(report);
   }

   return StiMvcViewer.ExportReportResult();
}
...
```

You can change the values of some parameters. After making changes, for the correct operation of the viewer, you should transfer the modified parameter object to the input of the resulting method.

HomeController.cs

```
...
public ActionResult ViewerInteraction()
{
   StiRequestParams requestParams = StiMvcViewer.GetRequestParams();
   if (requestParams.Action == StiAction.Variables)
   {
      requestParams.Interaction.Variables["Variable1"] = "MyValue";
      return StiMvcViewer.InteractionResult(requestParams);
   }
```

```
   return StiMvcViewer.InteractionResult();
}
...
```

The GetExportSettings() method

Returns all the parameters of the current report export. The type of the parameter object will correspond to the type of export selected in the viewer menu. Any export parameters can be changed and passed to the input of the resulting method. In this case, the report will be exported with the parameters transferred.

HomeController.cs

```
...
public ActionResult ExportReport()
{
   StiExportSettings settings = StiMvcViewer.GetExportSettings();
   if (settings.GetExportFormat() == StiExportFormat.Pdf)
   {
      StiPdfExportSettings pdfSettings = (StiPdfExportSettings)settings;
      pdfSettings.EmbeddedFonts = true;
      pdfSettings.AllowEditable = StiPdfAllowEditable.No;
      return StiMvcViewer.ExportReportResult(settings);
   }

   return StiMvcViewer.ExportReportResult();
}
...
```

4.1.19 Export and Printing from Code

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** provides the ability to print reports in various ways and export reports to various formats. These actions are performed using the viewer menu. If you want to print or export a report by using the code, for example, in the controller action, you can use the special **StiMvcReportResponse** class. This class contains a set of static methods that allow you to print or export a report from the code, and

the report viewer is not required.

Index.cshtml

```
...
@Html.ActionLink("Print Report from Code", "PrintReport")
<br />
@Html.ActionLink("Export Report from Code", "ExportReport")
...
```

HomeController.cs

```
...
private StiReport LoadSimpleList()
{
   DataSet dataSet = new DataSet();
   dataSet.ReadXml(Server.MapPath("Reports/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(Server.MapPath("Reports/SimpleList.mrt"));
   report.RegData(dataSet);

   return report;
}

public ActionResult PrintReport()
{
   StiReport report = LoadSimpleList();

   return StiMvcReportResponse.PrintAsPdf(report);
   //return StiMvcReportResponse.PrintAsHtml(report);
}

public ActionResult ExportReport()
{
   StiReport report = LoadSimpleList();

   return StiMvcReportResponse.ResponseAsPdf(report);
   //return StiMvcReportResponse.ResponseAsExcel2007(report);
   //return StiMvcReportResponse.ResponseAsText(report);
   //StiMvcReportResponse.ResponseAsJson(report);
}
...
```

The **StiMvcReportResponse** class contains methods for printing in PDF and HTML formats and methods to export the report in any of the supported formats. As arguments, methods can take various export settings, displaying modes and options for saving received files.

4.1.20 Timeout

When working with the **StiMvcViewer** component, you can set the timeout for various operations — storing the report in the cache, server response, and query execution. The timeout setting is done using the component properties and report options.

CacheTimeout Property
Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Server =
      {
         CacheTimeout = 10
      }
})
...
```

Using the cache will increase the speed of the report viewer. See the chapter Caching for more information

RequestTimeout Property
Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Server =
      {
         RequestTimeout = 30
      }
})
...
```

CommandTimeout Option

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to set the query timeout for the already created connection and data sources in the report.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Actions =
      {
         GetReport = "GetReport",
         ViewerEvent = "ViewerEvent"
      }
})
...
```

HomeController.cs

```
...
public ActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Report.mrt"));
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   return StiMvcViewer.GetReportResult(report);
}

public ActionResult ViewerEvent()
{
   return StiMvcViewer.ViewerEventResult();
}
...
```

4.1.21 Viewer Settings

The **HTML5 Viewer** is configured using properties that are located in the **StiMvcViewerOptions** class. All properties are divided into groups. Some of the groups contain subgroups for ease of use. The following is an example of setting the properties of the viewer.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcViewer("MvcViewer1",
   new StiMvcViewerOptions() {
      Theme = StiViewerTheme.Office2022WhiteTeal,
      Localization = "~/Content/Localization/en.xml",
      Actions =
      {
         GetReport = "GetReport",
         ViewerEvent = "ViewerEvent"
      },
      Appearance =
      {
         InterfaceType = StiInterfaceType.Auto,
         ScrollbarsMode = true,
         ShowTooltips = false
      },
      Exports =
      {
         DefaultSettings =
         {
            ExportToPdf =
            {
               CreatorString = "Company Name",
               ImageQuality = 0.75f
            }
         },
         ShowExportToDbf = false,
         ShowExportToDif = false
      }
})
...
```

Please note that all dashboard elements have their own save options and full-screen buttons for preview. There are no special options to control displaying them, but they can be disabled through the properties of the element. The code below should be added after loading the report before passing it to the viewer.

Default.aspx.cs

```
...
var dbsElementInteraction = (report.GetComponentByName("RegionMap1") as
Stimulsoft.Report.Dashboard.IStiElementInteraction).DashboardInteraction;
(dbsElementInteraction as
Stimulsoft.Report.Dashboard.IStiInteractionLayout).ShowFullScreenButton =
false;
(dbsElementInteraction as
Stimulsoft.Report.Dashboard.IStiInteractionLayout).ShowSaveButton = false;
...
```

Main settings (without groups)

Name	Description
Theme	Sets the viewer theme. The list of available themes can be found in the **StiViewerTheme** enumeration. The default value is **Office2022WhiteBlue**.
Localization	Sets the path to the XML localization file. The path can be absolute or relative. By default, the English localization is used. It is built into the viewer and does not require additional XML files.
Width	Sets the width of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. The default width is 100%.
Height	Sets the height of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the automatic height is set depending on the size of the report page, or 650 pixels in the view mode of the viewer with scrollbars.

Actions

Name	Description
GetReport	Specifies the name of the action method for preparing the rendered report. Specifies the name of the action method for preparing the constructed report. If report caching is enabled, this action will be called only once when the report is requested or if the requested report is not found in the server cache.

PrintReport	Specifies the name of the action method of report printing. This is not relevant when viewing dashboards.
ExportReport	Specifies the name of the action method of the export the report to the specified format.
EmailReport	Specifies the name of the action method of sending the report by email. This is not relevant when viewing dashboards.
Interaction	Specifies the name of the action method for the viewer to work with interactive operations, such as using parameters, dynamic sorting, collapsing, and drill-down.
DesignReport	Specifies the name of the action method to go to the specified view by clicking the Design button on the viewer panel.
ViewerEvent	Specifies the name of the action method of basic viewer events and the processing actions of the viewer, such as printing and exporting a report, working with parameters, and interactivity, if these actions are not specified separately. In addition, this action is used to load scripts and styles of the viewer. This action is mandatory.

Server

Name	Description
Controller	Specifies the name of the report controller for the report viewer. If this property is not specified, then the current controller will be used to process requests.
RouteTemplate	Sets the route template that is returned when the report viewer actions are executed. If the property is not set, then the MVC project template will be used instead. If the

	`UseRelativeUrls` property is set to `true`, the `BasePath` will not be respected for this property.The default value of this property is null.
RequestTimeout	Sets the response timeout from the server in seconds, after which an error will be generated. The default value is 20 seconds. For big reports, it is recommended to increase this value.
CacheTimeout	Sets the time in minutes that the server will store the report since the last action of the viewer. The default value is 20 minutes.
CacheMode	Sets the report caching mode. It can take one of the following values of the **StiServerCacheMode** enumeration: › **None** – caching is disabled. The report will be reloaded each time using the **GetReport** event; › **ObjectCache** – the cache is used as the storage, the report is stored as an object (default value); › **ObjectSession** – the session is used as the storage, the report is stored as an object; › **StringCache** – the server cache is used as the storage, the report is serialized to a packed string; › **StringSession** – the session is used as storage, the report is serialized into a packed string.
CacheItemPriority	Sets the priority of the report stored in the server cache. This property affects the automatic clearing of the server memory in case of a lack of memory. The lower the priority is, the greater is the chance of removing information from memory.
AllowAutoUpdateCache	Sets the mode for automatic cache update. The report stored in the cache or the server session

	will be automatically re-saved after a certain period of time when the viewer is idle (every 3 minutes). By default, the property is set to **true**.
UseRelativeUrls	Sets the viewer mode in which relative URLs are used for AJAX requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value that specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-1** removes the port number.
PassQueryParametersForResources	Enables transferring all request URL parameters when generating links to the resources of the viewer. If **false**, only the necessary parameters are used to request the resources of the viewer. This corresponds to the correct work of the browser cache. By default, the property is set to **true**.
PassQueryParametersToReport	Enables using all the URL parameters of the request as the variable values. The variable names must match the parameters. The default value of the property is **false**.
PassFormValues	Enables passing the values of the POST form to the client-side if these values are required to be used in the actions of the viewer. If you enable this property, the additional **GetFormValues()** method will return a collection of form parameters. By default, the property is **false**.
ShowServerErrorPage	Enables displaying an HTML page with the details of the error that occurred on the server-side. When the property is enabled, the details of the error will be displayed in the viewer window. If the property is disabled, only the numeric error code and a short error text in the dialog box will be displayed. By default, the property is set to **true**.
UseCompression	Enables compression of the viewer requests into the GZip stream. That allows to decrease the

	amount of internet traffic but slows down the viewer slightly. The default value of the property is **false**.
UseCacheForResources	Enables caching of the component resources on the server-side. The following resources are supported: scripts, styles, and images. This option improves the load speed of the component and also reduces the server load in multi-client environments. The default value is **true**.
UseLocalizedCache	Sets a value that enables the use of a different cache depending on the selected localization. The default value of the property is **false**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
CustomCss	Sets the path to the CSS file of the viewer's styles. The standard styles of the chosen theme will not be loaded if this property has got a value. The default value of the property is an empty string.
BackgroundColor	Sets the background color of the viewer. By default, it is set to **White**.
PageBorderColor	Sets the border color of the viewer. By default, it is set to **Gray**.
RightToLeft	Sets the **Right to Left** mode for viewer controls. By default, the property is set to **false**.
FullScreenMode	Sets the full-screen display mode of the viewer. By default, the property is set to **false**.

ScrollbarsMode	Sets the preview mode with scrollbars. By default, the property is set to **false**.
OpenLinksWindow	Sets the target window for opening links contained in the report. By default, the property is set to **Blank** (new window).
OpenExportedReportWindow	Sets the target window for opening the export file from the viewer. By default, the property is set to **Blank** (new window).
DesignWindow	Sets the destination window for launching the report designer. The default value of the property is **Self** (which is the current window).
ShowTooltips	Enables showing tips for the viewer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enables showing links to online documentation for the viewer controls. By default, the property is set to **true**.
ShowDialogsHelp	Sets a value that indicates that showing or hiding the help button in dialogs. By default, the property is set to **true**.
PageAlignment	Sets the position of the report page in the viewer window. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – the page will be aligned left; › **Center** – the page will be centered (default value); › **Right** – the page will be aligned right.
ShowPageShadow	Enables displaying shadow for report pages. By default the property is set to **true**.
BookmarksPrint	Enables printing of report bookmarks (besides the report itself). By default, the property is set to **false**.
BookmarksTreeWidth	Sets the width of the bookmarks panel in pixels.

	By default, the width is 180 pixels.
ParametersPanelPosition	Specifies the position of the report parameters panel. It can take one of the following **StiParametersPanelPosition** enumeration values: › **Top** - the panel will be docked to the top margin (default value); › **Left** - the panel will be docked to the left margin.
ParametersPanelMaxHeight	Sets the maximum height of the parameters bar in pixels. By default, the maximum height is 300 pixels.
ParametersPanelColumnsCount	Sets the number of columns to display report parameters. By default, there are two columns.
ParametersPanelSortDataItems	Gets or sets a value that indicates that variable items will be sorted. By default, the property is set to **true**.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
InterfaceType	Sets the type of interface used for the viewer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the viewer's interface is determined automatically depending on the device the report is displayed on. That is the default value. › **Mouse** – the standard interface with a mouse control will be used for all the screen types. › **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability.

	› **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with the smartphones.
AllowMobileMode	Enables or disables displaying a report or dashboard in the mobile mode. If the option is set to **false**, then the mobile view will not be used. If the option is set to **true**, the mobile view mode will be used when opening the viewer on mobile devices. By default, the option is set to **true**.
ChartRenderType	Sets the displaying mode of charts on the report page. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages. It can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN markup.

DatePickerFirstDayOfWeek	Sets the first day of the week for the date picker. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Monday** – the first day of the week is Monday (default value); › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default, the property is set to **false**.
AllowTouchZoom	Sets ability to change the scale of the report page by using the two-fingers gesture (Pinch to Zoom) for the touch-screens. The default value of the property is **true**.
ShowReportIsNotSpecifiedMessage	Sets a value that indicates that 'The report is not specified' message will be shown. The default value of the property is **true**.
PrintToPdfMode	Sets the Print to PDF mode. It has the following values: › **StiPrintToPdfMode.Hidden** - hidden print mode (default value); › **StiPrintToPdfMode.Popup** - the PDF document will be displayed before printing in a pop-up window.
ImagesQuality	Gets or sets the image quality that will be used on the viewer page. It has the following values: › **StiImagesQuality.Low** - low quality, used to speed up loading reports and saves memory; › **StiImagesQuality.Normal** - normal quality, suitable for most cases (default value); › **StiImagesQuality.High** - high quality, used for ultra-high-definition displays, but may slow down the loading of pages.
CombineReportPages	Sets a value that indicates that if a report contains several pages, then they will be combined in preview. By default, the property is set to **false**.

Toolbar

Name	Description
Visible	Enables displaying the viewer toolbar. By default, the property is set to **true**.
DisplayMode	Specifies the display mode of the toolbar of the viewer. It can take one of the following values of the **StiToolbarDisplayMode** enumeration: › **Simple** - all controls are located on the same control panel (default value); › **Separated** - the control panel is split into top and bottom panels.
BackgroundColor	Specifies the background color of the viewer toolbar. The default color of the selected theme is used.
BorderColor	Specifies the border color of the viewer toolbar. The default color of the selected theme is used.
FontColor	Specifies the text color for the toolbar and the viewer menu. The default color of the selected theme is used.
FontFamily	Specifies the font for the toolbar and the viewer menu. The default font of the selected theme is used.
Alignment	Sets the alignment mode for the controls on the viewer toolbar. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – elements will be aligned left; › **Center** – elements will be centered; › **Right** – elements will be aligned right; › **Default** – the alignment depends on the RightToLeft property (default value).

ShowButtonCaptions	Enables text of the buttons on the toolbar of the viewer. By default, the property is set to **true**.
ShowPrintButton	Enables showing the button - **Print** - on the viewer toolbar. By default, the property is set to **true**.
ShowOpenButton	Enables displaying the **Open** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to true.
ShowSendEmailButton	Enables showing the button - **Send Email** - on the viewer toolbar. By default, the property is set to **false**. Also, you should add the EmailReport action.
ShowFindButton	Enables showing the button - **Find** - on the viewer toolbar. By default, the property is set to **true**.
ShowBookmarksButton	Enables showing the button - **Bookmarks** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the bookmarks panel will not be displayed even if there are bookmarks in the report.
ShowParametersButton	Enables showing the button - **Parameters** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the parameters panel will not be displayed even if there are parameters in the report.
ShowResourcesButton	Enables showing the button - **Resources** - on the toolbar of the viewer. By default, the property is set to **true**. If the button is hidden, the resources panel will not be displayed even if there are resources in the report.
ShowEditorButton	Enables showing the button - **Editor** - on the

	viewer toolbar. By default, the property is set to **true**.
ShowFullScreenButton	Enables displaying the **Full Screen** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to **true**.
ShowFirstPageButton	Enables showing the button - **First Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowPreviousPageButton	Enables showing the button - **Previous Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowCurrentPageControl	Enables showing the current report page indicator. By default, the property is set to **true**.
ShowNextPageButton	Enables showing the button - **Next Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowLastPageButton	Enables showing the button - **Last Page** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowZoomButton	Enables showing the button to select the report zoom. By default, the property is set to **true**.
ShowViewModeButton	Enables showing the button to select the view mode of the report page. By default, the property is set to **true**.
ShowDesignButton	Enables displaying the **Design** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to **false**.
ShowAboutButton	Enables showing the button - **About** - on the viewer toolbar. By default, the property is set to **true**.
ShowRefreshButton	Sets a visibility of the **Refresh** button in the toolbar of the viewer. By default, the property is set to **true**.

ShowPinToolbarButton	Enables displaying of the **Pin Toolbar** button on the viewer's toolbar. The button is available only in the Mobile mode of the viewer's interface. The default value of the property is **true**.
PrintDestination	Sets the report printing mode. It can take one of the following values of the **StiPrintDestination** enumeration: › **Default** – a menu with a choice of printing modes will be displayed (default value); › **Pdf** – printing will be done in the PDF format; › **Direct** – printing will be done to the HTML format directly to the printer, the system print dialog will be displayed; › **PopupWindow** – printing will be done in the HTML format via the preview window of the report.
ViewMode	Sets the mode for displaying report pages. It can take one of the following **StiWebViewMode** enumeration values: › **SinglePage** - displays one page of the report selected in the toolbar of the viewer (default value); › **Continuous** - displays all pages of the report; › **MultiplePages** - displays all report pages as a table.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. The values are from 10 to 500 percent. You can also set one of the following values: › **StiZoomMode.PageWidth** – when the viewer runs, the zoom necessary to display the report by the page width will be set; › **StiZoomMode.PageHeight** – when the viewer runs, the zoom necessary to display the

	report by the page height will be set.
MenuAnimation	Enables animation when the viewer menu shows/hides. By default, the property is set to **true**.
ShowMenuMode	Sets the display mode of the viewer menu. It can take one of the following values of the **StiShowMenuMode** enumeration: › **Click** – shows menu by mouse click (default value); › **Hover** – shows menu by hovering the mouse cursor.
AutoHide	Enables auto-hiding of the viewer's toolbar. The property will work only for the Mobile mode of the viewer's interface. The default value of the property is **false**.

Export

Name	Description
DefaultSettings	This group of properties provides the ability to specify the default export settings for each export type. These settings will be applied to the export dialogs when the viewer runs or to the report if export dialogs are disabled.
StoreExportSettings	Enables saving selected settings in the export dialogs. Settings will be stored in browser cookies. By default, the property is set to **true**.
ShowExportDialog	Enables showing the export options dialog box. If the property is set to **false**, the export will be done with the default settings. By default, the property is set to **true**.
ShowExportToDocument	Enables the export menu item - **Document File**. By default, the property is set to **true**.
ShowExportToPdf	Enables displaying the **Adobe PDF file** export

	menu item when viewing reports, and the **Adobe PDF** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToXps	Enables the export menu item - **Microsoft XPS File**. By default, the property is set to **false**.
ShowExportToPowerPoint	Enables the export menu item - **Microsoft PowerPoint 2007/2010 File**. By default, the property is set to **true**.
ShowExportToHtml	Enables the export menu item - **HTML File**. By default, the property is set to **true**.
ShowExportToHtml5	Enables the export menu item - **HTML5 File**. By default, the property is set to **true**.
ShowExportToMht	Enables the export menu item - **MHT Web Archive**. By default, the property is set to **true**.
ShowExportToText	Enables the export menu item - **Text File**. By default, the property is set to **true**.
ShowExportToRtf	Enables the export menu item - **Rich Text File**. By default, the property is set to **true**.
ShowExportToWord2007	Enables the export menu item - **Microsoft Word 2007/2010 File**. By default, the property is set to **true**.
ShowExportToOpenDocumentWriter	Enables the export menu item - **OpenDocument Writer File**. By default, the property is set to **true**.
ShowExportToExcel	Enables the export menu item - **Microsoft Excel File**. By default, the property is set to **true**.
ShowExportToExcelXml	Enables the export menu item - **Microsoft Excel Xml File**. By default, the property is set to **true**.
ShowExportToExcel2007	Enables displaying the **Microsoft Excel 2007/2010 File**. export menu item when viewing reports, and the **Microsoft Excel** item when viewing dashboards. By default, the property is set to **true**.

ShowExportToOpenDocumentCalc	Enables the export menu item - **OpenDocument Calc File**. By default, the property is set to **true**.
ShowExportToCsv	Enables the export menu item - **CSV File**. By default, the property is set to **true**.
ShowExportToDbf	Enables the export menu item - **DBF File**. By default, the property is set to **true**.
ShowExportToXml	Enables the export menu item - **XML File**. By default, the property is set to **true**.
ShowExportToDif	Enables the export menu item - **Data Interchange Format (DIF) File**. By default, the property is set to **true**.
ShowExportToSylk	Enables the export menu item - **Symbolic Link (SYLK) File**. By default, the property is set to **true**.
ShowExportToJson	Enables the export menu item - **JSON File**. By default, the property is set to **true**.
ShowExportToImageBmp	Enables displaying the **BMP Image** export menu item when viewing reports and the **BMP Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageGif	Enables displaying the **GIF Image** export menu item when viewing reports and the **GIF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageJpeg	Enables displaying the **JPEG Image** export menu item when viewing reports and the **JPEG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePcx	Enables displaying the **PCX Image** export menu item when viewing reports and the **PCX Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePng	Enables displaying the **PNG Image** export menu item when viewing reports and the **PNG Image**

	item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageTiff	Enables displaying the **TIFF Image** export menu item when viewing reports and the **TIFF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageMetafile	Enables displaying the **Windows Metafile** export menu item when viewing reports and the **Windows Metafile** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvg	Enables displaying the **Scalable Vector Graphics (SVG) File** export menu item when viewing reports and the **Scalable Vector Graphics (SVG) File** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvgz	Enables displaying the **Compressed SVG (SVGZ) File** export menu item when viewing reports and the **Compressed SVG (SVGZ) File** item when viewing dashboards. By default, the property is set to **true**.
ShowOpenAfterExport	Enables displaying the **Open After Export** parameter in export settings menu. By default the property is set to **true**.

Email

Name	Description
ShowEmailDialog	Enables displaying settings for sending the report via email. If the dialog box is disabled, the email will be sent with the settings set on the server-side in the **EmailReport** action. By default, the property is set to **true**.
ShowExportDialog	Enables displaying export options dialog box

	when sending an email. If the property is set to **false**, the export will be done with the default settings. By default, the property is set to **true**.
DefaultEmailAddress	Sets the default recipient email, i.e., the address to which the email with the attached report will be sent.
DefaultEmailSubject	Sets the default email subject (header).
DefaultEmailMessage	Sets the default email message (text).

4.2 HTML5 Designer

YouTube

Watch videos for working with ASP.NET MVC HTML5 Designer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

Samples

See on GitHub examples of working with the ASP.NET MVC HTML5 Designer component. All examples are separate projects grouped into one solution for Visual Studio.

The **HTML5 Designer** (**StiMvcDesigner**) component is designed to create reports in the web browser. You do not need to install the .NET Framework, ActiveX components, or any special plug-ins on the client-side. All that is needed is any modern Web browser.

With the help of **HTML5 Designer**, you can create, edit, save and preview reports on any computer with any operating system installed. Since the designer only uses HTML and JavaScript technologies, it can be run on devices without Flash or Silverlight support - tablets, smartphones. Also, the designer supports the Touch interface, which is automatically enabled when using devices with a touch screen.

The **HTML5 Designer** component uses the AJAX technology to perform all actions

on reports, which allows you to get rid of reloading the entire page, save Web traffic, and speed up work.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use **HTML5 Designer** in a Web-project, the installation of the Stimulsoft.Reports.Web NuGet package is required:
- Select 'Manage NuGet Packages...' menu item in the project's pop-up menu;
- In the 'Browse' tab, type 'Stimulsoft.Reports.Web' in the search textbox;
- Click the Stimulsoft.Report.Web package, select the version of the package, and click **Install**. If the package should be updated, use the **Update** button.

If for some reason, that is not possible, the following assemblies should be added to the project:
- Stimulsoft.Base.dll
- Stimulsoft.Report.dll
- Stimulsoft.Report.Check.dll
- Stimulsoft.Report.Helper.dll
- Stimulsoft.Report.Mvc.dll
- Stimulsoft.Report.Web.dll
- Stimulsoft.Report.WebDesign.dll

To add the ability to create and edit dashboards in a Web project, install the NuGet package Stimulsoft.Dashboards.Web (this package is associated with the package Stimulsoft.Reports.Web. If it is missed it will be installed automatically):
- Select "Manage NuGet Packages ..." in the context menu of the project;
- Specify Stimulsoft.Dashboards.Web in the search bar on the Browse tab;
- Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

If, for any reason this is not possible, you should additionally add the following assemblies to the project:
- Stimulsoft.Dashboard.dll

- Stimulsoft.Dashboard.Drawing.dll
- Stimulsoft.Dashboard.Export.dll

4.2.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the designer, you need to place the **StiMvcDesigner** component on the page, set the necessary settings to it, and set the necessary actions in the view controller. When the report designer runs, the following actions occur:

- The .NET component generates HTML and JavaScript code that is necessary for displaying and running the designer;
- When the component is output, the JavaScript method is launched. It requests the report template on the server side displays it in the designer window;
- Various actions in the designer (for example, report preview, saving the report template, export reports, sorting, drill-down, etc.) call a certain action on the server-side. You can perform the necessary manipulations with the report.

4.2.2 Activation

YouTube

Watch videos that show how to activate the ASP.NET MVC HTML5 Designer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiMvcDesigner** component.

HomeController.cs

```
...
public class HomeController : Controller
{
   static HomeController()
   {
      //Activation with using license code
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";

      //Activation with using license file
      var path = System.Web.HttpContext.Current.Server.MapPath("~/Content/
      license.key");
      Stimulsoft.Base.StiLicense.LoadFromFile(path);
   }
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

4.2.3 Editing Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

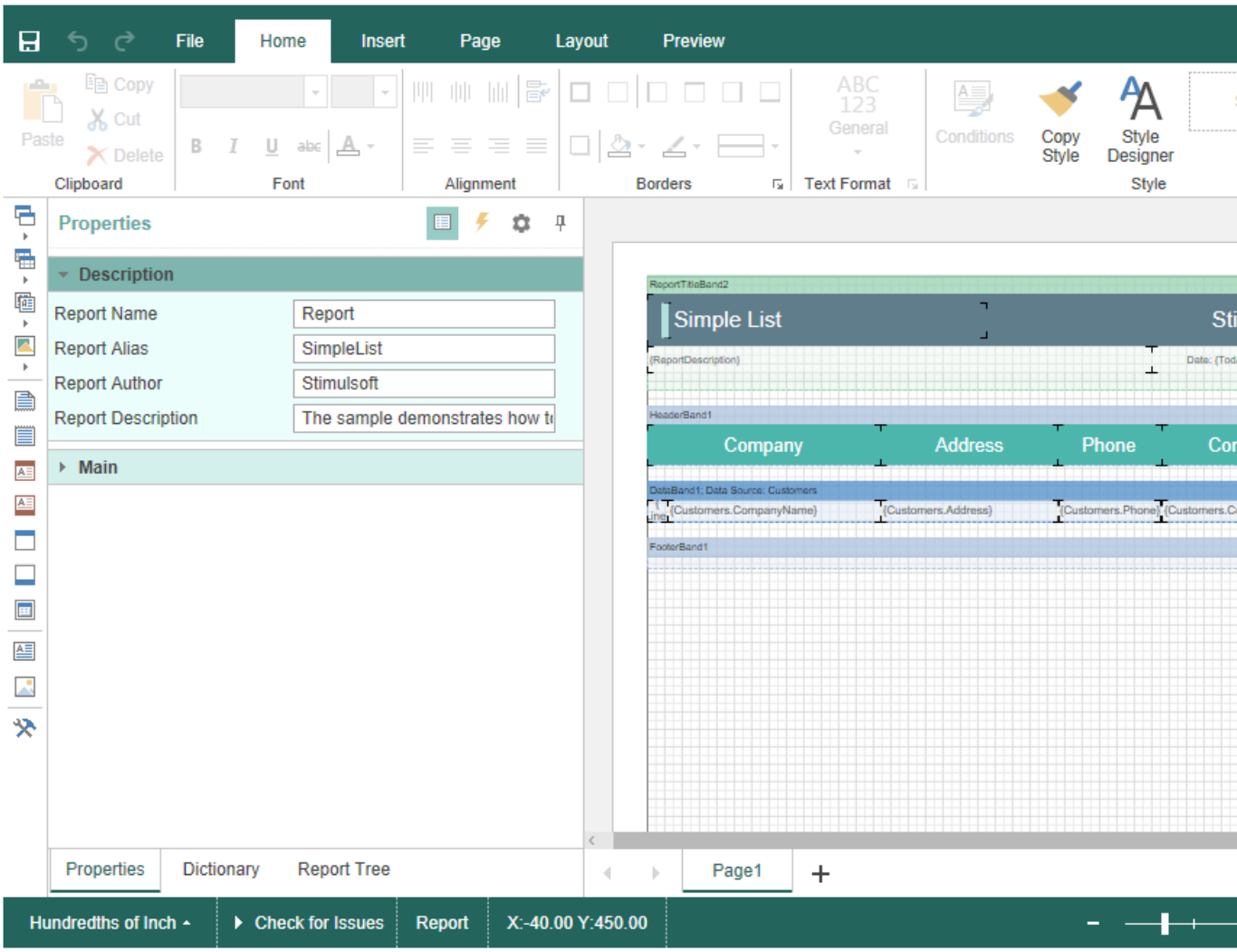
To edit a report template, you need to add the **StiMvcDesigner** component to the page, specify the minimum necessary settings, and define the actions required in the view controller.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Actions =
      {
         GetReport = "GetReport",
         DesignerEvent = "DesignerEvent"
      }
})
...
```

HomeController.cs

```
...
public ActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("~/Content/SimpleList.mrt"));
   //report.Load(Server.MapPath("~/Content/Dashboard.mrt"));

   return StiMvcDesigner.GetReportResult(report);
}

public ActionResult DesignerEvent()
{
   return StiMvcDesigner.DesignerEventResult();
}
...
```



The **GetReport** action is used to load an editable report template. It is called automatically after the report designer is loaded. The **DesignerEvent** action is designed to process various additional designer actions, such as working with data and components, previewing reports, and others.

Information

The **DesignerEvent** action is mandatory. Without it, the correct work of the designer is impossible.

4.2.4 Creating New Reports and New Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the report designer with a new (empty) report, it is enough to create a new report in the **GetReport** action and return it to the designer. If necessary, you can load data for the report or perform any other required actions.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Actions =
      {
         GetReport = "GetReport"
      }
})
...
```

HomeController.cs

```
...
public ActionResult GetReport()
{
   StiReport report = new StiReport();
   //var newDashboard = StiReport.CreateNewDashboard();

   return StiMvcDesigner.GetReportResult(report);
   //return StiMvcDesigner.GetReportResult(newDashboard);
}
...
```

You can also create a new report using the main menu of the designer. The **CreateReport** action is used to load data for a new report or perform any other necessary actions. This action will be called when creating a new empty report or when creating a report using the wizard.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Actions =
      {
         CreateReport = "CreateReport"
      }
})
...
```

HomeController.cs

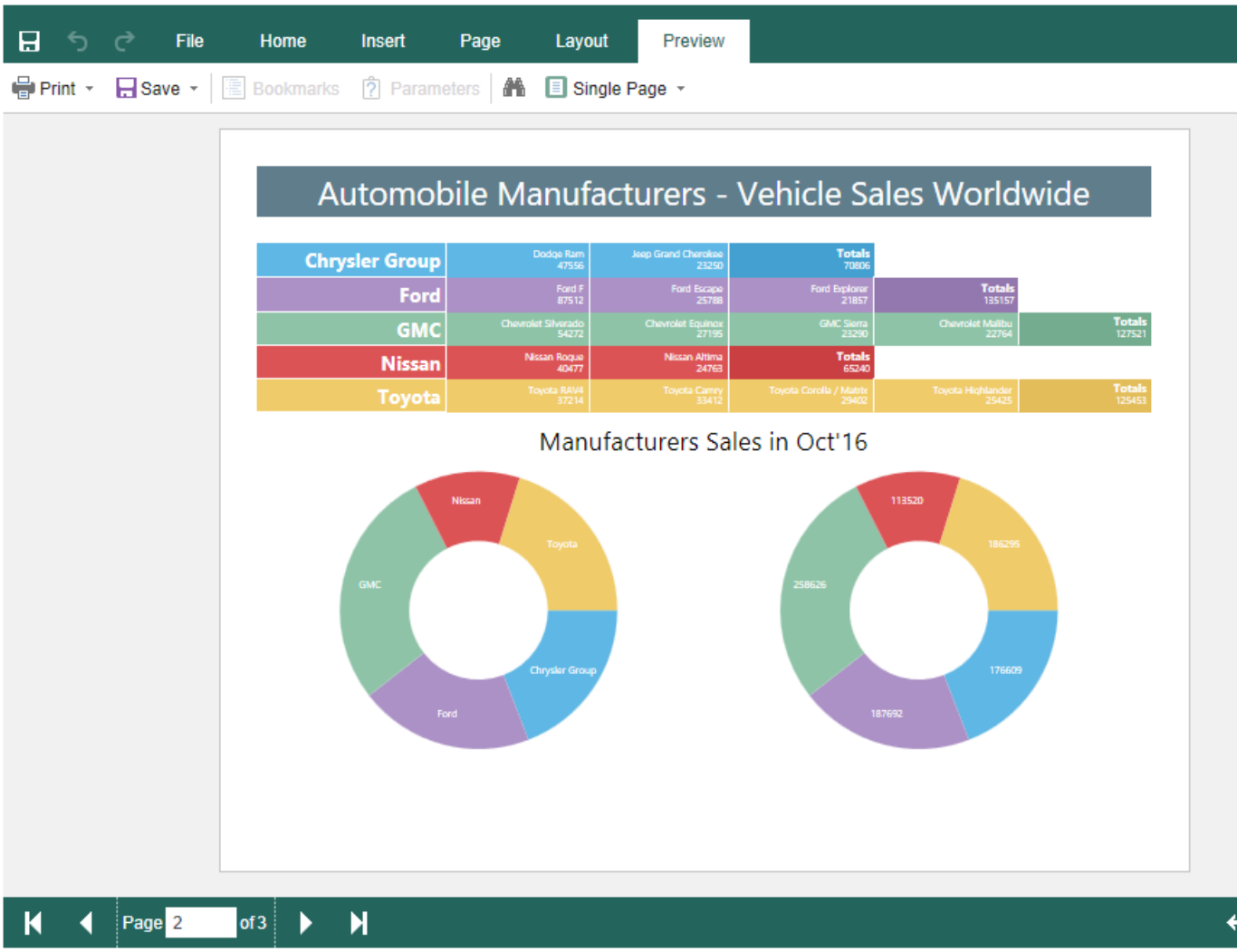
```
...
public ActionResult CreateReport()
{
   StiReport report = new StiReport();
   //var newDashboard = StiReport.CreateNewDashboard();

   // Register data for the new report, if necessary
   DataSet data = new DataSet("Demo");
   data.ReadXml(Server.MapPath("~/Content/Data/Demo.xml"));
   report.RegData(data);
   //newDashboard.RegData(data);
   report.Dictionary.Synchronize();
   //newDashboard.Dictionary.Synchronize();

   return StiMvcDesigner.GetReportResult(report);
   //return StiMvcDesigner.GetReportResult(newDashboard);
}
...
```

4.2.5 Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component provides the ability to preview reports. To preview the report, just go to the appropriate tab in the designer window. The report template will be transferred to the server-side, rendered, and displayed in the embedded viewer.



Before previewing the report, it is possible to perform any necessary actions, for example, connect data for the report. To do this, you can use the particular PreviewReport action that will be called before previewing the report. The **PreviewReport** action is called before preparing and rendering a report for viewing until it saves the cash.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Actions =
      {
         PreviewReport = "PreviewReport"
      }
})
```

```
...
```

HomeController.cs

```
...
public ActionResult PreviewReport()
{
   StiReport report = StiMvcDesigner.GetActionReportObject();
   //var Dashboard = StiMvcDesigner.GetActionReportObject();

   DataSet data = new DataSet("Demo");
   data.ReadXml(Server.MapPath("~/Content/Data/Demo.xml"));
   report.RegData(data);
   //Dashboard.RegData(data);

   return StiMvcDesigner.PreviewReportResult(report);
   //return StiMvcDesigner.PreviewReportResult(Dashboard);
}
...
```

If you need to make actions on your report immediately before displaying the report, you can use the **GetPreviewReport** action, which is called after the request of the prepared report from the cash.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Actions =
      {
         GetPreviewReport = "GetPreviewReport"
      }
})
...
```

HomeController.cs

```
...
public ActionResult GetPreviewReport()
{
   StiReport report = StiMvcDesigner.GetActionReportObject();
   //var Dashboard = StiMvcDesigner.GetActionReportObject();

   DataSet data = new DataSet("Demo");
   data.ReadXml(Server.MapPath("~/Content/Data/Demo.xml"));
   report.RegData(data);
   //Dashboard.RegData(data);
   //report.IsRendered = false;

   return StiMvcDesigner.PreviewReportResult(report);
```

```
   //return StiMvcDesigner.PreviewReportResult(Dashboard);
}
...
```

Information

So as in this event, a prepared report for viewing is transferred. If you need to render again, you should set the **report.IsRendered = false** flag.

4.2.6 Additional Features of Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The preview window of the **HTML5 Designer** component has a fully functional interactive HTML5 Viewer that can print and export reports, supports working with report parameters, dynamic sorting, interactive reports, collapsing, etc. To use these features, you do not need any additional settings for the report designer.

In any of the above actions, you can work with the report template, such as changing its properties and parameters and connecting new data for rendering.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Actions =
      {
         ExportReport = "ExportReport"
      }
})
...
```

HomeController.cs

```
...
public ActionResult ExportReport()
```

```
{
   StiReport report = StiMvcDesigner.GetActionReportObject();
   // ...

   return StiMvcDesigner.ExportReportResult(report);
}
...
```

Information

If you do not need any of these additional options to preview the report (for example, exporting or printing a report), you can disable them using the appropriate properties of the **HTML5 Designer** component.

4.2.7 Saving Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component provides two ways of saving the report which are available in the main menu and in the main panel of the designer - **Save Report** and **Save As**. In turn, each of these ways has its own modes and settings.

Saving a report and dashboard on the server side

To save the editable report on the server-side, you need to set the **SaveReport** action, which will be called when you select Save in the main menu or click the Save button on the main panel of the designer.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Actions =
      {
         SaveReport = "SaveReport"
      }
```

```
})
...
```

HomeController.cs

```
...
public ActionResult SaveReport()
{
   StiReport report = StiMvcDesigner.GetReportObject();

   // Save the report template
   // ...

   return StiMvcDesigner.SaveReportResult();
}
...
```

This action returns a response to the client-side of the designer about the result of saving the report. After saving the report, it is possible to display a dialog box with an error or a text message.

HomeController.cs

```
...
public ActionResult SaveReport()
{
   StiReport report = StiMvcDesigner.GetReportObject();

   // Save the report template
   // ...

   // Completion of the report saving with message dialog box
   return StiMvcDesigner.SaveReportResult("Some message after saving");
}
...
```

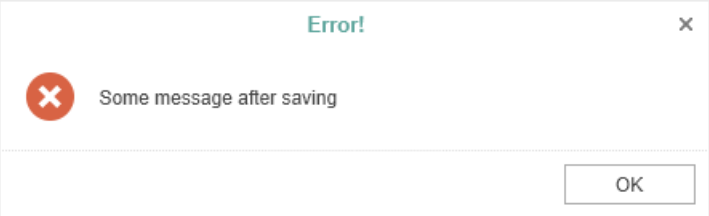
You can get a report name from the designer save dialog or an original report name.

HomeController.cs

```
...
public ActionResult SaveReport()
{
   var requestParams = StiMvcDesigner.GetRequestParams();
   var report = StiMvcDesigner.GetReportObject();

   //Report name from designer save dialog
   var savingReportName = requestParams.Designer.FileName;
```

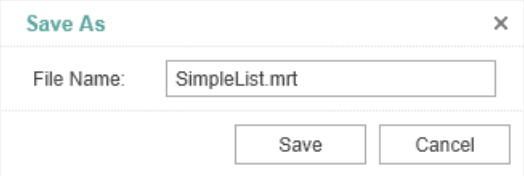
```
    //Original report name from properties
    var originalReportName = report.ReportName;

    return StiMvcDesigner.SaveReportResult();
}
...
```

In this case, the dialog with the specified text will be displayed. The text can contain both an error message of saving or a warning or any other message.



Saving reports and dashboards on the client side

To save the edited report on the client-side as a file, no additional designer settings are required. It is enough to click the **Save As** main menu item. The dialog box will be displayed. In this dialog, you can change the name of the report file. The file will be saved to the local disk of the computer.



The **HTML5 Designer** component provides the ability to change the behavior of the specified save option. For this purpose, the special **SaveReportAs** action is used in the designer. If you use this event, the report will be saved on the server-side. The work of this event will be similar to the **SaveReport** action.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
    new StiMvcDesignerOptions() {
        Actions =
        {
```

```
        SaveReportAs = "SaveReportAs"
      }
})
...
```

HomeController.cs

```
...
public ActionResult SaveReportAs()
{
   StiReport report = StiMvcDesigner.GetReportObject();

   // Save the report template
   // ...

   return StiMvcDesigner.SaveReportResult();
}
...
```

Saving settings

The report is saved in the background mode without reloading the page in the web browser window. Suppose you need to control the process of saving the report visually. In that case, you should change the value of the **SaveReportMode** (or **SaveReportAsMode**) property of the designer to one of the three specified values - **Hidden** (default value), **Visible**, or **NewWindow**.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Actions =
      {
         SaveReportAs = "SaveReportAs"
      },
      Behavior =
      {
         SaveReportAsMode = StiSaveMode.Visible
      }
})
...
```

If the **SaveReportMode** property is set to **Visible**, the report save action will be called in the current browser window in the normal (visible) mode using the POST request. If the **SaveReportMode** property is set to **NewWindow**, the report save event will be called in a new window of the web browser. By default, this property is set to **Hidden** - the report save event is called in the background using the AJAX

request and is not displayed in the browser window. The same values and behavior are applicable to the **SaveReportAsMode** property.

4.2.8 Localization

The **HTML5 Designer** component supports the complete localization of its interface. Use the special **Localization** property to localize the report designer interface. As a value of this property, you should specify the path to the localization XML file (relative or absolute).

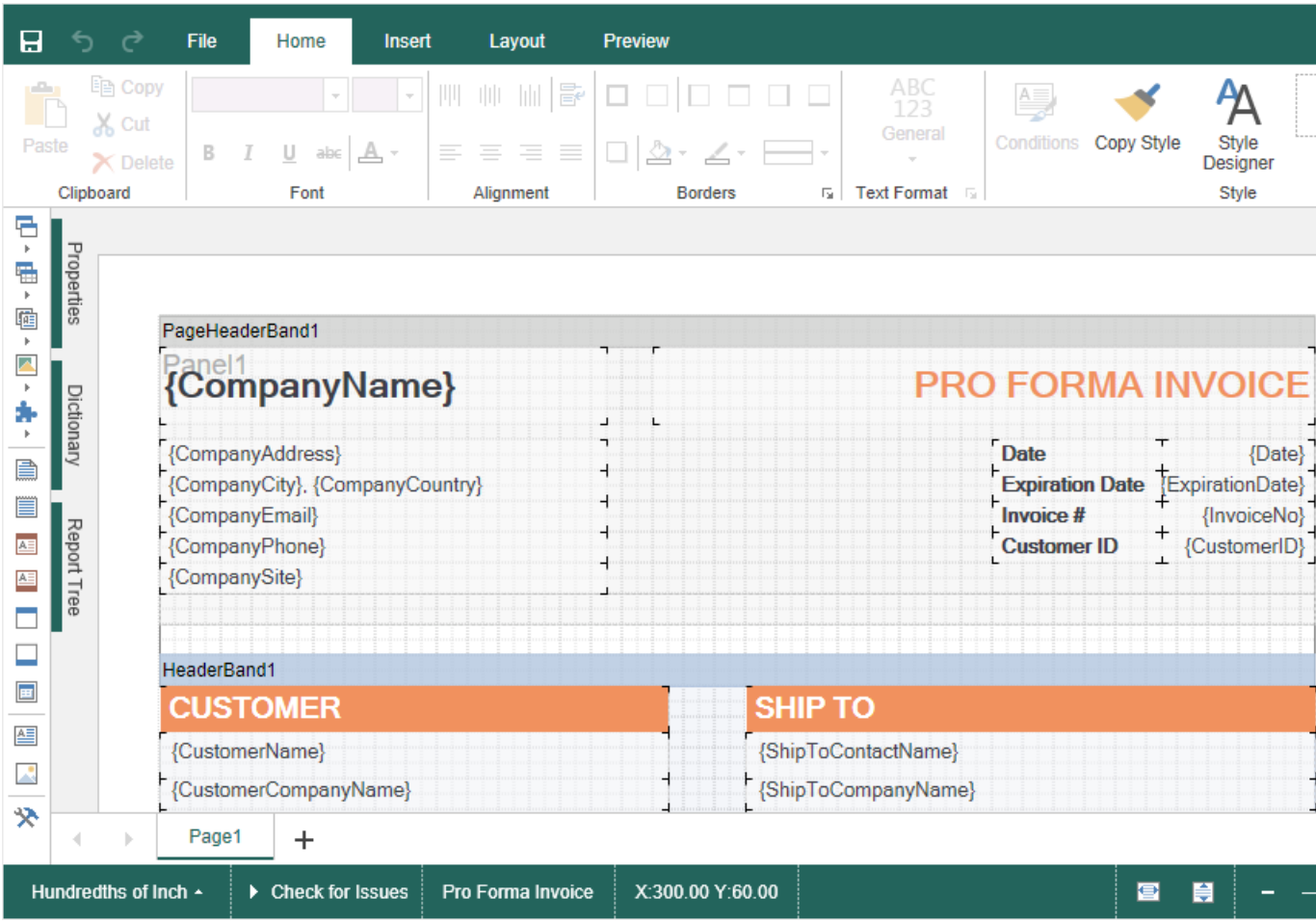
Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Localization = "~/Content/Localization/en.xml"
})
...
```

The interface of the report designer allows you to select the necessary localization from an access list. To do this, set value for the **LocalizationDirectory** property as the folder in which the localization XML files are stored.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Localization = "~/Content/Localization/en.xml",
      LocalizationDirectory = "~/Content/Localization"
})
...
```



Information

If the value for the **Localization** property is set, then when you run the report designer, the localization specified in this property will always be applied. If the property value is not set, the localization selected from the list of available localizations in the report designer panel will be automatically loaded.

4.2.9 Using Themes

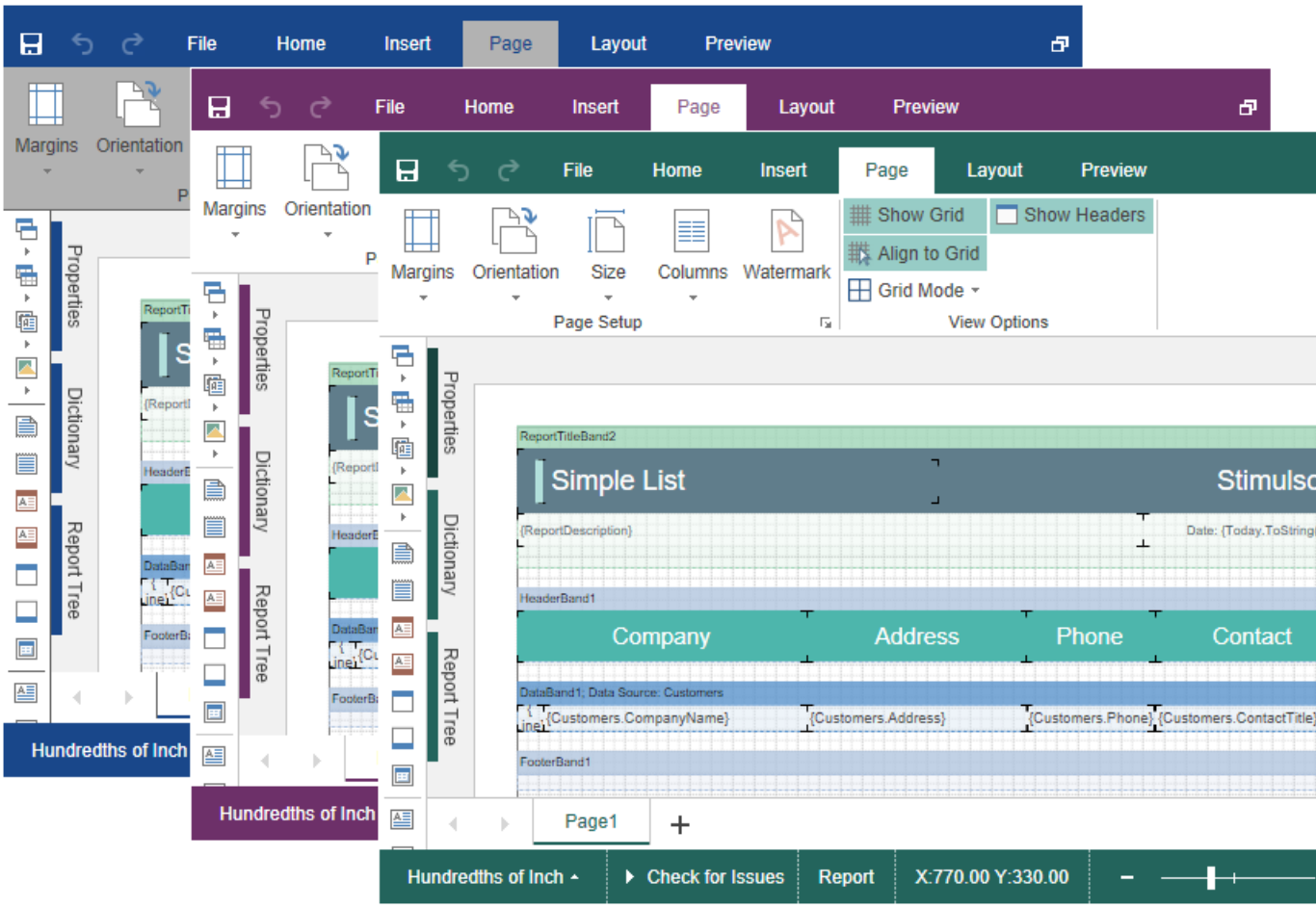
In the **HTML5 Designer** component, you can change the appearance of visual controls. To change the theme, you should use the **Theme** property.

Default.aspx

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
```

```
        Theme = StiDesignerTheme.Office2022WhiteTeal
})
...
```

There are currently **2 themes** available with different color accents. As a result, **more than 50** variants of the appearance are available. This allows you to customize the appearance of the designer for almost any design of the Web project.



4.2.10 Caching

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

HTM5 Designer uses the server cache to store the editable report template. It is necessary because the client part of the designer contains only a visual representation of components of a report template. The report object itself with all the parameters and properties is stored on the server-side.

You can manage caching with the following properties.

The CacheMode property

This property of the designer enables caching and sets its type. It can take one of the following values, specified in the **StiServerCacheMode** enumeration:

- **None** – caching is disabled;
- **ObjectCache** – for caching, the server cache is used. The report object is saved in this (set by default);
- **StringCache** – for caching, the server cache is used. The report is saved as a packed string in this cache;
- **ObjectSession** – the current session, in which the report object is saved, is used for caching;
- **StringSession** - for caching, the current session is used, in which the report is saved as a packed string.

The CacheItemPriority property

This property sets the priority of the report stored in the cache of the server. It affects the automatic clearing of the server memory in case of memory shortage. The lower the priority is, the greater is the chance of removing information from memory.

The CacheTimeout property

This property specifies the amount of time in minutes you want to store the report in the server cache. If, when using caching, the requested report is not found in the cache (time of storing this report expired), then it will be requested again using the special **GetReport** action. In this case, the unsaved changes may be lost.

The **HTML5 Designer** component provides the ability to specify its methods for working with report caching. For this purpose, a special **StiCacheHelper** class is used. It contains methods for obtaining a report from the cache and saving the report to the cache. It is necessary to create a new class inherited from **StiCacheHelper** and reload the above methods, which respectively have the names - **GetObject** and **SaveObject**.

HomeController.cs

```
...
public class DesignerController : Controller
{
   public class StiMyCacheHelper : StiCacheHelper
   {
      public override object GetObject(string guid)
      {
         string path =
         System.IO.Path.Combine(System.Web.HttpContext.Current.Server.MapPat
         h("~/"), "CacheFiles", guid);
         if (System.IO.File.Exists(path))
         {
            byte[] cacheData = System.IO.File.ReadAllBytes(path);
            return StiCacheHelper.GetObjectFromCacheData(cacheData);
         }
         return null;

         //return base.GetObject(guid);
      }

      public override void SaveObject(object obj, string guid)
      {
         byte[] cacheData = StiCacheHelper.GetCacheDataFromObject(obj);
         string path =
         System.IO.Path.Combine(System.Web.HttpContext.Current.Server.MapPat
         h("~/"), "CacheFiles", guid);
         System.IO.File.WriteAllBytes(path, cacheData);

         //base.SaveObject(obj, guid);
      }

      public override void RemoveReport(string guid)
      {
         string path =
         System.IO.Path.Combine(System.Web.HttpContext.Current.Server.MapPat
         h("~/"), "CacheFiles", guid);
         if (File.Exists(path))
         {
            File.Delete(path);
         }
      }
   }

   static DesignerController()
```

```
    {
        StiMvcDesigner.CacheHelper = new StiMyCacheHelper();
    }
}
...
```

To initialize the work with report caching using the created class, it is enough to set it as the value of the **StiMvcDesigner.CacheHelper** static property in the constructor of the controller.

4.2.11 Additional Methods

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

For **HTML5 Designer**, several additional methods are used to get the object of the currently edited report, parameters of the current state of the designer, and other useful data. These methods can be used in any actions of the designer.

The GetReportObject() Method

Returns the report object with which the designer is currently working. It is possible to perform the necessary actions - register new data sets, change report properties, assign parameters or load another report to the object. Then, the report can be returned to the designer, specifying it as a parameter in the resulting action method.

HomeController.cs

```
...
public ActionResult ExportReport()
{
    StiReport report = StiMvcDesigner.GetReportObject();
    report.ReportName = "MyReportName";

    return StiMvcDesigner.ExportReportResult(report);
}
...
```

The GetActionReportObject() method

Returns the report object that will be used for the particular action. For example, for the **OpenReport** action, this method returns a report loaded from the local disk of the computer. For the **PreviewReport** action, the method returns a prepared copy of the report for preview.

HomeController.cs

```
...
public ActionResult OpenReport()
{
   StiReport report = StiMvcDesigner.GetActionReportObject();

   // Register data for the opened report, if necessary
   DataSet data = new DataSet("Demo");
   data.ReadXml(Server.MapPath("~/Content/Data/Demo.xml"));
   report.RegData(data);
   report.Dictionary.Synchronize();

   return StiMvcDesigner.GetReportResult(report);
}
...
```

The GetRouteValues() method

Returns values for URLs with which the designer page was opened. Thus, it is possible to get the initial collection of page parameters to run the designer and use these values for any checks and conditions.

HomeController.cs

```
...
public ActionResult ExportReport()
{
   RouteValueDictionary routeValues = StiMvcDesigner.GetRouteValues();

   return StiMvcDesigner.ExportReportResult();
}
...
```

You can also get values of URL parameters by parameter name, specifying it as the parameter of the called action of the designer.

HomeController.cs

```
...
public ActionResult ExportReport(string id)
{
   return StiMvcDesigner.ExportReportResult();
}
...
```

The GetRequestParams() method

Returns all parameters of the current state of the designer passed to the server-side. They can be useful for determining the type of action that the designer is currently executing - for example, to determine the type of export, and all action parameters.

HomeController.cs

```
...
public ActionResult ExportReport()
{
   StiRequestParams requestParams = StiMvcDesigner.GetRequestParams();
   if (requestParams.ExportFormat == StiExportFormat.Pdf)
   {
      StiReport report = StiMvcDesigner.GetReportObject();

      // Some action with report for the PDF export
      // ...

      return StiMvcDesigner.ExportReportResult(report);
   }

   return StiMvcDesigner.ExportReportResult();
}
...
```

The GetExportSettings() method

Returns all parameters of the current report export. The parameter object type will correspond to the type of export selected in the report preview menu. Any export parameters can be changed and passed to the input of the resulting method. In this case, the report will be exported with the parameters transferred.

HomeController.cs

```
...
public ActionResult ExportReport()
{
```

```
    StiExportSettings settings = StiMvcDesigner.GetExportSettings();
    if (settings.GetExportFormat() == StiExportFormat.Pdf)
    {
        StiPdfExportSettings pdfSettings = (StiPdfExportSettings)settings;
        pdfSettings.EmbeddedFonts = true;
        pdfSettings.AllowEditable = StiPdfAllowEditable.No;
        return StiMvcDesigner.ExportReportResult(settings);
    }

    return StiMvcDesigner.ExportReportResult();
}
...
```

4.2.12 Timeout

When working with the **StiMvcDesigner** component, you can set the timeout for various operations — storing the report in the cache, server response, and query execution. The timeout setting is done using the component properties and report options.

CacheTimeout Property

Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
    new StiMvcDesignerOptions() {
        Server =
        {
            CacheTimeout = 10
        }
})
...
```

Using cache will increase the speed of the report designer. See the chapter Caching for more information

RequestTimeout Property

Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Server =
      {
         RequestTimeout = 30
      }
})
...
```

CommandTimeout Option

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to set the query timeout for the already created connection and data sources in the report.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Actions =
      {
         GetReport = "GetReport",
         DesignerEvent = "DesignerEvent"
      }
})
...
```

HomeController.cs

```
...
public ActionResult GetReport()
{

   StiReport report = new StiReport();
   report.Load(Server.MapPath("Report.mrt"));
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   return StiMvcDesigner.GetReportResult(report);
}

public ActionResult DesignerEvent()
{
   return StiMvcDesigner.DesignerEventResult();
```

```
}
...
```

4.2.13 Add custom functions

Information

See on GitHub example of adding a custom function in the ASP.NET MVC HTML5 Designer component.

You can add a custom function to the Dictionary in the report designer when integrating it into your application. After adding the custom function, you can use this in creating reports and dashboards. Below is the example of adding a function for converting text to uppercase.

DesignerController.cs

```
...
public static string MyFunc(string value)
{
   return value.ToUpper();
}
...
static DesignerController()
{
   var ParamNames = new string[1];
   var ParamTypes = new Type[1];
   var ParamDescriptions = new string[1];

   ParamNames[0] = "value";
   ParamDescriptions[0] = "Descriptions";
   ParamTypes[0] = typeof(string);

   // How to add my function
   StiFunctions.AddFunction(
   "MyCategory",
   "MyFunc",
   "MyFunc",
   "Description",
   typeof(DesignerController),
   typeof(string),
   "Return Description",
   ParamTypes,
   ParamNames,
   ParamDescriptions);
}
...
```

4.2.14 Settings

The **HTML5 Designer** configuration is done using properties that are located in the **StiMvcDesignerOptions** class. All properties are divided into groups. Some of the groups contain subgroups of properties for ease of use. Below is an example of setting some properties of the designer.

Index.cshtml

```
...
@Html.Stimulsoft().StiMvcDesigner("MvcDesigner1",
   new StiMvcDesignerOptions() {
      Theme = Stimulsoft.Report.Web.StiDesignerTheme.Office2022WhiteTeal,
      Localization = "~/Content/Localization/en.xml",
      Actions =
      {
         GetReport = "GetReport",
         PreviewReport = "PreviewReport",
         SaveReport = "SaveReport",
         DesignerEvent = "DesignerEvent"
      },
      Appearance =
      {
         InterfaceType = StiInterfaceType.Auto,
         ShowTooltipsHelp = false,
         ShowDialogsHelp = false,
         DefaultUnit = Stimulsoft.Report.StiReportUnitType.Centimeters
      },
      Dictionary =
      {
         PermissionBusinessObjects =
         Stimulsoft.Report.Web.StiDesignerPermissions.None,
         PermissionDataConnections =
         Stimulsoft.Report.Web.StiDesignerPermissions.View
      },
      Bands =
      {
         ShowChildBand = false,
         ShowEmptyBand = false,
         ShowOverlayBand = false
      }
})
...
```

Basic settings (without groups)

Name	Description
Theme	Specifies the theme of the report designer. The

	list of available themes is located in the **StiDesignerTheme** enumeration. The default value is **Office2022WhiteBlue**.
Localization	Specifies the path to the XML localization file. The path can be absolute or relative. By default, the English localization is used. It is built into the designer and does not require additional XML files.
LocalizationDirectory	Specifies the path to the directory with XML localization files. The localization files located in the specified folder will be loaded to the localization list in the designer panel.
Width	Sets the width of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the component is expanded to the entire area of the browser window.
Height	Sets the height of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the component is expanded to the entire area of the browser window.

Actions

Name	Description
GetReport	Specifies the name of the action method to prepare the report template when loading the designer.
OpenReport	Specifies the name of the action method to open the report template from the designer menu.

CreateReport	Specifies the name of the action method to prepare the report template when creating the new report in the designer.
SaveReport	Specifies the name of the action method to save the report template on the server-side.
SaveReportAs	Specifies the name of the action method to store the report template on the server-side when using the **Save As** menu item. If no action is specified, the built-in method of saving the report template to the local disk will be used.
PreviewReport	Specifies the name of the action method to prepare the rendered report in the preview window.
ExportReport	Specifies the name of the action method to export reports to the specified format.
Exit	Specifies the name of the action method to go to the desired view by clicking the Exit button in the main menu of the report designer.
DesignerEvent	Specifies the name of the action method of the report designer to handle additional designer actions, such as working with data, report components, and others. Also, this action is used to load scripts and designer styles.

Server

Name	Description
Controller	Sets the name of the controller to process requests. If this property is not set, then the current controller will be used to process the requests.
RouteTemplate	Sets the route template that is returned when the report designer actions are executed. If the property is not set, then the MVC project template will be used instead. If the

	`UseRelativeUrls` property is set to `true`, the `BasePath` will not be respected for this property.The default value of this property is null.
ShowServerErrorPage	Enables the display of an HTML page with error details that occurred on the server side. The error details will be displayed in the preview window if the property is enabled. If the property is disabled, the numeric error code and a short error description will be displayed in the dialog window. By default, the property is set to **true**.
AllowAutoUpdateCookies	Allows the designer to update the cookies automatically on every request to the server. By default, cookies are set when creating the designer, if they are not specified in the report. By default, the property is set to **false**.
AllowAntiforgeryToken	Allow the designer to automatically request and send the antiforgery token. By default, the property is set to **true**.
RequestTimeout	Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.
CacheTimeout	Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.
CacheMode	Sets the report caching mode. It can take one of the following values of the **StiServerCacheMode** enumeration: › **None** – caching is disabled in **HTML5 Designer**; › **ObjectCache** – the cache is used as the storage, the report is stored as an object (default value);

	› **ObjectSession** – the session is used as the storage, the report is stored as an object; › **StringCache** – the server cache is used as the storage, the report is serialized to a packed string; › **StringSession** – the session is used as a repository, the report is serialized into a packed string.
CacheItemPriority	Sets the priority of the report stored in the server cache. This property affects the automatic clearing of the server memory in case of lack of memory. The lower the priority is, the greater is the chance of removing information from memory.
AllowAutoUpdateCache	Sets the mode for automatic cache update. The report stored in the cache or server session will be automatically re-saved after a certain period of time if the designer is idle (about every 3 minutes). By default, the property is set to **true**.
UseRelativeUrls	Sets the designer mode in which relative URLs are used for requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value which specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-1** removes the port number.
PassQueryParametersForResources	Enables transferring all request URL parameters when generating links to the resources of the designer. If **false**, only the necessary parameters are used to request the resources of the designer. This corresponds to the more correct operation of the browser cache. By default, the property is set to **true**.
PassFormValues	Enables transferring the POST form values to the client side, if these values are to be used in the actions of the designer. If you enable this feature, the additional **GetFormValues()**

	method will return a collection of form parameters. By default, the property is **false**.
UseCompression	Enables compression of designer requests in the GZip stream. This allows you to reduce the amount of Internet traffic, but slows down the designer. By default, the property is **false**.
UseCacheForResources	Enables caching of the component resources on the server side. The following resources are supported: scripts, styles and images. This option improves the load speed of the component and also reduces the server load in multi-client environments. The default value is **true**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
CustomCss	Specifies the path to the CSS file of styles for the report designer. If this property is set, the standard styles of the selected theme will not be loaded. The default value is an empty value.
DefaultUnit	Sets the units for the size of the report and all its components. By default, centimeters are used.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. It can take one of the following values of the **StiZoomMode** enumeration: › **PageWidth** – when the designer runs, the zoom, necessary to display the report by the page width, will be set;

	› **PageHeight** – when the designer runs, the zoom, required to display the page height of the report, will be set.
ShowAnimation	Enables animation for various elements of the designer interface. By default, the property is set to **true**.
ShowOpenDialog	Allows to display the open dialog, or to open with the open event. By default, the property is set to **true**.
ShowTooltips	Enables displaying tooltips for designer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enable displaying links to online documentation in tooltips for designer controls. By default, the property is set to **true**.
ShowDialogsHelp	Enables displaying links to online documentation on the titles of dialog forms of the designer. By default, the property is set to **true**.
InterfaceType	Sets the type of interface used for the designer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the interface type of the designer will be selected automatically depending on the device used (default value); › **Mouse** – forced use of the interface to control the designer with the mouse; › **Touch** – forced use of the Touch interface to control the designer via the touch screen (mobile devices), also in this mode, the interface elements are increased.
DatePickerFirstDayOfWeek	Sets the first day of the week for the select date item. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Auto** – automatic detection of the first day of

	the week from the browser settings (default value); › **Monday** – the first day of the week is Monday; › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default, the property is set to **false**.
FormatForDateControls	This feature allows you to customize the format for date controls. By default, the current option does not have a specified value, and the date format is determined based on the browser's locale.
ShowReportTree	Enables displaying the tree of report components. By default, the property is set to **true**.
ChartRenderType	Gets or sets the type of the chart in the preview. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages in the preview tab. Can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value);

	› **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN markup.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
CloseDesignerWithoutAsking	Sets a value which indicates that the designer will be closed without asking. By default, the property is set to **true**.
ShowSystemFonts	Sets a visibility of the system fonts in the fonts list. By default, the property is set to **true**.
WizardTypeRunningAfterLoad	Calls the Report wizard after starting the report designer. It may have one of the following **StiWizardType** enumeration values: › **None** - runs the report designer without running the report wizard; › **StandardReport** - runs the Standard wizard; › **MasterDetailReport** - runs the Master-Detail wizard; › **LabelReport** - runs the Label wizard; › **InvoicesReport** - runs the Invoice wizard; › **OrdersReport** - runs the Order wizard; › **QuotationReport** - runs the Quote wizard.
AllowWordWrapTextEditors	Allows word wrap in the text editors. By default, the property is set to **true**.

Behavior

Name	Description
ShowSaveDialog	Enables displaying the dialog to insert a report name when it is saved. The name of the report will be transferred in the parameters of the report designer. By default, the property is set

	to **true**.
UndoMaxLevel	Sets the maximum number to cancel actions with the report (the Undo/Redo function). A big value of this property will consume memory on the server-side to store the undo parameters. The default value is **6**.
AllowChangeWindowTitle	Allows using the title of the browser window to display the file name of the edited report. By default, the property is set to **true**.
SaveReportMode	Sets the mode to save the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - saving of the report is called in the current web browser window in the visible mode using the POST request; › **NewWindow** - saving of the report is called in a new window (tab) of the web browser.
SaveReportAsMode	Sets the mode for saving the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - the saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - the saving of the report is called in the current web browser window in the visible mode using the POST request; › **NewWindow** - the saving of the report is called in a new window (tab) of the web browser.
CheckReportBeforePreview	Sets the value that allows running the report checker before preview.

FileMenu

Name	Description
Visible	Enables displaying the main menu of the report designer. By default, the property is set to **true**.
ShowNew	Enables showing the main menu item - **New**. By default, the property is set to **true**.
ShowFileMenuNewReport	Sets a visibility of the new report button in the designer. By default, the property is set to **true**.
ShowFileMenuNewDashboard	Sets a visibility of the new dashboard button in the designer. By default, the property is set to **true**.
ShowOpen	Enables showing the main menu item - **Open**. By default, the property is set to **true**.
ShowSave	Enables showing the main menu item - **Save**. By default, the property is set to **true**.
ShowSaveAs	Enables showing the main menu item - **Save As**. By default, the property is set to **true**.
ShowClose	Enables showing the main menu item - **Close**. By default, the property is set to **true**.
ShowExit	Enables showing the main menu item - **Exit**. By default, the property is set to **false**.
ShowReportSetup	Enables showing the main menu item - **Report Setup**. By default, the property is set to **true**.
ShowOptions	Enables showing the main menu item - **Options**. By default, the property is set to **true**.
ShowInfo	Enables showing the main menu item - **Info**. By default, the property is set to **true**.
ShowAbout	Enables showing the main menu item - **About**. By default, the property is set to **true**.
ShowHelp	Enables showing the main menu item - **Help**. By default, the property is set to **true**.

Dictionary

Name	Description
Visible	Enables showing the data dictionary of the report. By default, the property is set to **true**.
UseAliases	Allows you to use aliases in the data dictionary. It has the three values of the **StiUseAliases** enumeration: › **Auto** - defines the mode of using aliases from a saved value in cookies (default value); › **True** - sets the mode of using aliases in the data dictionary; › **False** - disables the mode of using aliases in the data dictionary.
NewReportDictionary	It allows you to create a new data dictionary or join the existing one when creating a new report in the designer. It has the three values of the **StiNewReportDictionary** enumeration: › **Auto** - defines the mode to create or join the data dictionary from a saved value in cookies (default value); › **DictionaryNew** - sets the mode to create a new data dictionary when creating a new report; › **DictionaryMerge** - sets the mode to join the existing data dictionary with a new one when creating a new report in the designer.
ShowDictionaryContextMenuProperties	Sets a visibility of the **Properties** item in the dictionary context menu. By default, the property is set to **true**.
ShowDictionaryActions	Sets a visibility of the **Actions** menu on the dictionary toolbar. By default, the property is set to **true**.
PermissionDataConnections	Sets the available actions to connect data to the report. It can take one or more values from the

	StiDesignerPermissions enumeration.
PermissionDataSources	Sets available actions on report data sources. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataColumns	Sets the available actions on data columns in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionBusinessObjects	Sets the available actions on the business objects in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataRelations	Sets the available actions to linking data in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionVariables	Sets available actions on report variables. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionResources	Sets the available actions for the resources in the Report Dictionary. Takes one or several values from the **StiDesignerPermissions** enumeration.
PermissionSqlParameters	Sets the available actions for the parameters of the SQL queries for the Report DataSources. Takes one or several values from **StiDesignerPermissions** enumeration.
DataTransformationsPermissions	Sets the available actions on data transformation. It can take one or more values from the **StiDesignerPermissions** enumeration.

The table below shows all available values for the **StiDesignerPermissions** enumeration, which can be set for the dictionary elements of the report.

Value	Description
None	Disables any action on the item of the data

	dictionary.
All	Allows any action on the item of the data dictionary.
Create	Allows creating a specific data dictionary item.
Delete	Allows deleting a specific data dictionary item.
Modify	Allows modifying a specific data dictionary item.
View	Allows viewing a specific data dictionary item.
ModifyView	Allows modifying and viewing a specific data dictionary item.

Toolbar

Name	Description
ShowToolbar	Enables displaying the toolbar of the designer. By default, the property is set to **true**.
ShowSetupToolboxButton	Enables displaying the button to call the dialog box with settings for the side toolbar. By default, the property is set to **true**.
ShowInsertButton	Enables displaying the **Insert** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowLayoutButton	Enables displaying the tab **Layout** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowPageButton	Enables displaying the tab **Page** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowPreviewButton	Enables displaying the tab **Preview** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the designer. By default, the property is set to **true**.

ShowAboutButton	Enables displaying the **About** on the toolbar of the designer. By default, the property is set to **false**.

PropertiesGrid

Name	Description
Visible	Enables displaying the property panel. By default, the property is set to **true**.
Width	Sets the width of the property panel. By default, the width is set to **370 px**.
LabelWidth	Specifies the width of the labels on the properties panel. By default, the width is set to **160 px**.
PropertiesGridPosition	Sets **Left** or **Right** position of the properties grid in the designer. It has the three values of the **StiPropertiesGridPosition** enumeration: › **Left**; › **Right**.
ShowPropertiesWhichUsedFromStyles	Sets a visibility of the properties which used from styles in the designer. By default, the property is set to **false**.

Components

Name	Description
ShowText	Enables displaying the **Text** component in the insert menu for report components. By default, the property is set to **true**.
ShowTextInCells	Enables displaying the **Text in Cells** component in the insert menu for report components. By default, the property is set to **true**.
ShowRichText	Enables displaying the **Rich Text** component in

	the insert menu for report components. By default, the property is set to **true**.
ShowImage	Enables displaying the **Image** component in the insert menu for report components. By default, the property is set to **true**.
ShowBarCode	Enables displaying the **Bar Code** component in the insert menu for report components. By default, the property is set to **true**.
ShowShape	Enables displaying the **Shape** component in the insert menu for report components. By default, the property is set to **true**.
ShowHorizontalLinePrimitive	Enables displaying the **Horizontal Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowVerticalLinePrimitive	Enables displaying the **Vertical Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowRectanglePrimitive	Enables displaying the **Rectangle** component in the insert menu for report components. By default, the property is set to **true**.
ShowRoundedRectanglePrimitive	Enables displaying the **Rounded Rectangle** component in the insert menu for report components. By default, the property is set to **true**.
ShowPanel	Enables displaying the **Panel** component in the insert menu for report components. By default, the property is set to **true**.
ShowClone	Enables displaying the **Clone** component in the insert menu for report components. By default, the property is set to **true**.
ShowCheckBox	Enables displaying the **Check Box** component in the insert menu for report components. By default, the property is set to **true**.
ShowSubReport	Enables displaying the **Sub Report** component

	in the insert menu for report components. By default, the property is set to **true**.
ShowZipCode	Enables displaying the **Zip Code** component in the insert menu for report components. By default, the property is set to **true**.
ShowTable	Enables displaying the **Table** component in the insert menu for report components. By default, the property is set to **true**.
ShowCrossTab	Enables displaying the **Cross-Tab** component in the insert menu for report components. By default, the property is set to **true**.
ShowChart	Enables displaying the **Chart** component in the insert menu for report components. It affects on all chart types. By default, the property is set to **true**.
ShowMap	Enables displaying the **Map** component in the insert menu for report components. By default, the property is set to **false**.
ShowGauge	Enables displaying the **Gauge** component in the insert menu for report components. By default, the property is set to **false**.
ShowSparkline	Enables displaying the **Sparkline** component in the insert menu for report components. By default, the property is set to **true**.
ShowMathFormula	Enables displaying the **Math Formula** component in the insert menu for report components. By default, the property is set to **true**.
ShowElectronicSignature	Enables displaying the **Electronic Signature** component in the insert menu for report components. By default, the property is set to **true**.
ShowPdfDigitalSignature	Enables displaying the **PDF Digital Signature** component in the insert menu for report components. By default, the property is set to

	true.

Bands

Name	Description
ShowReportTitleBands	Enables displaying the **Report Title** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowReportSummaryBand	Enables displaying the **Report Summary** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageHeaderBand	Enables displaying the **Page Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageFooterBand	Enables displaying the **Page Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowGroupHeaderBand	Enables displaying the **Group Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowGroupFooterBand	Enables displaying the **Group Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowHeaderBand	Enables displaying the **Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowFooterBand	Enables displaying the **Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowColumnHeaderBand	Enables displaying the **Column Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowColumnFooterBand	Enables displaying the **Column Footer** item in the **Bands** menu of the designer. By default, the

	property is set to **true**.
ShowDataBand	Enables displaying the **Data** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowHierarchicalBand	Enables displaying the **Hierarchical** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowChildBand	Enables displaying the **Child** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowEmptyBand	Enables displaying the **Empty** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowOverlayBand	Enables displaying the **Overlay** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowTableOfContents	Enables displaying the **Table of Contents** item in the **Bands** menu of the designer. By default, the property is set to **true**.

DashboardElements

Name	Description
ShowTableElement	Enables displaying the **Table** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowCardsElement	Enables displaying the **Cards** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowChartElement	Enables displaying the **Chart** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowGaugeElement	Enables displaying the **Gauge** element in the Dashboard Elements menu of the designer. By

	default, the property is set to **true**.
ShowPivotTableElement	Enables displaying the **Pivot** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowIndicatorElement	Enables displaying the **Indicator** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowProgressElement	Enables displaying the **Progress** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowRegionMapElement	Enables displaying the **Region Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowOnlineMapElement	Enables displaying the **Online Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowImageElement	Enables displaying the **Image** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTextElement	Enables displaying the **Text** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowPanelElement	Enables displaying the **Panel** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowShapeElement	Enables displaying the **Shape** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowButtonElement	Enables displaying the **Button** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowListBoxElement	Enables displaying the **List Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowComboBoxElement	Enables displaying the **Combo Box** element in

	the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTreeViewElement	Enables displaying the **Tree View** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTreeViewBoxElement	Enables displaying the **Tree View Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowDatePickerElement	Enables displaying the **Date Picker** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.

CrossBands

Name	Description
ShowCrossGroupHeaderBand	Enables displaying the **Cross Group Header** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossGroupFooterBand	Enables displaying the **Cross Group Footer** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossHeaderBand	Enables displaying the **Cross Header** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossFooterBand	Enables displaying the **Cross Footer** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossDataBand	Enables displaying the **Cross Data** section in the insert menu for report components. By default, the property is set to **true**.

Dashboards

Name	Description
ShowNewDashboardButton	Sets visibility of the **New Dashboard** button in the designer. By default, the property is set to **true**.

Pages

Name	Description
ShowNewPageButton	Sets visibility of the **New Page** button in the designer. By default, the property is set to **true**.

When designing a report or dashboard in the report designer, you can also define **ExportOptions**, **EmailOptions**, and **PreviewToolbarOptions** on the **Preview** tab. These options are similar to the report viewer options.

5 Reports and Dashboards for ASP.NET Core MVC

.NET Core is a cross-platform technology for creating Web applications for Windows, Linux, and macOS. **ASP.NET Core MVC** is a multifunctional platform for creating Web applications and APIs using the Model-View-Controller design structure. Stimulsoft company provides tools for creating, displaying, and transforming reports and dashboards using this technology.

Tools for creating and editing reports:	Tools for viewing and converting reports:	Tools for creating and editing dashboards:	Tools for viewing and converting dashboards:
HTML5 designer	HTML5 viewer	HTML5 designer	HTML5 viewer

5.1 HTML5 Viewer

YouTube

Watch videos .NET Core HTML5 Viewer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

Samples

See on GitHub examples for working with the .NET Core HTML5 Viewer component. All examples are separate projects, grouped into one solution for Visual Studio.

The **HTML5 Viewer** (**StiNetCoreViewer**) component is designed for viewing reports in the web browser. You do not need to install the .NET Framework, ActiveX components, or any special plug-ins on the client-side. All that is needed is any modern Web browser.

With the help of **HTML5 Viewer**, you can view, print, and export reports on any computer with any operating system installed. Since the viewer only uses HTML and JavaScript technologies, it can be run on devices with no Flash or Silverlight support - tablets, smartphones. Also, the viewer supports the Touch interface, which is automatically enabled when using devices with a touch screen.

The **HTML5 Viewer** component uses the AJAX technology to perform all actions (uploading a report, paging, scaling, interactivity in reports, etc.), which allows you to get rid of reloading the entire page save Web traffic, and speed up work. The report engine built using the .NET Core technology is used to render reports. This is a cross-platform technology. It allows you to deploy the application on servers that use operating systems like Windows, macOS, and Linux.

The **HTML5 Viewer** supports many themes, animated interface, bookmarks, interactive reports, editing of report elements on the page, full-screen mode, search panel, and other necessary features for viewing reports.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use the **HTML5 Viewer** in a Web project, you need to install the NuGet package of Stimulsoft.Reports.Web.NetCore:
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Reports.Web.NetCore in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

To add the ability to view and export dashboards in a Web project, install the NuGet package Stimulsoft.Dashboards.Web.NetCore (this package is associated with the package Stimulsoft.Reports.Web.NetCore. If it is missed, it will be installed automatically):
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Dashboards.Web.NetCore in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

- Work with Parameters
- Work with Bookmarks
- Viewer Settings
- Additional Methods
- Export and Printing from Code

5.1.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the viewer, you need to place the **StiNetCoreViewer** component on the page, set the necessary settings to it, and set the necessary actions in the view controller. When the report viewer runs, the following actions occur:

› The .NET Core component generates HTML and JavaScript code that is necessary for displaying and running the viewer;

› When the component is output, the JavaScript method is launched. It requests the first page of the report on the server-side or the entire report (depending on the selected mode) and the required report parameters;

› Each action in the viewer (for example, paging, printing or exporting a report, etc.) calls a certain action on the server-side, in which you can perform the necessary manipulations with the report;

› The viewer saves the report in cache or server session to speed up the work, which makes it impossible to re-build the report.

5.1.2 Activation

YouTube

Watch videos that show how to activate the ASP.NET Core HTML5 Viewer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiNetCoreViewer** component.

HomeController.cs

```
...
//Activation with using license code
public class HomeController : Controller
{
   static HomeController()
   {
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";
   }
}

//Activation with using license file
public class HomeController : Controller
{
   public HomeController(IHostingEnvironment hostEnvironment)
   {
      var path = Path.Combine(hostEnvironment.ContentRootPath, "Content\
      \license.key");
      Stimulsoft.Base.StiLicense.LoadFromFile(path);
   }
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

5.1.3 Showing Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `Report.Render()` method if you want to perform

specific actions with the rendered report before it is displayed in the viewer. Likewise, when using the compilation mode, you need to call the `Report.Compile()` method only if you have actions to perform with the compiled report before it is built and shown in the viewer.

To show the report, you need to add the **StiNetCoreViewer** component to the page and set it to the minimum required properties, and, in the view controller, specify the necessary actions.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      GetReport = "GetReport",
      ViewerEvent = "ViewerEvent"
   }
})
...
```

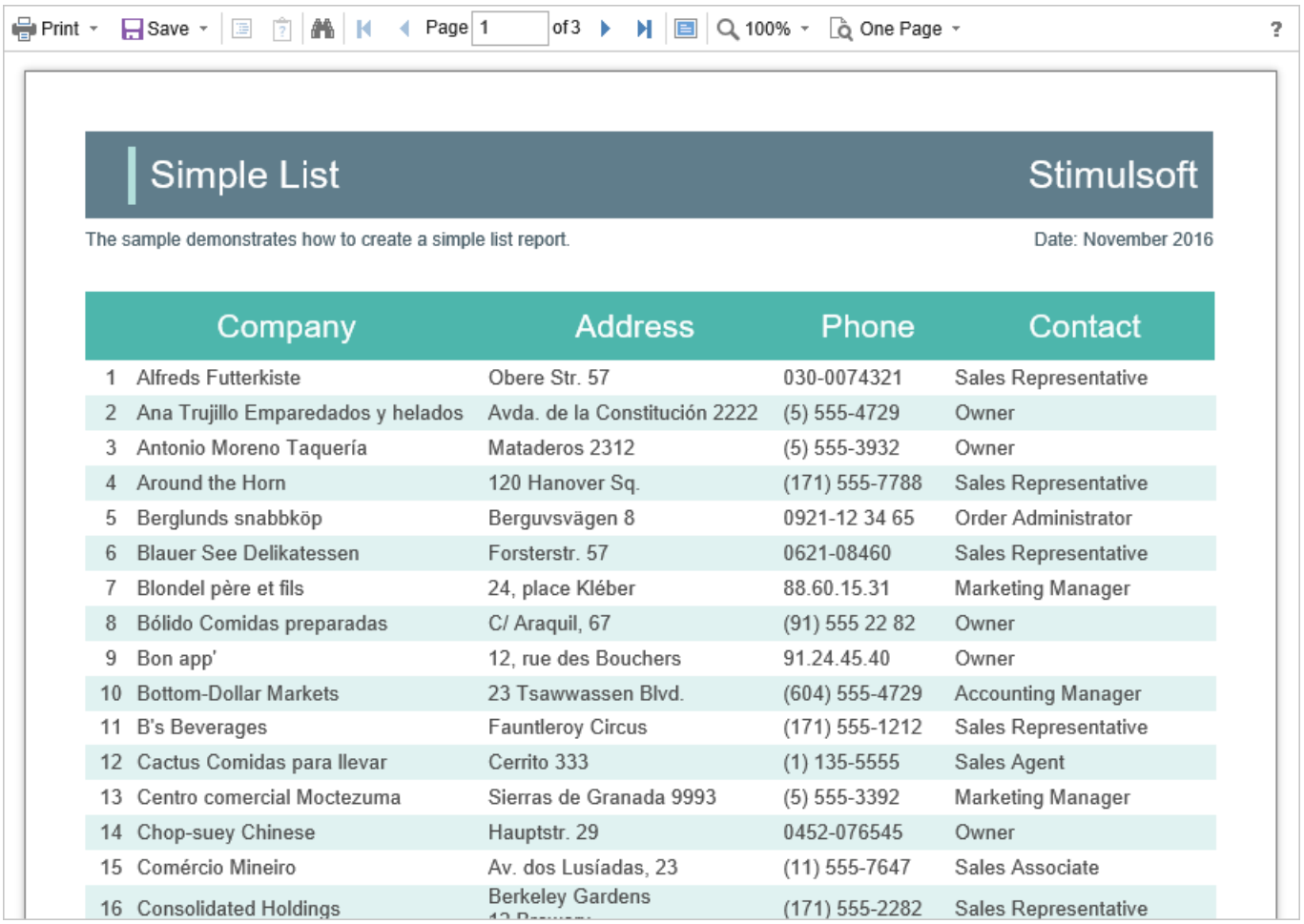
HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));
   //report.Load(StiNetCoreHelper.MapPath(this, "Reports/Dashboard.mrt"));

   return StiNetCoreViewer.GetReportResult(this, report);
}

public IActionResult ViewerEvent()
{
   return StiNetCoreViewer.ViewerEventResult(this);
}
...
```

In the above example, the processing of two actions of the viewer is added. The **GetReport** action returns the report prepared for preview. The **ViewerEvent** action handles the viewer events.

Information

The **ViewerEvent** action is mandatory. Without it, the correct operation of the viewer is not possible.



If the report was not rendered before showing, the **HTML5 Viewer** component will automatically render it. So you can use report templates and rendered reports to display reports.

HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.LoadDocument(StiNetCoreHelper.MapPath(this, "Reports/
   SimpleList.mdc"));
```

```
    return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Since the dashboard is not a static document and requires data to work, the format of the rendered MDC document is not available for it. Instead, it is possible to use a snapshot of the report in the MRT format, which contains all the data necessary for the dashboard to work and be correctly displayed in the viewer.

HomeController.cs

```
...
public ActionResult GetReport()
{
    StiReport report = new StiReport();
    report.Load(StiNetCoreHelper.MapPath("~/Reports/Snapshot.mrt"));

    return StiNetCoreViewer.GetReportResult(report);
}
...
```

Loading custom fonts

You can load custom fonts using the **StiFontCollection** class, having specified the file that contains a font. To do this, you should call the static method in the controller constructor to load a font.

ViewerController.cs

```
...
public class ViewerController : Controller
{
    static ViewerController()
    {
        Stimulsoft.Base.StiFontCollection.AddFontFile(StiNetCoreHelper.MapPat
        h(this, "/fonts/my-font/font-name.ttf"));
    }
}
...
```

5.1.4 Connecting Data

Information

Since dashboards and reports use the same unified template format - MRT,

methods for loading the template and working with data, the word “report” will be used in the documentation text.

Data to a report can be connected in various ways. The easiest way is to store connection settings in the report template. You can also connect the data from the code. This can be done when the report is loaded in the **GetReport** action.

HomeController.cs

```
...
public IActionResult GetReport()
{
   DataSet ds = new DataSet();
   ds.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/
   TwoSimpleLists.mrt"));
   report.Dictionary.Databases.Clear();
   report.RegData("Demo", ds);

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Data for the report can be connected not only when the report is loaded. For example, you can connect new data at the moment of interactive actions in the viewer (applying report parameters, sorting, drill-down, collapsing). To do this, you should set the **Interaction** action for the **HTML5 Viewer** component and, in the action handler, connect the data for the current report. In the same way, you can connect data in other actions of the viewer.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      GetReport = "GetReport",
      ViewerEvent = "ViewerEvent",
      Interaction = "ViewerInteraction"
   }
})
...
```

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   DataSet data = new DataSet();
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));

   StiReport report = StiNetCoreViewer.GetReportObject(this);
   report.RegData("Demo", data);

   return StiNetCoreViewer.InteractionResult(this, report);
}
...
```

If you want to connect new data only for a certain interactive action of the viewer, for example, only when you apply report parameters you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class. They are passed to any server-side on any request and contain all necessary information and states of the client part of the viewer. To determine the type of action of the viewer, it is enough to check the Action property of the viewer parameters.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   if (requestParams.Action == StiAction.Variables)
   {
      DataSet data = new DataSet();
      data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));

      StiReport report = StiNetCoreViewer.GetReportObject(this);
      report.RegData("Demo", data);

      return StiNetCoreViewer.InteractionResult(this, report);
   }

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

SQL data sources

The connection parameters to the SQL data source and any other ones can be

stored in the report template. Suppose you want to set the connection parameters from the code before rendering the report (for example, for security reasons or depending on the authorized user). In that case, you can use the example below.

HomeController.cs

```
...
public IActionResult GetReport()
{
   OracleConnection connection = new OracleConnection("Data
   Source=Oracle8i;Integrated Security=yes");
   connection.Open();
   OracleDataAdapter adapter = new OracleDataAdapter();
   adapter.SelectCommand = new OracleCommand("SELECT * FROM Products",
   connection);

   DataSet dataSet = new DataSet("productsDataSet");
   adapter.Fill(dataSet, "Products");

   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/
   SqlSampleReport.mrt"));
   report.RegData("Products", dataSet);

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Also, for SQL data sources used in the report, you can specify the Query Timeout in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to change the connection string for MS SQL, adjust the query, set the query timeout for the already created connection and data sources in the report.

HomeController.cs

```
...
public IActionResult GetReport()
{

   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath("Report.mrt"));
   ((StiSqlDatabase)
   report.Dictionary.Databases["Connection"]).ConnectionString = @"Data
   Source=server;Integrated Security=True;Initial Catalog=DataBase";
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).SqlCommand = "select *
```

```
   from Table where Column = 100";
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Information

For SQL data sources of other types, the connection is created similarly, and an adapter corresponding to the data source type is connected. For example, for the MS SQL data source, you should connect SqlDataAdapter. For OracleDataAdapter is required for Oracle. Also, you should specify a connection string that matches the connection type.

You can also use data for designing reports and dashboards obtained from OData storage. In this case, you can do the authorization using a username, user password, or using a token. Authorization parameters are specified in the connection string to the OData storage using the ";" separator.

HomeController.cs

```
...
public IActionResult GetReport()
{

   var report = new StiReport();

   //Authorization using a user account
   var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
   services.odata.org/V4/Northwind/
   Northwind.svc;AddressBearer=adress;UserName=UserName;Password=Password;C
   lient_Id=Your Client ID", false, null);

   //Authorization using a user token
   var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
   services.odata.org/V4/Northwind/Northwind.svc;Token=Enter your token",
   false, null);

   report.Dictionary.Databases.Add(oDataDatabase);
   oDataDatabase.Synchronize(report);

   //Query with data filter
   ((StiSqlSource)report.Dictionary.DataSources["Products"]).SqlCommand =
   "Products?$filter=ProductID eq 2";
```

```
    return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

The table below shows the connection string templates for different types of data sources.

Data Source	Connection String Template
MS SQL	Integrated Security=False; Data Source=myServerAddress;Initial Catalog=myDataBase; User ID=myUsername; Password=myPassword;
MySQL	Server=myServerAddress; Database=myDataBase;UserId=myUsername; Pwd=myPassword;
ODBC	Driver={SQL Server}; Server=myServerAddress;Database=myDataBase ; Uid=myUsername; Pwd=myPassword;
OLE DB	Provider=SQLOLEDB.1; Integrated Security=SSPI;Persist Security Info=False; Initial Catalog=myDataBase;Data Source=myServerAddress
Oracle	Data Source=TORCL;User Id=myUsername;Password=myPassword;
MS Access	Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Password=pass;Data Source=C:\ \myAccessFile.accdb;
PostgreSQL	Server=myServerAddress; Port=5432; Database=myDataBase;User Id=myUsername; Password=myPassword;
Firebird	User=SYSDBA; Password=masterkey; Database=SampleDatabase.fdb;DataSource=my ServerAddress; Port=3050; Dialect=3; Charset=NONE;Role=; Connection lifetime=15; Pooling=true; MinPoolSize=0;MaxPoolSize=50;

	Packet Size=8192; ServerType=0;
SQL CE	Data Source=c:\MyData.sdf; Persist Security Info=False;
SQLite	Data Source=c:\mydb.db; Version=3;
DB2	Server=myAddress:myPortNumber;Database=myDataBase;UID=myUsername;PWD=myPassword; Max Pool Size=100;Min Pool Size=10;
Infomix	Database=myDataBase;Host=192.168.10.10;Server=db_engine_tcp;Service=1492;Protocol=onsoctcp;UID=myUsername;Password=myPassword;
Sybase	Data Source=myASEserver;Port=5000;Database=myDataBase;Uid=myUsername;Pwd=myPassword;
Teradata	Data Source=myServerAddress;User ID=myUsername;Password=myPassword;
VistaDB	Data Source=D:\folder \myVistaDatabaseFile.vdb4;Open Mode=ExclusiveReadWrite;
Universal(dotConnect)	Provider=Oracle;direct=true;data source=192.168.0.1;port=1521;sid=sid;user=user; password=pass
MongoDB	mongodb://<user>:<password>@localhost/test
OData	http://services.odata.org/v3/odata/OData.svc/
Other...	The table shows the most commonly used templates for the connection string. You can view various connection string options at the special website.

Data from XML, JSON, Excel files

Connecting to XML and JSON data sources can be stored in the report template. If you want to specify data files from the code, you can use the example below.

HomeController.cs

```
...
public IActionResult GetReport()
{
   DataSet data = new DataSet();
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));
   report.RegData(data);

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

HomeController.cs

```
...
public IActionResult GetReport()
{
   DataSet data
   = StiJsonToDataSetConverterV2.GetDataSetFromFile(StiNetCoreHelper.MapPat
   h(this, "Data/Demo.json"));

   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));
   report.RegData(data);

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Information

The viewer has the possibility of obtaining data from an Excel file. To do this, you can use the following method.

```
DataSet dataSet = StiExcelConnector.Get().GetDataSet(new StiExcelOptions(ar
```

5.1.5 Localization

The **HTML5 Viewer** component supports the complete localization of its interface. To localize the report viewer interface, use the special **Localization** property. The value of this property should specify the path to the localization XML file (relative or absolute).

Index.cshtml

```
...
```

```
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Localization = "Localization/en.xml"
})
...
```

When you load the report viewer, the localization file will be loaded automatically.

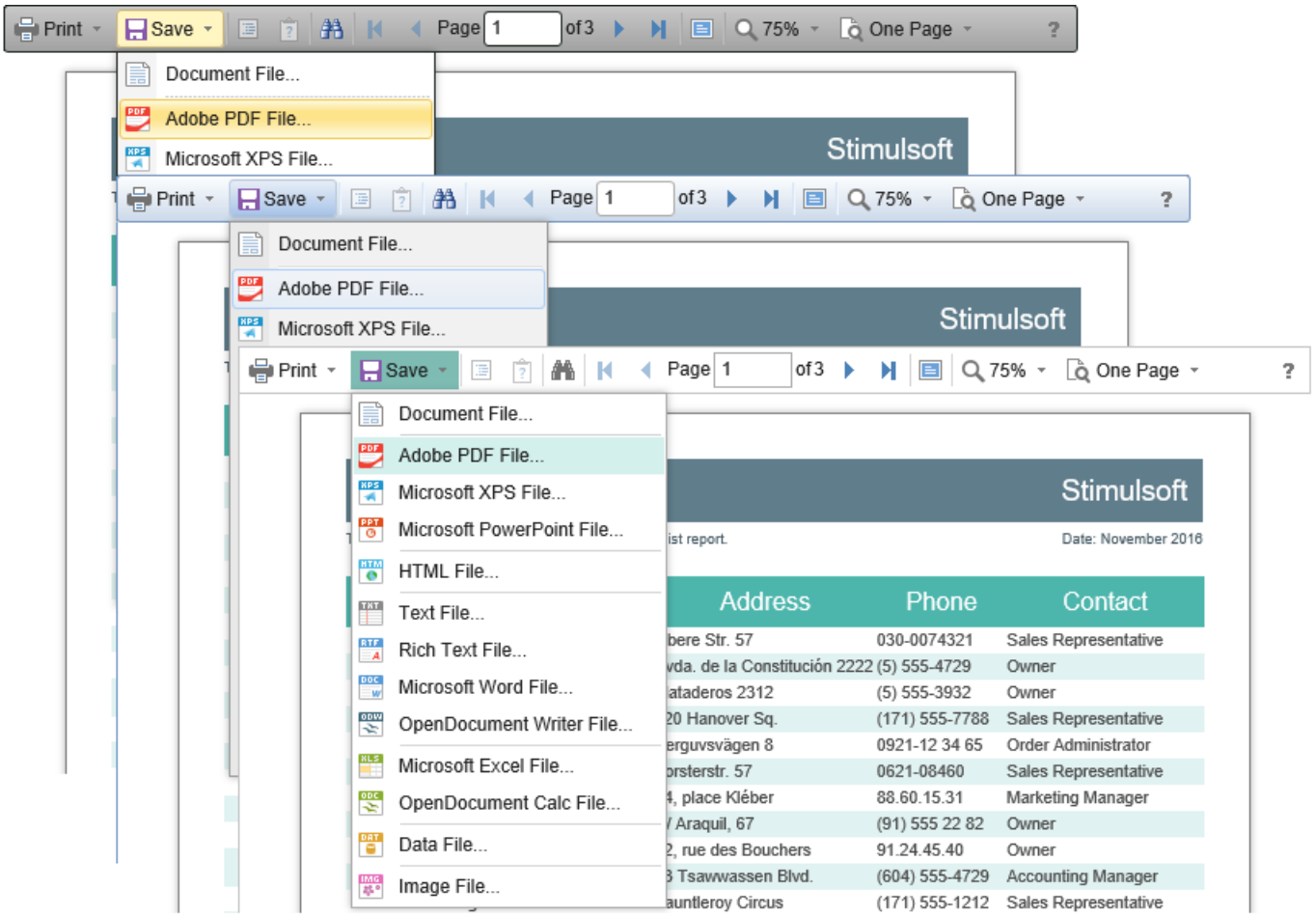
5.1.6 Using Themes

The **HTML5 Viewer** component can change the appearance of visual controls. To change the theme, use the **Theme** property, which can take one of the values of the **StiViewerTheme** enumeration.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Theme = StiViewerTheme.Office2022WhiteTeal
})
...
```

There are currently **8 themes** available with different color accents. As a result, **more than 60** variants of the appearance are available. This allows you to customize the appearance of the viewer for almost any design of the Web project.

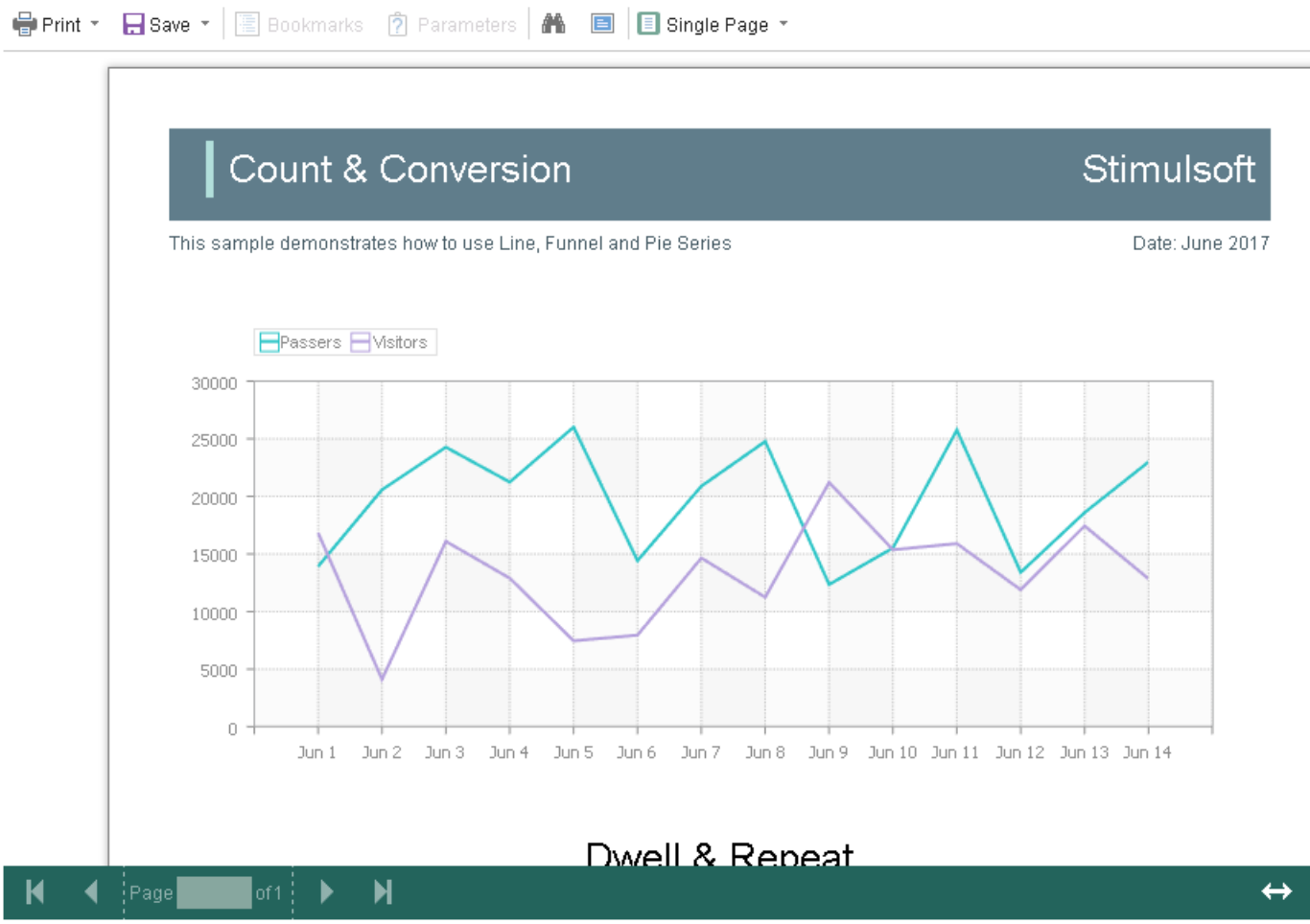


By default, the viewer has only the top toolbar on which all the report controls are located. If necessary, the toolbar can be split into top and bottom parts. The top panel will contain the menu for printing and exporting the report and the buttons for working with parameters and bookmarks. The bottom panel will contain controls to switch between the report pages and setting the zoom of pages. To enable this mode, enable the **ToolbarDisplayMode** property. It has values **Simple** and **Separated**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      ScrollbarsMode = true
   },
   Toolbar =
   {
```

```
      DisplayMode = StiToolbarDisplayMode.Separated
   }
})
...
```



In addition, it is possible to set the appearance parameters for the main elements of the viewer. For example, you can change the font and color of the control panel inscriptions of the viewer, set the background of the viewer, set the color of page borders, etc. Below is a list of available properties that change the appearance of the viewer, and their default values.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      BackgroundColor = Color.White,
```

```
      PageBorderColor = Color.Blue,
      ShowPageShadow = true
   },
   Toolbar =
   {
      BackgroundColor = Color.White,
      BorderColor = Color.Gray,
      FontColor = Color.Black,
      FontFamily = "Arial"
   }
})
...
```

5.1.7 Basic Features

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The main features of the viewer include the following operations:

- Displaying the report.
- Switching between the report pages.
- Changing the scale.
- Displaying the preview mode.

All specified operations are performed in the AJAX mode without restarting the browser page. For the correct work of these operations, you should define a special ViewerEvent action.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      ViewerEvent = "ViewerEvent"
   }
})
...
```

HomeController.cs

```
...
public IActionResult ViewerEvent()
```

```
{
   // Some code before viewer event
   // ...

   return StiNetCoreViewer.ViewerEventResult(this);
}
...
```

Information

This action is mandatory. Without it, the correct operation of the viewer is not possible.

The **ViewerEvent** action returns a prepared HTML page of the report (or set of pages), built taking into account the current state of the viewer. If necessary, you can change the parameters of the current report in the specified action and update the report data in case of interactive actions of the viewer.

5.1.8 Printing Reports

Information

Please note that the print option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component provides several options for printing a report. Each has its advantages and disadvantages.

Print to PDF
Printing will be done by exporting the report to PDF. The advantages are greater accuracy of positioning and printing of the report elements compared to other printing options. Among the drawbacks, one can mention the mandatory presence of a plug-in installed in a web browser for viewing PDF files (modern browsers have embedded PDF viewer and printer).

Print with Preview
The report will be printed in a separate pop-up browser window in HTML. The report

can be previewed and then sent to the printer or copied to another location as text or HTML code. Advantages - cross-browser compatibility when printing, no need to install special plug-ins. The disadvantage is the relatively low accuracy of the position of the report elements due to the peculiarities of the implementation of HTML formatting.

Print without Preview
The report will be printed directly to the printer without preview. After selecting this menu item, the system print dialog is displayed. Since printing in this mode is carried out in the HTML format, the print quality is similar to printing a report with a preview.

Information

When printing to the **HTML format**, you should check the compliance of report page settings and printer parameters (paper size, orientation, margins, indents), as well as check your browser print settings, such as margins, headers, footers, watermarks printing, color printing.

The print function does not require additional settings for the viewer. If you need to perform any actions before printing a report, you can define a special **PrintReport** action.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      PrintReport = "PrintReport"
   }
})
...
```

HomeController.cs

```
...
public IActionResult PrintReport()
{
   // Some code before print
   // ...
```

```
    return StiNetCoreViewer.PrintReportResult(this);
}
...
```

Print setup

If you choose printing a report in the viewer panel, a menu with printing options is displayed. The **HTML5 Viewer** component is able to force the required printing mode. To do this, set the **PrintDestination** property to one of the following values of the **StiPrintDestination** enumeration.

- **Default** – the menu will be displayed (the default property value);
- **Pdf** – print to the PDF format;
- **Direct** – printing to the HTML format directly to the printer, the system print dialog will be displayed;
- **WithPreview** – print to the HTML format with preview in a pop-up window.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
    Toolbar =
    {
        PrintDestination = StiPrintDestination.Default
    }
})
...
```

The **HTML5 Viewer** component is able to completely disable report printing. To do this, set the value of the **ShowPrintButton** property to **false**.

Index.cshtml

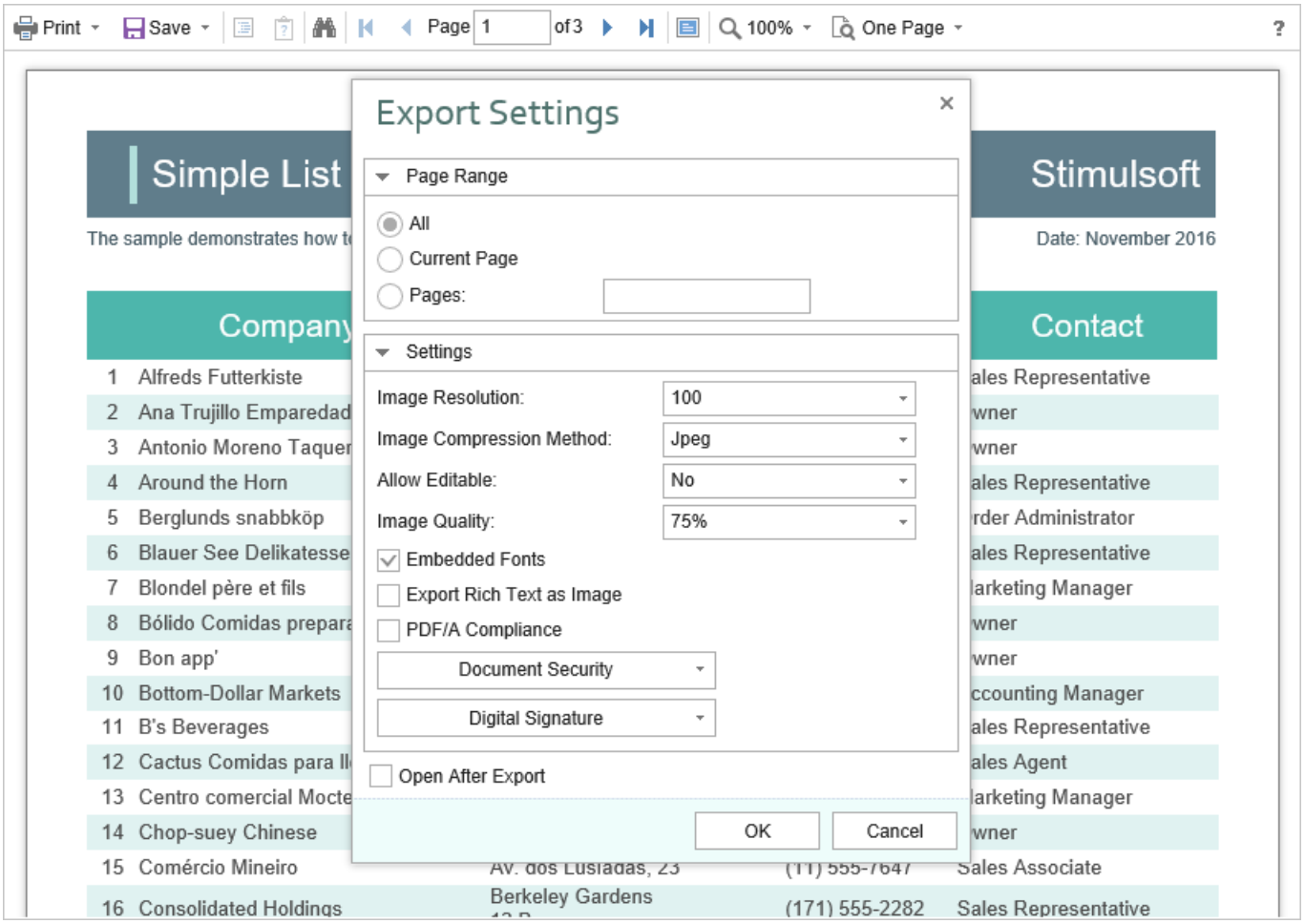
```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
    Toolbar =
    {
        ShowPrintButton = false
    }
})
...
```

5.1.9 Exporting Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component allows you to export the displayed report to three dozen various formats, such as **PDF**, **HTML**, **Word**, **Excel**, **XPS**, **RTF**, **images**, **text**, and others. You may export the dashboard to PDF, Excel, image files.



The export function does not require additional settings for the viewer. If you need to perform any actions before exporting the report, you can define a special **ExportReport** action.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      ExportReport = "ExportReport"
   }
})
...
```

HomeController.cs

```
...
public IActionResult ExportReport()
{
   // Some code before export
   // ...

   return StiNetCoreViewer.ExportReportResult(this);
}
...
```

Export settings

Each report export format of the **HTML5 Viewer** component has a lot of settings, and each setting has its default values. Sometimes you need to set other default values. For this purpose, a special **DefaultSettings** property of the viewer is used. It is a container for all the default export settings.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Exports =
   {
      DefaultSettings =
      {
         ExportToPdf =
         {
            ImageQuality = 0.75f,
            ImageFormat = Stimulsoft.Report.Export.StiImageFormat.Color
         },
         ExportToHtml =
         {
            ExportMode = Stimulsoft.Report.Export.StiHtmlExportMode.Div,
            UseEmbeddedImages = true
         }
      }
   }
})
```

```
...
```

If it is required, you can completely hide export dialogs. Exporting will always be done with default settings. For this, it is enough to set the value of the **ShowExportDialog** property to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Exports =
   {
      ShowExportDialog = false
   }
})
...
```

The **HTML5 Viewer** component contains 30+ export formats, and sometimes you need to disable unwanted formats. This allows you to simplify UI and the use of the viewer. To disable unused export formats, it is enough to set the values for the corresponding properties of the viewer listed in the list below to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Exports =
   {
      ShowExportToDocument = true,
      ShowExportToPdf = true,
      ShowExportToXps = true,
      ShowExportToPowerPoint = true,
      ShowExportToHtml = true,
      ShowExportToHtml5 = true,
      ShowExportToMht = true,
      ShowExportToText = true,
      ShowExportToRtf = true,
      ShowExportToWord2007 = true,
      ShowExportToOpenDocumentWriter = true,
      ShowExportToExcel = true,
      ShowExportToExcelXml = true,
      ShowExportToExcel2007 = true,
      ShowExportToOpenDocumentCalc = true,
      ShowExportToCsv = true,
      ShowExportToDbf = true,
      ShowExportToXml = true,
      ShowExportToDif = true,
      ShowExportToSylk = true,
```

```
    ShowExportToImageBmp = true,
    ShowExportToImageGif = true,
    ShowExportToImageJpeg = true,
    ShowExportToImagePcx = true,
    ShowExportToImagePng = true,
    ShowExportToImageTiff = true,
    ShowExportToImageMetafile = true,
    ShowExportToImageSvg = true,
    ShowExportToImageSvgz = true
  }
})
...
```

The **HTML5 Viewer** component can completely disable the export menu. To do this, set the value of the **ShowSaveButton** property to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
  Toolbar =
  {
    ShowSaveButton = false
  }
})
...
```

5.1.10 Viewing Modes

The **HTML5 Viewer** component has two modes for displaying reports - with and without scrollbars. By default, the view mode without scrollbars is set. To enable the scrollbar view mode, set the value of the **ScrollbarsMode** property to **true**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
  Appearance =
  {
    ScrollbarsMode = true
  }
})
...
```

In the first mode (without scrollbars), the viewer displays a page or report as a whole, automatically stretching the preview space. If the width and height are specified, the viewer will truncate the page that is out of bounds. In the second mode, unlike the first one, when the page is out of bounds of the viewer's size, no truncation will be

performed. Scrollbars will appear, using which you can view the entire page or report.

Information

In the report mode with scrollbars, you should set the height of the viewer, otherwise the default height will be set to **650 pixels**.

The **HTML5 Viewer** component provides the full-screen report and dashboard mode. By default, the standard view mode is enabled, the viewer has the specified dimensions in the settings. To enable the full-screen mode, set the **FullScreenMode** property to **true**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      FullScreenMode = true
   }
})
...
```

Also, to enable or disable the full-screen mode, you can use the corresponding button on the control panel of the viewer.

The **HTML5 Viewer** component has three modes to display reports - page-by-page, entire report, and tabular display of report pages. To control the modes, the **ViewMode** property is used. It can have one of the specified values - **SinglePage**, **Continuous**, **MultiplePages**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Toolbar =
   {
      ViewMode = StiWebViewMode.SinglePage
   }
})
```

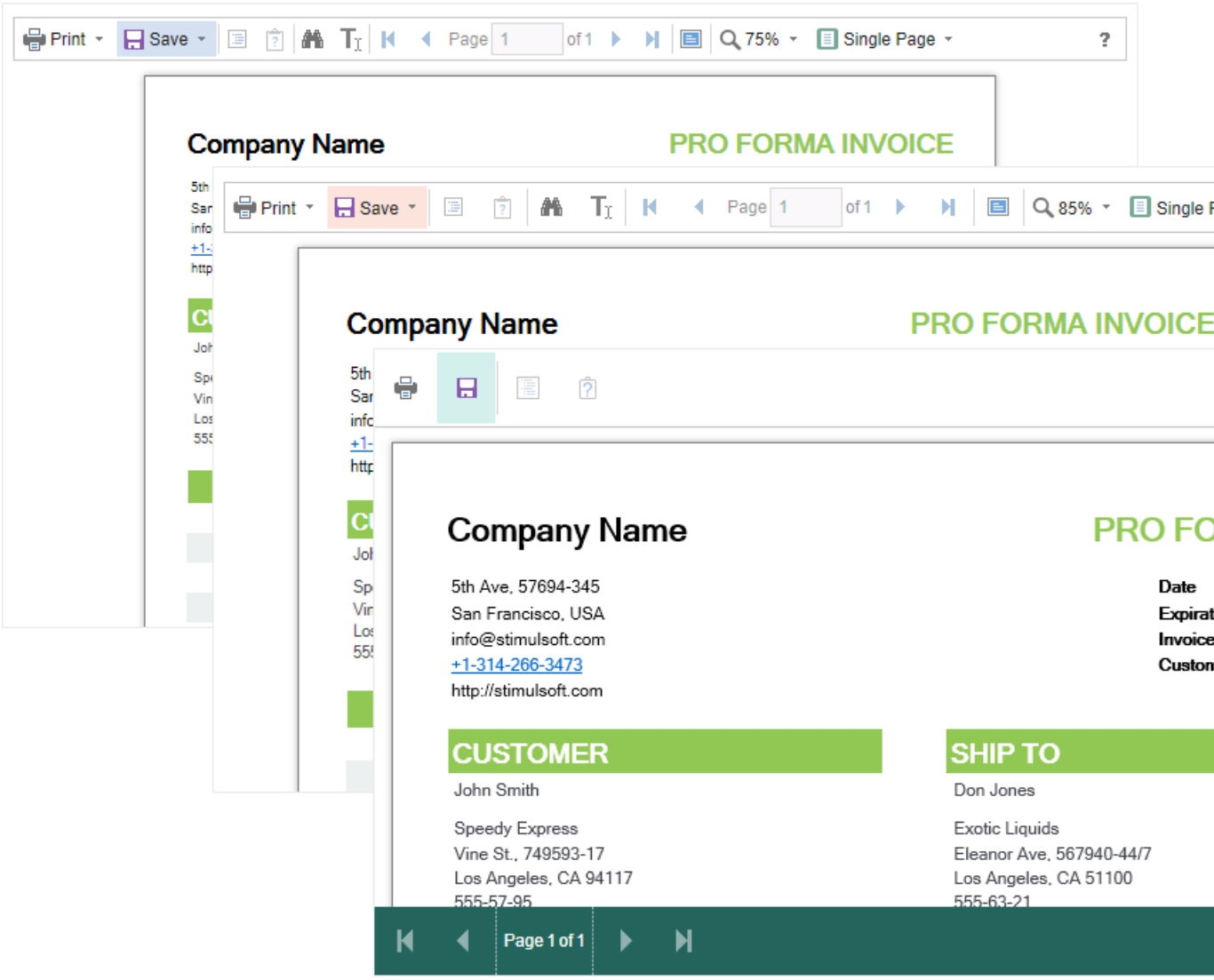
```
...
```

The **HTML5 Viewer** component supports interaction on a regular PC display and works with a touchscreen of screens and mobile devices. The **InterfaceType** property allows controlling the interface modes. The property can have one of the following values:

- **Auto** – the viewer's interface is determined automatically depending on the device the report is displayed on. That is the default value.
- **Mouse** – the standard interface with a mouse control will be used for all the screen types.
- **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability.
- **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with smartphones.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      InterfaceType = StiInterfaceType.Auto
   }
})
...
```



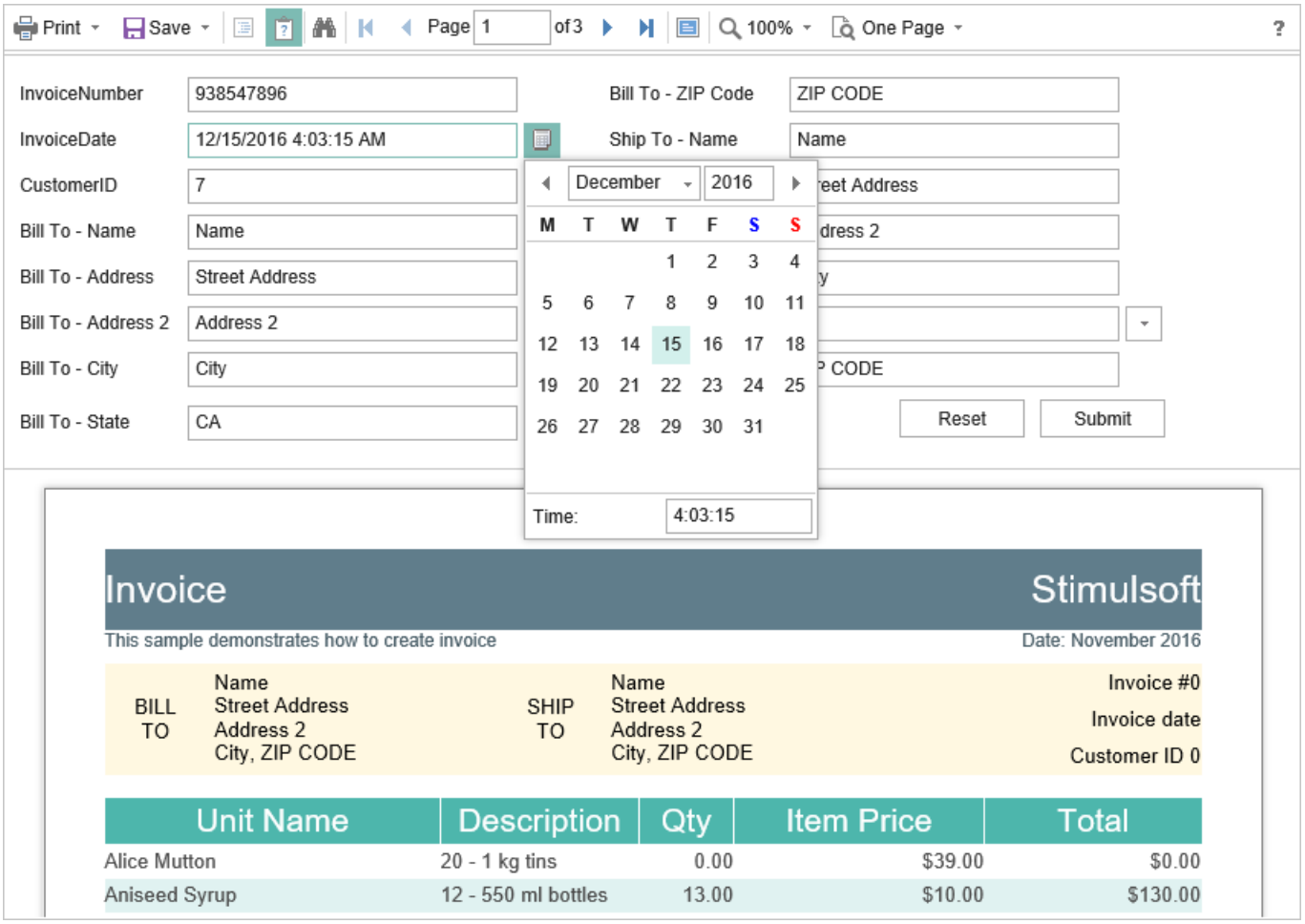
5.1.11 Work with Parameters

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To work with report parameters in the **HTML5 Viewer**, there is a special settings panel. To add a parameter to the panel, you need to define a variable in a report requested by the user. When viewing a report in the viewer, such a variable will be

automatically added to the settings panel. It supports all report variables (normal variables, date and time, borders, lists, etc.).



To work with reports with parameters, no additional viewer settings are required. If you need to perform some actions before applying the parameters, you can define a special **Interaction** action.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      Interaction = "ViewerInteraction"
   }
})
...
```

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   // Some code before any interaction
   // ...

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

This action is called during any interactive actions of the viewer. If you need to perform any actions only when applying report parameters, you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class. They are passed to any server-side on any request and contain all necessary information and states of the client part of the viewer. To determine the type of action of the viewer, it is enough to check the **Action** property of the viewer parameters.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   if (requestParams.Action == StiAction.Variables)
   {
      // Some code before apply parameters
   }

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

If you do not need to work with parameters, you can completely disable this feature. To do this, use the **ShowParametersButton** property in the **Toolbar** section of properties, which should be set to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Toolbar =
   {
      ShowParametersButton = false
```

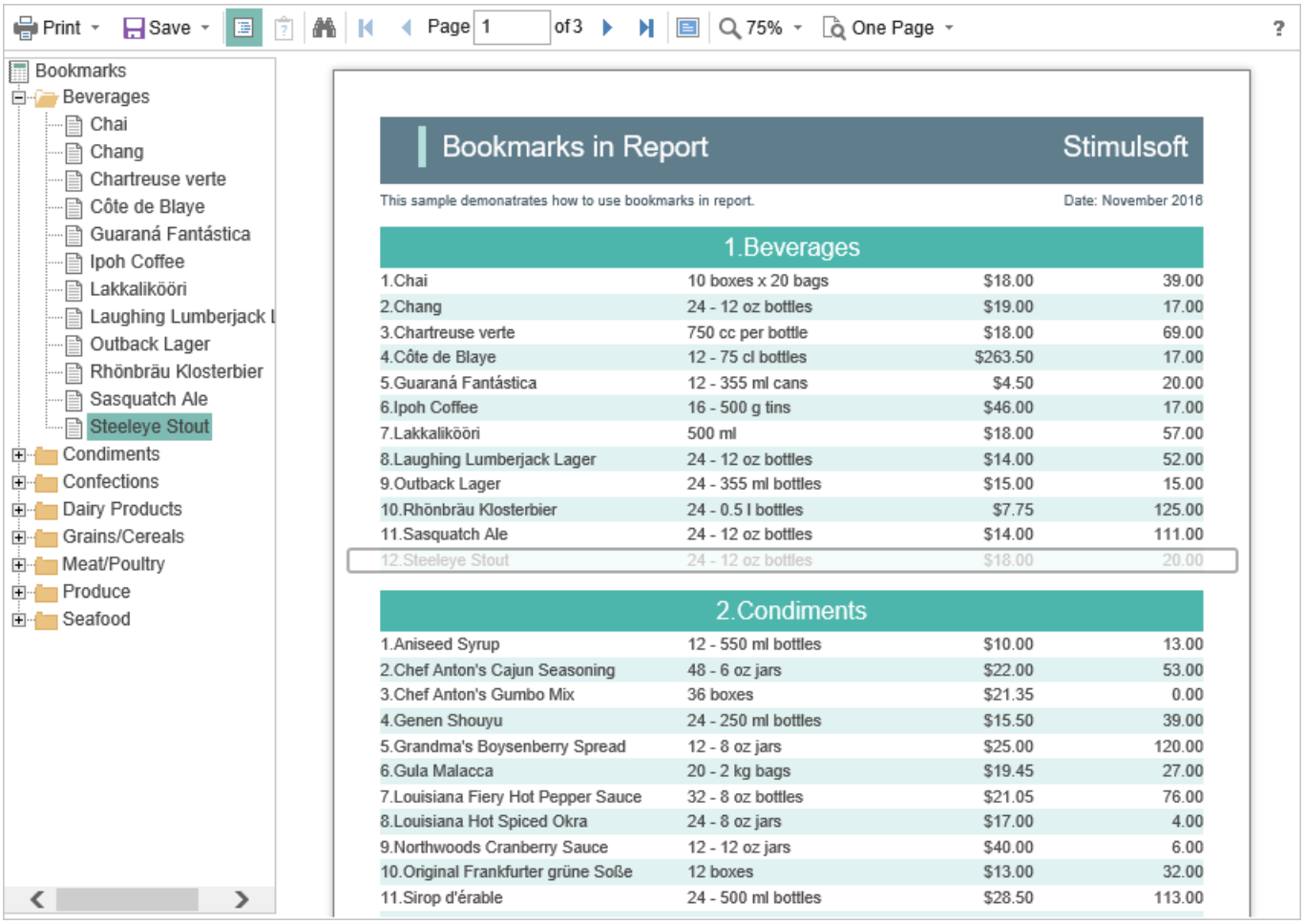
```
    }
})
...
```

Information

The options panel will not be displayed with such a viewer configuration, even if the parameters are present in the displayed report.

5.1.12 Work with Bookmarks

The **HTML5 Viewer** component supports report bookmarks. A panel with bookmarks will be displayed when displaying such a report on the left side of the page. When you select a bookmark of the report, the viewer will carry out an automatic transition to the specified page, and the report item with a bookmark is highlighted.



By default, the bookmarks bar width is 180 pixels. The **HTML5 Viewer** component allows you to change this value. For this, the **BookmarksTreeWidth** property, which value is specified in pixels, is used.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      BookmarksTreeWidth = 200
   }
})
...
```

If work with report bookmarks is not required, you can disable this feature. For this, set the **ShowBookmarksButton** property to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Toolbar =
   {
      ShowBookmarksButton = false
   }
})
...
```

Information

In this case, report bookmarks will not be displayed, even if they are present in the displayed report. This property has no effect on printing and exporting reports.

When printing a report with bookmarks, the bookmark tree will be hidden. If you want to print bookmarks with the report, it is necessary to set the **BookmarksPrint** property to **true**.

Index.cshtml

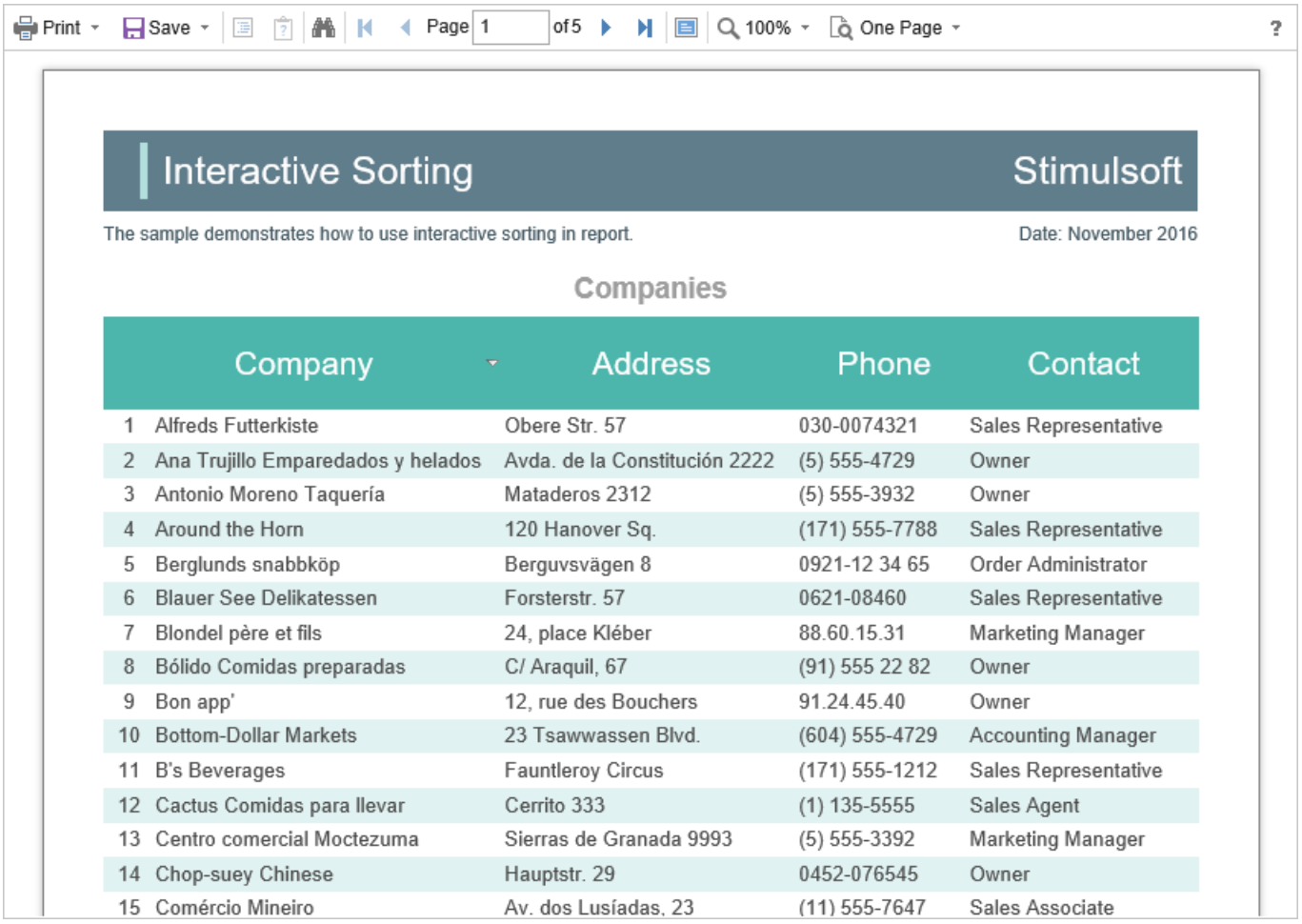
```
...
```

```
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions () {
   Appearance =
   {
      BookmarksPrint = true
   }
})
...
```

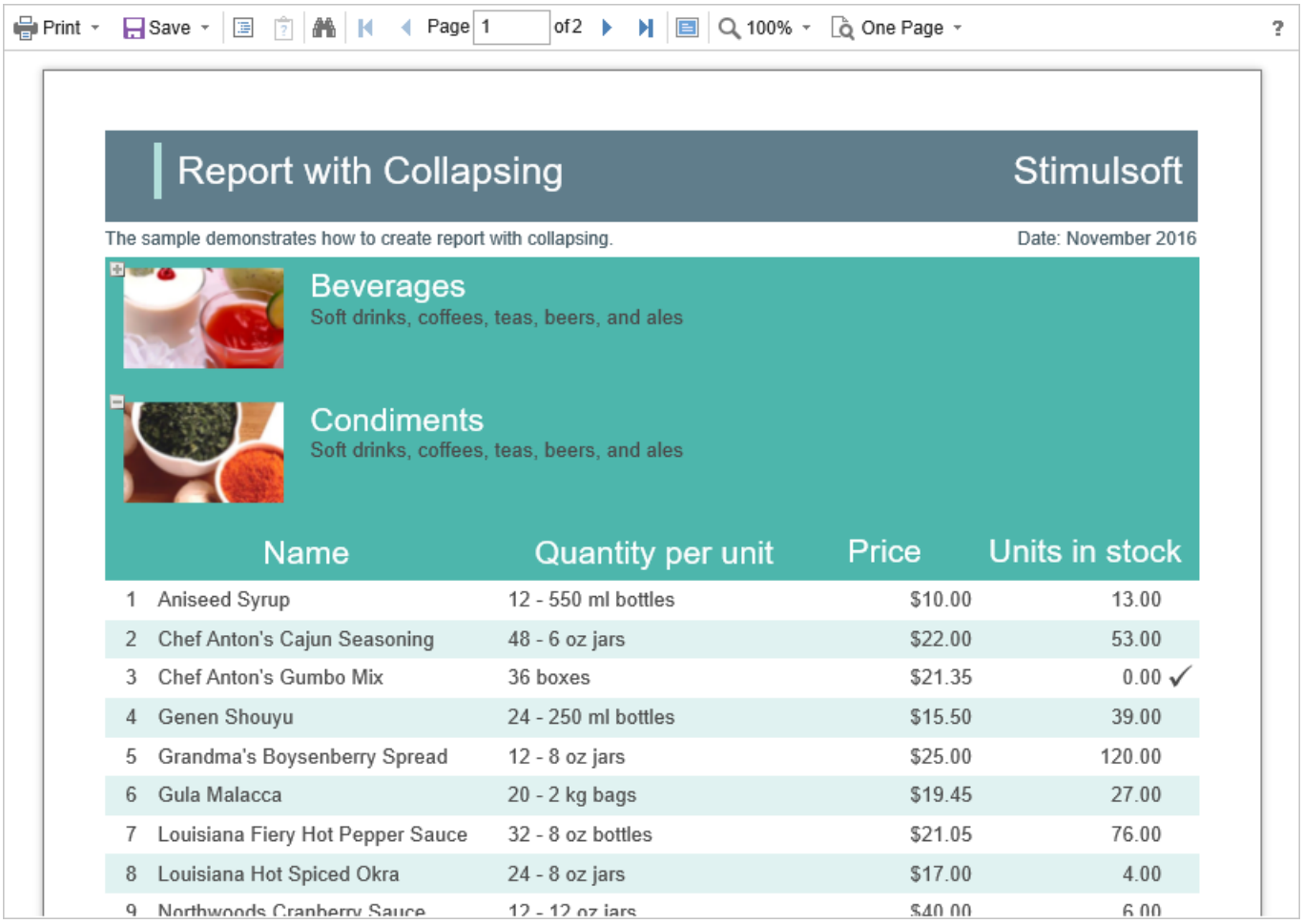
5.1.13 Dynamic Sorting, Collapsing, and Drill-Down

The **HTML5 Viewer** component supports dynamic sorting, collapsing, and drill-down of reports. Dynamic sorting provides the ability to change the direction of sorting in a rendered report. To do this, click on the component that has dynamic sorting enabled. Dynamic sorting is carried out in the following directions - **Ascending** and **Descending**. Each time the component is clicked, the sorting direction is reversed.

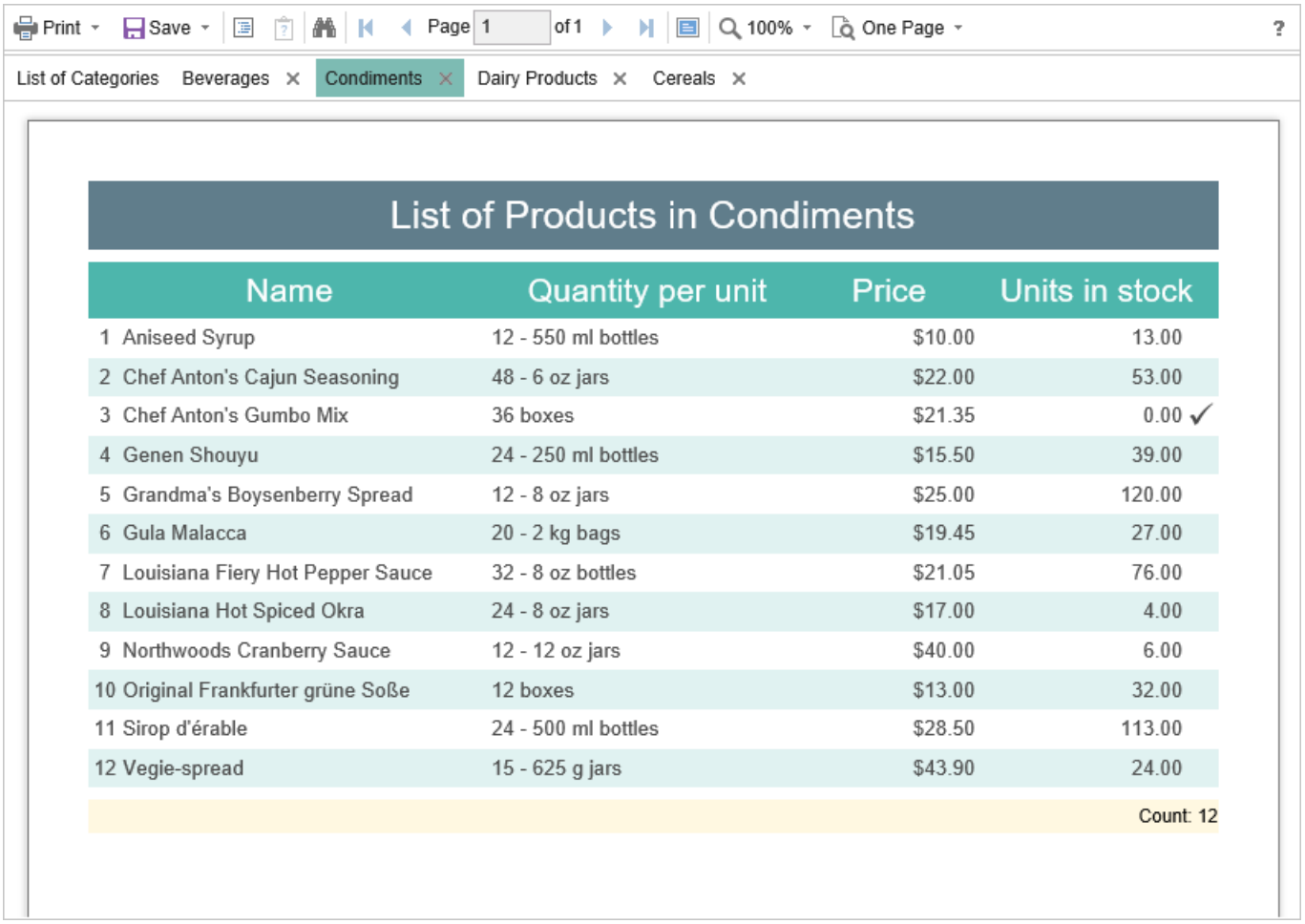
Multi-level sorting is allowed in the report. To do this, hold down the **Ctrl** key and sequentially click on the sorted components in the report. To reset sorting, you can click on any sorted component without holding down the **Ctrl** key.



A report with dynamic collapsing is an interactive report in which blocks can collapse/expand their content when you click on the block title. Report elements, which can be collapsed/expanded, are indicated by special icons - **[-]** or **[+]**.



When using drill-down, under the main panel of the viewer, the drill-down panel with tabs for drill-down reports will be displayed. The currently displayed report will be highlighted.



To work with dynamic sorting, collapsing, and drill-down reports, no additional viewer settings are required. A special **Interaction** action is used to perform any actions before sorting, collapsing, or drill-down of the report. It will be called when interactive action of the viewer.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      Interaction = "ViewerInteraction"
   }
})
...
```

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   // Some code before any interaction
   // ...

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

To get the type of action, you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class. They are passed to any server-side by any request and contain all necessary information and states of the client part of the viewer. For each type of interactivity, the viewer has a certain type of action:

- **Sorting** – when using column sorting;
- **DrillDown** – when using drill-down in reports;
- **Collapsing** – when using collapsing report blocks.

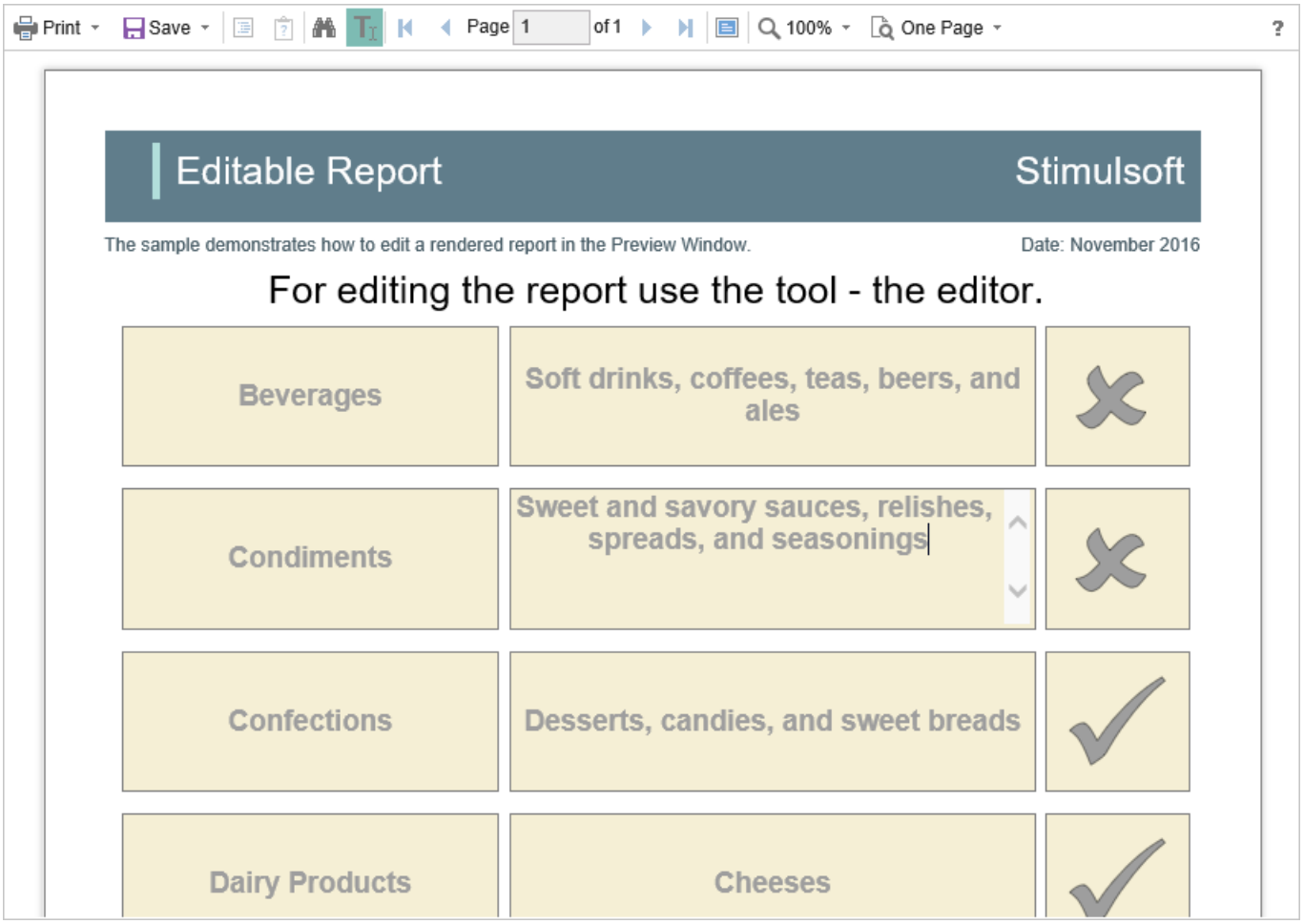
HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   switch (requestParams.Action)
   {
      case StiAction.Sorting:
         break;

      case StiAction.DrillDown:
         break;

      case StiAction.Collapsing:
         break;
   }

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

5.1.14 Editing Report

The **HTML5 Viewer** component has the ability to edit report items, such as text boxes and checkboxes. You should mark the required components as editable in the report template for the editing to be possible. After displaying a report in the viewer, you need to click the corresponding button on the viewer panel to start editing. After editing, it is necessary to click the button once more, and all changes will be

applied to the report.



For the report edit mode, no special settings of the viewer required.

Information

The edited settings will be applied when you print or export a report, and the original report remains unchanged. After restarting the viewer, all the values will be returned to the initial ones.

5.1.15 Sending Report by Email

Information

Please note that the Send Report by Email option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component provides the ability to send reports by email. To activate this feature, you should set the **ShowSendEmailButton** property of the viewer to **true** and define the **EmailReport** action.

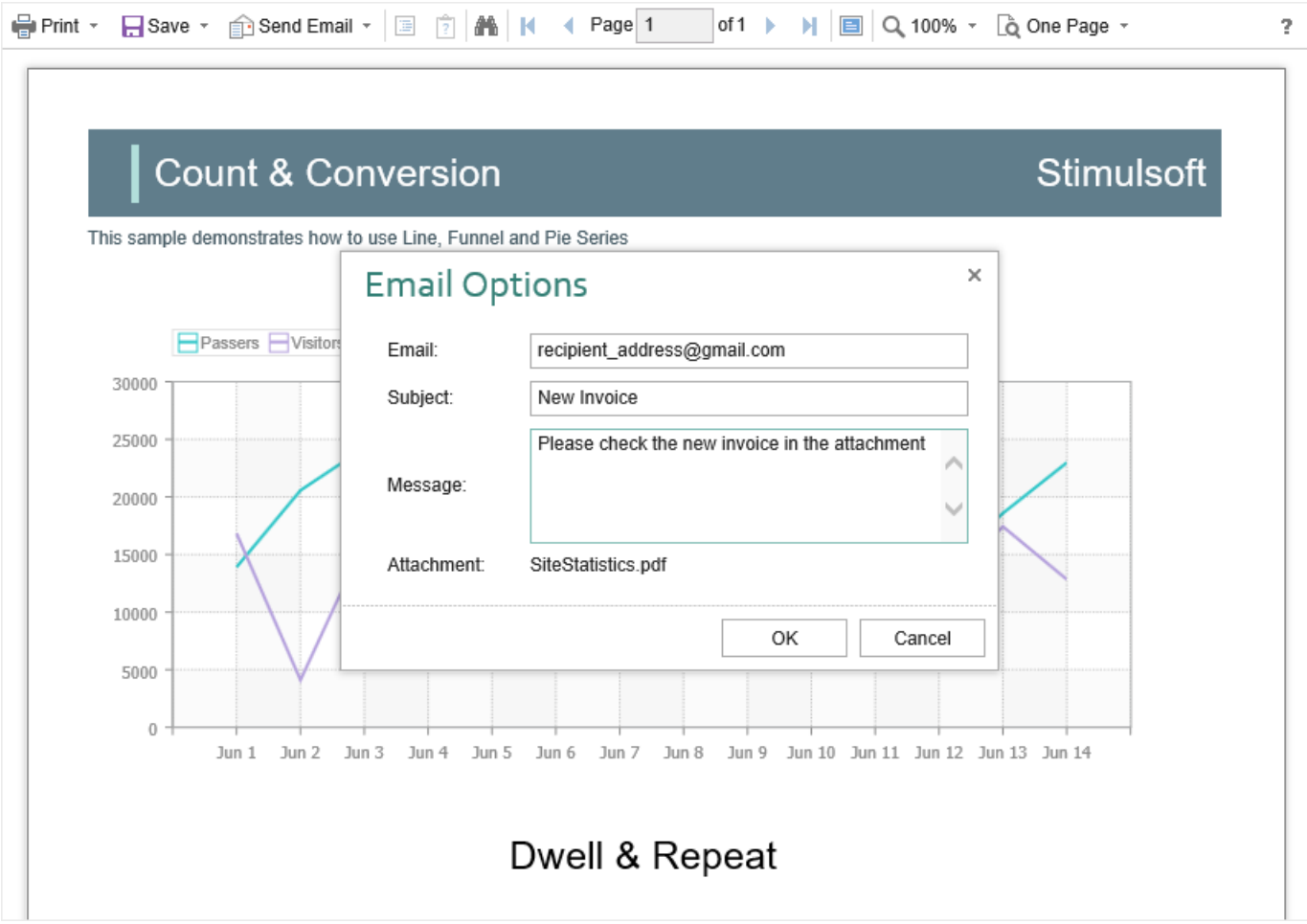
Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      EmailReport = "EmailReport"
   },
   Toolbar =
   {
      ShowSendEmailButton = true
   }
})
...
```

HomeController.cs

```
...
public IActionResult EmailReport()
{
   StiEmailOptions options = StiNetCoreViewer.GetEmailOptions(this);

   // Passed from the viewer, can be checked and changed
   // options.AddressTo = "";
   // options.Subject = "";
   // options.Body = "";

   // Should be filled here
   options.AddressFrom = "admin_address@test.com";
   options.Host = "smtp.test.com";
   options.Port = 465;
   options.UserName = "admin_address@test.com";
   options.Password = "admin_password";

   // options.CC.Add("email@test.com");
   // options.BCC.Add("email@test.com");
   // options.EnableSsl = true;

   return StiNetCoreViewer.EmailReportResult(this, options);
}
...
```

When sending a report by email, the menu to select the attachment format is displayed. It corresponds to the menu for selecting the format for exporting the report. After selecting the format, the dialog to enter the send email parameters, such as the recipient's email, subject, and text of the message, is displayed.



After confirmation of sending the email, the above described **EmailReport** event will be called. You can check and correct the data entered in this form. The exported report file will be attached to the email automatically.

The **HTML5 Viewer** component allows you to set default values for the send email form. The **DefaultEmailAddress**, **DefaultEmailSubject**, and **DefaultEmailMessage** properties can be used for this. By default, these properties are empty.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Email =
   {
      DefaultEmailAddress = "recipient_address@gmail.com",
      DefaultEmailSubject = "New Invoice",
      DefaultEmailMessage = "Please check the new invoice in the
      attachment"
   }
})
...
```

5.1.16 Calling Designer from Viewer

The **HTML5 Viewer** component has the ability to call the report designer. The special **Design** button in the toolbar of the viewer (the button is disabled by default) should be used. To use this feature, you should set the **ShowDesignButton** property to true and define the **DesignReport** event handler.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      DesignReport = "DesignReport"
   },
   Toolbar =
   {
      ShowDesignButton = true
   }
})
...
```

HomeController.cs

```
...
public IActionResult DesignReport()
{
   StiReport report = StiNetCoreViewer.GetReportObject(this);
   ViewBag.ReportName = report.ReportName;

   return View("Designer");
}
...
```

Information

The viewer does not run the designer. It only calls the specified action, in which you can get all the necessary parameters. Then, in action, you can implement a

redirection to another View, which contains the report designer.

5.1.17 Caching

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component allows you to use the server cache to store rendered reports. If you do not use caching, you should load the report, connect data, and render it again every time you request a page. If you use caching, the previously rendered report will be loaded from the cache every time you refresh the page.

When using caching, it should be taken into account that every report saved in the cache takes up server memory and, with a large number of requests to reports, this can become a critical issue. Therefore, you need to choose between two options: low memory requirements but high in performance or low-performance requirements but high in memory.

You need to connect modules to work with a session or cache on the server-side to use caching. To do this, just add the following services to the project in the start file of a project.

Startup.cs

```
...
public void ConfigureServices(IServiceCollection services)
{
   services.AddMemoryCache();
   services.AddSession();
   services.AddMvc();
}
...
```

You can manage caching with the following properties.

The CacheMode property

This property of the viewer enables caching and sets its type. It can take one of the following values, specified in the **StiServerCacheMode** enumeration:

- **None** – caching is disabled. Each action of the viewer requires loading the report from the file and, if it is a report template, then render it;
- **ObjectCache** – for caching, the server cache is used. The report object is saved in this cache (set by default);
- **StringCache** – for caching, the server cache is used. The report is saved as a packed string in this cache;
- **ObjectSession** – the current session, in which the report object is saved, is used for caching;
- **StringSession** – for caching, the current session is used. The report is saved as a packed string in this cache.

The CacheItemPriority property

This property sets the priority of the report stored in the server's cache. It affects the automatic clearing of the server memory in case of a lack of memory. The lower the priority is, the greater is the chance of removing information from memory.

The CacheTimeout property

This property specifies the amount of time in minutes for which you want to save the report in the server cache. If you use caching and the requested report is not found in the cache (the objects storage time has expired), then it will be requested again using a special **GetReport** event, then connect the report data and render it.

StiCacheHelper

The **HTML5 Viewer** component provides the ability to define your methods of working with report caching. For this purpose, a special class **StiCacheHelper** is used. It contains methods for obtaining a report from the cache and saving the report to

the cache. It is necessary to create a new class inherited from **StiCacheHelper** and reload the above methods, which respectively have the names - **GetReport**, **SaveReport** and **RemoveReport**.

HomeController.cs

```
...
public class ViewerController : Controller
{
   public class StiMyCacheHelper : StiCacheHelper
   {
      public override StiReport GetReport(string guid)
      {
         string path =
         System.IO.Path.Combine(this.HttpContext.Server.MapPath("CacheFiles"
         ), guid);
         if (System.IO.File.Exists(path))
         {
            StiReport report = new StiReport();
            string packedReport = System.IO.File.ReadAllText(path);
            if (guid.EndsWith("template"))
            report.LoadPackedReportFromString(packedReport);
            else report.LoadPackedDocumentFromString(packedReport);

            return report;
         }
         return null;

         //return base.GetReport(guid);
      }

      public override void SaveReport(StiReport report, string guid)
      {
         string packedReport = guid.EndsWith("template") ?
         report.SavePackedReportToString() :
         report.SavePackedDocumentToString();
         string path =
         System.IO.Path.Combine(this.HttpContext.Server.MapPath("CacheFiles"
         ), guid);
         System.IO.File.WriteAllText(path, packedReport);

         //base.SaveReport(report, guid);
      }

      public override void RemoveReport(string guid)
      {
         var path = Path.Combine(HttpContext.Server.MapPath("CacheFiles"),
         guid);
         if (File.Exists(path))
            File.Delete(path);
      }
   }

   static ViewerController()
   {
      StiNetCoreViewer.CacheHelper = new StiMyCacheHelper();
```

```
    }
}
...
```

To initialize the work with report caching using the created class, it is enough to set it as a value of the static **StiNetCoreViewer.CacheHelper** property in the controller constructor.

Information

If report caching is disabled (the **CacheMode** property of the viewer is set to **None**), the specified class will not be used.

5.1.18 Additional Methods

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

For **HTML5 Viewer**, several additional methods are used to get the object of the currently viewed report, parameters of the current state of the viewer, and other useful data. These methods can be used in any actions of the viewer.

The GetReportObject() method

Returns the report object with which the viewer is currently working. It is possible to perform the necessary actions - register new data sets, change report properties, assign parameters or load another report to the object. Then, the report can be returned to the viewer, specifying it as a parameter in the resulting action method.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
```

```
   StiReport report = StiNetCoreViewer.GetReportObject(this);
   report.ReportName = "MyReportName";

   return StiNetCoreViewer.InteractionResult(this, report);
}
...
```

The GetRouteValues() method

Returns values for URLs with which the viewer page was opened. Thus, it is possible to get the initial collection of run page parameters in any viewer action and use these values for any checks and conditions.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   RouteValueDictionary routeValues =
   StiNetCoreViewer.GetRouteValues(this);

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

You can also get values of URL parameters by parameter name, specifying it as the parameter of the called action of the viewer.

HomeController.cs

```
...
public IActionResult ViewerInteraction(string id)
{
   return StiNetCoreViewer.InteractionResult(this);
}
...
```

The GetFormValues() method

Returns the values of the form that initiated (opened by the POST request) a page of the viewer. Thus, it is possible to get a collection of form parameters in any action of the viewer.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   NameValueCollection formValues = StiNetCoreViewer.GetFormValues(this);

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

By default, this feature is disabled to optimize requests of the client-side of the viewer to the server. To enable it, set the **PassFormValues** property to **true**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Server =
   {
      PassFormValues = true
   }
})
...
```

The GetRequestParams() method

Returns all parameters of the current state of the viewer passed to the server-side. They can be useful for determining the type of action that the viewer is currently executing - for example, to determine the type of export and all action parameters.

HomeController.cs

```
...
public IActionResult ExportReport()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   if (requestParams.ExportFormat == StiExportFormat.Pdf)
   {
      StiReport report = StiNetCoreViewer.GetReportObject(this);

      // Some action with report for the PDF export
      // ...

      return StiNetCoreViewer.ExportReportResult(this, report);
   }

   return StiNetCoreViewer.ExportReportResult(this);
```

```
}
...
```

You can change the values of some parameters. After making changes, for the correct operation of the viewer, you should transfer the modified parameter object to the input of the resulting method.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   if (requestParams.Action == StiAction.Variables)
   {
      requestParams.Interaction.Variables["Variable1"] = "MyValue";
      return StiNetCoreViewer.InteractionResult(this, requestParams);
   }

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

The GetExportSettings() method

Returns all the parameters of the current report export. The type of the parameter object will correspond to the type of export selected in the viewer menu. Any export parameters can be changed and passed to the input of the resulting method. In this case, the report will be exported with the parameters transferred.

HomeController.cs

```
...
public IActionResult ExportReport()
{
   StiExportSettings settings = StiNetCoreViewer.GetExportSettings(this);
   if (settings.GetExportFormat() == StiExportFormat.Pdf)
   {
      StiPdfExportSettings pdfSettings = (StiPdfExportSettings)settings;
      pdfSettings.EmbeddedFonts = true;
      pdfSettings.AllowEditable = StiPdfAllowEditable.No;
      return StiNetCoreViewer.ExportReportResult(this, settings);
   }

   return StiNetCoreViewer.ExportReportResult(this);
}
...
```

The MapPath() and MapWebRootPath() methods

Returns the absolute path, respectively, to the application or wwwroot directory. You can use this to upload report templates files, data files, etc. These methods are located in the StiNetCoreHelper static class.

HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

5.1.19 Export and Printing from Code

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** provides the ability to print reports in various ways and export reports to various formats. These actions are performed using the viewer menu. If you want to print or export a report using the code, for example, in the controller action, you can use the special **StiNetCoreReportResponse** class. This class contains a set of static methods that allow you to print or export a report from the code, and the report viewer is not required.

Index.cshtml

```
...
@Html.ActionLink("Print Report from Code", "PrintReport")
<br />
@Html.ActionLink("Export Report from Code", "ExportReport")
...
```

HomeController.cs

```
...
private StiReport LoadSimpleList()
{
   DataSet dataSet = new DataSet();
   dataSet.ReadXml(Server.MapPath("Reports/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(Server.MapPath("Reports/SimpleList.mrt"));
   report.RegData(dataSet);

   return report;
}

public IActionResult PrintReport()
{
   StiReport report = LoadSimpleList();

   return StiNetCoreReportResponse.PrintAsPdf(report);
   //return StiNetCoreReportResponse.PrintAsHtml(report);
}

public IActionResult ExportReport()
{
   StiReport report = LoadSimpleList();

   return StiNetCoreReportResponse.ResponseAsPdf(report);
   //return StiNetCoreReportResponse.ResponseAsExcel2007(report);
   //return StiNetCoreReportResponse.ResponseAsText(report);
   //StiNetCoreReportResponse.ResponseAsJson(report);
}
...
```

The **StiNetCoreReportResponse** class contains methods for printing in PDF and HTML formats and methods to export the report in any of the supported formats. As arguments, methods can take various export settings, displaying modes and options for saving received files.

5.1.20 Timeout

When working with the **StiNetCoreViewer** component, you can set the timeout for various operations — storing the report in the cache, server response, and query execution. The timeout setting is done using the component properties and report options.

CacheTimeout Property
Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Server =
   {
      CacheTimeout = 10
   }
})
...
```

Using the cache will increase the speed of the report viewer. See the chapter Caching for more information.

RequestTimeout Property

Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Server =
   {
      RequestTimeout = 30
   }
})
...
```

CommandTimeout Option

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to set the query timeout for the already created connection and data sources in the report.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
```

```
   {
      GetReport = "GetReport",
      ViewerEvent = "ViewerEvent"
   }
   })
...
```

HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Report.mrt"));
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   return StiNetCoreViewer.GetReportResult(this, report);
}

public IActionResult ViewerEvent()
{
   return StiNetCoreViewer.ViewerEventResult(this);
}
...
```

5.1.21 Viewer Settings

The **HTML5 Viewer** is configured using properties that are located in the **StiNetCoreViewerOptions** class. All properties are divided into groups. Some of the groups contain subgroups for ease of use. The following is an example of setting the properties of the viewer.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Theme = StiViewerTheme.Office2022WhiteTeal,
   Localization = "Localization/en.xml",
   Actions =
   {
      GetReport = "GetReport",
      ViewerEvent = "ViewerEvent"
   },
   Appearance =
   {
      InterfaceType = StiInterfaceType.Auto,
      ScrollbarsMode = true,
      ShowTooltips = false
   },
   Exports =
   {
      DefaultSettings =
      {
```

```
        ExportToPdf =
        {
           CreatorString = "Company Name",
           ImageQuality = 0.75f
        }
     },
     ShowExportToDbf = false,
     ShowExportToDif = false
   }
})
...
```

Please note that all dashboard elements have their own save options and full-screen buttons for preview. There are no special options to control displaying them, but they can be disabled through the properties of the element. The code below should be added after loading the report before passing it to the viewer.

HomeController.cs

```
...
var dbsElementInteraction = (report.GetComponentByName("RegionMap1") as
Stimulsoft.Report.Dashboard.IStiElementInteraction).DashboardInteraction;
(dbsElementInteraction as
Stimulsoft.Report.Dashboard.IStiInteractionLayout).ShowFullScreenButton =
false;
(dbsElementInteraction as
Stimulsoft.Report.Dashboard.IStiInteractionLayout).ShowSaveButton = false;
...
```

Main settings (without groups)

Name	Description
Theme	Sets the viewer theme. The list of available themes can be found in the **StiViewerTheme** enumeration. The default value is **Office2022WhiteBlue**.
Localization	Sets the path to the XML localization file. The path can be absolute or relative. By default, the English localization is used. It is built into the viewer and does not require additional XML files.
Width	Sets the width of the component in the required

	units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. The default width is 100%.
Height	Sets the height of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the automatic height is set depending on the size of the report page, or 650 pixels in the view mode of the viewer with scrollbars.

Actions

Name	Description
GetReport	Specifies the name of the action method for preparing the rendered report. Specifies the name of the action method for preparing the constructed report. If report caching is enabled, this action will be called only once when the report is requested or if the requested report is not found in the server cache.
PrintReport	Specifies the name of the action method of report printing. This is not relevant when viewing dashboards.
ExportReport	Specifies the name of the action method of the export the report to the specified format.
EmailReport	Specifies the name of the action method of sending the report by email. This is not relevant when viewing dashboards.
Interaction	Specifies the name of the action method for the viewer to work with interactive operations, such as using parameters, dynamic sorting, collapsing, and drill-down.

DesignReport	Specifies the name of the action method to go to the specified view by clicking the Design button on the viewer panel.
ViewerEvent	Specifies the name of the action method of basic viewer events and the processing actions of the viewer, such as printing and exporting a report, working with parameters, and interactivity, if these actions are not specified separately. In addition, this action is used to load scripts and styles of the viewer. This action is mandatory.

Server

Name	Description
Controller	Specifies the name of the report controller for the report viewer. If this property is not specified, then the current controller will be used to process requests.
RouteTemplate	Sets the route template that is returned when the report viewer actions are executed. If the property is not set, then the MVC project template will be used instead. If the `UseRelativeUrls` property is set to `true`, the `BasePath` will not be respected for this property.The default value of this property is null.
RequestTimeout	Sets the response timeout from the server in seconds, after which an error will be generated. The default value is 20 seconds. For big reports, it is recommended to increase this value.
CacheTimeout	Sets the time in minutes that the server will store the report since the last action of the viewer. The default value is 20 minutes.

CacheMode	Sets the report caching mode. It can take one of the following values of the **StiServerCacheMode** enumeration: › **None** – caching is disabled, the report will be reloaded each time using the **GetReport** event; › **ObjectCache** – the cache is used as the storage, the report is stored as an object (default value); › **ObjectSession** – the session is used as the storage, the report is stored as an object; › **StringCache** – the server cache is used as the storage, the report is serialized to a packed string; › **StringSession** – the session is used as storage, the report is serialized into a packed string.
CacheItemPriority	Sets the priority of the report stored in the server cache. This property affects the automatic clearing of the server memory in case of lack of memory. The lower the priority is, the greater is the chance of removing information from memory.
AllowAutoUpdateCache	Sets the mode for automatic cache update. The report stored in the cache or the server session will be automatically re-saved after a certain period of time when the viewer is idle (every 3 minutes). By default, the property is set to **true**.
UseRelativeUrls	Sets the viewer mode in which relative URLs are used for AJAX requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value that specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-1** removes the port number.
PassQueryParametersForResource	Enables transferring all request URL parameters

s	when generating links to the resources of the viewer. If **false**, only the necessary parameters are used to request the resources of the viewer. This corresponds to the correct work of the browser cache. By default, the property is set to **true**.
PassQueryParametersToReport	Enables using all the URL parameters of the request as the variable values. The variable names must match the parameters. The default value of the property is false.
PassFormValues	Enables passing the values of the POST form to the client-side, if these values are required to be used in the actions of the viewer. If you enable this property, the additional **GetFormValues()** method will return a collection of form parameters. By default, the property is **false**.
ShowServerErrorPage	Enables displaying an HTML page with the details of the error that occurred on the server-side. When the property is enabled, the details of the error will be displayed in the viewer window. If the property is disabled, only the numeric error code and a short error text in the dialog box will be displayed. By default, the property is set to **true**.
UseCompression	Enables compression of the viewer requests into the GZip stream. Enables compression of the viewer requests into the GZip stream. That allows to decrease the amount of internet traffic but slows down the viewer slightly. The default value of the property is **false**.
UseCacheForResources	Enables caching of the component resources on the server-side. The following resources are supported - scripts, styles, and images. This option improves the load speed of the component and also reduces the server load in multi-client environments. The default value is **true**.

UseLocalizedCache	Sets a value that enables the use of a different cache depending on the selected localization. The default value of the property is **false**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
CustomCss	Sets the path to the CSS file of the viewer's styles. The standard styles of the chosen theme will not be loaded if this property has got a value. The default value of the property is an empty string.
BackgroundColor	Sets the background color of the viewer. By default, it is set to **White**.
PageBorderColor	Sets the border color of the viewer. By default it is set to **Gray**.
RightToLeft	Sets the **Right to Left** mode for viewer controls. By default, the property is set to **false**. By default, the property is set to **false**.
FullScreenMode	Sets the full-screen display mode of the viewer. By default, the property is set to **false**.
ScrollbarsMode	Sets the preview mode with scrollbars. By default, the property is set to **false**.
OpenLinksWindow	Sets the target window for opening links contained in the report. By default, the property is set to **Blank** (new window).
OpenExportedReportWindow	Sets the target window for opening the export file from the viewer. By default, the property is set to **Blank** (new window).
DesignWindow	Sets the destination window for launching the

	report designer. The default value of the property is **Self** (which is the current window).
ShowTooltips	Enables showing tips for the viewer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enables showing links to online documentation for the viewer controls. By default, the property is set to **true**.
ShowDialogsHelp	Sets a value that indicates that showing or hiding the help button in dialogs. By default, the property is set to **true**.
PageAlignment	Sets the position of the report page in the viewer window. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – the page will be aligned left; › **Center** – the page will be centered (default value); › **Right** – the page will be aligned right.
ShowPageShadow	Enables displaying shadow for report pages. By default, the property is set to **true**.
BookmarksPrint	Enables printing of report bookmarks (besides the report itself). By default, the property is set to **false**.
BookmarksTreeWidth	Sets the width of the bookmarks panel in pixels. By default, the width is 180 pixels.
ParametersPanelPosition	Specifies the position of the report parameters panel. It can take one of the following **StiParametersPanelPosition** enumeration values: › **Top** - the panel will be docked to the top margin (default value); › **Left** - the panel will be docked to the left margin.

ParametersPanelMaxHeight	Sets the maximum height of the parameters bar in pixels. By default, the maximum height is 300 pixels.
ParametersPanelColumnsCount	Sets the number of columns to display report parameters. By default, there are 2 columns.
ParametersPanelSortDataItems	Gets or sets a value which indicates that variable items will be sorted. By default, the property is set to **true**.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
InterfaceType	Sets the type of interface used for the viewer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the viewer's interface is determined automatically depending of the device that is report is displayed on. That is the default value. › **Mouse** – the standard interface with a mouse control will be used for all the screen types. › **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability. › **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with the smartphones.
AllowMobileMode	Enables or disables displaying a report or dashboard in the mobile mode. If the option is set to **false**, then the mobile view will not be used. If the option is set to **true**, the mobile

	view mode will be used when opening the viewer on mobile devices. By default, the option is set to **true**.
ChartRenderType	Sets the displaying mode of charts on the report page. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages. It can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN markup.
DatePickerFirstDayOfWeek	Sets the first day of the week for the date picker. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Monday** – the first day of the week is Monday (default value); › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default, the property is set to **false**.

AllowTouchZoom	Sets ability to change the scale of the report page by using the two-fingers gesture (Pinch to Zoom) for the touch-screens. The default value of the property is **true**.
ShowReportIsNotSpecifiedMessage	Sets a value which indicates that 'The report is not specified' message will be shown. The default value of the property is **true**.
PrintToPdfMode	Sets the Print to PDF mode. It has the following values: › **StiPrintToPdfMode.Hidden** - hidden print mode (default value); › **StiPrintToPdfMode.Popup** - the PDF document will be displayed before printing in a pop-up window.
ImagesQuality	Gets or sets the image quality that will be used on the viewer page. It has the following values: › **StiImagesQuality.Low** - low quality, used to speed up loading reports and saves memory; › **StiImagesQuality.Normal** - normal quality, suitable for most cases (default value); › **StiImagesQuality.High** - high quality, used for ultra high-definition displays, but may slow down the loading of pages.
CombineReportPages	Sets a value which indicates that if a report contains several pages, then they will be combined in preview. By default, the property is set to **false**.

Toolbar

Name	Description
Visible	Enables displaying the viewer toolbar. By default, the property is set to **true**.
DisplayMode	Specifies the display mode of the toolbar of the viewer. It can take one of the following values of the **StiToolbarDisplayMode** enumeration:

	› **Simple** - all controls are located on the same control panel (default value); › **Separated** - the control panel is split into top and bottom panels.
BackgroundColor	Specifies the background color of the viewer toolbar. The default color of the selected theme is used.
BorderColor	Specifies the border color of the viewer toolbar. The default color of the selected theme is used.
FontColor	Specifies the text color for the toolbar and the viewer menu. The default color of the selected theme is used.
FontFamily	Specifies the font for the toolbar and the viewer menu. The default font of the selected theme is used.
Alignment	Sets the alignment mode for the controls on the viewer toolbar. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – elements will be aligned left; › **Center** – elements will be centered; › **Right** – elements will be aligned right; › **Default** – the alignment depends on the RightToLeft property (default value).
ShowButtonCaptions	Enables text of the buttons on the toolbar of the viewer. By default, the property is set to **true**.
ShowPrintButton	Enables showing the button - **Print** - on the viewer toolbar. By default, the property is set to **true**.
ShowOpenButton	Enables displaying the **Open** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the

	toolbar of the viewer when viewing reports or dashboards. By default, the property is set to true.
ShowSendEmailButton	Enables showing the button - **Send Email** - on the viewer toolbar. By default, the property is set to **false**. Also, you should add the EmailReport action.
ShowFindButton	Enables showing the button - **Find** - on the viewer toolbar. By default, the property is set to **true**.
ShowBookmarksButton	Enables showing the button - **Bookmarks** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the bookmarks panel will not be displayed even if there are bookmarks in the report.
ShowParametersButton	Enables showing the button - **Parameters** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the parameters panel will not be displayed even if there are parameters in the report.
ShowResourcesButton	Enables showing the button - **Resources** - on the toolbar of the viewer. By default, the property is set to **true**. If the button is hidden, the resources panel will not be displayed even if there are resources in the report.
ShowEditorButton	Enables showing the button - **Editor** - on the viewer toolbar. By default, the property is set to **true**.
ShowFullScreenButton	Enables displaying the **Full Screen** button on the toolbar of the viewer when viewing reports or dashboards. . By default, the property is set to **true**.
ShowFirstPageButton	Enables showing the button - **First Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowPreviousPageButton	Enables showing the button - **Previous Page** -

	on the viewer toolbar. By default, the property is set to **true**.
ShowCurrentPageControl	Enables showing the current report page indicator. By default, the property is set to **true**.
ShowNextPageButton	Enables showing the button - **Next Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowLastPageButton	Enables showing the button - **Last Page** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowZoomButton	Enables showing the button to select the report zoom. By default, the property is set to **true**.
ShowViewModeButton	Enables showing the button to select the view mode of the report page. By default, the property is set to **true**.
ShowDesignButton	Enables displaying the **Design** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to **false**.
ShowAboutButton	Enables showing the button - **About** - on the viewer toolbar. By default, the property is set to **true**.
ShowRefreshButton	Sets a visibility of the **Refresh** button in the toolbar of the viewer. By default, the property is set to **true**.
ShowPinToolbarButton	Enables displaying of the **Pin Toolbar** button on the viewer's toolbar. The button is available only in the Mobile mode of the viewer's interface. The default value of the property is **true**.
PrintDestination	Sets the report printing mode. It can take one of the following values of the **StiPrintDestination** enumeration: › **Default** – a menu with a choice of printing

	modes will be displayed (default value); › **Pdf** – printing will be done in the PDF format; › **Direct** – printing will be done to the HTML format directly to the printer, the system print dialog will be displayed; › **PopupWindow** – printing will be done in the HTML format via the preview window of the report.
ViewMode	Sets the mode for displaying report pages. It can take one of the following **StiWebViewMode** enumeration values: › **SinglePage** - displays one page of the report selected in the toolbar of the viewer (default value); › **Continuous** - displays all pages of the report; › **MultiplePages** - displays all report pages as a table.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. The values are from 10 to 500 percent. You can also set one of the following values: › **StiZoomMode.PageWidth** – when the viewer runs, the zoom, necessary to display the report by the page width, will be set; › **StiZoomMode.PageHeight** – when the viewer runs, the zoom, necessary to display the report by the page height, will be set.
MenuAnimation	Enables animation when the viewer menu shows/hides. By default the property is set to **true**.
ShowMenuMode	Sets the display mode of the viewer menu. It can take one of the following values of the **StiShowMenuMode** enumeration: › **Click** – shows menu by mouse click (default value);

	› **Hover** – shows menu by hovering the mouse cursor.
AutoHide	Enables auto-hiding of the viewer's toolbar. The property will work only for the Mobile mode of the viewer's interface. The default value of the property is **false**.

Export

Name	Description
DefaultSettings	This group of properties provides the ability to specify the default export settings for each export type. These settings will be applied to the export dialogs when the viewer runs or to the report, if export dialogs are disabled.
StoreExportSettings	Enables saving selected settings in the export dialogs. Settings will be stored in browser cookies. By default the property is set to **true**.
ShowExportDialog	Enables showing the export options dialog box. If the property is set to **false**, the export will be done with the default settings. By default the property is set to **true**.
ShowExportToDocument	Enables the export menu item - **Document File**. By default, the property is set to **true**.
ShowExportToPdf	Enables displaying the **Adobe PDF file** export menu item when viewing reports, and the **Adobe PDF** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToXps	Enables the export menu item - **Microsoft XPS File**. By default, the property is set to **false**.
ShowExportToPowerPoint	Enables the export menu item - **Microsoft PowerPoint 2007/2010 File**. By default, the property is set to **true**.
ShowExportToHtml	Enables the export menu item - **HTML File**. By

	default, the property is set to **true**.
ShowExportToHtml5	Enables the export menu item - **HTML5 File**. By default, the property is set to **true**.
ShowExportToMht	Enables the export menu item - **MHT Web Archive**. By default, the property is set to **true**.
ShowExportToText	Enables the export menu item - **Text File**. By default, the property is set to **true**.
ShowExportToRtf	Enables the export menu item - **Rich Text File**. By default, the property is set to **true**.
ShowExportToWord2007	Enables the export menu item - **Microsoft Word 2007/2010 File**. By default, the property is set to **true**.
ShowExportToOpenDocumentWriter	Enables the export menu item - **OpenDocument Writer File**. By default, the property is set to **true**.
ShowExportToExcel	Enables the export menu item - **Microsoft Excel File**. By default, the property is set to **true**.
ShowExportToExcelXml	Enables the export menu item - **Microsoft Excel Xml File**. By default, the property is set to **true**.
ShowExportToExcel2007	Enables displaying the **Microsoft Excel 2007/2010 File** export menu item when viewing reports, and the **Microsoft Excel** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToOpenDocumentCalc	Enables the export menu item - **OpenDocument Calc File**. By default, the property is set to **true**.
ShowExportToCsv	Enables the export menu item - **CSV File**. By default, the property is set to **true**.
ShowExportToDbf	Enables the export menu item - **DBF File**. By default, the property is set to **true**.
ShowExportToXml	Enables the export menu item - **XML File**. By default, the property is set to **true**.

ShowExportToDif	Enables the export menu item - **Data Interchange Format (DIF) File**. By default, the property is set to **true**.
ShowExportToSylk	Enables the export menu item - **Symbolic Link (SYLK) File**. By default, the property is set to **true**.
ShowExportToJson	Enables the export menu item - **JSON File**. By default, the property is set to **true**.
ShowExportToImageBmp	Enables displaying the **BMP Image** export menu item when viewing reports, and the **BMP Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageGif	Enables displaying the **GIF Image** export menu item when viewing reports, and the **GIF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageJpeg	Enables displaying the **JPEG Image** export menu item when viewing reports, and the **JPEG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePcx	Enables displaying the **PCX Image** export menu item when viewing reports, and the **PCX Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePng	Enables displaying the **PNG Image** export menu item when viewing reports, and the **PNG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageTiff	Enables displaying the **TIFF Image** export menu item when viewing reports, and the **TIFF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvg	Enables displaying the **Scalable Vector Graphics (SVG) File** export menu item when viewing reports, and the **Scalable Vector Graphics (SVG) File** item when viewing

	dashboards. By default, the property is set to **true**.
ShowExportToImageSvgz	Enables displaying the **Compressed SVG (SVGZ) File** export menu item when viewing reports, and the **Compressed SVG (SVGZ) File** item when viewing dashboards. By default, the property is set to **true**.
ShowOpenAfterExport	Enables displaying the **Open After Export** parameter in export settings menu. By default, the property is set to **true**.

Email

Name	Description
ShowEmailDialog	Enables displaying settings for sending the report via email. If the dialog box is disabled, the email will be sent with the settings set on the server side in the **EmailReport** action. By default the property is set to **true**.
ShowExportDialog	Enables displaying export options dialog box when sending email. If the property is set to **false**, the export will be done with the default settings. By default the property is set to **true**.
DefaultEmailAddress	Sets the default recipient email, i.e. the address to which the email with the attached report will be sent.
DefaultEmailSubject	Sets the default email subject (header).
DefaultEmailMessage	Sets the default email message (text).

5.2 HTML5 Designer

YouTube

Watch videos for working with .NET Core HTML5 Designer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your

questions and suggestions in the comments to the video.

Samples

See on GitHub examples of working with the .NET Core HTML5 Designer component. All examples are separate projects, grouped into one solution for Visual Studio.

The **HTML5 Designer** (**StiNetCoreDesigner**) component is designed to create reports in the web browser. You do not need to install the .NET Framework, ActiveX components or any special plug-ins on the client side. All that is needed is any modern Web browser.

With help of **HTML5 Designer** you can create, edit, save and preview reports on any computer with any operating system installed. Since the designer only uses HTML and JavaScript technologies, it can be run on devices where there is no Flash or Silverlight support - tablets, smartphones. Also, the designer supports the Touch interface, which is automatically enabled when using devices with a touch screen.

The **HTML5 Designer** component uses the AJAX technology to perform all actions on reports, which allows you to get rid of reloading the entire page, save Web traffic and speed up work. The report engine built using the .NET Core technology is used to render reports. This is a cross-platform technology. It allows you to deploy the application on servers that use the operating systems like Windows, macOS, and Linux.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use the **HTML5 Designer** in a Web project, you need to install the NuGet

package of Stimulsoft.Reports.Web.NetCore:
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Reports.Web.NetCore in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

To add the ability to create and edit dashboards in a Web project, install the NuGet package Stimulsoft.Dashboards.Web.NetCore (this package is associated with the package Stimulsoft.Reports.Web.NetCore. If it is missed it will be installed automatically):
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Dashboards.Web.NetCore in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

5.2.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the designer, you need to place the **StiNetCoreDesigner** component on the page, set the necessary settings to it, and set the necessary actions in the view controller. When the report designer runs, the following actions occur:

- The .NET Core component generates HTML and JavaScript code that is necessary for displaying and running the designer;
- When the component is output, the JavaScript method is launched. It requests the report template on the server side displays it in the designer window;
- Various actions in the designer (for example, report preview, saving the report template, export reports, sorting, drill-down etc.) calls a certain action on the server side, in which you can perform the necessary manipulations with the report.

5.2.2 Activation

YouTube

Watch videos which show how to activate the ASP.NET Core HTML5 Designer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiNetCoreDesigner** component.

HomeController.cs

```
...
//Activation with using license code
public class HomeController : Controller
{
   static HomeController()
   {
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";
   }
}

//Activation with using license file
public class HomeController : Controller
{
   public HomeController(IHostingEnvironment hostEnvironment)
   {
      var path = Path.Combine(hostEnvironment.ContentRootPath, "Content\
      \license.key");
      Stimulsoft.Base.StiLicense.LoadFromFile(path);
```

```
    }
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

5.2.3 Editing Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To edit a report template, you need to add the **StiNetCoreDesigner** component to the page, specify the minimum necessary settings for it, and define the necessary actions in the view controller.

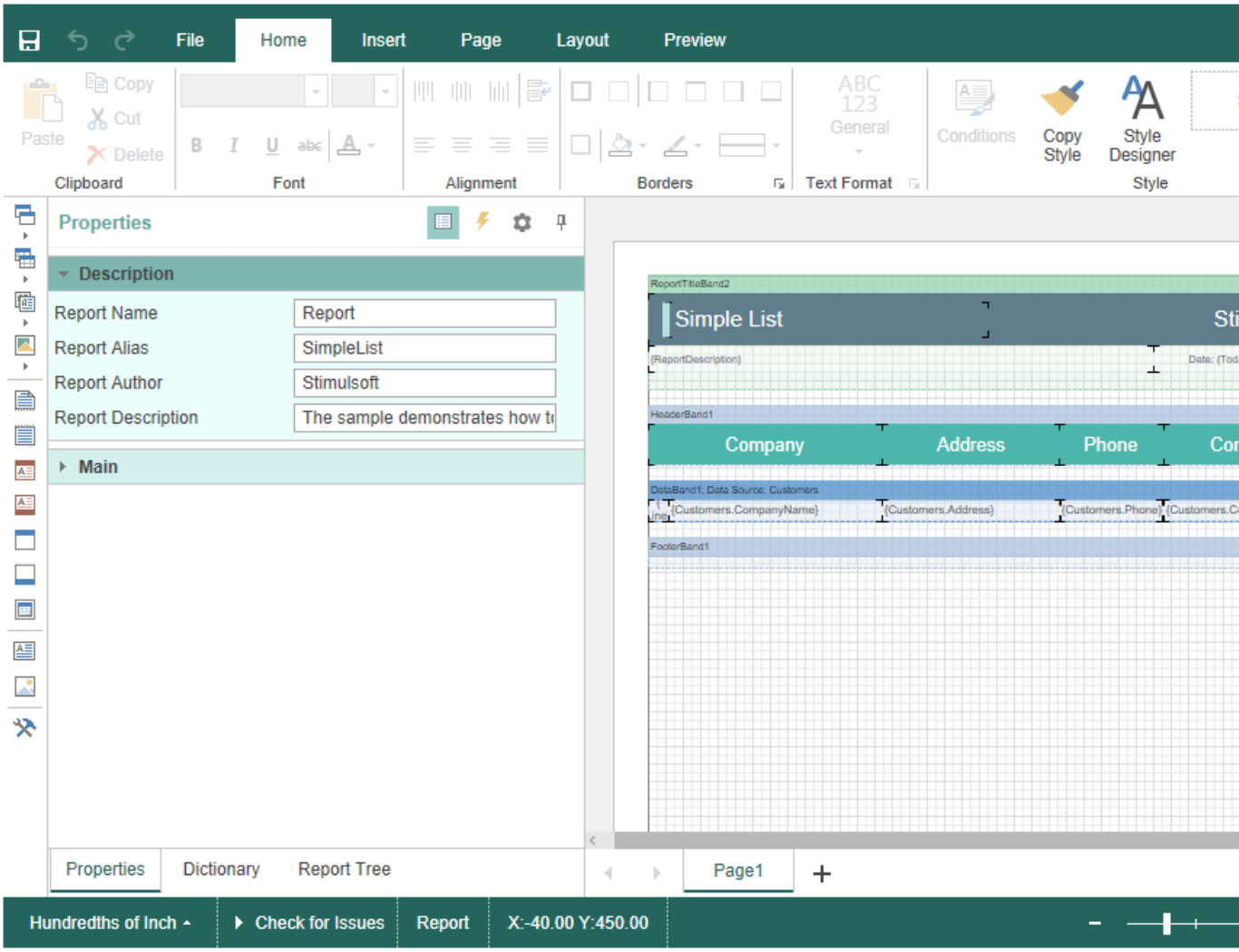
Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      GetReport = "GetReport",
      DesignerEvent = "DesignerEvent"
   }
})
...
```

HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));
   //report.Load(StiNetCoreHelper.MapPath(this, "Reports/Dashboard.mrt"));

   return StiNetCoreDesigner.GetReportResult(this, report);
}

public IActionResult DesignerEvent()
```

```
{
   return StiNetCoreDesigner.DesignerEventResult(this);
}
...
```



The **GetReport** action is used to load an editable report template. It is called automatically after the report designer is loaded. The **DesignerEvent** action is designed to process various additional designer actions, such as working with data and components, previewing reports and others.

Information

The **DesignerEvent** action is mandatory. Without it, the correct work of the

designer is impossible.

5.2.4 Creating New Reports and New Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the report designer with a new (empty) report, it is enough to create a new report in the **GetReport** action and return it to the designer. If necessary, you can load data for the report, or perform any other necessary actions.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      GetReport = "GetReport"
   }
})
...
```

HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();

   return StiNetCoreDesigner.GetReportResult(this, report);
}
...
```

You can also create a new report using the main menu of the designer. The **CreateReport** action is used to load data for a new report or perform any other necessary actions. This action will be called when creating a new empty report or when creating a report using the wizard.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      CreateReport = "CreateReport"
   }
})
...
```

HomeController.cs

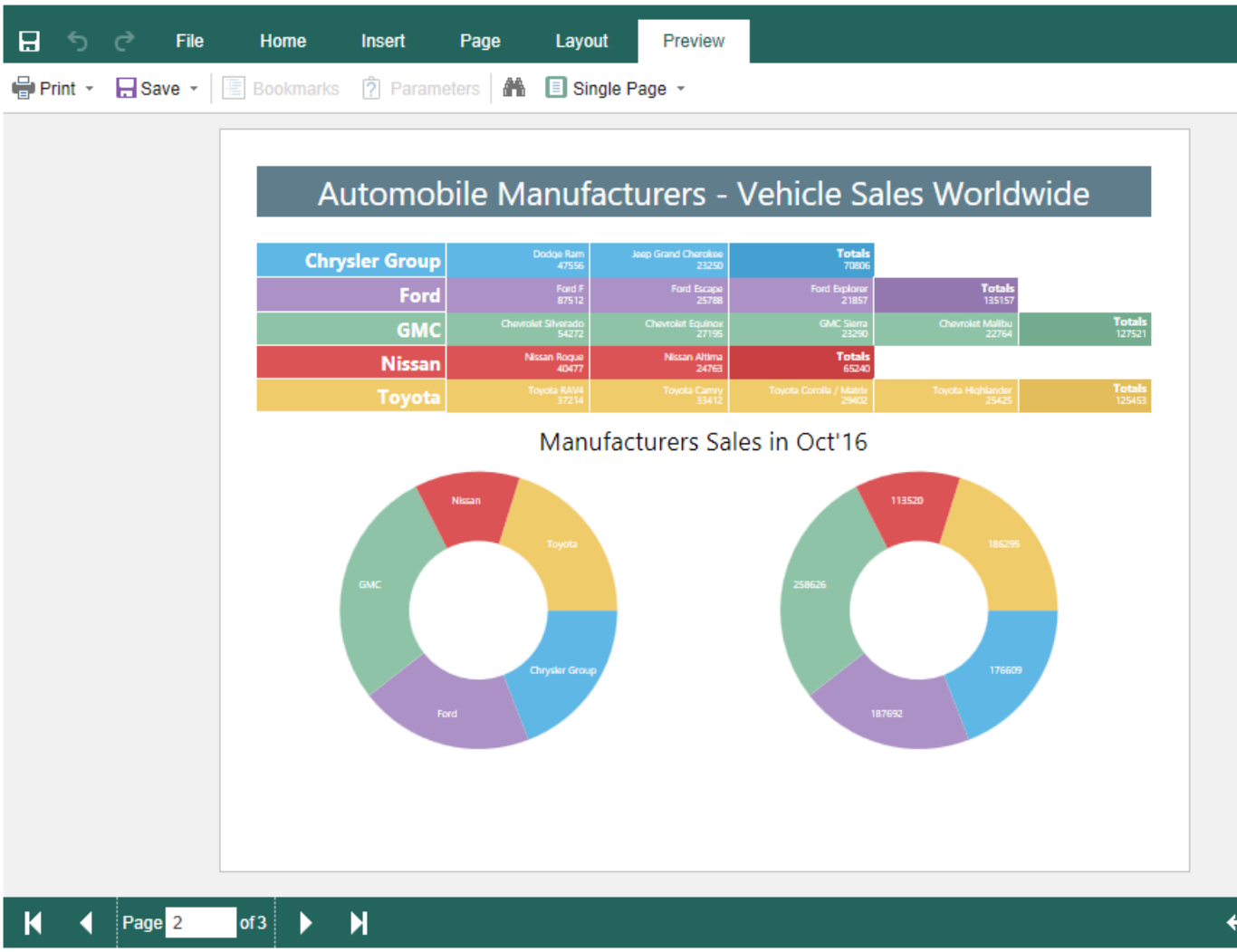
```
...
public IActionResult CreateReport()
{
   StiReport report = new StiReport();
   //var newDashboard = StiReport.CreateNewDashboard();

   // Register data for the new report, if necessary
   DataSet data = new DataSet("Demo");
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));
   report.RegData(data);
   //newDashboard.RegData(data);
   report.Dictionary.Synchronize();
   //newDashboard.Dictionary.Synchronize();

   return StiNetCoreDesigner.GetReportResult(this, report);
   //return StiNetCoreDesigner.GetReportResult(this, newDashboard);
}
...
```

5.2.5 Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component provides the ability to preview reports. To preview the report, just go to the appropriate tab in the designer window. The report template will be transferred to the server side, rendered and displayed in the embedded viewer.



Before previewing the report, it is possible to perform any necessary actions, for example, connect data for the report. To do this, you can use the special **PreviewReport** action that will be called before previewing the report. The **PreviewReport** action is called before preparing and rendering a report for viewing till its saving to the cash.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      PreviewReport = "PreviewReport"
   }
})
...
```

HomeController.cs

```
...
public IActionResult PreviewReport()
{
   StiReport report = StiNetCoreDesigner.GetActionReportObject(this);

   DataSet data = new DataSet("Demo");
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));
   report.RegData(data);

   return StiNetCoreDesigner.PreviewReportResult(this, report);
}
...
```

If you need to make actions on your report immediately before displaying the report, you can use the **GetPreviewReport** action, which is called after the request of the prepared report from the cash.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      GetPreviewReport = "GetPreviewReport"
   }
})
...
```

HomeController.cs

```
...
public IActionResult GetPreviewReport()
{
   StiReport report = StiNetCoreDesigner.GetActionReportObject(this);

   DataSet data = new DataSet("Demo");
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));
   report.RegData(data);
   //report.IsRendered = false;

   return StiNetCoreDesigner.PreviewReportResult(this, report);
}
...
```

Information

So as in this event a prepared report for viewing is transferred, if you need to render again you should set the **report.IsRendered = false** flag.

5.2.6 Additional Features of Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The preview window of the **HTML5 Designer** component has a fully functional interactive **HTML5 Viewer** that can print and export reports, supports working with report parameters, dynamic sorting, interactive reports, collapsing and etc. To use these features, you do not need any additional settings for the report designer.

In any of the above actions, you can work with the report template, for example, change its properties and parameters, connect new data for rendering.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      ExportReport = "ExportReport"
   }
})
...
```

HomeController.cs

```
...
public IActionResult ExportReport()
{
   StiReport report = StiNetCoreDesigner.GetActionReportObject(this);
   // ...

   return StiNetCoreDesigner.ExportReportResult(this, report);
}
...
```

Information

If you do not need any of these additional options to preview the report (for example, exporting or printing a report), you can disable them using the appropriate properties of the **HTML5 Designer** component.

5.2.7 Saving Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component provides two ways of saving the report which are available in the main menu and in the main panel of the designer - **Save Report** and **Save As**. In turn, each of these ways has its own modes and settings.

Saving a report and dashboard on the server side

To save the editable report on the server side, you need to set the **SaveReport** action, which will be called when you select **Save** in the main menu, or click the **Save** button on the main panel of the designer.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      SaveReport = "SaveReport"
   }
})
...
```

HomeController.cs

```
...
public IActionResult SaveReport()
{
   StiReport report = StiNetCoreDesigner.GetReportObject(this);
```

```
    // Save the report template
    // ...

    return StiNetCoreDesigner.SaveReportResult(this);
}
...
```

This action returns a response to the client side of the designer about the result of saving the report. After saving the report, it is possible to display a dialog box with an error or a text message.

HomeController.cs

```
...
public IActionResult SaveReport()
{
    StiReport report = StiNetCoreDesigner.GetReportObject(this);

    // Save the report template
    // ...

    // Completion of the report saving with message dialog box
    return StiNetCoreDesigner.SaveReportResult(this, "Some message after
    saving");
    //return Content("{"infoMessage":"Some info message after saving"}");
    //return Content("{\"warningMessage\":\"Some info message after saving
    \"}");
}
...
```

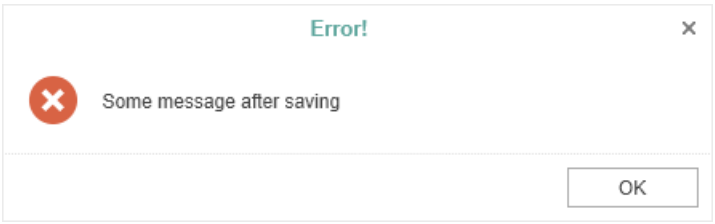
You can get a report name from the designer save dialog or an original report name.

HomeController.cs

```
...
public IActionResult SaveReport()
{
    var requestParams = StiNetCoreDesigner.GetRequestParams();
    var report = StiNetCoreDesigner.GetReportObject();

    //Report name from designer save dialog
    var savingReportName = requestParams.Designer.FileName;

    //Original report name from properties
    var originalReportName = report.ReportName;

    return StiNetCoreDesigner.SaveReportResult(this, "Some message after
    saving");
}
```

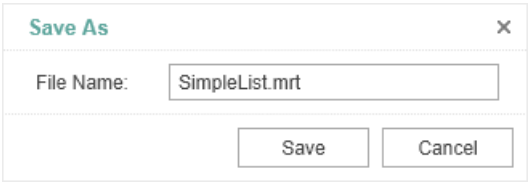
```
...
```

In this case, the dialog with the specified text will be displayed. The text can contain both an error message of saving or a warning, or any other message.



Saving reports and dashboards on the client side

To save the edited report on the client side as a file, no additional designer settings are required. It is enough to click the **Save As** main menu item. The dialog box will be displayed. In this dialog you can change the name of the report file. The file will be saved to the local disk of the computer.



The **HTML5 Designer** component provides the ability to change the behavior of the specified save option. For this purpose, the special **SaveReportAs** action is used in the designer. If you use this event, the report will be saved on the server side. Work of this event will be similar to the **SaveReport** action.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      SaveReportAs = "SaveReportAs"
   }
})
...
```

HomeController.cs

```
...
public IActionResult SaveReportAs()
{
   StiReport report = StiNetCoreDesigner.GetReportObject(this);

   // Save the report template
   // ...

   return StiNetCoreDesigner.SaveReportResult(this);
}
...
```

Use the following code to get the report name from the Save dialog.

HomeController.cs

```
public IActionResult SaveReport()
{
   StiReport report = StiNetCoreDesigner.GetReportObject(this);
   var requestParams = StiNetCoreDesigner.GetRequestParams(this);
   var reportName = requestParams.Designer.FileName;

   return StiNetCoreDesigner.SaveReportResult(this);
}
```

Saving settings

The report is saved in the background mode without reloading the page in the web browser window. If you need to visually control the process of saving the report, you should change the value of the **SaveReportMode** (or **SaveReportAsMode**) property of the designer to one of the three specified values - **Hidden** (default value), **Visible** or **NewWindow**.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      SaveReportAs = "SaveReportAs"
   },
   Behavior =
   {
      SaveReportAsMode = StiSaveMode.Visible
```

```
    }
})
...
```

If the **SaveReportMode** property is set to **Visible**, the report save action will be called in the current browser window in the normal (visible) mode using the POST request. If the **SaveReportMode** property is set to **NewWindow**, the report save event will be called in a new window of the web browser. By default, this property is set to **Hidden** - the report save event is called in the background using the AJAX request and is not displayed in the browser window. The same values and behavior are applicable to the **SaveReportAsMode** property.

5.2.8 Localization

The **HTML5 Designer** component supports the complete localization of its interface. Use the special **Localization** property to localize the report designer interface. As a value of this property, you should specify the path to the localization XML file (relative or absolute).

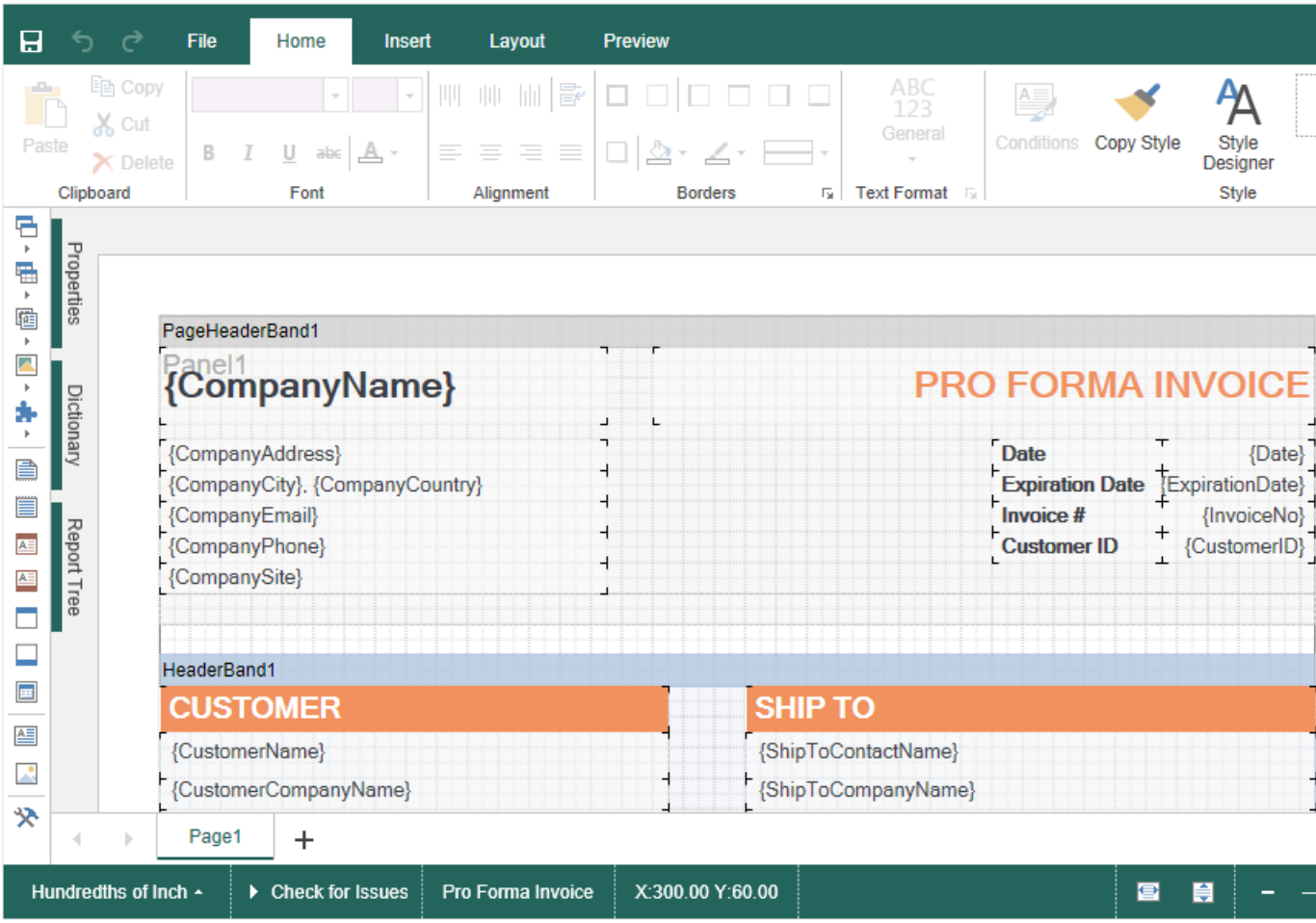
Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
    Localization = "Localization/en.xml"
})
...
```

The interface of the report designer allows you to select the necessary localization from an accessible list. To do this, set value for the **LocalizationDirectory** property as the folder in which the localization XML files are stored.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
    Localization = "Localization/en.xml",
    LocalizationDirectory = "Localization"
})
...
```



Information

If the value for the **Localization** property is set, then when you run the report designer, the localization specified in this property will always be applied. If the property value is not set, the localization selected from the list of available localizations in the report designer panel will be automatically loaded.

5.2.9 Using Themes

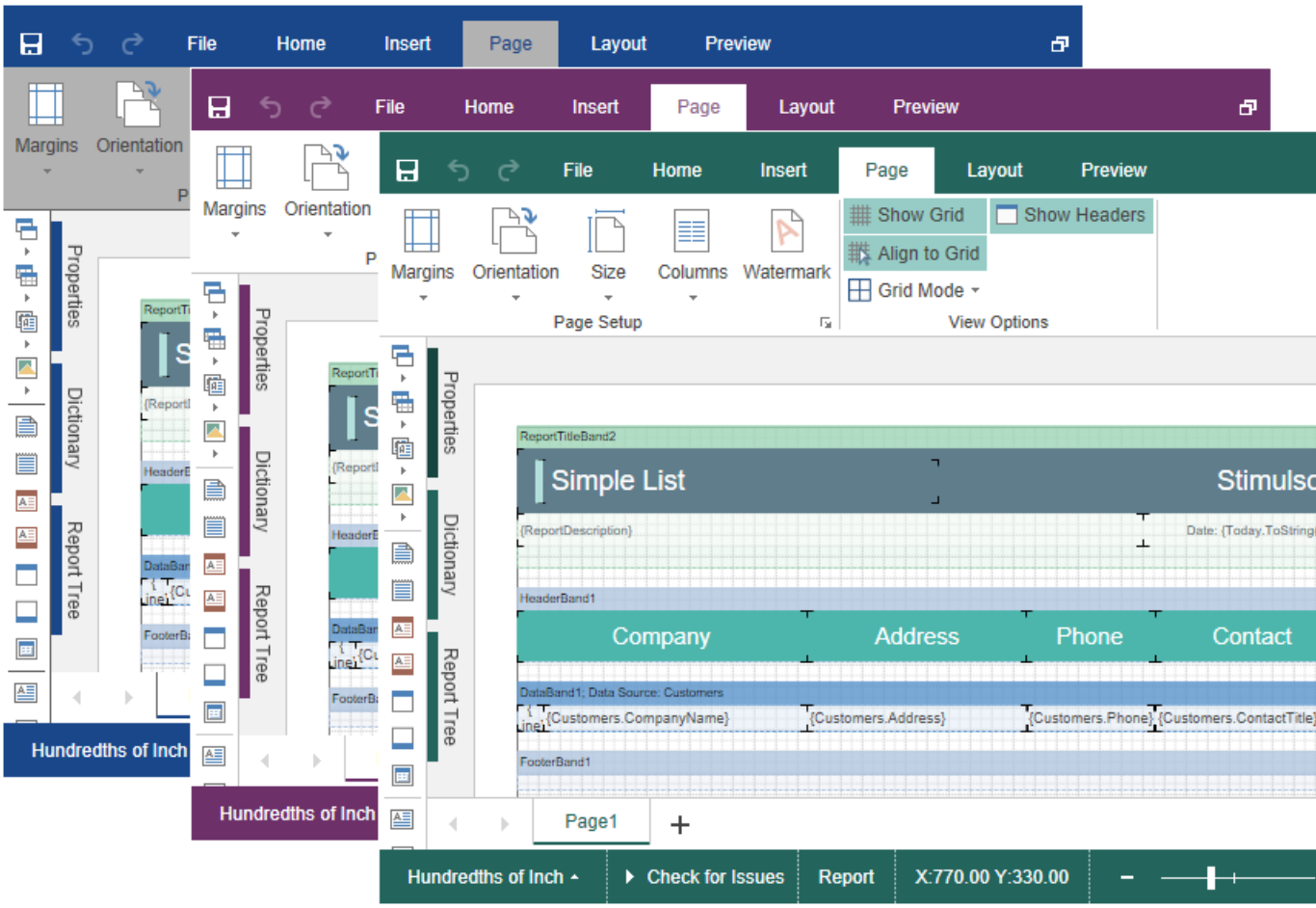
In the **HTML5 Designer** component you can change the appearance of visual controls. To change the theme, you should use the **Theme** property.

Default.aspx

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Theme = StiDesignerTheme.Office2022WhiteTeal
```

```
})
...
```

There are currently **2 themes** available with different color accents. As a result, **more than 50** variants of the appearance are available. This allows you to customize the appearance of the designer for almost any design of the Web project.



5.2.10 Caching

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

HTM5 Designer uses the server cache to store the editable report template. It is necessary because the client part of the designer contains only a visual representation of components of a report template. The report object itself with all the parameters and properties is stored on the server side.

To use caching, you need to connect modules to work with a session or cache on the server side. To do this, just add the following services to the project in the start file of a project.

Startup.cs

```
...
public void ConfigureServices(IServiceCollection services)
{
   services.AddMemoryCache();
   services.AddSession();
   services.AddMvc();
}
...
```

You can manage caching with the following properties.

The CacheMode property

This property of the designer enables caching and sets its type. It can take one of the following values, specified in the **StiServerCacheMode** enumeration:

- **None** – caching is disabled;
- **ObjectCache** – for caching, the server cache is used. The report object is saved in this (set by default);
- **StringCache** – for caching, the server cache is used. The report is saved as a packed string in this cache;
- **ObjectSession** – the current session, in which the report object is saved, is used for caching;
- **StringSession** - for caching, the current session is used, in which the report is saved as a packed string.

The CacheItemPriority property

This property sets the priority of the report stored in the cache of the server. It affects the automatic clearing of the server memory in case of memory shortage. The lower the priority is, the greater is the chance of removing information from memory.

The CacheTimeout property

This property specifies the amount of time in minutes you want to store the report in the server cache. If, when using caching, the requested report is not found in the cache (time of storing this report expired), then it will be requested again using the special **GetReport** action. In this case the unsaved changes may be lost.

The **HTML5 Designer** component provides the ability to specify its own methods for working with report caching. For this purpose, a special **StiCacheHelper** class is used. It contains methods for obtaining a report from the cache and saving the report to the cache. It is necessary to create a new class inherited from **StiCacheHelper** and reload the above methods which respectively have the names - **GetObject, SaveObject** and **RemoveObject**..

HomeController.cs

```
...
public class DesignerController : Controller
{
   public class StiMyCacheHelper : StiCacheHelper
   {
      public override object GetObject(string guid)
      {
         string path =
         System.IO.Path.Combine(this.HttpContext.Server.MapPath("CacheFiles"
         ), guid);
         if (System.IO.File.Exists(path))
         {
            byte[] cacheData = System.IO.File.ReadAllBytes(path);
            return StiCacheHelper.GetObjectFromCacheData(cacheData);
         }
         return null;

         //return base.GetObject(guid);
      }

      public override void SaveObject(object obj, string guid)
      {
         byte[] cacheData = StiCacheHelper.GetCacheDataFromObject(obj);
```

```
            string path =
            System.IO.Path.Combine(this.HttpContext.Server.MapPath("CacheFiles"
            ), guid);
            System.IO.File.WriteAllBytes(path, cacheData);

            //base.SaveObject(obj, guid);
        }

        public override void RemoveObject(string guid)
        {
            string path =
            System.IO.Path.Combine(this.HttpContext.Server.MapPath("CacheFiles"
            ), guid);
            if (File.Exists(path))
            {
                File.Delete(path);
            }
        }
    }

    static DesignerController()
    {
        StiNetCoreDesigner.CacheHelper = new StiMyCacheHelper();
    }
}
...
```

To initialize the work with report caching using the created class, it is enough to set it as the value of the **StiNetCoreDesigner.CacheHelper** static property in the constructor of the controller.

5.2.11 Additional Methods

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

For **HTML5 Designer**, there are several additional methods that are used to get the object of the currently edited report, parameters of the current state of the designer and other useful data. These methods can be used in any actions of the designer.

The GetReportObject() Method

Returns the report object with which the designer is currently working. It is possible to perform the necessary actions with it - register new data sets, change report properties, assign parameters or load another report to the object. Then, the report can be returned to the designer, specifying it as a parameter in the resulting action method.

HomeController.cs

```
...
public IActionResult ExportReport()
{
   StiReport report = StiNetCoreDesigner.GetReportObject(this);
   report.ReportName = "MyReportName";

   return StiNetCoreDesigner.ExportReportResult(this, report);
}
...
```

The GetActionReportObject() method

Returns the report object that will be used for the particular action. For example, for the **OpenReport** action, this method returns a report loaded from the local disk of the computer. For the **PreviewReport** action, the method returns a prepared copy of the report for preview.

HomeController.cs

```
...
public IActionResult OpenReport()
{
   StiReport report = StiNetCoreDesigner.GetActionReportObject(this);

   // Register data for the opened report, if necessary
   DataSet data = new DataSet("Demo");
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));
   report.RegData(data);
   report.Dictionary.Synchronize();

   return StiNetCoreDesigner.GetReportResult(this, report);
}
...
```

The GetRouteValues() method

Returns values for URLs with which the designer page was opened. Thus, it is

possible to get the initial collection of page parameters to run the designer and use these values for any checks and conditions.

HomeController.cs

```
...
public IActionResult ExportReport()
{
   RouteValueDictionary routeValues =
   StiNetCoreDesigner.GetRouteValues(this);

   return StiNetCoreDesigner.ExportReportResult(this);
}
...
```

You can also get values of URL parameters by parameter name, specifying it as the parameter of the called action of the designer.

HomeController.cs

```
...
public IActionResult ExportReport(string id)
{
   return StiNetCoreDesigner.ExportReportResult(this);
}
...
```

The GetFormValues() method

Returns the values of the form that initiated (opened by the POST request) a page of the designer. Thus, it is possible to get a collection of form parameters in any action of the designer.

Index.cshtml.cs

```
...
public IActionResult DesignerInteraction()
{
   NameValueCollection formValues = StiNetCoreDesigner.GetFormValues(this);

   return StiNetCoreDesigner.InteractionResult(this);
}
...
```

By default, this feature is disabled to optimize requests of the client-side of the

designer to the server. To enable it, set the **PassFormValues** property to **true**.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Server =
   {
      PassFormValues = true
   }
})
...
```

The GetRequestParams() method

Returns all parameters of the current state of the designer passed to the server side. They can be useful for determining the type of action that the designer is currently executing - for example, to determine the type of export, as well as all action parameters.

HomeController.cs

```
...
public IActionResult ExportReport()
{
   StiRequestParams requestParams =
   StiNetCoreDesigner.GetRequestParams(this);
   if (requestParams.ExportFormat == StiExportFormat.Pdf)
   {
      StiReport report = StiNetCoreDesigner.GetReportObject(this);

      // Some action with report for the PDF export
      // ...

      return StiNetCoreDesigner.ExportReportResult(this, report);
   }

   return StiNetCoreDesigner.ExportReportResult(this);
}
...
```

The GetExportSettings() method

Returns all parameters of the current report export. The type of the parameter object will correspond to the type of export selected in the report preview menu. Any export parameters can be changed and passed to the input of the resulting method.

In this case, the report will be exported with the parameters transferred.

HomeController.cs

```
...
public IActionResult ExportReport()
{
   StiExportSettings settings = StiNetCoreDesigner.GetExportSettings(this);
   if (settings.GetExportFormat() == StiExportFormat.Pdf)
   {
      StiPdfExportSettings pdfSettings = (StiPdfExportSettings)settings;
      pdfSettings.EmbeddedFonts = true;
      pdfSettings.AllowEditable = StiPdfAllowEditable.No;
      return StiNetCoreDesigner.ExportReportResult(this, settings);
   }

   return StiNetCoreDesigner.ExportReportResult(this);
}
...
```

The MapPath() and MapWebRootPath() methods

Returns the absolute path, respectively, to the application or wwwroot directory. You can use this to upload report templates files, data files, etc. These methods are located in the **StiNetCoreHelper** static class.

HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));

   return StiNetCoreDesigner.GetReportResult(this, report);
}
...
```

5.2.12 Timeout

When working with the **StiNetCoreDesigner** component, you can set the timeout for various operations — storing the report in the cache, server response, and query execution. The timeout setting is done using the component properties and report options.

CacheTimeout Property

Sets the time in minutes that the server will store the rendered report since the last

action of the viewer. The default setting is 10 minutes.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Server =
   {
      CacheTimeout = 10
   }
})
...
```

Using cache will increase the speed of the report designer. See the chapter Caching for more information

RequestTimeout Property

Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Server =
   {
      RequestTimeout = 30
   }
})
...
```

CommandTimeout Option

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to set the query timeout for the already created connection, and data sources in the report.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      GetReport = "GetReport",
      DesignerEvent = "DesignerEvent"
   }
})
...
```

HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Report.mrt"));
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   return StiNetCoreDesigner.GetReportResult(this, report);
}

public IActionResult DesignerEvent()
{
   return StiNetCoreDesigner.DesignerEventResult(this);
}
...
```

5.2.13 Add custom functions

Information

See on GitHub example of adding a custom function in the ASP.NET MVC HTML5 Designer component.

You can add a custom function to the Dictionary in the report designer when you integrate it into your application. After adding the custom function, you can use this in creating reports and dashboards. Below is the example of adding a function for calculating the sum total.

DesignerController.cs

```
...
public static decimal MySum(object value)
{
   if (!ListExt.IsList(value))
   return Stimulsoft.Base.Helpers.StiValueHelper.TryToDecimal(value);
```

```
    return Stimulsoft.Data.Functions.Funcs.SkipNulls(ListExt.ToList(value))
    .TryCastToDecimal()
    .Sum();
}
...
static DesignerController()
{
    StiFunctions.AddFunction("MyCategory", "MySum",
    "description", typeof(DesignerController),
    typeof(decimal), "Calculates a sum of the specified set of values.",
    new[] { typeof(object) },
    new[] { "values" },
    new[] { "A set of values" }).UseFullPath = false;
}
...
```

5.2.14 Settings

The **HTML5 Designer** configuration is done using properties that are located in the **StiNetCoreDesignerOptions** class. All properties are divided into groups, some of the groups contain subgroups of properties for ease of use. Below is an example of setting some properties of the designer.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
    Theme = Stimulsoft.Report.Web.StiDesignerTheme.Office2022WhiteTeal,
    Localization = "Localization/en.xml",
    Actions =
    {
        GetReport = "GetReport",
        PreviewReport = "PreviewReport",
        SaveReport = "SaveReport",
        DesignerEvent = "DesignerEvent"
    },
    Appearance =
    {
        InterfaceType = StiInterfaceType.Auto,
        ShowTooltipsHelp = false,
        ShowDialogsHelp = false,
        DefaultUnit = Stimulsoft.Report.StiReportUnitType.Centimeters
    },
    Dictionary =
    {
        PermissionBusinessObjects =
        Stimulsoft.Report.Web.StiDesignerPermissions.None,
        PermissionDataConnections =
        Stimulsoft.Report.Web.StiDesignerPermissions.View
    },
    Bands =
```

```
    {
        ShowChildBand = false,
        ShowEmptyBand = false,
        ShowOverlayBand = false
    }
})
...
```

Basic settings (without groups)

Name	Description
Theme	Specifies the theme of the report designer. The list of available themes is located in the **StiDesignerTheme** enumeration. The default value is **Office2022WhiteBlue**.
Localization	Specifies the path to the XML localization file. The path can be absolute or relative. By default, English localization is used. It is built into the designer and does not require additional XML files.
LocalizationDirectory	Specifies the path to the directory with XML localization files. The localization files located in the specified folder will be loaded to the localization list in the designer panel.
Width	Sets the width of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the component is expanded to the entire area of the browser window.
Height	Sets the height of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the component is expanded to the entire area of the browser window.

Actions

Name	Description
GetReport	Specifies the name of the action method to prepare the report template when loading the designer.
OpenReport	Specifies the name of the action method to open the report template from the designer menu.
CreateReport	Specifies the name of the action method to prepare the report template when creating the new report in the designer.
SaveReport	Specifies the name of the action method to save the report template on the server side..
SaveReportAs	Specifies the name of the action method to store the report template on the server side when using the **Save As** menu item. If no action is specified, the built-in method of saving the report template to the local disk will be used.
PreviewReport	Specifies the name of the action method to prepare the rendered report in the preview window.
GetPreviewReport	Specifies the name of the action method just before a report is displayed in the preview window.
ExportReport	Specifies the name of the action method to export reports to the specified format.
Exit	Specifies the name of the action method to go to the desired view by clicking the Exit button in the main menu of the report designer.
DesignerEvent	Specifies the name of the action method of the report designer to handle additional designer actions, such as working with data, report

	components, and others. Also, this action is used to load scripts and designer styles.

Server

Name	Description
Controller	Sets the name of the controller to process requests. If this property is not set, then the current controller will be used to process the requests.
RouteTemplate	Sets the route template that is returned when the report designer actions are executed. If the property is not set, then the MVC project template will be used instead. The default value of the property is null. If the `UseRelativeUrls` property is set to `true`, the `BasePath` will not be respected for this property.The default value of this property is null.
ShowServerErrorPage	Enables the display of an HTML page with error details that occurred on the server side. The error details will be displayed in the preview window if the property is enabled. If the property is disabled, the numeric error code and a short error description will be displayed in the dialog window. By default, the property is set to **true**.
AllowAutoUpdateCookies	Allows the designer to update the cookies automatically on every request to the server. By default, cookies are set when creating the designer, if they are not specified in the report. By default, the property is set to **false**.
AllowAntiforgeryToken	Allow the designer to automatically request and send the antiforgery token. By default, the property is set to **true**.
RequestTimeout	Sets the time to wait for a response from the server in seconds, after which an error will be

	generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.
CacheTimeout	Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.
CacheMode	Sets the report caching mode. It can take one of the following values of the **StiServerCacheMode** enumeration: › **None** – caching is disabled in **HTML5 Designer**; › **ObjectCache** – the cache is used as the storage, the report is stored as an object (default value); › **ObjectSession** – the session is used as the storage, the report is stored as an object; › **StringCache** – the server cache is used as the storage, the report is serialized to a packed string; › **StringSession** – the session is used as a repository, the report is serialized into a packed string.
CacheItemPriority	Sets the priority of the report stored in the server cache. This property affects the automatic clearing of the server memory in case of lack of memory. The lower the priority is, the greater is the chance of removing information from memory.
AllowAutoUpdateCache	Sets the mode for automatic cache update. The report stored in the cache or server session will be automatically re-saved after a certain period of time if the designer is idle (about every 3 minutes). By default, the property is set to **true**.
UseRelativeUrls	Sets the designer mode in which relative URLs are used for requests to the server. By default, the property is set to **true**.

PortNumber	Gets or sets a value which specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-1** removes the port number.
PassQueryParametersForResources	Enables transferring all request URL parameters when generating links to the resources of the designer. If **false**, only the necessary parameters are used to request the resources of the designer. This corresponds to the more correct operation of the browser cache. By default, the property is set to **true**.
PassFormValues	Enables transferring the POST form values to the client side, if these values are to be used in the actions of the designer. If you enable this feature, the additional **GetFormValues()** method will return a collection of form parameters. By default, the property is **false**.
UseCompression	Enables compression of designer requests in the GZip stream. This allows you to reduce the amount of Internet traffic, but slows down the designer. By default, the property is **false**.
UseCacheForResources	Enables caching of the component resources on the server side. The following resources are supported: scripts, styles and images. This option improves the load speed of the component and also reduces the server load in multi-client environments. The default value is **true**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description

CustomCss	Specifies the path to the CSS file of styles for the report designer. If this property is set, the standard styles of the selected theme will not be loaded. The default value is an empty value.
DefaultUnit	Sets the units for the size of the report and all its components. By default, centimeters are used.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. It can take one of the following values of the **StiZoomMode** enumeration: › **PageWidth** – when the designer runs, the zoom, necessary to display the report by the page width, will be set; › **PageHeight** – when the designer runs, the zoom, required to display the page height of the report, will be set.
ShowAnimation	Enables animation for various elements of the designer interface. By default, the property is set to **true**.
ShowOpenDialog	Allows to display the open dialog, or to open with the open event. By default, the property is set to **true**.
ShowTooltips	Enables displaying tooltips for designer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enable displaying links to online documentation in tooltips for designer controls. By default, the property is set to **true**.
ShowDialogsHelp	Enables displaying links to online documentation on the titles of dialog forms of the designer. By default, the property is set to **true**.
InterfaceType	Sets the type of interface used for the designer. It can take one of the following

	StiInterfaceType enumeration values: › **Auto** – the interface type of the designer will be selected automatically depending on the device used (default value); › **Mouse** – forced use of the interface to control the designer with the mouse; › **Touch** – forced use of the Touch interface to control the designer via the touch screen (mobile devices), also in this mode, the interface elements are increased.
DatePickerFirstDayOfWeek	Sets the first day of the week for the select date item. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Auto** – automatic detection of the first day of the week from the browser settings (default value); › **Monday** – the first day of the week is Monday; › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default, the property is set to **false**.
FormatForDateControls	This feature allows you to customize the format for date controls. By default, the current option does not have a specified value, and the date format is determined based on the browser's locale.
ShowReportTree	Enables displaying the tree of report components. By default, the property is set to **true**.
ChartRenderType	Gets or sets the type of the chart in the preview. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images;

	› **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages in the preview tab. Can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN markup.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
CloseDesignerWithoutAsking	Sets a value which indicates that the designer will be closed without asking. By default, the property is set to **true**.
ShowSystemFonts	Sets a visibility of the system fonts in the fonts list. By default, the property is set to **true**.
WizardTypeRunningAfterLoad	Calls the Report wizard after starting the report designer. It may have one of the following **StiWizardType** enumeration values: › **None** - runs the report designer without running the report wizard; › **StandardReport** - runs the Standard wizard; › **MasterDetailReport** - runs the Master-Detail

	wizard; › **LabelReport** - runs the Label wizard; › **InvoicesReport** - runs the Invoice wizard; › **OrdersReport** - runs the Order wizard; › **QuotationReport** - runs the Quote wizard.
AllowWordWrapTextEditors	Allows word wrap in the text editors. By default, the property is set to **true**.

Behavior

Name	Description
ShowSaveDialog	Enables displaying the dialog to insert a report name when it is saved. The name of the report will be transferred in the parameters of the report designer. By default, the property is set to **true**.
UndoMaxLevel	Sets the maximum number to cancel actions with the report (the Undo/Redo function). A big value of this property will consume memory on the server side to store the undo parameters. The default value is **6**.
AllowChangeWindowTitle	Allows using the title of the browser window to display the file name of the edited report. By default, the property is set to **true**.
SaveReportMode	Sets the mode to save the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - saving of the report is called in the current web browser window in the visible mode using the POST request; › **NewWindow** - saving of the report is called in a new window (tab) of the web browser.

SaveReportAsMode	Sets the mode for saving the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - saving of the report is called in the current web browser window in the visible mode using the POST request; › **NewWindow** - saving of the report is called in a new window (tab) of the web browser.
CheckReportBeforePreview	Sets the value that allows running the report checker before preview.

FileMenu

Name	Description
Visible	Enables displaying the main menu of the report designer. By default, the property is set to **true**.
ShowNew	Enables showing the main menu item - **New**. By default, the property is set to **true**.
ShowFileMenuNewReport	Sets a visibility of the new report button in the designer. By default, the property is set to **true**.
ShowFileMenuNewDashboard	Sets a visibility of the new dashboard button in the designer. By default, the property is set to **true**.
ShowOpen	Enables showing the main menu item - **Open**. By default, the property is set to **true**.
ShowSave	Enables showing the main menu item - **Save**. By default, the property is set to **true**.
ShowSaveAs	Enables showing the main menu item - **Save As**. By default, the property is set to **true**.
ShowClose	Enables showing the main menu item - **Close**.

	By default, the property is set to **true**.
ShowExit	Enables showing the main menu item - **Exit**. By default, the property is set to **false**.
ShowReportSetup	Enables showing the main menu item - **Report Setup**. By default, the property is set to **true**.
ShowOptions	Enables showing the main menu item - **Options**. By default, the property is set to **true**.
ShowInfo	Enables showing the main menu item - **Info**. By default, the property is set to **true**.
ShowAbout	Enables showing the main menu item - **About**. By default, the property is set to **true**.
ShowHelp	Enables showing the main menu item - **Help**. By default, the property is set to **true**.

Dictionary

Name	Description
Visible	Enables showing the data dictionary of the report. By default, the property is set to **true**.
UseAliases	Allows you to use aliases in the data dictionary. It has the three values of the **StiUseAliases** enumeration: › **Auto** - defines the mode of using aliases from a saved value in cookies (default value); › **True** - sets the mode of using aliases in the data dictionary; › **False** - disables the mode of using aliases in the data dictionary.
NewReportDictionary	It allows you to create a new data dictionary or join the existing one when creating a new report in the designer. It has the three values of the **StiNewReportDictionary** enumeration:

	› **Auto** - defines the mode to create or join the data dictionary from a saved value in cookies (default value); › **DictionaryNew** - sets the mode to create a new data dictionary when creating a new report; › **DictionaryMerge** - sets the mode to join the existing data dictionary with a new one when creating a new report in the designer.
ShowDictionaryContextMenuProperties	Sets a visibility of the **Properties** item in the dictionary context menu. By default, the property is set to **true**.
ShowDictionaryActions	Sets a visibility of the **Actions** menu on the dictionary toolbar. By default, the property is set to **true**.
PermissionDataConnections	Sets the available actions to connect data to the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataSources	Sets available actions on report data sources. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataColumns	Sets the available actions on data columns in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionBusinessObjects	Sets the available actions on the business objects in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataRelations	Sets the available actions to linking data in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionVariables	Sets available actions on report variables. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionResources	Sets the available actions for the resources in the Report Dictionary. Takes one or several values from the **StiDesignerPermissions**

	enumeration.
PermissionSqlParameters	Sets the available actions for the parameters of the SQL queries for the Report DataSources. Takes one or several values from **StiDesignerPermissions** enumeration.
DataTransformationsPermissions	Sets the available actions on data transformation. It can take one or more values from the **StiDesignerPermissions** enumeration.

The table below shows all available values for the **StiDesignerPermissions** enumeration, which can be set for the dictionary elements of the report.

Value	Description
None	Disables any action on the item of the data dictionary.
All	Allows any action on the item of the data dictionary.
Create	Allows creating a specific data dictionary item.
Delete	Allows deleting a specific data dictionary item.
Modify	Allows modifying a specific data dictionary item.
View	Allows viewing a specific data dictionary item.
ModifyView	Allows modifying and viewing a specific data dictionary item.

Toolbar

Name	Description
ShowToolbar	Enables displaying the toolbar of the designer. By default, the property is set to **true**.
ShowSetupToolboxButton	Enables displaying the button to call the dialog box with settings for the side toolbar. By default, the property is set to **true**.

ShowInsertButton	Enables displaying the **Insert** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowLayoutButton	Enables displaying the tab **Layout** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowPageButton	Enables displaying the tab **Page** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowPreviewButton	Enables displaying the tab **Preview** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the designer. By default, the property is set to **true**.
ShowAboutButton	Enables displaying the **About** on the toolbar of the designer. By default, the property is set to **false**.

PropertiesGrid

Name	Description
Visible	Enables displaying the property panel. By default, the property is set to **true**.
Width	Sets the width of the property panel. By default, the width is set to **370 px**.
LabelWidth	Specifies the width of the labels on the properties panel. By default, the width is set to **160 px**.
PropertiesGridPosition	Sets **Left** or **Right** position of the properties grid in the designer. It has the three values of the **StiPropertiesGridPosition** enumeration: › **Left**; › **Right**.

ShowPropertiesWhichUsedFromStyles	Sets a visibility of the properties which used from styles in the designer. By default, the property is set to **false**.

Components

Name	Description
ShowText	Enables displaying the **Text** component in the insert menu for report components. By default, the property is set to **true**.
ShowTextInCells	Enables displaying the **Text in Cells** component in the insert menu for report components. By default, the property is set to **true**.
ShowRichText	Enables displaying the **Rich Text** component in the insert menu for report components. By default, the property is set to **true**.
ShowImage	Enables displaying the **Image** component in the insert menu for report components. By default, the property is set to **true**.
ShowBarCode	Enables displaying the **Bar Code** component in the insert menu for report components. By default, the property is set to **true**.
ShowShape	Enables displaying the **Shape** component in the insert menu for report components. By default, the property is set to **true**.
ShowHorizontalLinePrimitive	Enables displaying the **Horizontal Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowVerticalLinePrimitive	Enables displaying the **Vertical Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowRectanglePrimitive	Enables displaying the **Rectangle** component in the insert menu for report components. By

	default, the property is set to **true**.
ShowRoundedRectanglePrimitive	Enables displaying the **Rounded Rectangle** component in the insert menu for report components. By default, the property is set to **true**.
ShowPanel	Enables displaying the **Panel** component in the insert menu for report components. By default, the property is set to **true**.
ShowClone	Enables displaying the **Clone** component in the insert menu for report components. By default, the property is set to **true**.
ShowCheckBox	Enables displaying the **Check Box** component in the insert menu for report components. By default, the property is set to **true**.
ShowSubReport	Enables displaying the **Sub Report** component in the insert menu for report components. By default, the property is set to **true**.
ShowZipCode	Enables displaying the **Zip Code** component in the insert menu for report components. By default, the property is set to **true**.
ShowTable	Enables displaying the **Table** component in the insert menu for report components. By default, the property is set to **true**.
ShowCrossTab	Enables displaying the **Cross-Tab** component in the insert menu for report components. By default, the property is set to **true**.
ShowChart	Enables displaying the **Chart** component in the insert menu for report components. It affects on all chart types. By default, the property is set to **true**.
ShowMap	Enables displaying the **Map** component in the insert menu for report components. By default, the property is set to **false**.
ShowGauge	Enables displaying the **Gauge** component in the insert menu for report components. By default,

	the property is set to **false**.
ShowSparkline	Enables displaying the **Sparkline** component in the insert menu for report components. By default, the property is set to **true**.
ShowMathFormula	Enables displaying the **Math Formula** component in the insert menu for report components. By default, the property is set to **true**.
ShowElectronicSignature	Enables displaying the **Electronic Signature** component in the insert menu for report components. By default, the property is set to **true**.
ShowPdfDigitalSignature	Enables displaying the **PDF Digital Signature** component in the insert menu for report components. By default, the property is set to **true**.

Bands

Name	Description
ShowReportTitleBands	Enables displaying the **Report Title** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowReportSummaryBand	Enables displaying the **Report Summary** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageHeaderBand	Enables displaying the **Page Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageFooterBand	Enables displaying the **Page Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowGroupHeaderBand	Enables displaying the **Group Header** item in the **Bands** menu of the designer. By default, the

	property is set to **true**.
ShowGroupFooterBand	Enables displaying the **Group Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowHeaderBand	Enables displaying the **Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowFooterBand	Enables displaying the **Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowColumnHeaderBand	Enables displaying the **Column Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowColumnFooterBand	Enables displaying the **Column Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowDataBand	Enables displaying the **Data** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowHierarchicalBand	Enables displaying the **Hierarchical** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowChildBand	Enables displaying the **Child** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowEmptyBand	Enables displaying the **Empty** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowOverlayBand	Enables displaying the **Overlay** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowTableOfContents	Enables displaying the **Table of Contents** item in the **Bands** menu of the designer. By default, the property is set to **true**.

DashboardElements

Name	Description
ShowTableElement	Enables displaying the **Table** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowCardsElement	Enables displaying the **Cards** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowChartElement	Enables displaying the **Chart** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowGaugeElement	Enables displaying the **Gauge** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowPivotTableElement	Enables displaying the **Pivot** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowIndicatorElement	Enables displaying the **Indicator** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowProgressElement	Enables displaying the **Progress** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowRegionMapElement	Enables displaying the **Region Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowOnlineMapElement	Enables displaying the **Online Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowImageElement	Enables displaying the **Image** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTextElement	Enables displaying the **Text** element in the

	Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowPanelElement	Enables displaying the **Panel** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowShapeElement	Enables displaying the **Shape** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowButtonElement	Enables displaying the **Button** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowListBoxElement	Enables displaying the **List Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowComboBoxElement	Enables displaying the **Combo Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTreeViewElement	Enables displaying the **Tree View** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTreeViewBoxElement	Enables displaying the **Tree View Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowDatePickerElement	Enables displaying the **Date Picker** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.

CrossBands

Name	Description
ShowCrossGroupHeaderBand	Enables displaying the **Cross Group Header** section in the insert menu for report components. By default, the property is set to **true**.

ShowCrossGroupFooterBand	Enables displaying the **Cross Group Footer** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossHeaderBand	Enables displaying the **Cross Header** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossFooterBand	Enables displaying the **Cross Footer** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossDataBand	Enables displaying the **Cross Data** section in the insert menu for report components. By default, the property is set to **true**.

Dashboards

Name	Description
ShowNewDashboardButton	Sets a visibility of the **New Dashboard** button in the designer. By default, the property is set to **true**.

Pages

Name	Description
ShowNewPageButton	Sets a visibility of the **New Page** button in the designer. By default, the property is set to **true**.

When designing a report or dashboard in the report designer, you can also define **ExportOptions**, **EmailOptions**, and **PreviewToolbarOptions** on the **Preview** tab. These options are similar to the report viewer options.

6 Reports and Dashboards for ASP.NET Core Razor

.NET Core is a cross-platform technology for creating Web applications for Windows, Linux, and macOS. **ASP.NET Core Razor Pages** is a technology that is similar to the Model-View-Controller (MVC) system. **Razor Pages** allow you to create pages with code that can process various requests. This technology makes the creation of scripts for pages easier and more efficient in comparison with the use of the MVC solution. Stimulsoft company provides tools for creating, displaying, and transforming reports and dashboards using this technology.

Tools for creating and editing reports:	Tools for viewing and converting reports:	Tools for creating and editing dashboards:	Tools for viewing and converting dashboards:
HTML5 designer	HTML5 viewer	HTML5 designer	HTML5 viewer

6.1 HTML5 Viewer

Samples

See on GitHub examples for working with the ASP.NET Core Razor HTML5 Viewer component. All examples are separate projects, grouped into one solution for Visual Studio.

The **HTML5 Viewer** (**StiNetCoreViewer**) component is designed for viewing reports in the web browser. You do not need to install the .NET Framework, ActiveX components, or any special plug-ins on the client-side. All that is needed is any modern Web browser.

With the help of **HTML5 Viewer**, you can view, print, and export reports on any computer with any operating system installed. Since the viewer only uses HTML and JavaScript technologies, it can be run on devices with no Flash or Silverlight support

- tablets, smartphones. Also, the viewer supports the Touch interface, which is automatically enabled when using devices with a touch screen.

The **HTML5 Viewer** component uses the AJAX technology to perform all actions (uploading a report, paging, scaling, interactivity in reports, etc.), which allows you to get rid of reloading the entire page save Web traffic, and speed up work. The report engine built using the .NET Core technology is used to render reports. This is a cross-platform technology. It allows you to deploy the application on servers that use operating systems like Windows, macOS, and Linux.

The **HTML5 Viewer** supports many themes, animated interface, bookmarks, interactive reports, editing of report elements on the page, full-screen mode, search panel, and other necessary features for viewing reports.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use the **HTML5 Viewer** in a Web project, you need to install the NuGet package of Stimulsoft.Reports.Web.NetCore:
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Reports.Web.NetCore in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

To add the ability to view and export dashboards in a Web project, install the NuGet package Stimulsoft.Dashboards.Web.NetCore (this package is associated with the package Stimulsoft.Reports.Web.NetCore. If it is missed, it will be installed automatically):
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Dashboards.Web.NetCore in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

6.1.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the viewer, you should place the **StiNetCoreViewer** component on a page, set the necessary properties, and define necessary actions in the page event handler. When running the report viewer, the following actions occur:

› The .NET Core component generates HTML and JavaScript code that is necessary for displaying and running the viewer;

› When the component is output, the JavaScript method is launched. It requests the first page of the report on the server-side or the entire report (depending on the selected mode) and the required report parameters;

› Each action in the viewer (for example, paging, printing or exporting a report, etc.) calls a certain action on the server-side, in which you can perform the necessary

manipulations with the report;

› The viewer saves the report in cache or server session to speed up the work, which makes it impossible to re-build the report.

6.1.2 Activation

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiNetCoreViewer** component.

Index.cshtml.cs

```
...
//Activation with using license code
public class IndexModel : PageModel
{
   static IndexModel()
   {
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";
   }
}

//Activation with using license file
public class IndexModel : PageModel
{
   public IndexModel(IWebHostEnvironment webHostEnvironment)
   {
      var path = Path.Combine(webHostEnvironment.ContentRootPath, "Content\
      \license.key");
      Stimulsoft.Base.StiLicense.LoadFromFile(path);
   }
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

6.1.3 Showing Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `Report.Render()` method if you want to perform specific actions with the rendered report before it is displayed in the viewer. Likewise, when using the compilation mode, you need to call the `Report.Compile()` method only if you have actions to perform with the compiled report before it is built and shown in the viewer.

To display a report, you should add the **StiNetCoreViewer** component to a page, set the minimum required properties, and define necessary actions in the page event handler.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      GetReport = "GetReport",
      ViewerEvent = "ViewerEvent"
   }
})
...
```

Index.cshtml.cs

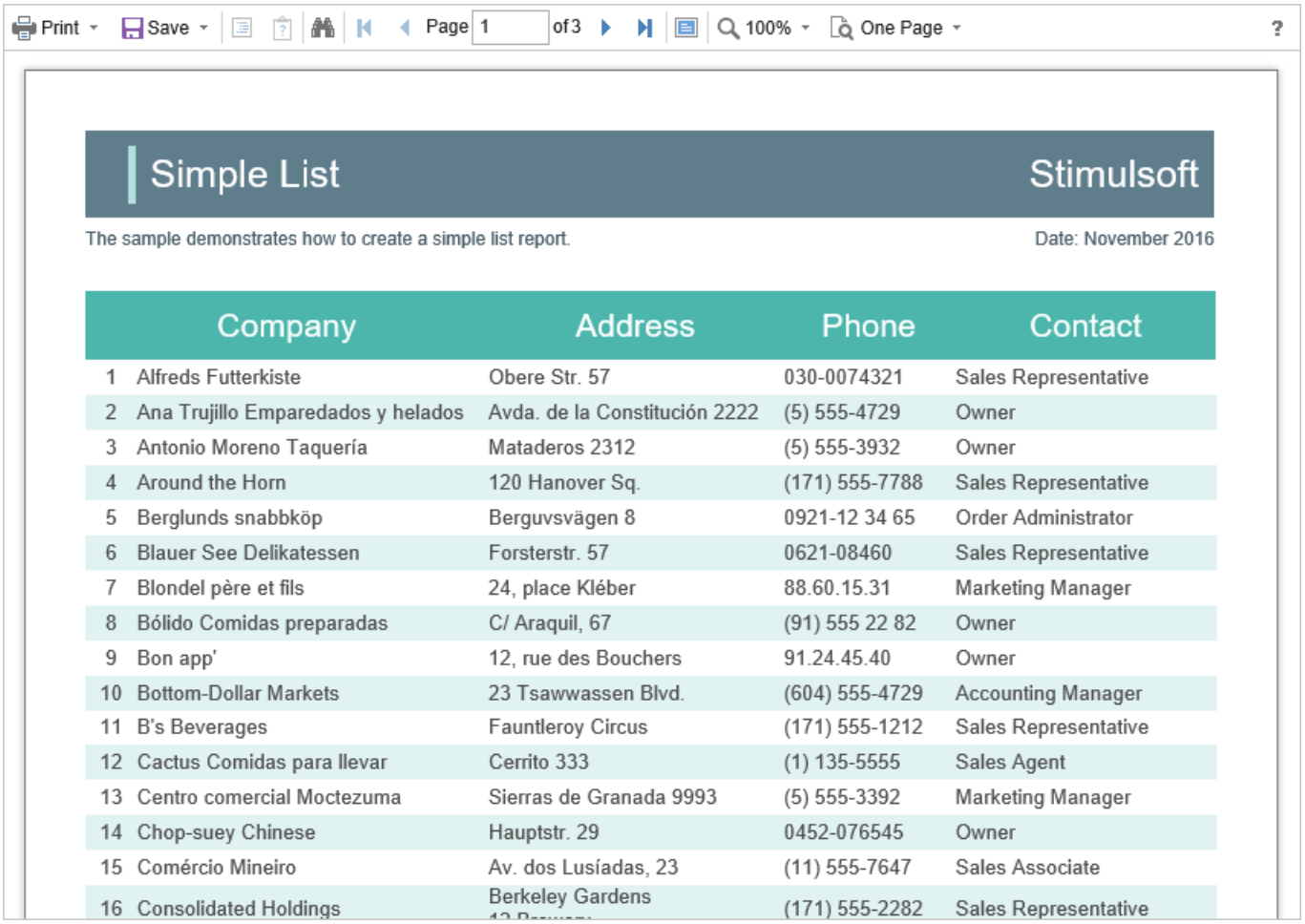
```
...
public IActionResult OnPostGetReport()
{
   // Create the report object
   StiReport report = new StiReport();

   // Load report or dashboard
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));
   //report.Load(StiNetCoreHelper.MapPath(this, "Reports/Dashboard.mrt"));

   return StiNetCoreViewer.GetReportResult(this, report);
}

public IActionResult OnGetViewerEvent()
{
   return StiNetCoreViewer.ViewerEventResult(this);
}
```

```
public IActionResult OnPostViewerEvent()
{
   return StiNetCoreViewer.ViewerEventResult(this);
}
...
```

In the above example, the processing of two actions of the viewer is added. The **GetReport** action returns the report prepared for preview. The **ViewerEvent** action handles the viewer events.

Information

The **ViewerEvent** action is mandatory. Without it, the viewer can't work correctly. The action is called for two types of requests: **OnGet** – the component requests the resources necessary for work, such as CSS styles, JS scripts and images; **OnPost** – all other actions of the viewer.



	Company	Address	Phone	Contact
1	Alfreds Futterkiste	Obere Str. 57	030-0074321	Sales Representative
2	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	(5) 555-4729	Owner
3	Antonio Moreno Taquería	Mataderos 2312	(5) 555-3932	Owner
4	Around the Horn	120 Hanover Sq.	(171) 555-7788	Sales Representative
5	Berglunds snabbköp	Berguvsvägen 8	0921-12 34 65	Order Administrator
6	Blauer See Delikatessen	Forsterstr. 57	0621-08460	Sales Representative
7	Blondel père et fils	24, place Kléber	88.60.15.31	Marketing Manager
8	Bólido Comidas preparadas	C/ Araquil, 67	(91) 555 22 82	Owner
9	Bon app'	12, rue des Bouchers	91.24.45.40	Owner
10	Bottom-Dollar Markets	23 Tsawwassen Blvd.	(604) 555-4729	Accounting Manager
11	B's Beverages	Fauntleroy Circus	(171) 555-1212	Sales Representative
12	Cactus Comidas para llevar	Cerrito 333	(1) 135-5555	Sales Agent
13	Centro comercial Moctezuma	Sierras de Granada 9993	(5) 555-3392	Marketing Manager
14	Chop-suey Chinese	Hauptstr. 29	0452-076545	Owner
15	Comércio Mineiro	Av. dos Lusíadas, 23	(11) 555-7647	Sales Associate
16	Consolidated Holdings	Berkeley Gardens	(171) 555-2282	Sales Representative

If the report was not rendered before showing, the **HTML5 Viewer** component will automatically render it. So you can use report templates and rendered reports to display reports.

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
   StiReport report = new StiReport();
   report.LoadDocument(StiNetCoreHelper.MapPath(this, "Reports/
   SimpleList.mdc"));

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Since the dashboard is not a static document and requires data to work, the format of the rendered MDC document is not available for it. Instead, it is possible to use a snapshot of the report in the MRT format, which contains all the data necessary for the dashboard to work and be correctly displayed in the viewer.

Index.cshtml.cs

```
...
public ActionResult OnPostGetReport()
{
   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath("~/Reports/Snapshot.mrt"));

   return StiNetCoreViewer.GetReportResult(report);
}
...
```

Loading custom fonts

You can connect custom fonts using the **StiFontCollection** class, having specified a file that contains a font. To do this, you should invoke the static method in the page constructor to load the font.

Index.cshtml.cs

```
...
public class ViewerController : Controller
{
   static ViewerController()
   {
      Stimulsoft.Base.StiFontCollection.AddFontFile(StiNetCoreHelper.MapPat
      h(this, "/fonts/my-font/font-name.ttf"));
   }
}
...
```

6.1.4 Connecting Data

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Data to a report can be connected in various ways. The easiest way is to store

connection settings in the report template. You can also connect the data from the code. This can be done when the report is loaded in the **GetReport** action.

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
   DataSet ds = new DataSet();
   ds.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/
   TwoSimpleLists.mrt"));
   report.Dictionary.Databases.Clear();
   report.RegData("Demo", ds);

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Data for the report can be connected not only when the report is loaded. For example, you can connect new data at the moment of interactive actions in the viewer (applying report parameters, sorting, drill-down, collapsing). To do this, you should set the **Interaction** action for the **HTML5 Viewer** component and, in the action handler, connect the data for the current report. In the same way, you can connect data in other actions of the viewer.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      GetReport = "GetReport",
      ViewerEvent = "ViewerEvent",
      Interaction = "ViewerInteraction"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostViewerInteraction()
{
   DataSet data = new DataSet();
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));
```

```
   StiReport report = StiNetCoreViewer.GetReportObject(this);
   report.RegData("Demo", data);

   return StiNetCoreViewer.InteractionResult(this, report);
}
...
```

If you want to connect new data only for a certain interactive action of the viewer, for example, only when you apply report parameters you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class. They are passed to any server-side on any request and contain all necessary information and states of the client part of the viewer. To determine the type of action of the viewer, it is enough to check the Action property of the viewer parameters.

Index.cshtml.cs

```
...
public IActionResult OnPostViewerInteraction()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   if (requestParams.Action == StiAction.Variables)
   {
      DataSet data = new DataSet();
      data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));

      StiReport report = StiNetCoreViewer.GetReportObject(this);
      report.RegData("Demo", data);

      return StiNetCoreViewer.InteractionResult(this, report);
   }

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

SQL data sources

The connection parameters to the SQL data source and any other ones can be stored in the report template. Suppose you want to set the connection parameters from the code before rendering the report (for example, for security reasons or depending on the authorized user). In that case, you can use the example below.

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
   OracleConnection connection = new OracleConnection("Data
   Source=Oracle8i;Integrated Security=yes");
   connection.Open();
   OracleDataAdapter adapter = new OracleDataAdapter();
   adapter.SelectCommand = new OracleCommand("SELECT * FROM Products",
   connection);

   DataSet dataSet = new DataSet("productsDataSet");
   adapter.Fill(dataSet, "Products");

   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/
   SqlSampleReport.mrt"));
   report.RegData("Products", dataSet);

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Also, for SQL data sources used in the report, you can specify the Query Timeout in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to change the connection string for MS SQL, adjust the query, set the query timeout for the already created connection and data sources in the report.

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{

   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath("Report.mrt"));
   ((StiSqlDatabase)
   report.Dictionary.Databases["Connection"]).ConnectionString = @"Data
   Source=server;Integrated Security=True;Initial Catalog=DataBase";
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).SqlCommand = "select *
   from Table where Column = 100";
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Information

For SQL data sources of other types, the connection is created similarly, and an adapter corresponding to the data source type is connected. For example, for the MS SQL data source, you should connect SqlDataAdapter. For OracleDataAdapter is required for Oracle. Also, you should specify a connection string that matches the connection type.

You can also use data for designing reports and dashboards obtained from OData storage. In this case, you can do the authorization using a username, user password, or using a token. Authorization parameters are specified in the connection string to the OData storage using the ";" separator.

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{

   var report = new StiReport();

   //Authorization using a user account
   var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
   services.odata.org/V4/Northwind/
   Northwind.svc;AddressBearer=adress;UserName=UserName;Password=Password;C
   lient_Id=Your Client ID", false, null);

   //Authorization using a user token
   var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
   services.odata.org/V4/Northwind/Northwind.svc;Token=Enter your token",
   false, null);

   report.Dictionary.Databases.Add(oDataDatabase);
   oDataDatabase.Synchronize(report);

   //Query with data filter
   ((StiSqlSource)report.Dictionary.DataSources["Products"]).SqlCommand =
   "Products?$filter=ProductID eq 2";

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

The table below shows the connection string templates for different types of data

sources.

Data Source	Connection String Template
MS SQL	Integrated Security=False; Data Source=myServerAddress;Initial Catalog=myDataBase; User ID=myUsername; Password=myPassword;
MySQL	Server=myServerAddress; Database=myDataBase;UserId=myUsername; Pwd=myPassword;
ODBC	Driver={SQL Server}; Server=myServerAddress;Database=myDataBase; Uid=myUsername; Pwd=myPassword;
OLE DB	Provider=SQLOLEDB.1; Integrated Security=SSPI;Persist Security Info=False; Initial Catalog=myDataBase;Data Source=myServerAddress
Oracle	Data Source=TORCL;User Id=myUsername;Password=myPassword;
MS Access	Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Password=pass;Data Source=C:\ \myAccessFile.accdb;
PostgreSQL	Server=myServerAddress; Port=5432; Database=myDataBase;User Id=myUsername; Password=myPassword;
Firebird	User=SYSDBA; Password=masterkey; Database=SampleDatabase.fdb;DataSource=myServerAddress; Port=3050; Dialect=3; Charset=NONE;Role=; Connection lifetime=15; Pooling=true; MinPoolSize=0;MaxPoolSize=50; Packet Size=8192; ServerType=0;
SQL CE	Data Source=c:\MyData.sdf; Persist Security Info=False;
SQLite	Data Source=c:\mydb.db; Version=3;
DB2	Server=myAddress:myPortNumber;Database=m

	yDataBase;UID=myUsername;PWD=myPassword;Max Pool Size=100;Min Pool Size=10;
Infomix	Database=myDataBase;Host=192.168.10.10;Server=db_engine_tcp;Service=1492;Protocol=onsoctcp;UID=myUsername;Password=myPassword;
Sybase	Data Source=myASEserver;Port=5000;Database=myDataBase;Uid=myUsername;Pwd=myPassword;
Teradata	Data Source=myServerAddress;User ID=myUsername;Password=myPassword;
VistaDB	Data Source=D:\folder \myVistaDatabaseFile.vdb4;Open Mode=ExclusiveReadWrite;
Universal(dotConnect)	Provider=Oracle;direct=true;data source=192.168.0.1;port=1521;sid=sid;user=user;password=pass
MongoDB	mongodb://<user>:<password>@localhost/test
OData	http://services.odata.org/v3/odata/OData.svc/
Other...	The table shows the most commonly used templates for the connection string. You can view various connection string options at the special website.

Data from XML, JSON, Excel files

Connecting to XML and JSON data sources can be stored in the report template. If you want to specify data files from the code, you can use the example below.

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
   DataSet data = new DataSet();
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));
```

```
    report.RegData(data);

    return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
    DataSet data
    = StiJsonToDataSetConverterV2.GetDataSetFromFile(StiNetCoreHelper.MapPat
    h(this, "Data/Demo.json"));

    StiReport report = new StiReport();
    report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));
    report.RegData(data);

    return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

Information

The viewer has the possibility of obtaining data from an Excel file. To do this, you can use the following method.

```
DataSet dataSet = StiExcelConnector.Get().GetDataSet(new StiExcelOptions(ar
```

6.1.5 Localization

The **HTML5 Viewer** component supports the complete localization of its interface. To localize the report viewer interface, use the special **Localization** property. The value of this property should specify the path to the localization XML file (relative or absolute).

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
    Localization = "Localization/en.xml"
})
...
```

When you load the report viewer, the localization file will be loaded automatically.

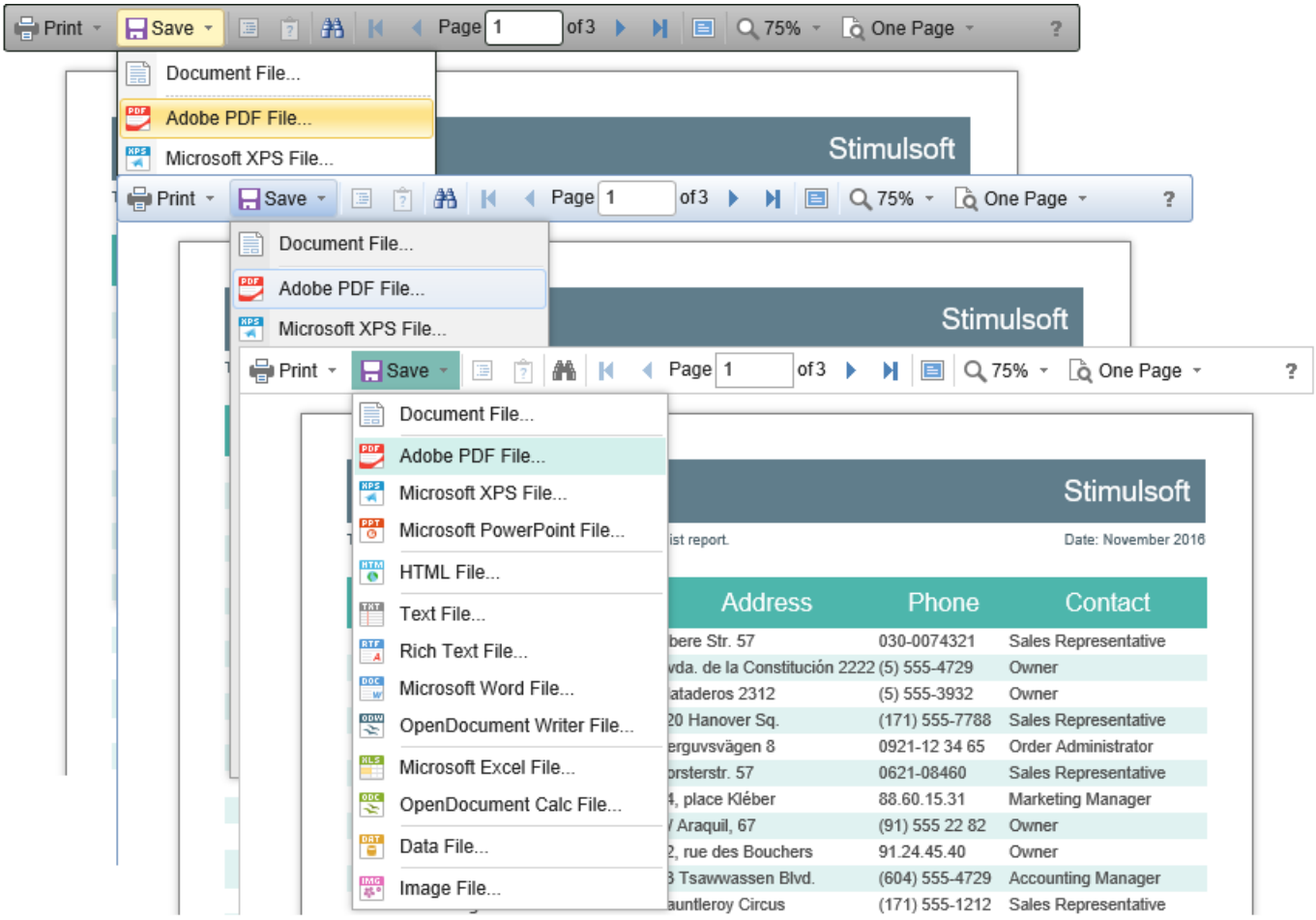
6.1.6 Using Themes

The **HTML5 Viewer** component can change the appearance of visual controls. To change the theme, use the **Theme** property, which can take one of the values of the **StiViewerTheme** enumeration.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Theme = StiViewerTheme.Office2022WhiteTeal
})
...
```

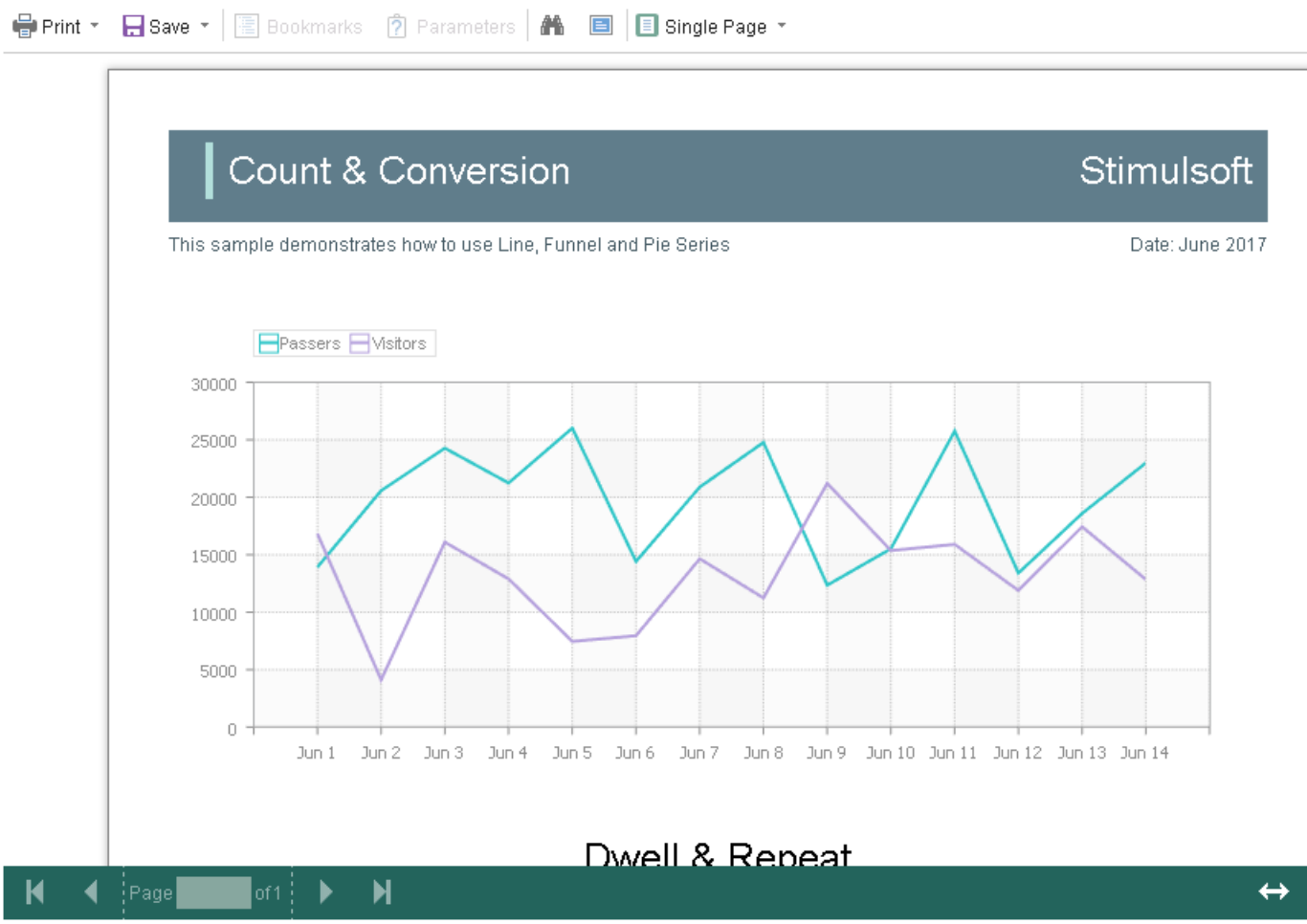
There are currently **8 themes** available with different color accents. As a result, **more than 60** variants of the appearance are available. This allows you to customize the appearance of the viewer for almost any design of the Web project.



By default, the viewer has only the top toolbar on which all the report controls are located. If necessary, the toolbar can be split into top and bottom parts. The top panel will contain the menu for printing and exporting the report and the buttons for working with parameters and bookmarks. The bottom panel will contain controls to switch between the report pages and setting the zoom of pages. To enable this mode, enable the **ToolbarDisplayMode** property. It has values **Simple** and **Separated**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      ScrollbarsMode = true
   },
   Toolbar =
   {
      DisplayMode = StiToolbarDisplayMode.Separated
   }
})
...
```



In addition, it is possible to set the appearance parameters for the main elements of the viewer. For example, you can change the font and color of the control panel inscriptions of the viewer, set the background of the viewer, set the color of page borders, etc. Below is a list of available properties that change the appearance of the viewer, and their default values.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      BackgroundColor = Color.White,
      PageBorderColor = Color.Blue,
      ShowPageShadow = true
   },
   Toolbar =
   {
      BackgroundColor = Color.White,
```

```
    BorderColor = Color.Gray,
    FontColor = Color.Black,
    FontFamily = "Arial"
  }
})
...
```

6.1.7 Basic Features

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The main features of the viewer include the following operations:

- Displaying the report.
- Switching between the report pages.
- Changing the scale.
- Displaying the preview mode.

All specified operations are performed in the AJAX mode without restarting the browser page. For the correct work of these operations, you should define a special ViewerEvent action.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
  Actions =
  {
    ViewerEvent = "ViewerEvent"
  }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnGetViewerEvent()
{
  // Some code before loading the viewer resources
  // ...

  return StiNetCoreViewer.ViewerEventResult(this);
}
```

```
public IActionResult OnPostViewerEvent()
{
   // Some code before the viewer actions
   // ...

   return StiNetCoreViewer.ViewerEventResult(this);
}
...
```

Information

The **ViewerEvent** action is mandatory. Without it, the viewer can't work correctly. The action is called for two types of requests: **OnGet** – the component requests the resources necessary for work, such as CSS styles, JS scripts and images; **OnPost** – all other actions of the viewer.

The **ViewerEvent** action returns a prepared HTML page of the report (or set of pages), built taking into account the current state of the viewer. If necessary, you can change the parameters of the current report in the specified action and update the report data in case of interactive actions of the viewer.

6.1.8 Printing Reports

Information

Please note that the print option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component provides several options for printing a report. Each has its advantages and disadvantages.

Print to PDF

Printing will be done by exporting the report to PDF. The advantages are greater accuracy of positioning and printing of the report elements compared to other printing options. Among the drawbacks, one can mention the mandatory presence of a plug-in installed in a web browser for viewing PDF files (modern browsers have embedded PDF viewer and printer).

Print with Preview
The report will be printed in a separate pop-up browser window in HTML. The report can be previewed and then sent to the printer or copied to another location as text or HTML code. Advantages - cross-browser compatibility when printing, no need to install special plug-ins. The disadvantage is the relatively low accuracy of the position of the report elements due to the peculiarities of the implementation of HTML formatting.

Print without Preview
The report will be printed directly to the printer without preview. After selecting this menu item, the system print dialog is displayed. Since printing in this mode is carried out in the HTML format, the print quality is similar to printing a report with a preview.

Information

When printing to the **HTML format**, you should check the compliance of report page settings and printer parameters (paper size, orientation, margins, indents), as well as check your browser print settings, such as margins, headers, footers, watermarks printing, color printing.

The print function does not require additional settings for the viewer. If you need to perform any actions before printing a report, you can define a special **PrintReport** action.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      PrintReport = "PrintReport"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostPrintReport()
```

```
{
   // Some code before print
   // ...

   return StiNetCoreViewer.PrintReportResult(this);
}
...
```

Print setup

If you choose printing a report in the viewer panel, a menu with printing options is displayed. The **HTML5 Viewer** component is able to force the required printing mode. To do this, set the **PrintDestination** property to one of the following values of the **StiPrintDestination** enumeration.

- **Default** – the menu will be displayed (the default property value);
- **Pdf** – print to the PDF format;
- **Direct** – printing to the HTML format directly to the printer, the system print dialog will be displayed;
- **WithPreview** – print to the HTML format with preview in a pop-up window.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Toolbar =
   {
      PrintDestination = StiPrintDestination.Default
   }
})
...
```

The **HTML5 Viewer** component is able to completely disable report printing. To do this, set the value of the **ShowPrintButton** property to **false**.

Index.cshtml

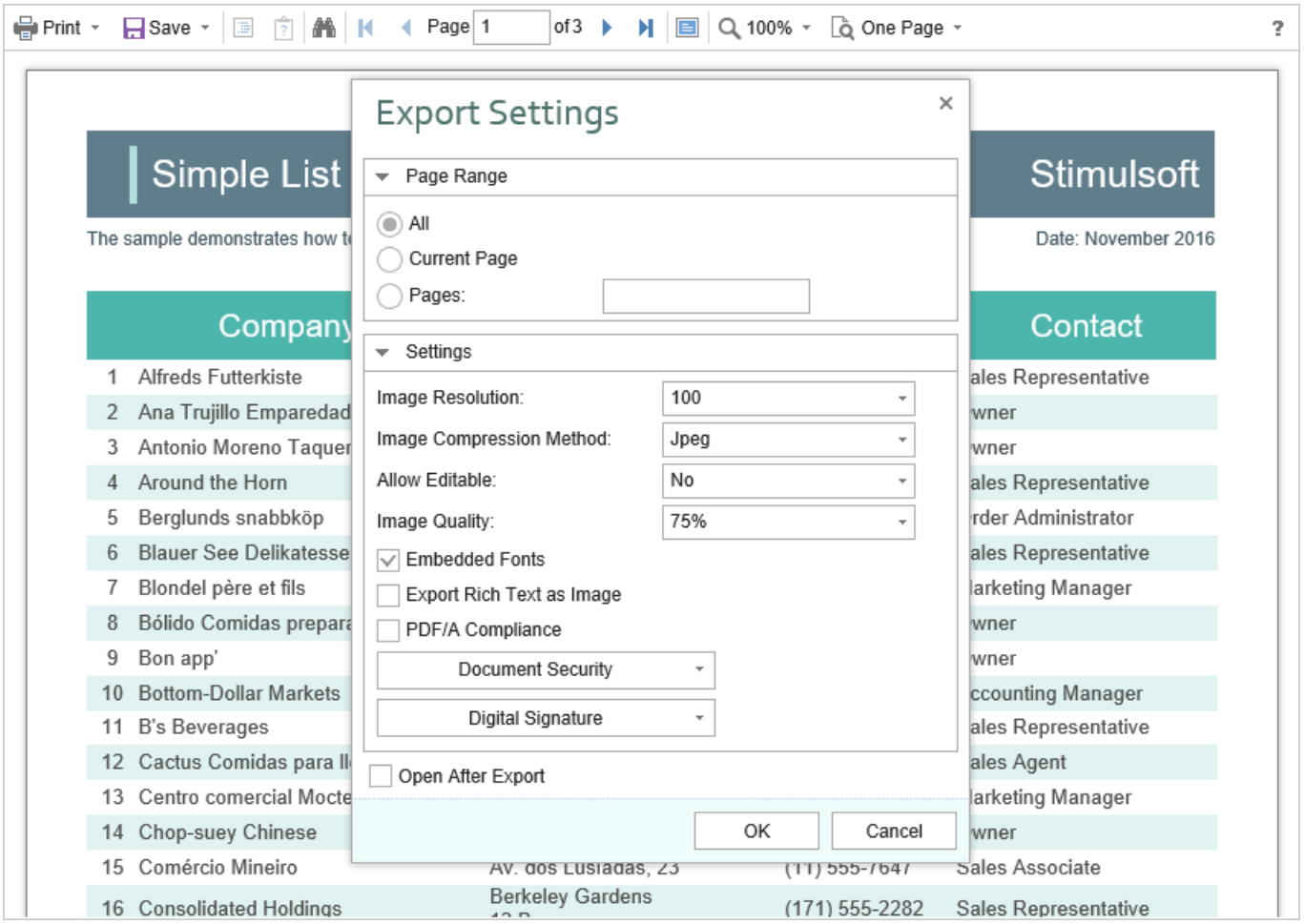
```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Toolbar =
   {
      ShowPrintButton = false
   }
})
...
```

6.1.9 Exporting Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component allows you to export the displayed report to three dozen various formats, such as **PDF**, **HTML**, **Word**, **Excel**, **XPS**, **RTF**, **images**, **text**, and others. You may export the dashboard to PDF, Excel, image files.



The export function does not require additional settings for the viewer. If you need

to perform any actions before exporting the report, you can define a special **ExportReport** action.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      ExportReport = "ExportReport"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostExportReport()
{
   // Some code before export
   // ...

   return StiNetCoreViewer.ExportReportResult(this);
}
...
```

Export settings

Each report export format of the **HTML5 Viewer** component has a lot of settings, and each setting has its default values. Sometimes you need to set other default values. For this purpose, a special **DefaultSettings** property of the viewer is used. It is a container for all the default export settings.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Exports =
   {
      DefaultSettings =
      {
         ExportToPdf =
         {
            ImageQuality = 0.75f,
            ImageFormat = Stimulsoft.Report.Export.StiImageFormat.Color
         },
         ExportToHtml =
         {
            ExportMode = Stimulsoft.Report.Export.StiHtmlExportMode.Div,
```

```
            UseEmbeddedImages = true
        }
    }
}
...
```

If it is required, you can completely hide export dialogs. Exporting will always be done with default settings. For this, it is enough to set the value of the **ShowExportDialog** property to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
    Exports =
    {
        ShowExportDialog = false
    }
})
...
```

The **HTML5 Viewer** component contains 30+ export formats, and sometimes you need to disable unwanted formats. This allows you to simplify UI and the use of the viewer. To disable unused export formats, it is enough to set the values for the corresponding properties of the viewer listed in the list below to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
    Exports =
    {
        ShowExportToDocument = true,
        ShowExportToPdf = true,
        ShowExportToXps = true,
        ShowExportToPowerPoint = true,
        ShowExportToHtml = true,
        ShowExportToHtml5 = true,
        ShowExportToMht = true,
        ShowExportToText = true,
        ShowExportToRtf = true,
        ShowExportToWord2007 = true,
        ShowExportToOpenDocumentWriter = true,
        ShowExportToExcel = true,
        ShowExportToExcelXml = true,
        ShowExportToExcel2007 = true,
        ShowExportToOpenDocumentCalc = true,
```

```
    ShowExportToCsv = true,
    ShowExportToDbf = true,
    ShowExportToXml = true,
    ShowExportToDif = true,
    ShowExportToSylk = true,
    ShowExportToImageBmp = true,
    ShowExportToImageGif = true,
    ShowExportToImageJpeg = true,
    ShowExportToImagePcx = true,
    ShowExportToImagePng = true,
    ShowExportToImageTiff = true,
    ShowExportToImageMetafile = true,
    ShowExportToImageSvg = true,
    ShowExportToImageSvgz = true
  }
})
...
```

The **HTML5 Viewer** component can completely disable the export menu. To do this, set the value of the **ShowSaveButton** property to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
  Toolbar =
  {
    ShowSaveButton = false
  }
})
...
```

6.1.10 Viewing Modes

The **HTML5 Viewer** component has two modes for displaying reports - with and without scrollbars. By default, the view mode without scrollbars is set. To enable the scrollbar view mode, set the value of the **ScrollbarsMode** property to **true**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
  Appearance =
  {
    ScrollbarsMode = true
  }
})
...
```

In the first mode (without scrollbars), the viewer displays a page or report as a whole, automatically stretching the preview space. If the width and height are specified, the viewer will truncate the page that is out of bounds. In the second mode, unlike the first one, when the page is out of bounds of the viewer's size, no truncation will be performed. Scrollbars will appear, using which you can view the entire page or report.

Information

In the report mode with scrollbars, you should set the height of the viewer, otherwise the default height will be set to **650 pixels**.

The **HTML5 Viewer** component provides the full-screen report and dashboard mode. By default, the standard view mode is enabled, the viewer has the specified dimensions in the settings. To enable the full-screen mode, set the **FullScreenMode** property to **true**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      FullScreenMode = true
   }
})
...
```

Also, to enable or disable the full-screen mode, you can use the corresponding button on the control panel of the viewer.

The **HTML5 Viewer** component has three modes to display reports - page-by-page, entire report, and tabular display of report pages. To control the modes, the **ViewMode** property is used. It can have one of the specified values - **SinglePage**, **Continuous**, **MultiplePages**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
```

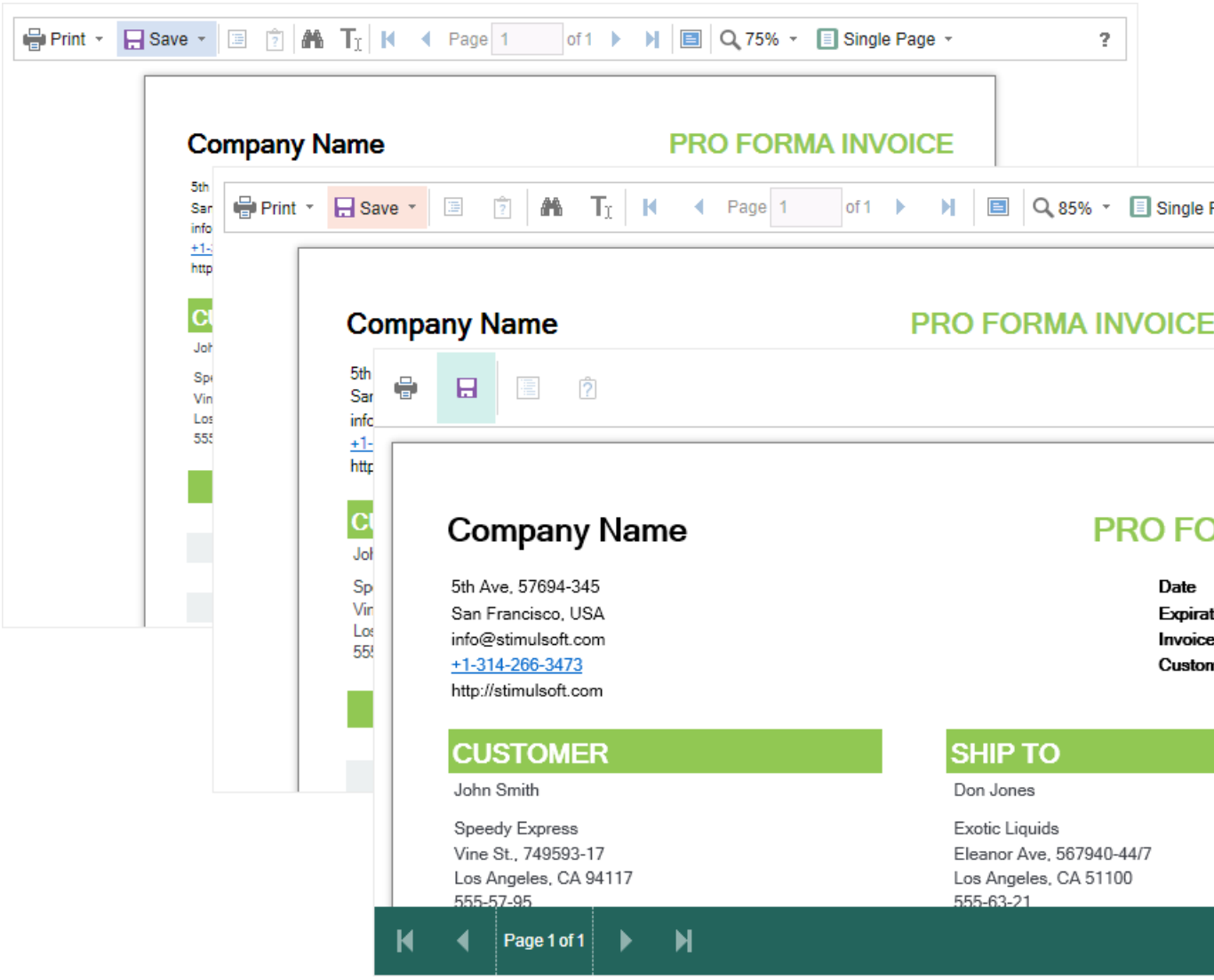
```
   Toolbar =
   {
      ViewMode = StiWebViewMode.SinglePage
   }
})
...
```

The **HTML5 Viewer** component supports interaction on a regular PC display and works with a touchscreen of screens and mobile devices. The **InterfaceType** property allows controlling the interface modes. The property can have one of the following values:

- **Auto** – the viewer's interface is determined automatically depending on the device the report is displayed on. That is the default value.
- **Mouse** – the standard interface with a mouse control will be used for all the screen types.
- **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability.
- **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with smartphones.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      InterfaceType = StiInterfaceType.Auto
   }
})
...
```



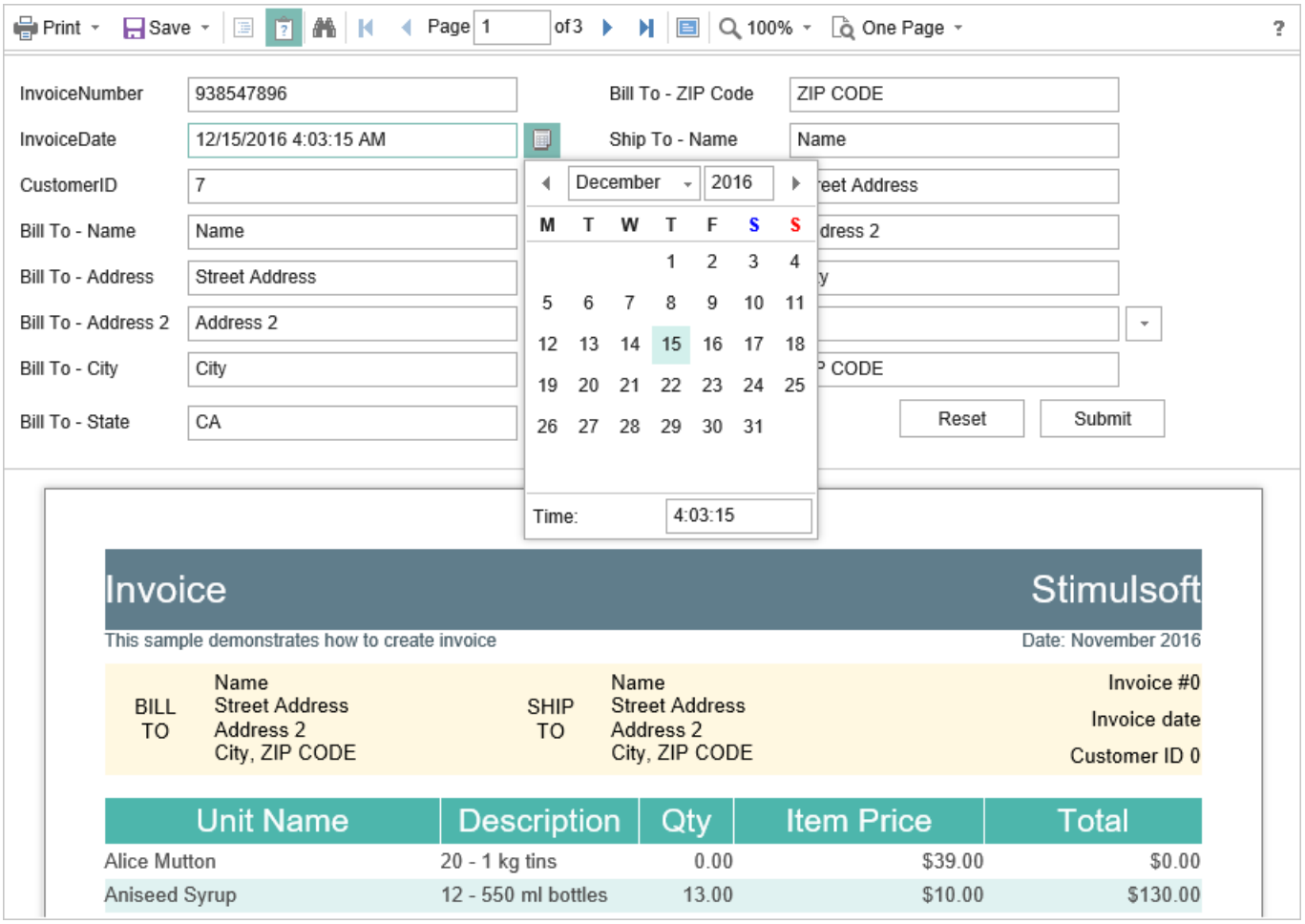
6.1.11 Work with Parameters

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To work with report parameters in the **HTML5 Viewer**, there is a special settings panel. To add a parameter to the panel, you need to define a variable in a report requested by the user. When viewing a report in the viewer, such a variable will be

automatically added to the settings panel. It supports all report variables (normal variables, date and time, borders, lists, etc.).



To work with reports with parameters, no additional viewer settings are required. If you need to perform some actions before applying the parameters, you can define a special **Interaction** action.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      Interaction = "ViewerInteraction"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostViewerInteraction()
{
   // Some code before any interaction
   // ...

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

This action is called during any interactive actions of the viewer. If you need to perform any actions only when applying report parameters, you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class. They are passed to any server-side on any request and contain all necessary information and states of the client part of the viewer. To determine the type of action of the viewer, it is enough to check the **Action** property of the viewer parameters.

Index.cshtml.cs

```
...
public IActionResult OnPostViewerInteraction()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   if (requestParams.Action == StiAction.Variables)
   {
      // Some code before apply parameters
   }

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

If you do not need to work with parameters, you can completely disable this feature. To do this, use the **ShowParametersButton** property in the **Toolbar** section of properties, which should be set to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Toolbar =
   {
      ShowParametersButton = false
```

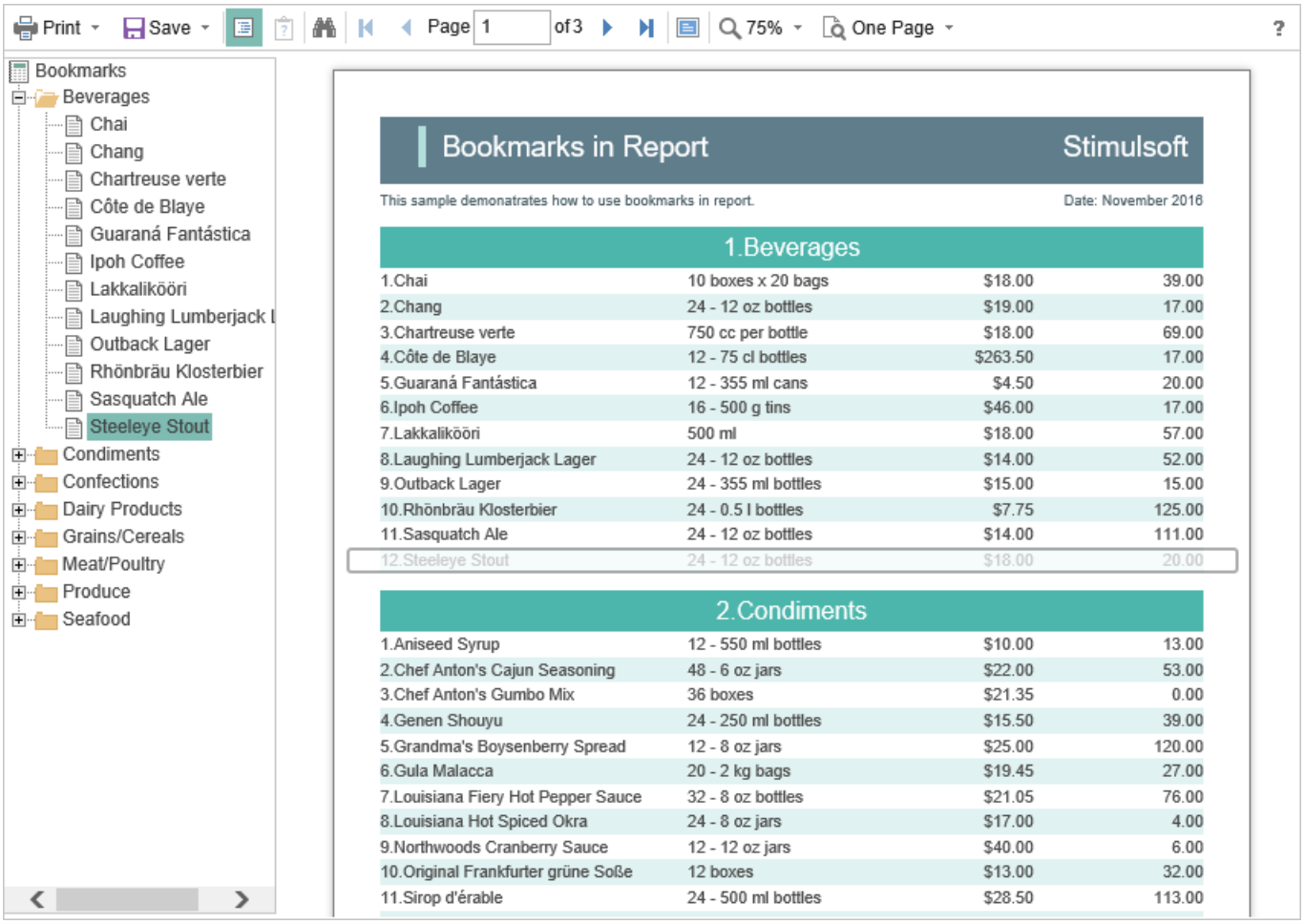
```
    }
})
...
```

Information

The options panel will not be displayed with such a viewer configuration, even if the parameters are present in the displayed report.

6.1.12 Work with Bookmarks

The **HTML5 Viewer** component supports report bookmarks. A panel with bookmarks will be displayed when displaying such a report on the left side of the page. When you select a bookmark of the report, the viewer will carry out an automatic transition to the specified page, and the report item with a bookmark is highlighted.



By default, the bookmarks bar width is 180 pixels. The **HTML5 Viewer** component allows you to change this value. For this, the **BookmarksTreeWidth** property, which value is specified in pixels, is used.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Appearance =
   {
      BookmarksTreeWidth = 200
   }
})
...
```

If work with report bookmarks is not required, you can disable this feature. For this, set the **ShowBookmarksButton** property to **false**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Toolbar =
   {
      ShowBookmarksButton = false
   }
})
...
```

Information

In this case, report bookmarks will not be displayed, even if they are present in the displayed report. This property has no effect on printing and exporting reports.

When printing a report with bookmarks, the bookmark tree will be hidden. If you want to print bookmarks with the report, it is necessary to set the **BookmarksPrint** property to **true**.

Index.cshtml

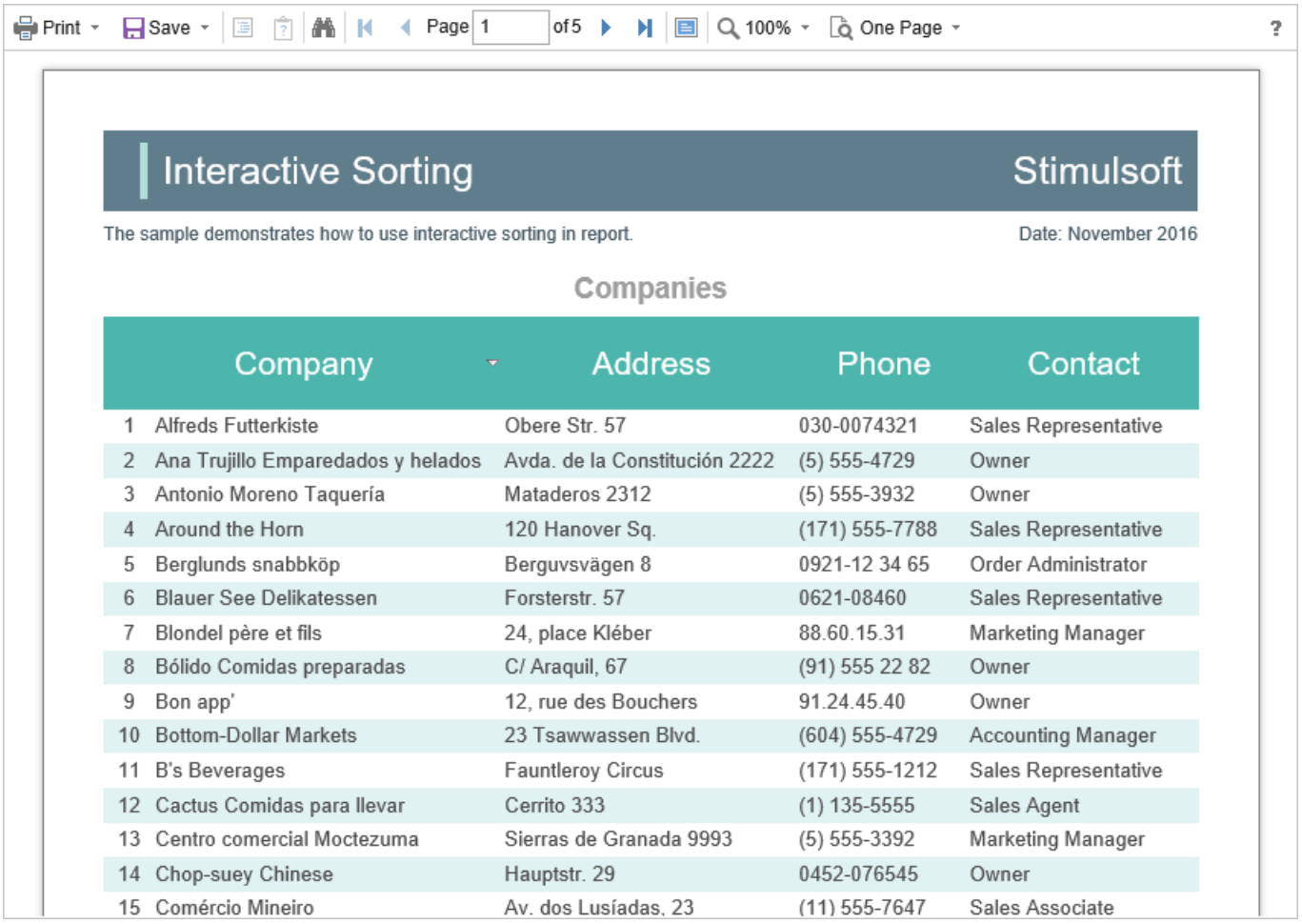
```
...
```

```
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions () {
   Appearance =
   {
      BookmarksPrint = true
   }
})
...
```

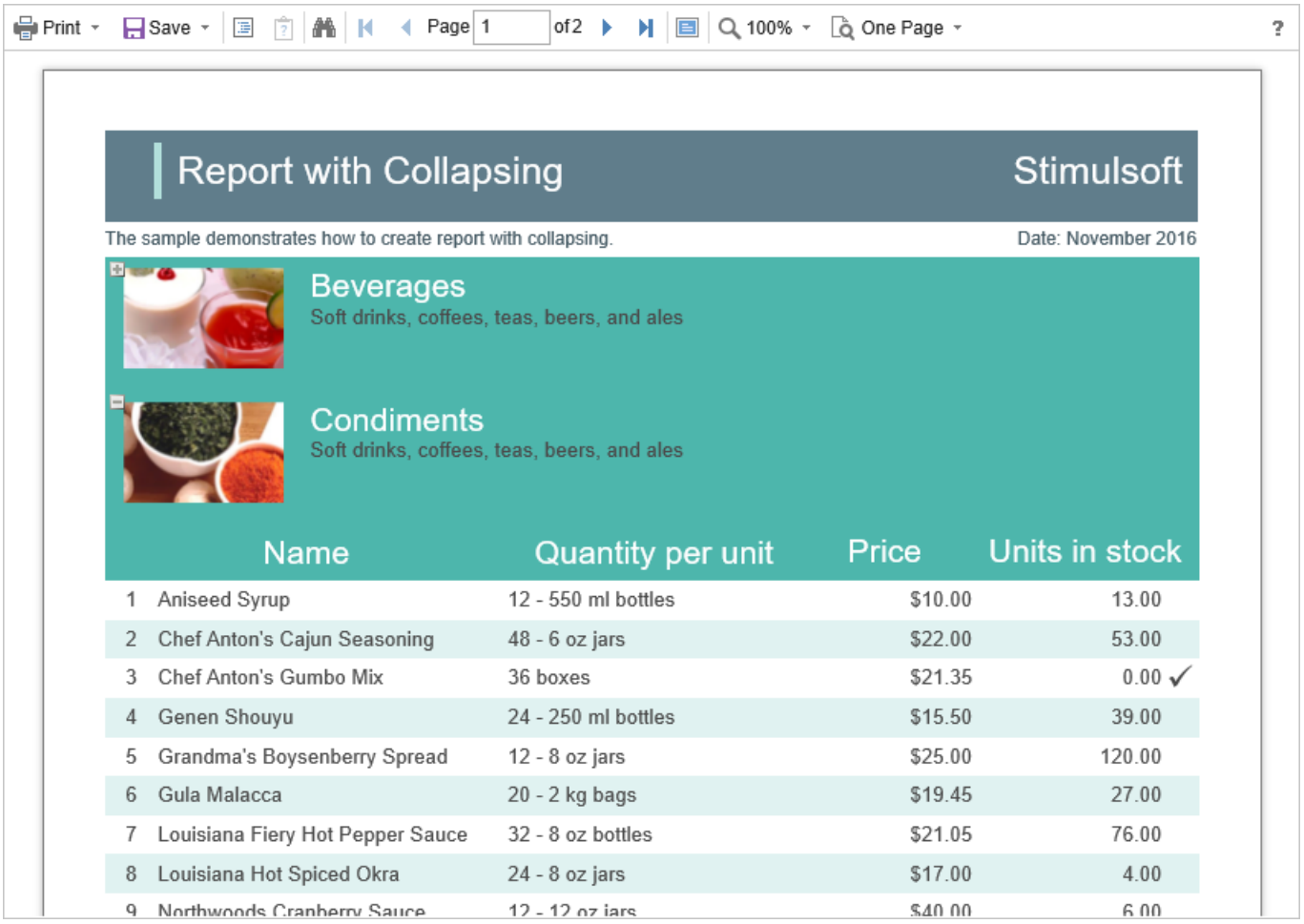
6.1.13 Dynamic Sorting, Collapsing, and Drill-Down

The **HTML5 Viewer** component supports dynamic sorting, collapsing, and drill-down of reports. Dynamic sorting provides the ability to change the direction of sorting in a rendered report. To do this, click on the component that has dynamic sorting enabled. Dynamic sorting is carried out in the following directions - **Ascending** and **Descending**. Each time the component is clicked, the sorting direction is reversed.

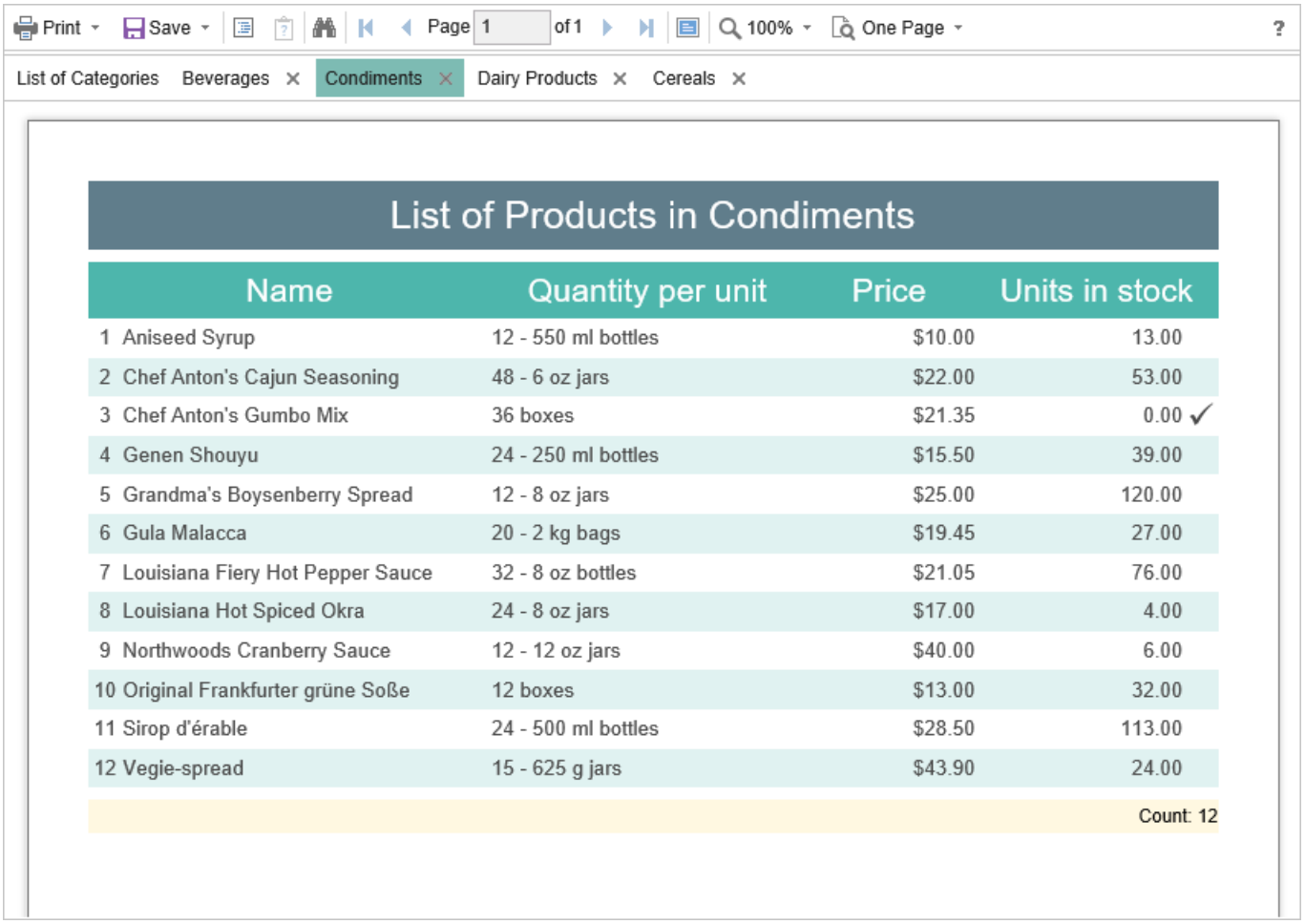
Multi-level sorting is allowed in the report. To do this, hold down the **Ctrl** key and sequentially click on the sorted components in the report. To reset sorting, you can click on any sorted component without holding down the **Ctrl** key.



A report with dynamic collapsing is an interactive report in which blocks can collapse/expand their content when you click on the block title. Report elements, which can be collapsed/expanded, are indicated by special icons - **[-]** or **[+]**.



When using drill-down, under the main panel of the viewer, the drill-down panel with tabs for drill-down reports will be displayed. The currently displayed report will be highlighted.



To work with dynamic sorting, collapsing, and drill-down reports, no additional viewer settings are required. A special **Interaction** action is used to perform any actions before sorting, collapsing, or drill-down of the report. It will be called when interactive action of the viewer.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      Interaction = "ViewerInteraction"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostViewerInteraction()
{
   // Some code before any interaction
   // ...

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

To get the type of action, you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class. They are passed to any server-side by any request and contain all necessary information and states of the client part of the viewer. For each type of interactivity, the viewer has a certain type of action:

- **Sorting** – when using column sorting;
- **DrillDown** – when using drill-down in reports;
- **Collapsing** – when using collapsing report blocks.

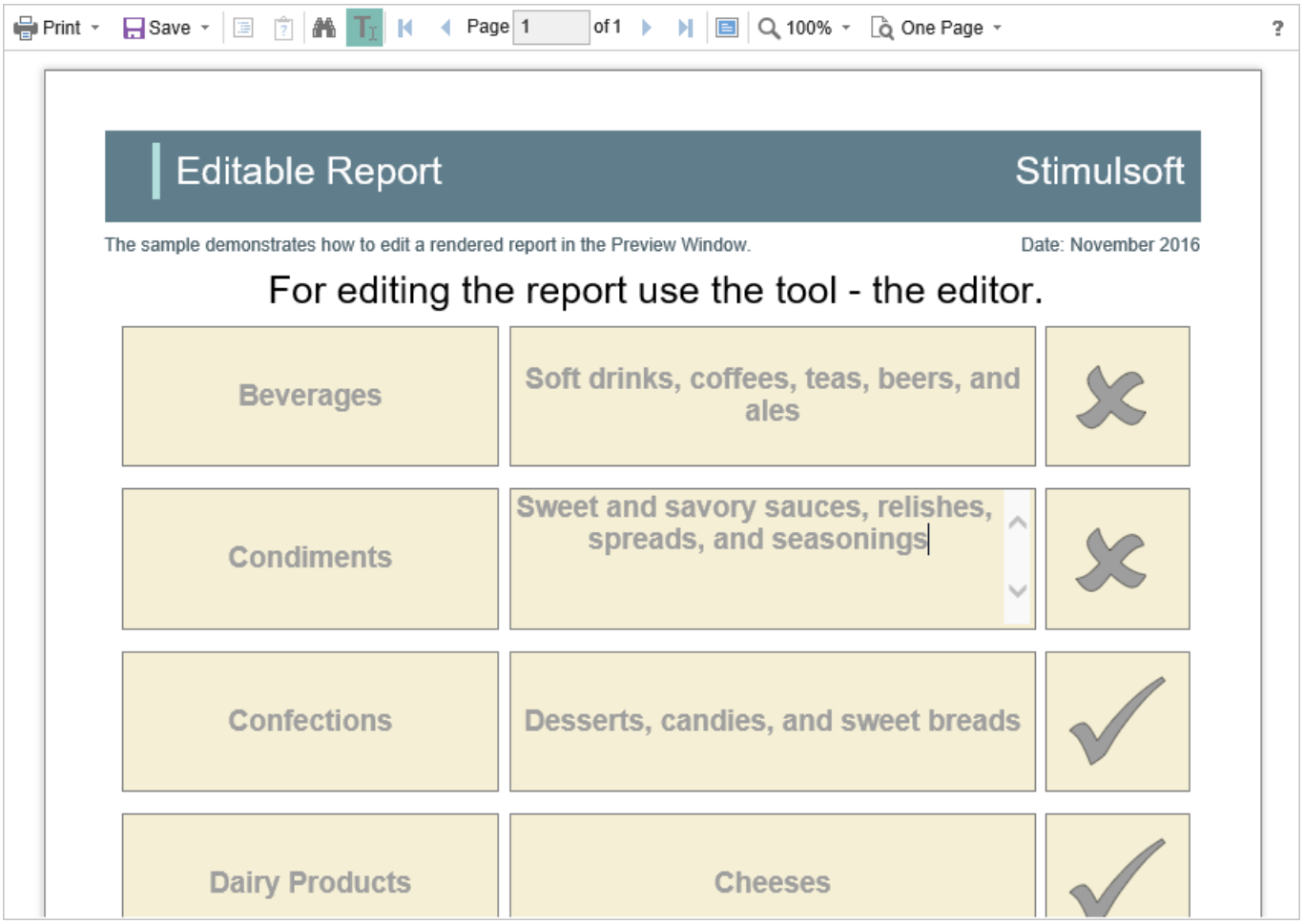
Index.cshtml.cs

```
...
public IActionResult OnPostViewerInteraction()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   switch (requestParams.Action)
   {
      case StiAction.Sorting:
         break;

      case StiAction.DrillDown:
         break;

      case StiAction.Collapsing:
         break;
   }

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

6.1.14 Editing Report

The **HTML5 Viewer** component has the ability to edit report items, such as text boxes and checkboxes. You should mark the required components as editable in the report template for the editing to be possible. After displaying a report in the viewer, you need to click the corresponding button on the viewer panel to start editing. After editing, it is necessary to click the button once more, and all changes will be

applied to the report.



For the report edit mode, no special settings of the viewer required.

Information

The edited settings will be applied when you print or export a report, and the original report remains unchanged. After restarting the viewer, all the values will be returned to the initial ones.

6.1.15 Sending Report by Email

Information

Please note that the Send Report by Email option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component provides the ability to send reports by email. To activate this feature, you should set the **ShowSendEmailButton** property of the viewer to **true** and define the **EmailReport** action.

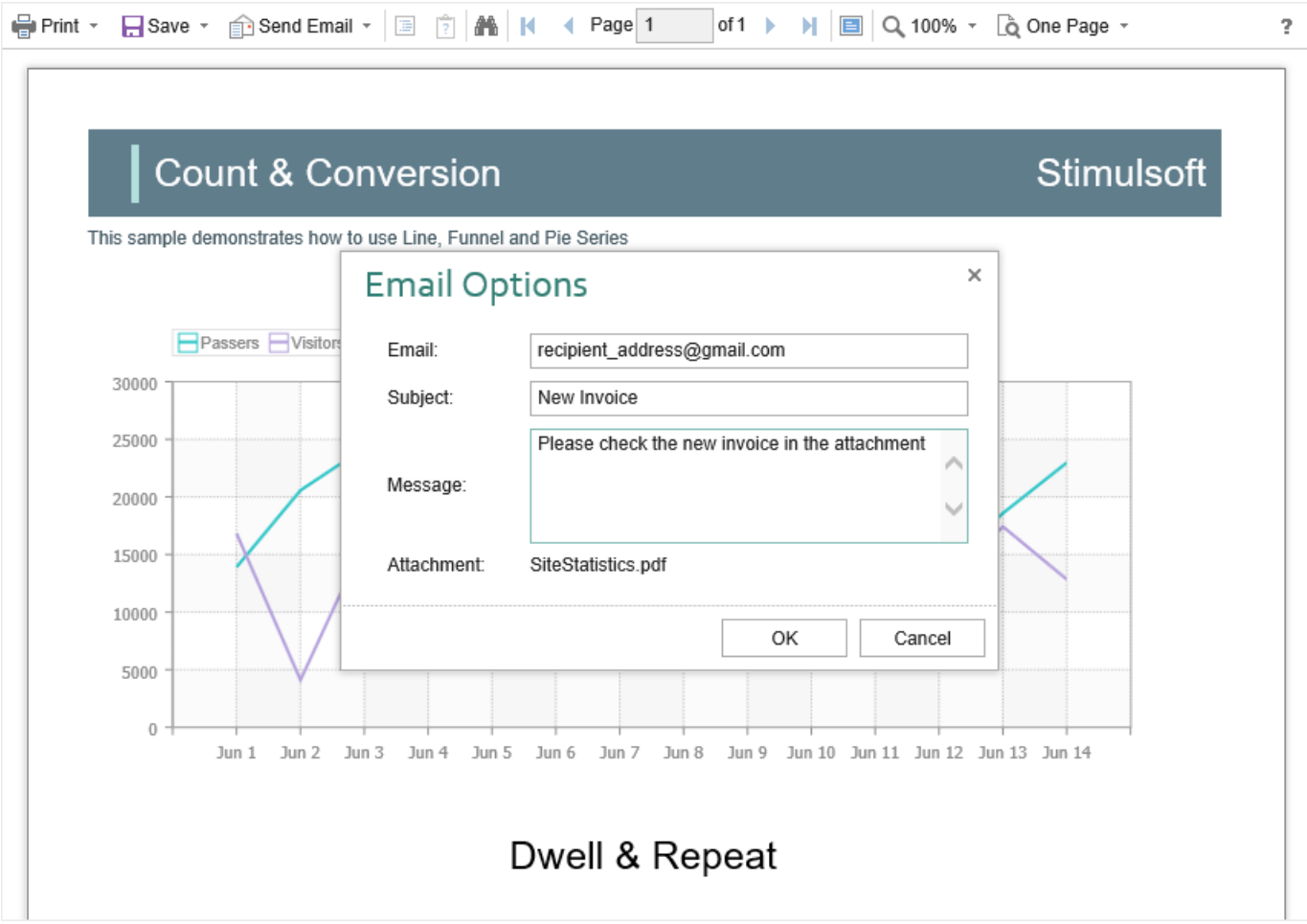
Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      EmailReport = "EmailReport"
   },
   Toolbar =
   {
      ShowSendEmailButton = true
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostEmailReport()
{
   StiEmailOptions options = StiNetCoreViewer.GetEmailOptions(this);

   // Passed from the viewer, can be checked and changed
   // options.AddressTo = "";
   // options.Subject = "";
   // options.Body = "";

   // Should be filled here
   options.AddressFrom = "admin_address@test.com";
   options.Host = "smtp.test.com";
   options.Port = 465;
   options.UserName = "admin_address@test.com";
   options.Password = "admin_password";

   // options.CC.Add("email@test.com");
   // options.BCC.Add("email@test.com");
   // options.EnableSsl = true;

   return StiNetCoreViewer.EmailReportResult(this, options);
}
...
```

When sending a report by email, the menu to select the attachment format is displayed. It corresponds to the menu for selecting the format for exporting the report. After selecting the format, the dialog to enter the send email parameters, such as the recipient's email, subject, and text of the message, is displayed.



After confirmation of sending the email, the above described **EmailReport** event will be called. You can check and correct the data entered in this form. The exported report file will be attached to the email automatically.

The **HTML5 Viewer** component allows you to set default values for the send email form. The **DefaultEmailAddress**, **DefaultEmailSubject**, and **DefaultEmailMessage** properties can be used for this. By default, these properties are empty.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Email =
   {
      DefaultEmailAddress = "recipient_address@gmail.com",
      DefaultEmailSubject = "New Invoice",
      DefaultEmailMessage = "Please check the new invoice in the
      attachment"
   }
})
...
```

6.1.16 Calling Designer from Viewer

The **HTML5 Viewer** component has the ability to call the report designer. The special **Design** button in the toolbar of the viewer (the button is disabled by default) should be used. To use this feature, you should set the **ShowDesignButton** property to true and define the **DesignReport** event handler.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      DesignReport = "DesignReport"
   },
   Toolbar =
   {
      ShowDesignButton = true
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostDesignReport()
{
   StiReport report = StiNetCoreViewer.GetReportObject(this);
   TempData["ReportName"] = report.ReportName;

   return RedirectToPage("Designer");
}
...
```

Information

The viewer does not run the designer. It only calls the specified action, in which you can get all the necessary parameters. Then, in action, you can implement a

redirection to another View, which contains the report designer.

6.1.17 Caching

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component allows you to use the server cache to store rendered reports. If you do not use caching, you should load the report, connect data, and render it again every time you request a page. If you use caching, the previously rendered report will be loaded from the cache every time you refresh the page.

When using caching, it should be taken into account that every report saved in the cache takes up server memory and, with a large number of requests to reports, this can become a critical issue. Therefore, you need to choose between two options: low memory requirements but high in performance or low-performance requirements but high in memory.

You can manage caching with the following properties.

The CacheMode property

This property of the viewer enables caching and sets its type. It can take one of the following values, specified in the **StiServerCacheMode** enumeration:

- **None** – caching is disabled. Each action of the viewer requires loading the report from the file and, if it is a report template, then render it;
- **ObjectCache** – for caching, the server cache is used. The report object is saved in this cache (set by default);
- **StringCache** – for caching, the server cache is used. The report is saved as a packed string in this cache;
- **ObjectSession** – the current session, in which the report object is saved, is used for caching;

- **StringSession** – for caching, the current session is used. The report is saved as a packed string in this cache.

The CacheItemPriority property

This property sets the priority of the report stored in the server's cache. It affects the automatic clearing of the server memory in case of a lack of memory. The lower the priority is, the greater is the chance of removing information from memory.

The CacheTimeout property

This property specifies the amount of time in minutes for which you want to save the report in the server cache. If you use caching and the requested report is not found in the cache (the objects storage time has expired), then it will be requested again using a special **GetReport** event, then connect the report data and render it.

StiCacheHelper

The **HTML5 Viewer** component provides the ability to define your methods of working with report caching. For this purpose, a special class **StiCacheHelper** is used. It contains methods for obtaining a report from the cache and saving the report to the cache. It is necessary to create a new class inherited from **StiCacheHelper** and reload the above methods, which respectively have the names - **GetReport**, **SaveReport** and **RemoveReport**.

Index.cshtml.cs

```
...
public class IndexModel : PageModel
{
   public class StiMyCacheHelper : StiCacheHelper
   {
      public override StiReport GetReport(string guid)
      {
         var path = Path.Combine(HttpContext.Server.MapPath("CacheFiles"),
         guid);
         if (File.Exists(path))
         {
            StiReport report = new StiReport();
```

```
            var packedReport = File.ReadAllText(path);
            if (guid.EndsWith(GUID_ReportTemplate))
            report.LoadPackedReportFromString(packedReport);
            else report.LoadPackedDocumentFromString(packedReport);

            return report;
          }
          return null;
        }

        public override void SaveReport(StiReport report, string guid)
        {
          var packedReport = guid.EndsWith(GUID_ReportTemplate) ?
          report.SavePackedReportToString() :
          report.SavePackedDocumentToString();
          var path = Path.Combine(HttpContext.Server.MapPath("CacheFiles"),
          guid);
          File.WriteAllText(path, packedReport);
        }

        public override void RemoveReport(string guid)
        {
          var path = Path.Combine(HttpContext.Server.MapPath("CacheFiles"),
          guid);
          if (File.Exists(path))
            File.Delete(path);
        }
      }

      static IndexModel()
      {
        StiNetCoreViewer.CacheHelper = new StiMyCacheHelper();
      }
    }
    ...
```

To initialize the work with report caching using the created class, it is enough to set it as a value of the static **StiNetCoreViewer.CacheHelper** property in the controller constructor.

Information

If report caching is disabled (the **CacheMode** property of the viewer is set to **None**), the specified class will not be used.

6.1.18 Additional Methods

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

For **HTML5 Viewer**, several additional methods are used to get the object of the currently viewed report, parameters of the current state of the viewer, and other useful data. These methods can be used in any actions of the viewer.

The GetReportObject() method

Returns the report object with which the viewer is currently working. It is possible to perform the necessary actions - register new data sets, change report properties, assign parameters or load another report to the object. Then, the report can be returned to the viewer, specifying it as a parameter in the resulting action method.

Index.cshtml.cs

```
...
public IActionResult OnPostViewerInteraction()
{
   StiReport report = StiNetCoreViewer.GetReportObject(this);
   report.ReportName = "MyReportName";

   return StiNetCoreViewer.InteractionResult(this, report);
}
...
```

The GetFormValues() method

Returns the values of the form that initiated (opened by the POST request) a page of the viewer. Thus, it is possible to get a collection of form parameters in any action of the viewer.

Index.cshtml.cs

```
...
public IActionResult OnPostViewerInteraction()
{
   NameValueCollection formValues = StiNetCoreViewer.GetFormValues(this);

   return StiNetCoreViewer.InteractionResult(this);
```

```
}
...
```

By default, this feature is disabled to optimize requests of the client-side of the viewer to the server. To enable it, set the **PassFormValues** property to **true**.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Server =
   {
      PassFormValues = true
   }
})
...
```

The GetRequestParams() method

Returns all parameters of the current state of the viewer passed to the server-side. They can be useful for determining the type of action that the viewer is currently executing - for example, to determine the type of export and all action parameters.

Index.cshtml.cs

```
...
public IActionResult OnPostExportReport()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   if (requestParams.ExportFormat == StiExportFormat.Pdf)
   {
      StiReport report = StiNetCoreViewer.GetReportObject(this);

      // Some action with report for the PDF export
      // ...

      return StiNetCoreViewer.ExportReportResult(this, report);
   }

   return StiNetCoreViewer.ExportReportResult(this);
}
...
```

You can change the values of some parameters. After making changes, for the correct operation of the viewer, you should transfer the modified parameter object to the input of the resulting method.

Index.cshtml.cs

```
...
public IActionResult OnPostViewerInteraction()
{
   StiRequestParams requestParams =
   StiNetCoreViewer.GetRequestParams(this);
   if (requestParams.Action == StiAction.Variables)
   {
      requestParams.Interaction.Variables["Variable1"] = "MyValue";
      return StiNetCoreViewer.InteractionResult(this, requestParams);
   }

   return StiNetCoreViewer.InteractionResult(this);
}
...
```

The GetExportSettings() method

Returns all the parameters of the current report export. The type of the parameter object will correspond to the type of export selected in the viewer menu. Any export parameters can be changed and passed to the input of the resulting method. In this case, the report will be exported with the parameters transferred.

Index.cshtml.cs

```
...
public IActionResult OnPostExportReport()
{
   StiExportSettings settings = StiNetCoreViewer.GetExportSettings(this);
   if (settings.GetExportFormat() == StiExportFormat.Pdf)
   {
      StiPdfExportSettings pdfSettings = (StiPdfExportSettings)settings;
      pdfSettings.EmbeddedFonts = true;
      pdfSettings.AllowEditable = StiPdfAllowEditable.No;
      return StiNetCoreViewer.ExportReportResult(this, settings);
   }

   return StiNetCoreViewer.ExportReportResult(this);
}
...
```

The MapPath() and MapWebRootPath() methods

Returns the absolute path, respectively, to the application or wwwroot directory. You can use this to upload report templates files, data files, etc. These methods are

located in the StiNetCoreHelper static class.

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
   StiReport report = new StiReport();
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));

   return StiNetCoreViewer.GetReportResult(this, report);
}
...
```

6.1.19 Export and Printing from Code

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** provides the ability to print reports in various ways and export reports to various formats. These actions are performed using the viewer menu. If you want to print or export a report using the code, for example, in the specific page event, you can use the special **StiNetCoreReportResponse** class. This class contains a set of static methods that allow you to print or export a report from the code, and the report viewer is not required.

Index.cshtml

```
page "{handler?}"
...
<a href="PrintReport">Print Report from Code</a>
<br />
<a href="ExportReport">Export Report from Code</a>
...
```

Index.cshtml.cs

```
...
private StiReport LoadSimpleList()
{
   DataSet dataSet = new DataSet();
   dataSet.ReadXml(Server.MapPath("Reports/Demo.xml"));
```

```
    StiReport report = new StiReport();
    report.Load(Server.MapPath("Reports/SimpleList.mrt"));
    report.RegData(dataSet);

    return report;
}

public IActionResult OnGetPrintReport()
{
    StiReport report = LoadSimpleList();

    return StiNetCoreReportResponse.PrintAsPdf(report);
    //return StiNetCoreReportResponse.PrintAsHtml(report);
}

public IActionResult OnGetExportReport()
{
    StiReport report = LoadSimpleList();

    return StiNetCoreReportResponse.ResponseAsPdf(report);
    //return StiNetCoreReportResponse.ResponseAsExcel2007(report);
    //return StiNetCoreReportResponse.ResponseAsText(report);
    //StiNetCoreReportResponse.ResponseAsJson(report);
}
...
```

The **StiNetCoreReportResponse** class contains methods for printing in PDF and HTML formats and methods to export the report in any of the supported formats. As arguments, methods can take various export settings, displaying modes and options for saving received files.

6.1.20 Timeout

When working with the **StiNetCoreViewer** component, you can set the timeout for various operations — storing the report in the cache, server response, and query execution. The timeout setting is done using the component properties and report options.

CacheTimeout Property

Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
    Server =
```

```
   {
      CacheTimeout = 10
   }
})
...
```

Using the cache will increase the speed of the report viewer. See the chapter Caching for more information.

RequestTimeout Property

Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Server =
   {
      RequestTimeout = 30
   }
})
...
```

CommandTimeout Option

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to set the query timeout for the already created connection and data sources in the report.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Actions =
   {
      GetReport = "GetReport",
      ViewerEvent = "ViewerEvent"
   }
})
```

```
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Report.mrt"));
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   return StiNetCoreViewer.GetReportResult(this, report);
}

public IActionResult OnGetViewerEvent()
{
   return StiNetCoreViewer.ViewerEventResult(this);
}

public IActionResult OnPostViewerEvent()
{
   return StiNetCoreViewer.ViewerEventResult(this);
}
...
```

6.1.21 Viewer Settings

The **HTML5 Viewer** is configured using properties that are located in the **StiNetCoreViewerOptions** class. All properties are divided into groups. Some of the groups contain subgroups for ease of use. The following is an example of setting the properties of the viewer.

Index.cshtml

```
...
@Html.StiNetCoreViewer(new StiNetCoreViewerOptions() {
   Theme = StiViewerTheme.Office2022WhiteTeal,
   Localization = "Localization/en.xml",
   Actions =
   {
      GetReport = "GetReport",
      ViewerEvent = "ViewerEvent"
   },
   Appearance =
   {
      InterfaceType = StiInterfaceType.Auto,
      ScrollbarsMode = true,
      ShowTooltips = false
   },
   Exports =
   {
      DefaultSettings =
      {
```

```
        ExportToPdf =
        {
           CreatorString = "Company Name",
           ImageQuality = 0.75f
        }
     },
     ShowExportToDbf = false,
     ShowExportToDif = false
   }
})
...
```

Please note that all dashboard elements have their own save options and full-screen buttons for preview. There are no special options to control displaying them, but they can be disabled through the properties of the element. The code below should be added after loading the report before passing it to the viewer.

Index.cshtml.cs

```
...
var dbsElementInteraction = (report.GetComponentByName("RegionMap1") as
Stimulsoft.Report.Dashboard.IStiElementInteraction).DashboardInteraction;
(dbsElementInteraction as
Stimulsoft.Report.Dashboard.IStiInteractionLayout).ShowFullScreenButton =
false;
(dbsElementInteraction as
Stimulsoft.Report.Dashboard.IStiInteractionLayout).ShowSaveButton = false;
...
```

Main settings (without groups)

Name	Description
Theme	Sets the viewer theme. The list of available themes can be found in the **StiViewerTheme** enumeration. The default value is **Office2022WhiteBlue**.
Localization	Sets the path to the XML localization file. The path can be absolute or relative. By default, the English localization is used. It is built into the viewer and does not require additional XML files.
Width	Sets the width of the component in the required

	units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. The default width is 100%.
Height	Sets the height of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the automatic height is set depending on the size of the report page, or 650 pixels in the view mode of the viewer with scrollbars.

Actions

Name	Description
GetReport	Specifies the name of the action method for preparing the rendered report. Specifies the name of the action method for preparing the constructed report. If report caching is enabled, this action will be called only once when the report is requested or if the requested report is not found in the server cache.
PrintReport	Specifies the name of the action method of report printing. This is not relevant when viewing dashboards.
ExportReport	Specifies the name of the action method of the export the report to the specified format.
EmailReport	Specifies the name of the action method of sending the report by email. This is not relevant when viewing dashboards.
Interaction	Specifies the name of the action method for the viewer to work with interactive operations, such as using parameters, dynamic sorting, collapsing, and drill-down.

DesignReport	Specifies the name of the action method to go to the specified view by clicking the Design button on the viewer panel.
ViewerEvent	Specifies the name of the action method of basic viewer events and the processing actions of the viewer, such as printing and exporting a report, working with parameters, and interactivity, if these actions are not specified separately. In addition, this action is used to load scripts and styles of the viewer. This action is mandatory.

Server

Name	Description
RouteTemplate	Sets the route template that is returned when the report viewer actions are executed. If the property is not set, then the Razor project template will be used instead. If the `UseRelativeUrls` property is set to `true`, the `BasePath` will not be respected for this property.The default value of this property is null.
AllowAutoUpdateCookies	Allows the viewer to update the cookies automatically on every request to the server. By default, cookies are set when creating the viewer, if they are not specified in the report. By default, the property is set to **false**.
AllowAntiforgeryToken	Allow the viewer to automatically request and send the antiforgery token. By default, the property is set to **true**.
RequestTimeout	Sets the response timeout from the server in seconds, after which an error will be generated. The default value is 20 seconds. For big reports, it is recommended to increase this value.

CacheTimeout	Sets the time in minutes that the server will store the report since the last action of the viewer. The default value is 20 minutes.
CacheMode	Sets the report caching mode. It can take one of the following values of the **StiServerCacheMode** enumeration: › **None** – caching is disabled, the report will be reloaded each time using the **GetReport** event; › **ObjectCache** – the cache is used as the storage, the report is stored as an object (default value); › **ObjectSession** – the session is used as the storage, the report is stored as an object; › **StringCache** – the server cache is used as the storage, the report is serialized to a packed string; › **StringSession** – the session is used as storage, the report is serialized into a packed string.
CacheItemPriority	Sets the priority of the report stored in the server cache. This property affects the automatic clearing of the server memory in case of lack of memory. The lower the priority is, the greater is the chance of removing information from memory.
AllowAutoUpdateCache	Sets the mode for automatic cache update. The report stored in the cache or the server session will be automatically re-saved after a certain period of time when the viewer is idle (every 3 minutes). By default, the property is set to **true**.
UseRelativeUrls	Sets the viewer mode in which relative URLs are used for AJAX requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value that specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-**

	1 removes the port number.
PassQueryParametersToReport	Enables using all the URL parameters of the request as the variable values. The variable names must match the parameters. The default value of the property is **false**.
PassQueryParametersForResources	Enables transferring all request URL parameters when generating links to the resources of the viewer. If **false**, only the necessary parameters are used to request the resources of the viewer. This corresponds to the correct work of the browser cache. By default, the property is set to **true**.
PassFormValues	Enables passing the values of the POST form to the client-side, if these values are required to be used in the actions of the viewer. If you enable this property, the additional **GetFormValues()** method will return a collection of form parameters. By default, the property is **false**.
ShowServerErrorPage	Enables displaying an HTML page with the details of the error that occurred on the server-side. When the property is enabled, the details of the error will be displayed in the viewer window. If the property is disabled, only the numeric error code and a short error text in the dialog box will be displayed. By default, the property is set to **true**.
UseCompression	Enables compression of the viewer requests into the GZip stream. Enables compression of the viewer requests into the GZip stream. That allows to decrease the amount of internet traffic but slows down the viewer slightly. The default value of the property is **false**.
UseCacheForResources	Enables caching of the component resources on the server-side. The following resources are supported - scripts, styles, and images. This option improves the load speed of the component and also reduces the server load in

	multi-client environments. The default value is **true**.
UseLocalizedCache	Sets a value that enables the use of a different cache depending on the selected localization. The default value of the property is **false**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
CustomCss	Sets the path to the CSS file of the viewer's styles. The standard styles of the chosen theme will not be loaded if this property has got a value. The default value of the property is an empty string.
BackgroundColor	Sets the background color of the viewer. By default, it is set to **White**.
PageBorderColor	Sets the border color of the viewer. By default it is set to **Gray**.
RightToLeft	Sets the **Right to Left** mode for viewer controls. By default, the property is set to **false**. By default, the property is set to **false**.
FullScreenMode	Sets the full-screen display mode of the viewer. By default, the property is set to **false**.
ScrollbarsMode	Sets the preview mode with scrollbars. By default, the property is set to **false**.
OpenLinksWindow	Sets the target window for opening links contained in the report. By default, the property is set to **Blank** (new window).
OpenExportedReportWindow	Sets the target window for opening the export file from the viewer. By default, the property is

	set to **Blank** (new window).
DesignWindow	Sets the destination window for launching the report designer. The default value of the property is **Self** (which is the current window).
ShowTooltips	Enables showing tips for the viewer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enables showing links to online documentation for the viewer controls. By default, the property is set to **true**.
ShowDialogsHelp	Sets a value that indicates that showing or hiding the help button in dialogs. By default, the property is set to **true**.
PageAlignment	Sets the position of the report page in the viewer window. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – the page will be aligned left; › **Center** – the page will be centered (default value); › **Right** – the page will be aligned right.
ShowPageShadow	Enables displaying shadow for report pages. By default, the property is set to **true**.
BookmarksPrint	Enables printing of report bookmarks (besides the report itself). By default, the property is set to **false**.
BookmarksTreeWidth	Sets the width of the bookmarks panel in pixels. By default, the width is 180 pixels.
ParametersPanelPosition	Specifies the position of the report parameters panel. It can take one of the following **StiParametersPanelPosition** enumeration values: › **Top** - the panel will be docked to the top margin (default value);

	› **Left** - the panel will be docked to the left margin.
ParametersPanelMaxHeight	Sets the maximum height of the parameters bar in pixels. By default, the maximum height is 300 pixels.
ParametersPanelColumnsCount	Sets the number of columns to display report parameters. By default, there are 2 columns.
ParametersPanelSortDataItems	Gets or sets a value which indicates that variable items will be sorted. By default, the property is set to **true**.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
InterfaceType	Sets the type of interface used for the viewer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the viewer's interface is determined automatically depending of the device that is report is displayed on. That is the default value. › **Mouse** – the standard interface with a mouse control will be used for all the screen types. › **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability. › **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with the smartphones.
AllowMobileMode	Enables or disables displaying a report or dashboard in the mobile mode. If the option is

	set to **false**, then the mobile view will not be used. If the option is set to **true**, the mobile view mode will be used when opening the viewer on mobile devices. By default, the option is set to **true**.
ChartRenderType	Sets the displaying mode of charts on the report page. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages. It can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN markup.
DatePickerFirstDayOfWeek	Sets the first day of the week for the date picker. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Auto** – automatic detection of the first day of the week from the browser settings (default value); › **Monday** – the first day of the week is Monday;

	› **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default, the property is set to **false**.
AllowTouchZoom	Sets ability to change the scale of the report page by using the two-fingers gesture (Pinch to Zoom) for the touch-screens. The default value of the property is **true**.
ShowReportIsNotSpecifiedMessage	Sets a value which indicates that 'The report is not specified' message will be shown. The default value of the property is **true**.
PrintToPdfMode	Sets the Print to PDF mode. It has the following values: › **StiPrintToPdfMode.Hidden** - hidden print mode (default value); › **StiPrintToPdfMode.Popup** - the PDF document will be displayed before printing in a pop-up window.
ImagesQuality	Gets or sets the image quality that will be used on the viewer page. It has the following values: › **StiImagesQuality.Low** - low quality, used to speed up loading reports and saves memory; › **StiImagesQuality.Normal** - normal quality, suitable for most cases (default value); › **StiImagesQuality.High** - high quality, used for ultra high-definition displays, but may slow down the loading of pages.
CombineReportPages	Sets a value which indicates that if a report contains several pages, then they will be combined in preview. By default, the property is set to **false**.

Toolbar

Name	Description

Visible	Enables displaying the viewer toolbar. By default, the property is set to **true**.
DisplayMode	Specifies the display mode of the toolbar of the viewer. It can take one of the following values of the **StiToolbarDisplayMode** enumeration: › **Simple** - all controls are located on the same control panel (default value); › **Separated** - the control panel is split into top and bottom panels.
BackgroundColor	Specifies the background color of the viewer toolbar. The default color of the selected theme is used.
BorderColor	Specifies the border color of the viewer toolbar. The default color of the selected theme is used.
FontColor	Specifies the text color for the toolbar and the viewer menu. The default color of the selected theme is used.
FontFamily	Specifies the font for the toolbar and the viewer menu. The default font of the selected theme is used.
Alignment	Sets the alignment mode for the controls on the viewer toolbar. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – elements will be aligned left; › **Center** – elements will be centered; › **Right** – elements will be aligned right; › **Default** – the alignment depends on the RightToLeft property (default value).
ShowButtonCaptions	Enables text of the buttons on the toolbar of the viewer. By default, the property is set to **true**.
ShowPrintButton	Enables showing the button - **Print** - on the viewer toolbar. By default, the property is set to **true**.

ShowOpenButton	Enables displaying the **Open** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to true.
ShowSendEmailButton	Enables showing the button - **Send Email** - on the viewer toolbar. By default, the property is set to **false**. Also, you should add the EmailReport action.
ShowFindButton	Enables showing the button - **Find** - on the viewer toolbar. By default, the property is set to **true**.
ShowBookmarksButton	Enables showing the button - **Bookmarks** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the bookmarks panel will not be displayed even if there are bookmarks in the report.
ShowParametersButton	Enables showing the button - **Parameters** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the parameters panel will not be displayed even if there are parameters in the report.
ShowResourcesButton	Enables showing the button - **Resources** - on the toolbar of the viewer. By default, the property is set to **true**. If the button is hidden, the resources panel will not be displayed even if there are resources in the report.
ShowEditorButton	Enables showing the button - **Editor** - on the viewer toolbar. By default, the property is set to **true**.
ShowFullScreenButton	Enables displaying the **Full Screen** button on the toolbar of the viewer when viewing reports or dashboards. . By default, the property is set

	to **true**.
ShowFirstPageButton	Enables showing the button - **First Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowPreviousPageButton	Enables showing the button - **Previous Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowCurrentPageControl	Enables showing the current report page indicator. By default, the property is set to **true**.
ShowNextPageButton	Enables showing the button - **Next Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowLastPageButton	Enables showing the button - **Last Page** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowZoomButton	Enables showing the button to select the report zoom. By default, the property is set to **true**.
ShowViewModeButton	Enables showing the button to select the view mode of the report page. By default, the property is set to **true**.
ShowDesignButton	Enables displaying the **Design** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to **false**.
ShowAboutButton	Enables showing the button - **About** - on the viewer toolbar. By default, the property is set to **true**.
ShowRefreshButton	Sets a visibility of the **Refresh** button in the toolbar of the viewer. By default, the property is set to **true**.
ShowPinToolbarButton	Enables displaying of the **Pin Toolbar** button on the viewer's toolbar. The button is available only in the Mobile mode of the viewer's interface. The default value of the property is **true**.

PrintDestination	Sets the report printing mode. It can take one of the following values of the **StiPrintDestination** enumeration: › **Default** – a menu with a choice of printing modes will be displayed (default value); › **Pdf** – printing will be done in the PDF format; › **Direct** – printing will be done to the HTML format directly to the printer, the system print dialog will be displayed; › **PopupWindow** – printing will be done in the HTML format via the preview window of the report.
ViewMode	Sets the mode for displaying report pages. It can take one of the following **StiWebViewMode** enumeration values: › **SinglePage** - displays one page of the report selected in the toolbar of the viewer (default value); › **Continuous** - displays all pages of the report; › **MultiplePages** - displays all report pages as a table.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. The values are from 10 to 500 percent. You can also set one of the following values: › **StiZoomMode.PageWidth** – when the viewer runs, the zoom, necessary to display the report by the page width, will be set; › **StiZoomMode.PageHeight** – when the viewer runs, the zoom, necessary to display the report by the page height, will be set.
MenuAnimation	Enables animation when the viewer menu shows/hides. By default the property is set to **true**.
ShowMenuMode	Sets the display mode of the viewer menu. It

	can take one of the following values of the **StiShowMenuMode** enumeration: › **Click** – shows menu by mouse click (default value); › **Hover** – shows menu by hovering the mouse cursor.
AutoHide	Enables auto-hiding of the viewer's toolbar. The property will work only for the Mobile mode of the viewer's interface. The default value of the property is **false**.

Export

Name	Description
DefaultSettings	This group of properties provides the ability to specify the default export settings for each export type. These settings will be applied to the export dialogs when the viewer runs or to the report, if export dialogs are disabled.
StoreExportSettings	Enables saving selected settings in the export dialogs. Settings will be stored in browser cookies. By default the property is set to **true**.
ShowExportDialog	Enables showing the export options dialog box. If the property is set to **false**, the export will be done with the default settings. By default the property is set to **true**.
ShowExportToDocument	Enables the export menu item - **Document File**. By default, the property is set to **true**.
ShowExportToPdf	Enables displaying the **Adobe PDF file** export menu item when viewing reports, and the **Adobe PDF** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToXps	Enables the export menu item - **Microsoft XPS File**. By default, the property is set to **false**.

ShowExportToPowerPoint	Enables the export menu item - **Microsoft PowerPoint 2007/2010 File**. By default, the property is set to **true**.
ShowExportToHtml	Enables the export menu item - **HTML File**. By default, the property is set to **true**.
ShowExportToHtml5	Enables the export menu item - **HTML5 File**. By default, the property is set to **true**.
ShowExportToMht	Enables the export menu item - **MHT Web Archive**. By default, the property is set to **true**.
ShowExportToText	Enables the export menu item - **Text File**. By default, the property is set to **true**.
ShowExportToRtf	Enables the export menu item - **Rich Text File**. By default, the property is set to **true**.
ShowExportToWord2007	Enables the export menu item - **Microsoft Word 2007/2010 File**. By default, the property is set to **true**.
ShowExportToOpenDocumentWriter	Enables the export menu item - **OpenDocument Writer File**. By default, the property is set to **true**.
ShowExportToExcel	Enables the export menu item - **Microsoft Excel File**. By default, the property is set to **true**.
ShowExportToExcelXml	Enables the export menu item - **Microsoft Excel Xml File**. By default, the property is set to **true**.
ShowExportToExcel2007	Enables displaying the **Microsoft Excel 2007/2010 File** export menu item when viewing reports, and the **Microsoft Excel** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToOpenDocumentCalc	Enables the export menu item - **OpenDocument Calc File**. By default, the property is set to **true**.
ShowExportToCsv	Enables the export menu item - **CSV File**. By default, the property is set to **true**.

ShowExportToDbf	Enables the export menu item - **DBF File**. By default, the property is set to **true**.
ShowExportToXml	Enables the export menu item - **XML File**. By default, the property is set to **true**.
ShowExportToDif	Enables the export menu item - **Data Interchange Format (DIF) File**. By default, the property is set to **true**.
ShowExportToSylk	Enables the export menu item - **Symbolic Link (SYLK) File**. By default, the property is set to **true**.
ShowExportToJson	Enables the export menu item - **JSON File**. By default, the property is set to **true**.
ShowExportToImageBmp	Enables displaying the **BMP Image** export menu item when viewing reports, and the **BMP Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageGif	Enables displaying the **GIF Image** export menu item when viewing reports, and the **GIF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageJpeg	Enables displaying the **JPEG Image** export menu item when viewing reports, and the **JPEG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePcx	Enables displaying the **PCX Image** export menu item when viewing reports, and the **PCX Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePng	Enables displaying the **PNG Image** export menu item when viewing reports, and the **PNG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageTiff	Enables displaying the **TIFF Image** export menu item when viewing reports, and the **TIFF Image** item when viewing dashboards. By default, the

	property is set to **true**.
ShowExportToImageSvg	Enables displaying the **Scalable Vector Graphics (SVG) File** export menu item when viewing reports, and the **Scalable Vector Graphics (SVG) File** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvgz	Enables displaying the **Compressed SVG (SVGZ) File** export menu item when viewing reports, and the **Compressed SVG (SVGZ) File** item when viewing dashboards. By default, the property is set to **true**.
ShowOpenAfterExport	Enables displaying the **Open After Export** parameter in export settings menu. By default, the property is set to **true**.

Email

Name	Description
ShowEmailDialog	Enables displaying settings for sending the report via email. If the dialog box is disabled, the email will be sent with the settings set on the server side in the **EmailReport** action. By default the property is set to **true**.
ShowExportDialog	Enables displaying export options dialog box when sending email. If the property is set to **false**, the export will be done with the default settings. By default the property is set to **true**.
DefaultEmailAddress	Sets the default recipient email, i.e. the address to which the email with the attached report will be sent.
DefaultEmailSubject	Sets the default email subject (header).
DefaultEmailMessage	Sets the default email message (text).
DefaultEmailReplyTo	Gets or sets the default text of the replyTo of

	the message created in the viewer.

6.2 HTML5 Designer

Samples

See on GitHub examples of working with the ASP.NET Core Razor HTML5 Designer component. All examples are separate projects, grouped into one solution for Visual Studio.

The **HTML5 Designer** (**StiNetCoreDesigner**) component is designed to create reports in the web browser. You do not need to install the .NET Framework, ActiveX components or any special plug-ins on the client side. All that is needed is any modern Web browser.

With help of **HTML5 Designer** you can create, edit, save and preview reports on any computer with any operating system installed. Since the designer only uses HTML and JavaScript technologies, it can be run on devices where there is no Flash or Silverlight support - tablets, smartphones. Also, the designer supports the Touch interface, which is automatically enabled when using devices with a touch screen.

The **HTML5 Designer** component uses the AJAX technology to perform all actions on reports, which allows you to get rid of reloading the entire page, save Web traffic and speed up work. The report engine built using the .NET Core technology is used to render reports. This is a cross-platform technology. It allows you to deploy the application on servers that use the operating systems like Windows, macOS, and Linux.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use the **HTML5 Designer** in a Web project, you need to install the NuGet

package of Stimulsoft.Reports.Web.NetCore:
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Reports.Web.NetCore in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

To add the ability to create and edit dashboards in a Web project, install the NuGet package Stimulsoft.Dashboards.Web.NetCore (this package is associated with the package Stimulsoft.Reports.Web.NetCore. If it is missed it will be installed automatically):
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Dashboards.Web.NetCore in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

6.2.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the designer, you should place the **StiNetCoreDesigner** component on a page, set the necessary settings, and define the actions in the page event handler. When running the report designer, the following actions occur:

› The .NET Core component generates HTML and JavaScript code that is necessary for displaying and running the designer;

› When the component is output, the JavaScript method is launched. It requests the report template on the server side displays it in the designer window;

› Various actions in the designer (for example, report preview, saving the report template, export reports, sorting, drill-down etc.) calls a certain action on the server side, in which you can perform the necessary manipulations with the report.

6.2.2 Activation

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiNetCoreDesigner** component.

Index.cshtml.cs

```
...
//Activation with using license code
public class IndexModel : PageModel
{
   static IndexModel()
   {
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";
   }
}

//Activation with using license file
public class IndexModel : PageModel
{
   public IndexModel(IWebHostEnvironment webHostEnvironment)
   {
      var path = Path.Combine(webHostEnvironment.ContentRootPath, "Content\
      \license.key");
      Stimulsoft.Base.StiLicense.LoadFromFile(path);
   }
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

6.2.3 Editing Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To edit a report template, you should add the **StiNetCoreDesigner** component to a page, set the minimum required settings for it, and define necessary actions in the page event handler.

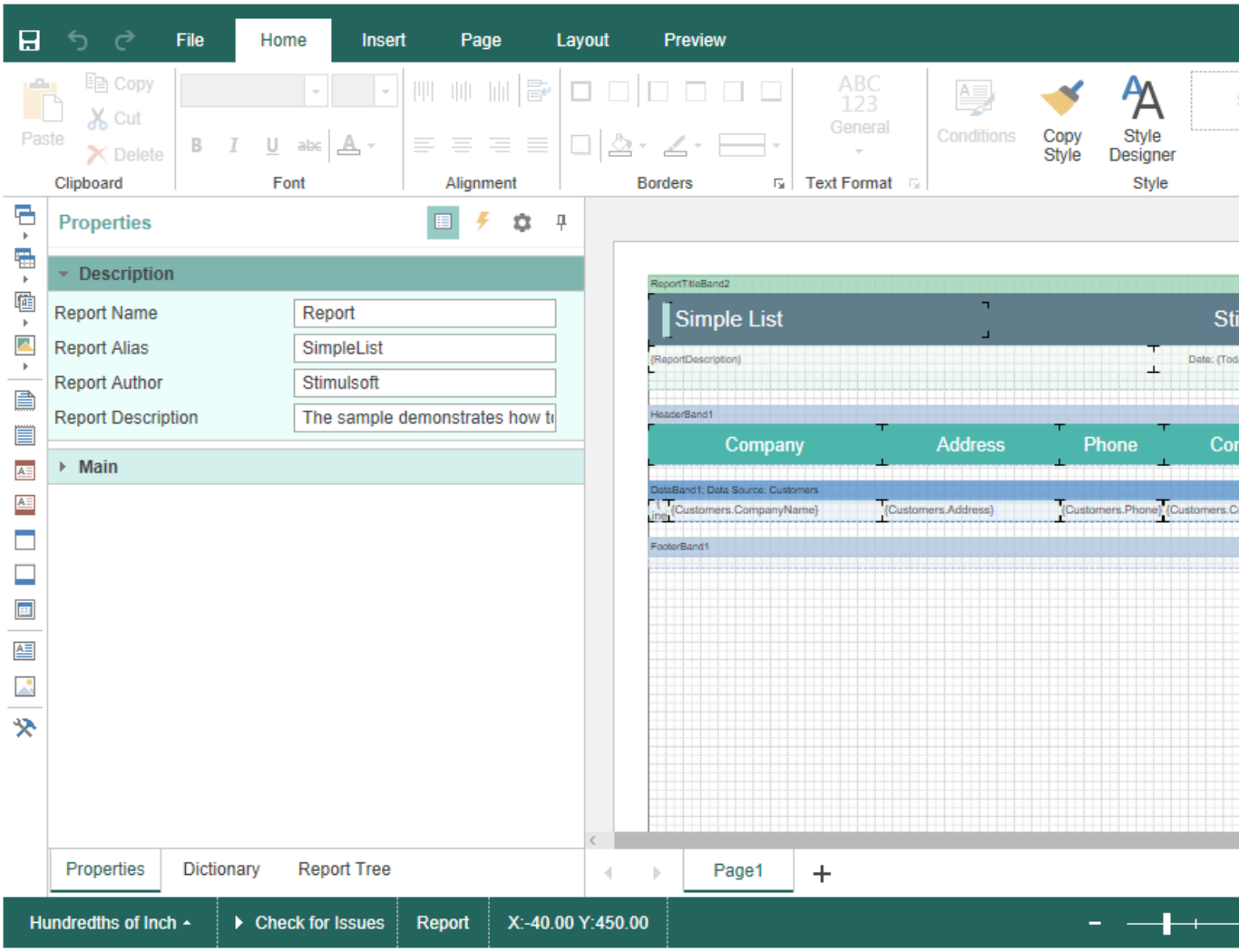
Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      GetReport = "GetReport",
      DesignerEvent = "DesignerEvent"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
   // Create the report object
   StiReport report = new StiReport();

   // Load report or dashboard
   report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));
   //report.Load(StiNetCoreHelper.MapPath(this, "Reports/Dashboard.mrt"));

   return StiNetCoreDesigner.GetReportResult(this, report);
}

public IActionResult OnGetDesignerEvent()
{
   return StiNetCoreDesigner.DesignerEventResult(this);
}

public IActionResult OnPostDesignerEvent()
{
   return StiNetCoreDesigner.DesignerEventResult(this);
}
```

```
. . .
```



The **GetReport** action is used to load an editable report template. It is called automatically after the report designer is loaded. The **DesignerEvent** action is designed to process various additional designer actions, such as working with data and components, previewing reports and others.

Information

The **DesignerEvent** action is mandatory. Without it, the correct work of the designer is impossible. The action is called for two types of requests: **OnGet** – a component requests necessary resources for work, such as CSS styles, JS scripts,

and images; **OnPost** – all other actions of the designer.

6.2.4 Creating New Reports and New Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To run the report designer with a new (empty) report, it is enough to create a new report in the **GetReport** action and return it to the designer. If necessary, you can load data for the report, or perform any other necessary actions.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      GetReport = "GetReport"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
   StiReport report = new StiReport();

   return StiNetCoreDesigner.GetReportResult(this, report);
}
...
```

You can also create a new report using the main menu of the designer. The **CreateReport** action is used to load data for a new report or perform any other necessary actions. This action will be called when creating a new empty report or when creating a report using the wizard.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      CreateReport = "CreateReport"
   }
})
...
```

Index.cshtml.cs

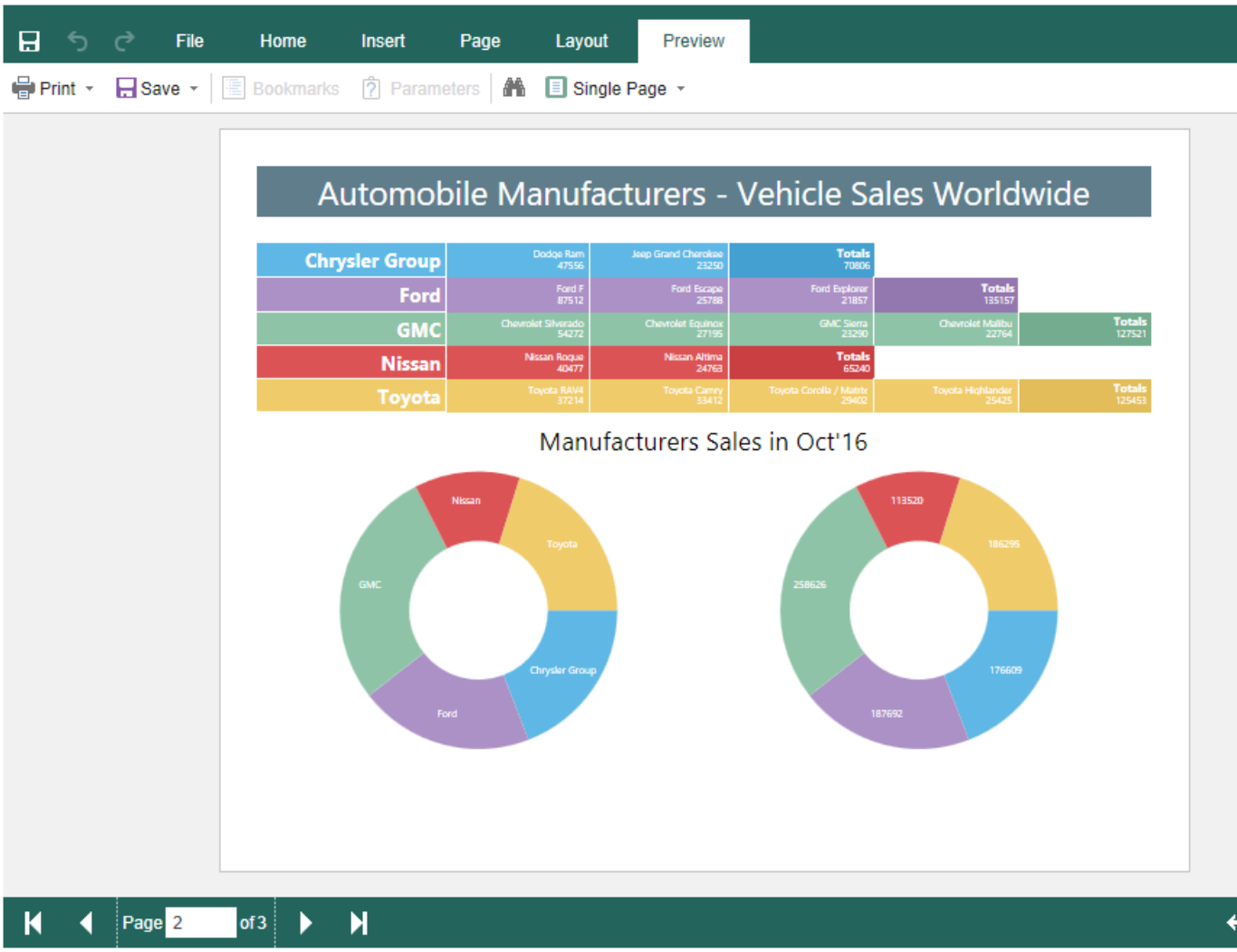
```
...
public IActionResult OnPostCreateReport()
{
   StiReport report = new StiReport();
   //var newDashboard = StiReport.CreateNewDashboard();

   // Register data for the new report, if necessary
   DataSet data = new DataSet("Demo");
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));
   report.RegData(data);
   //newDashboard.RegData(data);
   report.Dictionary.Synchronize();
   //newDashboard.Dictionary.Synchronize();

   return StiNetCoreDesigner.GetReportResult(this, report);
   //return StiNetCoreDesigner.GetReportResult(this, newDashboard);
}
...
```

6.2.5 Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component provides the ability to preview reports. To preview the report, just go to the appropriate tab in the designer window. The report template will be transferred to the server side, rendered and displayed in the embedded viewer.



Before previewing the report, it is possible to perform any necessary actions, for example, connect data for the report. To do this, you can use the special **PreviewReport** action that will be called before previewing the report. The **PreviewReport** action is called before preparing and rendering a report for viewing till its saving to the cash.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      PreviewReport = "PreviewReport"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostPreviewReport()
{
   StiReport report = StiNetCoreDesigner.GetActionReportObject(this);

   DataSet data = new DataSet("Demo");
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));
   report.RegData(data);

   return StiNetCoreDesigner.PreviewReportResult(this, report);
}
...
```

If you need to make actions on your report immediately before displaying the report, you can use the **GetPreviewReport** action, which is called after the request of the prepared report from the cash.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      GetPreviewReport = "GetPreviewReport"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostGetPreviewReport()
{
   StiReport report = StiNetCoreDesigner.GetActionReportObject(this);

   DataSet data = new DataSet("Demo");
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));
   report.RegData(data);
   //report.IsRendered = false;

   return StiNetCoreDesigner.PreviewReportResult(this, report);
}
...
```

Information

So as in this event a prepared report for viewing is transferred, if you need to render again you should set the **report.IsRendered = false** flag.

6.2.6 Additional Features of Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The preview window of the **HTML5 Designer** component has a fully functional interactive **HTML5 Viewer** that can print and export reports, supports working with report parameters, dynamic sorting, interactive reports, collapsing and etc. To use these features, you do not need any additional settings for the report designer.

In any of the above actions, you can work with the report template, for example, change its properties and parameters, connect new data for rendering.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      ExportReport = "ExportReport"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostExportReport()
{
   StiReport report = StiNetCoreDesigner.GetActionReportObject(this);
   // ...

   return StiNetCoreDesigner.ExportReportResult(this, report);
}
...
```

Information

If you do not need any of these additional options to preview the report (for example, exporting or printing a report), you can disable them using the appropriate properties of the **HTML5 Designer** component.

6.2.7 Saving Reports and Dashboards

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component provides two ways of saving the report which are available in the main menu and in the main panel of the designer - **Save Report** and **Save As**. In turn, each of these ways has its own modes and settings.

Saving a report and dashboard on the server side

To save the editable report on the server side, you need to set the **SaveReport** action, which will be called when you select **Save** in the main menu, or click the **Save** button on the main panel of the designer.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      SaveReport = "SaveReport"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostSaveReport()
{
   StiReport report = StiNetCoreDesigner.GetReportObject(this);
```

```
    // Save the report template
    // ...

    return StiNetCoreDesigner.SaveReportResult(this);
}
...
```

This action returns a response to the client side of the designer about the result of saving the report. After saving the report, it is possible to display a dialog box with an error or a text message.

Index.cshtml.cs

```
...
public IActionResult OnPostSaveReport()
{
    StiReport report = StiNetCoreDesigner.GetReportObject(this);

    // Save the report template
    // ...

    // Completion of the report saving with message dialog box
    return StiNetCoreDesigner.SaveReportResult(this, "Some message after
    saving");
    //return Content("{"infoMessage":"Some info message after saving"}");
    //return Content("{\"warningMessage\":\"Some info message after saving
    \"}");
}
...
```

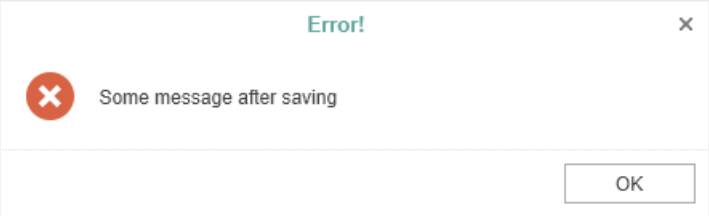
You can get a report name from the designer save dialog or an original report name.

Index.cshtml.cs

```
...
public IActionResult OnPostSaveReport()
{
    var requestParams = StiNetCoreDesigner.GetRequestParams();
    var report = StiNetCoreDesigner.GetReportObject();

    //Report name from designer save dialog
    var savingReportName = requestParams.Designer.FileName;

    //Original report name from properties
    var originalReportName = report.ReportName;

    return StiNetCoreDesigner.SaveReportResult(this, "Some message after
    saving");
}
```

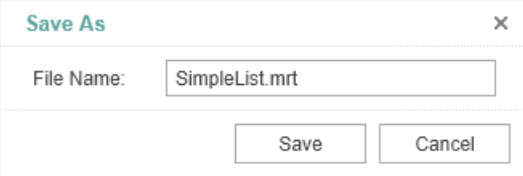
```
...
```

In this case, the dialog with the specified text will be displayed. The text can contain both an error message of saving or a warning, or any other message.



Saving reports and dashboards on the client side

To save the edited report on the client side as a file, no additional designer settings are required. It is enough to click the **Save As** main menu item. The dialog box will be displayed. In this dialog you can change the name of the report file. The file will be saved to the local disk of the computer.



The **HTML5 Designer** component provides the ability to change the behavior of the specified save option. For this purpose, the special **SaveReportAs** action is used in the designer. If you use this event, the report will be saved on the server side. Work of this event will be similar to the **SaveReport** action.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      SaveReportAs = "SaveReportAs"
   }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnPostSaveReportAs()
{
   StiReport report = StiNetCoreDesigner.GetReportObject(this);

   // Save the report template
   // ...

   return StiNetCoreDesigner.SaveReportResult(this);
}
...
```

Use the following code to get the report name from the Save dialog.

Index.cshtml.cs

```
public IActionResult OnPostSaveReportAs()
{
   StiReport report = StiNetCoreDesigner.GetReportObject(this);
   var requestParams = StiNetCoreDesigner.GetRequestParams(this);
   var reportName = requestParams.Designer.FileName;

   return StiNetCoreDesigner.SaveReportResult(this);
}
```

Saving settings

The report is saved in the background mode without reloading the page in the web browser window. If you need to visually control the process of saving the report, you should change the value of the **SaveReportMode** (or **SaveReportAsMode**) property of the designer to one of the three specified values - **Hidden** (default value), **Visible** or **NewWindow**.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Actions =
   {
      SaveReportAs = "SaveReportAs"
   },
   Behavior =
   {
      SaveReportAsMode = StiSaveMode.Visible
```

```
    }
})
...
```

If the **SaveReportMode** property is set to **Visible**, the report save action will be called in the current browser window in the normal (visible) mode using the POST request. If the **SaveReportMode** property is set to **NewWindow**, the report save event will be called in a new window of the web browser. By default, this property is set to **Hidden** - the report save event is called in the background using the AJAX request and is not displayed in the browser window. The same values and behavior are applicable to the **SaveReportAsMode** property.

6.2.8 Localization

The **HTML5 Designer** component supports the complete localization of its interface. Use the special **Localization** property to localize the report designer interface. As a value of this property, you should specify the path to the localization XML file (relative or absolute).

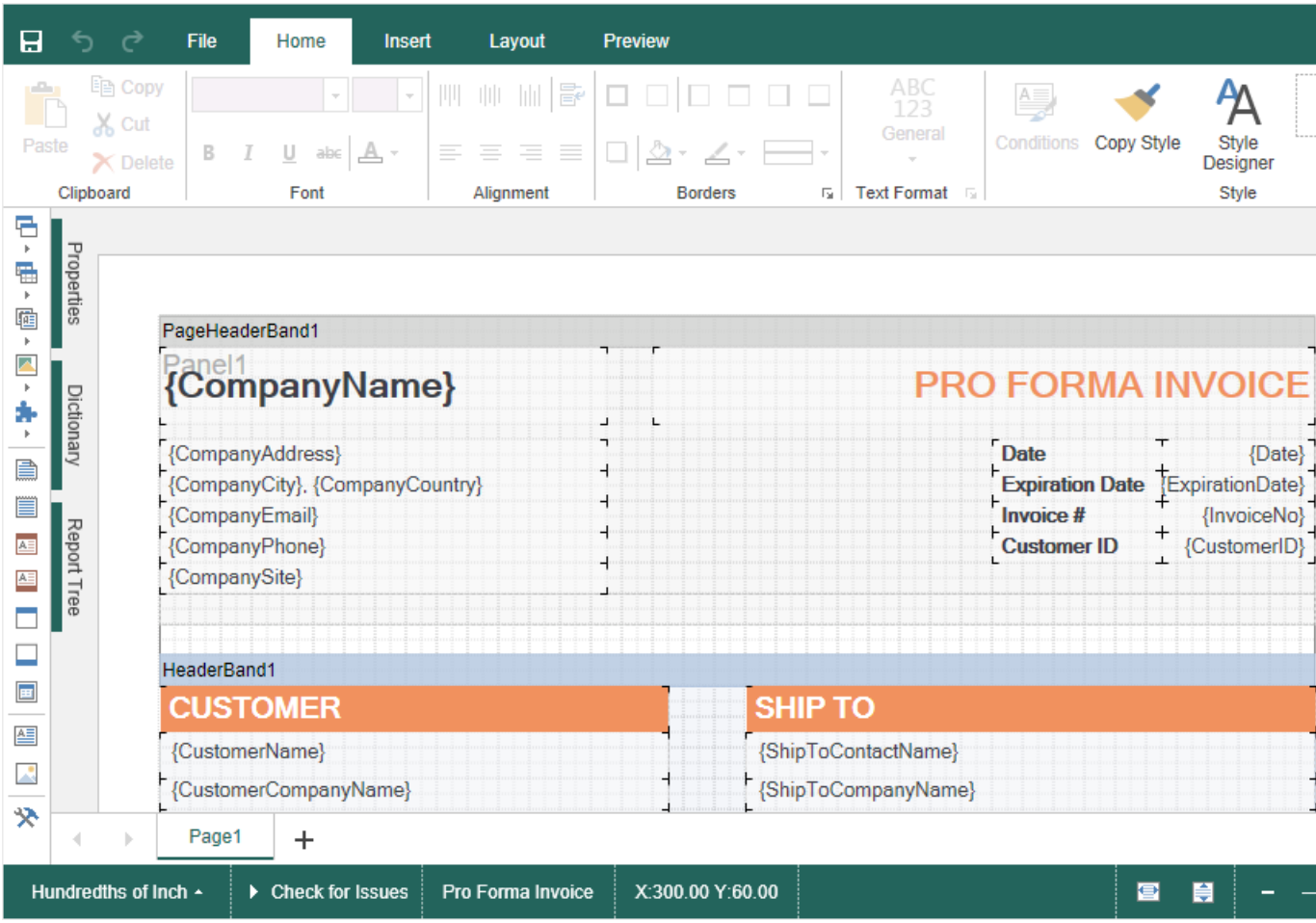
Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
    Localization = "Localization/en.xml"
})
...
```

The interface of the report designer allows you to select the necessary localization from an accessible list. To do this, set value for the **LocalizationDirectory** property as the folder in which the localization XML files are stored.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
    Localization = "Localization/en.xml",
    LocalizationDirectory = "Localization"
})
...
```



Information

If the value for the **Localization** property is set, then when you run the report designer, the localization specified in this property will always be applied. If the property value is not set, the localization selected from the list of available localizations in the report designer panel will be automatically loaded.

6.2.9 Using Themes

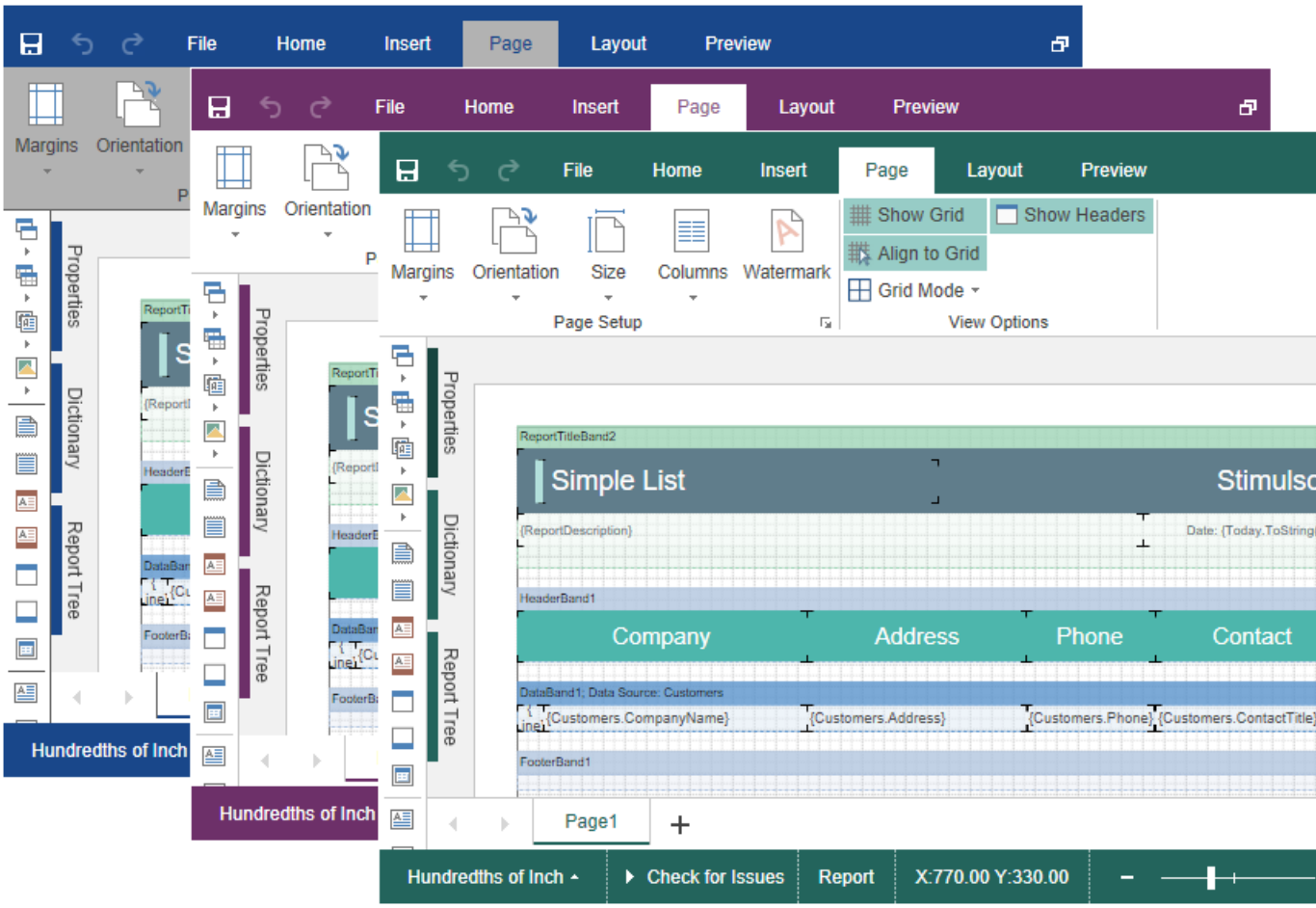
In the **HTML5 Designer** component you can change the appearance of visual controls. To change the theme, you should use the **Theme** property.

Default.aspx

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Theme = StiDesignerTheme.Office2022WhiteTeal
```

```
})
...
```

There are currently **2 themes** available with different color accents. As a result, **more than 50** variants of the appearance are available. This allows you to customize the appearance of the designer for almost any design of the Web project.



6.2.10 Caching

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

HTM5 Designer uses the server cache to store the editable report template. It is necessary because the client part of the designer contains only a visual representation of components of a report template. The report object itself with all the parameters and properties is stored on the server side.

You can manage caching with the following properties.

The CacheMode property

This property of the designer enables caching and sets its type. It can take one of the following values, specified in the **StiServerCacheMode** enumeration:

- **None** – caching is disabled;
- **ObjectCache** – for caching, the server cache is used. The report object is saved in this (set by default);
- **StringCache** – for caching, the server cache is used. The report is saved as a packed string in this cache;
- **ObjectSession** – the current session, in which the report object is saved, is used for caching;
- **StringSession** - for caching, the current session is used, in which the report is saved as a packed string.

The CacheItemPriority property

This property sets the priority of the report stored in the cache of the server. It affects the automatic clearing of the server memory in case of memory shortage. The lower the priority is, the greater is the chance of removing information from memory.

The CacheTimeout property

This property specifies the amount of time in minutes you want to store the report in the server cache. If, when using caching, the requested report is not found in the cache (time of storing this report expired), then it will be requested again using the special **GetReport** action. In this case the unsaved changes may be lost.

The **HTML5 Designer** component provides the ability to specify its own methods for working with report caching. For this purpose, a special **StiCacheHelper** class is used. It contains methods for obtaining a report from the cache and saving the report to the cache. It is necessary to create a new class inherited from **StiCacheHelper** and reload the above methods which respectively have the names - **GetObject, SaveObject** and **RemoveObject**.

Index.cshtml.cs

```
...
public class IndexModel : PageModel
{
   public class StiMyCacheHelper : StiCacheHelper
   {
      public override object GetObject(string guid)
      {
         var path = Path.Combine(HttpContext.Server.MapPath("CacheFiles"),
         guid);
         if (File.Exists(path))
         {
            byte[] cacheData = System.IO.File.ReadAllBytes(path);
            return StiCacheHelper.GetObjectFromCacheData(cacheData);
         }
         return null;
      }

      public override void SaveObject(object obj, string guid)
      {
         byte[] cacheData = StiCacheHelper.GetCacheDataFromObject(obj);
         var path = Path.Combine(HttpContext.Server.MapPath("CacheFiles"),
         guid);
         File.WriteAllBytes(path, cacheData);
      }

      public override void RemoveObject(string guid)
      {
         var path = Path.Combine(HttpContext.Server.MapPath("CacheFiles"),
         guid);
         if (File.Exists(path))
            File.Delete(path);
      }
   }

   static IndexModel()
   {
      StiNetCoreDesigner.CacheHelper = new StiMyCacheHelper();
   }
}
...
```

To initialize the work with report caching using the created class, it is enough to set it as the value of the **StiNetCoreDesigner.CacheHelper** static property in the constructor of the controller.

6.2.11 Additional Methods

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

For **HTML5 Designer**, there are several additional methods that are used to get the object of the currently edited report, parameters of the current state of the designer and other useful data. These methods can be used in any actions of the designer.

The GetReportObject() Method

Returns the report object with which the designer is currently working. It is possible to perform the necessary actions with it - register new data sets, change report properties, assign parameters or load another report to the object. Then, the report can be returned to the designer, specifying it as a parameter in the resulting action method.

Index.cshtml.cs

```
...
public IActionResult OnPostExportReport()
{
   StiReport report = StiNetCoreDesigner.GetReportObject(this);
   report.ReportName = "MyReportName";

   return StiNetCoreDesigner.ExportReportResult(this, report);
}
...
```

The GetActionReportObject() method

Returns the report object that will be used for the particular action. For example, for the **OpenReport** action, this method returns a report loaded from the local disk of

the computer. For the **PreviewReport** action, the method returns a prepared copy of the report for preview.

Index.cshtml.cs

```
...
public IActionResult OnPostOpenReport()
{
   StiReport report = StiNetCoreDesigner.GetActionReportObject(this);

   // Register data for the opened report, if necessary
   DataSet data = new DataSet("Demo");
   data.ReadXml(StiNetCoreHelper.MapPath(this, "Data/Demo.xml"));
   report.RegData(data);
   report.Dictionary.Synchronize();

   return StiNetCoreDesigner.GetReportResult(this, report);
}
...
```

The GetFormValues() method

Returns the values of the form that initiated (opened by the POST request) a page of the designer. Thus, it is possible to get a collection of form parameters in any action of the designer.

Index.cshtml.cs

```
...
public IActionResult OnPostDesignerInteraction()
{
   NameValueCollection formValues = StiNetCoreDesigner.GetFormValues(this);

   return StiNetCoreDesigner.InteractionResult(this);
}
...
```

By default, this feature is disabled to optimize requests of the client-side of the designer to the server. To enable it, set the **PassFormValues** property to **true**.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Server =
   {
      PassFormValues = true
```

```
    }
})
...
```

The GetRequestParams() method

Returns all parameters of the current state of the designer passed to the server side. They can be useful for determining the type of action that the designer is currently executing - for example, to determine the type of export, as well as all action parameters.

Index.cshtml.cs

```
...
public IActionResult OnPostExportReport()
{
   StiRequestParams requestParams =
   StiNetCoreDesigner.GetRequestParams(this);
   if (requestParams.ExportFormat == StiExportFormat.Pdf)
   {
      StiReport report = StiNetCoreDesigner.GetReportObject(this);

      // Some action with report for the PDF export
      // ...

      return StiNetCoreDesigner.ExportReportResult(this, report);
   }

   return StiNetCoreDesigner.ExportReportResult(this);
}
...
```

The GetExportSettings() method

Returns all parameters of the current report export. The type of the parameter object will correspond to the type of export selected in the report preview menu. Any export parameters can be changed and passed to the input of the resulting method. In this case, the report will be exported with the parameters transferred.

Index.cshtml.cs

```
...
public IActionResult OnPostExportReport()
{
   StiExportSettings settings = StiNetCoreDesigner.GetExportSettings(this);
   if (settings.GetExportFormat() == StiExportFormat.Pdf)
```

```
    {
        StiPdfExportSettings pdfSettings = (StiPdfExportSettings)settings;
        pdfSettings.EmbeddedFonts = true;
        pdfSettings.AllowEditable = StiPdfAllowEditable.No;
        return StiNetCoreDesigner.ExportReportResult(this, settings);
    }

    return StiNetCoreDesigner.ExportReportResult(this);
}
...
```

The MapPath() and MapWebRootPath() methods

Returns the absolute path, respectively, to the application or wwwroot directory. You can use this to upload report templates files, data files, etc. These methods are located in the **StiNetCoreHelper** static class.

Index.cshtml.cs

```
...
public IActionResult OnPostGetReport()
{
    StiReport report = new StiReport();
    report.Load(StiNetCoreHelper.MapPath(this, "Reports/SimpleList.mrt"));

    return StiNetCoreDesigner.GetReportResult(this, report);
}
...
```

6.2.12 Timeout

When working with the **StiNetCoreDesigner** component, you can set the timeout for various operations — storing the report in the cache, server response, and query execution. The timeout setting is done using the component properties and report options.

CacheTimeout Property

Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
    Server =
    {
```

```
    CacheTimeout = 10
  }
})
...
```

Using cache will increase the speed of the report designer. See the chapter Caching for more information

RequestTimeout Property

Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
  Server =
  {
    RequestTimeout = 30
  }
})
...
```

CommandTimeout Option

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to set the query timeout for the already created connection, and data sources in the report.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
  Actions =
  {
    GetReport = "GetReport",
    DesignerEvent = "DesignerEvent"
  }
})
...
```

Index.cshtml.cs

```
...
public IActionResult OnGetDesignerEvent()
{
   return StiNetCoreDesigner.DesignerEventResult(this);
}

public IActionResult OnPostDesignerEvent()
{
   return StiNetCoreDesigner.DesignerEventResult(this);
}
...
```

6.2.13 Add custom functions

Information

See on GitHub example of adding a custom function in the ASP.NET MVC HTML5 Designer component.

You can add a custom function to the Dictionary in the report designer when you integrate it into your application. After adding the custom function, you can use this in creating reports and dashboards. Below is the example of adding a function for calculating the sum total.

Index.cshtml.cs

```
...
public static decimal MySum(object value)
{
   if (!ListExt.IsList(value))
   return Stimulsoft.Base.Helpers.StiValueHelper.TryToDecimal(value);

   return Stimulsoft.Data.Functions.Funcs.SkipNulls(ListExt.ToList(value))
   .TryCastToDecimal()
   .Sum();
}
...
static IndexModel()
{
   StiFunctions.AddFunction("MyCategory", "MySum",
   "description", typeof(DesignerController),
   typeof(decimal), "Calculates a sum of the specified set of values.",
   new[] { typeof(object) },
   new[] { "values" },
   new[] { "A set of values" }).UseFullPath = false;
```

```
}
...
```

6.2.14 Settings

The **HTML5 Designer** configuration is done using properties that are located in the **StiNetCoreDesignerOptions** class. All properties are divided into groups, some of the groups contain subgroups of properties for ease of use. Below is an example of setting some properties of the designer.

Index.cshtml

```
...
@Html.StiNetCoreDesigner(new StiNetCoreDesignerOptions() {
   Theme = Stimulsoft.Report.Web.StiDesignerTheme.Office2022WhiteTeal,
   Localization = "Localization/en.xml",
   Actions =
   {
      GetReport = "GetReport",
      PreviewReport = "PreviewReport",
      SaveReport = "SaveReport",
      DesignerEvent = "DesignerEvent"
   },
   Appearance =
   {
      InterfaceType = StiInterfaceType.Auto,
      ShowTooltipsHelp = false,
      ShowDialogsHelp = false,
      DefaultUnit = Stimulsoft.Report.StiReportUnitType.Centimeters
   },
   Dictionary =
   {
      PermissionBusinessObjects =
      Stimulsoft.Report.Web.StiDesignerPermissions.None,
      PermissionDataConnections =
      Stimulsoft.Report.Web.StiDesignerPermissions.View
   },
   Bands =
   {
      ShowChildBand = false,
      ShowEmptyBand = false,
      ShowOverlayBand = false
   }
})
...
```

Basic settings (without groups)

Name	Description
Theme	Specifies the theme of the report designer. The

	list of available themes is located in the **StiDesignerTheme** enumeration. The default value is **Office2022WhiteBlue**.
Localization	Specifies the path to the XML localization file. The path can be absolute or relative. By default, English localization is used. It is built into the designer and does not require additional XML files.
LocalizationDirectory	Specifies the path to the directory with XML localization files. The localization files located in the specified folder will be loaded to the localization list in the designer panel.
Width	Sets the width of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the component is expanded to the entire area of the browser window.
Height	Sets the height of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the component is expanded to the entire area of the browser window.

Actions

Name	Description
GetReport	Specifies the name of the action method to prepare the report template when loading the designer.
OpenReport	Specifies the name of the action method to open the report template from the designer menu.

CreateReport	Specifies the name of the action method to prepare the report template when creating the new report in the designer.
SaveReport	Specifies the name of the action method to save the report template on the server side..
SaveReportAs	Specifies the name of the action method to store the report template on the server side when using the **Save As** menu item. If no action is specified, the built-in method of saving the report template to the local disk will be used.
PreviewReport	Specifies the name of the action method to prepare the rendered report in the preview window.
GetPreviewReport	Specifies the name of the action method just before a report is displayed in the preview window.
ExportReport	Specifies the name of the action method to export reports to the specified format.
Exit	Specifies the name of the action method to go to the desired view by clicking the Exit button in the main menu of the report designer.
DesignerEvent	Specifies the name of the action method of the report designer to handle additional designer actions, such as working with data, report components, and others. Also, this action is used to load scripts and designer styles.

Server

Name	Description
RouteTemplate	Sets the route template that is returned when the report designer actions are executed. If the property is not set, then the Razor project template will be used instead. If the `UseRelativeUrls` property is set to `true`, the

	`BasePath` will not be respected for this property.The default value of this property is null.
ShowServerErrorPage	Enables the display of an HTML page with error details that occurred on the server side. The error details will be displayed in the preview window if the property is enabled. If the property is disabled, the numeric error code and a short error description will be displayed in the dialog window. By default, the property is set to **true**.
AllowAutoUpdateCookies	Allows the designer to update the cookies automatically on every request to the server. By default, cookies are set when creating the designer, if they are not specified in the report. By default, the property is set to **false**.
AllowAntiforgeryToken	Allow the designer to automatically request and send the antiforgery token. By default, the property is set to **true**.
RequestTimeout	Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.
CacheTimeout	Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.
CacheMode	Sets the report caching mode. It can take one of the following values of the **StiServerCacheMode** enumeration: › **None** – caching is disabled in **HTML5 Designer**; › **ObjectCache** – the cache is used as the storage, the report is stored as an object (default value); › **ObjectSession** – the session is used as the

	storage, the report is stored as an object; › **StringCache** – the server cache is used as the storage, the report is serialized to a packed string; › **StringSession** – the session is used as a repository, the report is serialized into a packed string.
CacheItemPriority	Sets the priority of the report stored in the server cache. This property affects the automatic clearing of the server memory in case of lack of memory. The lower the priority is, the greater is the chance of removing information from memory.
AllowAutoUpdateCache	Sets the mode for automatic cache update. The report stored in the cache or server session will be automatically re-saved after a certain period of time if the designer is idle (about every 3 minutes). By default, the property is set to **true**.
UseRelativeUrls	Sets the designer mode in which relative URLs are used for requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value which specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-1** removes the port number.
PassQueryParametersForResources	Enables transferring all request URL parameters when generating links to the resources of the designer. If **false**, only the necessary parameters are used to request the resources of the designer. This corresponds to the more correct operation of the browser cache. By default, the property is set to **true**.
PassFormValues	Enables transferring the POST form values to the client side, if these values are to be used in the actions of the designer. If you enable this feature, the additional **GetFormValues()** method will return a collection of form

	parameters. By default, the property is **false**.
UseCompression	Enables compression of designer requests in the GZip stream. This allows you to reduce the amount of Internet traffic, but slows down the designer. By default, the property is **false**.
UseCacheForResources	Enables caching of the component resources on the server side. The following resources are supported: scripts, styles and images. This option improves the load speed of the component and also reduces the server load in multi-client environments. The default value is **true**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
CustomCss	Specifies the path to the CSS file of styles for the report designer. If this property is set, the standard styles of the selected theme will not be loaded. The default value is an empty value.
DefaultUnit	Sets the units for the size of the report and all its components. By default, centimeters are used.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. It can take one of the following values of the **StiZoomMode** enumeration: › **PageWidth** – when the designer runs, the zoom, necessary to display the report by the page width, will be set; › **PageHeight** – when the designer runs, the

	zoom, required to display the page height of the report, will be set.
ShowAnimation	Enables animation for various elements of the designer interface. By default, the property is set to **true**.
ShowOpenDialog	Allows to display the open dialog, or to open with the open event. By default, the property is set to **true**.
ShowTooltips	Enables displaying tooltips for designer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enable displaying links to online documentation in tooltips for designer controls. By default, the property is set to **true**.
ShowDialogsHelp	Enables displaying links to online documentation on the titles of dialog forms of the designer. By default, the property is set to **true**.
InterfaceType	Sets the type of interface used for the designer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the interface type of the designer will be selected automatically depending on the device used (default value); › **Mouse** – forced use of the interface to control the designer with the mouse; › **Touch** – forced use of the Touch interface to control the designer via the touch screen (mobile devices), also in this mode, the interface elements are increased.
DatePickerFirstDayOfWeek	Sets the first day of the week for the select date item. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Auto** – automatic detection of the first day of the week from the browser settings (default

	value); › **Monday** – the first day of the week is Monday; › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default, the property is set to **false**.
FormatForDateControls	This feature allows you to customize the format for date controls. By default, the current option does not have a specified value, and the date format is determined based on the browser's locale.
ShowReportTree	Enables displaying the tree of report components. By default, the property is set to **true**.
ChartRenderType	Gets or sets the type of the chart in the preview. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages in the preview tab. Can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV

	markup; › **Span** - report items are exported using SPAN markup.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
CloseDesignerWithoutAsking	Sets a value which indicates that the designer will be closed without asking. By default, the property is set to **true**.
ShowSystemFonts	Sets a visibility of the system fonts in the fonts list. By default, the property is set to **true**.
WizardTypeRunningAfterLoad	Calls the Report wizard after starting the report designer. It may have one of the following **StiWizardType** enumeration values: › **None** - runs the report designer without running the report wizard; › **StandardReport** - runs the Standard wizard; › **MasterDetailReport** - runs the Master-Detail wizard; › **LabelReport** - runs the Label wizard; › **InvoicesReport** - runs the Invoice wizard; › **OrdersReport** - runs the Order wizard; › **QuotationReport** - runs the Quote wizard.
AllowWordWrapTextEditors	Allows word wrap in the text editors. By default, the property is set to **true**.

Behavior

Name	Description
ShowSaveDialog	Enables displaying the dialog to insert a report name when it is saved. The name of the report will be transferred in the parameters of the report designer. By default, the property is set to **true**.

UndoMaxLevel	Sets the maximum number to cancel actions with the report (the Undo/Redo function). A big value of this property will consume memory on the server side to store the undo parameters. The default value is **6**.
AllowChangeWindowTitle	Allows using the title of the browser window to display the file name of the edited report. By default, the property is set to **true**.
SaveReportMode	Sets the mode to save the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - saving of the report is called in the current web browser window in the visible mode using the POST request; › **NewWindow** - saving of the report is called in a new window (tab) of the web browser.
SaveReportAsMode	Sets the mode for saving the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - saving of the report is called in the current web browser window in the visible mode using the POST request; › **NewWindow** - saving of the report is called in a new window (tab) of the web browser.
CheckReportBeforePreview	Sets the value that allows running the report checker before preview.

FileMenu

Name	Description
Visible	Enables displaying the main menu of the report designer. By default, the property is set to **true**.
ShowNew	Enables showing the main menu item - **New**. By default, the property is set to **true**.
ShowFileMenuNewReport	Sets a visibility of the new report button in the designer. By default, the property is set to **true**.
ShowFileMenuNewDashboard	Sets a visibility of the new dashboard button in the designer. By default, the property is set to **true**.
ShowOpen	Enables showing the main menu item - **Open**. By default, the property is set to **true**.
ShowSave	Enables showing the main menu item - **Save**. By default, the property is set to **true**.
ShowSaveAs	Enables showing the main menu item - **Save As**. By default, the property is set to **true**.
ShowClose	Enables showing the main menu item - **Close**. By default, the property is set to **true**.
ShowExit	Enables showing the main menu item - **Exit**. By default, the property is set to **false**.
ShowReportSetup	Enables showing the main menu item - **Report Setup**. By default, the property is set to **true**.
ShowOptions	Enables showing the main menu item - **Options**. By default, the property is set to **true**.
ShowInfo	Enables showing the main menu item - **Info**. By default, the property is set to **true**.
ShowAbout	Enables showing the main menu item - **About**. By default, the property is set to **true**.
ShowHelp	Enables showing the main menu item - **Help**. By default, the property is set to **true**.

Dictionary

Name	Description
Visible	Enables showing the data dictionary of the report. By default, the property is set to **true**.
UseAliases	Allows you to use aliases in the data dictionary. It has the three values of the **StiUseAliases** enumeration: › **Auto** - defines the mode of using aliases from a saved value in cookies (default value); › **True** - sets the mode of using aliases in the data dictionary; › **False** - disables the mode of using aliases in the data dictionary.
NewReportDictionary	It allows you to create a new data dictionary or join the existing one when creating a new report in the designer. It has the three values of the **StiNewReportDictionary** enumeration: › **Auto** - defines the mode to create or join the data dictionary from a saved value in cookies (default value); › **DictionaryNew** - sets the mode to create a new data dictionary when creating a new report; › **DictionaryMerge** - sets the mode to join the existing data dictionary with a new one when creating a new report in the designer.
ShowDictionaryContextMenuProperties	Sets a visibility of the **Properties** item in the dictionary context menu. By default, the property is set to **true**.
ShowDictionaryActions	Sets a visibility of the **Actions** menu on the dictionary toolbar. By default, the property is set to **true**.
PermissionDataConnections	Sets the available actions to connect data to the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataSources	Sets available actions on report data sources. It can take one or more values from the

	StiDesignerPermissions enumeration.
PermissionDataColumns	Sets the available actions on data columns in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionBusinessObjects	Sets the available actions on the business objects in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataRelations	Sets the available actions to linking data in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionVariables	Sets available actions on report variables. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionResources	Sets the available actions for the resources in the Report Dictionary. Takes one or several values from the **StiDesignerPermissions** enumeration.
PermissionSqlParameters	Sets the available actions for the parameters of the SQL queries for the Report DataSources. Takes one or several values from **StiDesignerPermissions** enumeration.
DataTransformationsPermissions	Sets the available actions on data transformation. It can take one or more values from the **StiDesignerPermissions** enumeration.

The table below shows all available values for the **StiDesignerPermissions** enumeration, which can be set for the dictionary elements of the report.

Value	Description
None	Disables any action on the item of the data dictionary.
All	Allows any action on the item of the data dictionary.

Create	Allows creating a specific data dictionary item.
Delete	Allows deleting a specific data dictionary item.
Modify	Allows modifying a specific data dictionary item.
View	Allows viewing a specific data dictionary item.
ModifyView	Allows modifying and viewing a specific data dictionary item.

Toolbar

Name	Description
ShowToolbar	Enables displaying the toolbar of the designer. By default, the property is set to **true**.
ShowSetupToolboxButton	Enables displaying the button to call the dialog box with settings for the side toolbar. By default, the property is set to **true**.
ShowInsertButton	Enables displaying the **Insert** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowLayoutButton	Enables displaying the tab **Layout** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowPageButton	Enables displaying the tab **Page** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowPreviewButton	Enables displaying the tab **Preview** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the designer. By default, the property is set to **true**.
ShowAboutButton	Enables displaying the **About** on the toolbar of the designer. By default, the property is set to **false**.

PropertiesGrid

Name	Description
Visible	Enables displaying the property panel. By default, the property is set to **true**.
Width	Sets the width of the property panel. By default, the width is set to **370 px**.
LabelWidth	Specifies the width of the labels on the properties panel. By default, the width is set to **160 px**.
PropertiesGridPosition	Sets **Left** or **Right** position of the properties grid in the designer. It has the three values of the **StiPropertiesGridPosition** enumeration: › **Left**; › **Right**.
ShowPropertiesWhichUsedFromStyles	Sets a visibility of the properties which used from styles in the designer. By default, the property is set to **false**.

Components

Name	Description
ShowText	Enables displaying the **Text** component in the insert menu for report components. By default, the property is set to **true**.
ShowTextInCells	Enables displaying the **Text in Cells** component in the insert menu for report components. By default, the property is set to **true**.
ShowRichText	Enables displaying the **Rich Text** component in the insert menu for report components. By default, the property is set to **true**.
ShowImage	Enables displaying the **Image** component in the

	insert menu for report components. By default, the property is set to **true**.
ShowBarCode	Enables displaying the **Bar Code** component in the insert menu for report components. By default, the property is set to **true**.
ShowShape	Enables displaying the **Shape** component in the insert menu for report components. By default, the property is set to **true**.
ShowHorizontalLinePrimitive	Enables displaying the **Horizontal Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowVerticalLinePrimitive	Enables displaying the **Vertical Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowRectanglePrimitive	Enables displaying the **Rectangle** component in the insert menu for report components. By default, the property is set to **true**.
ShowRoundedRectanglePrimitive	Enables displaying the **Rounded Rectangle** component in the insert menu for report components. By default, the property is set to **true**.
ShowPanel	Enables displaying the **Panel** component in the insert menu for report components. By default, the property is set to **true**.
ShowClone	Enables displaying the **Clone** component in the insert menu for report components. By default, the property is set to **true**.
ShowCheckBox	Enables displaying the **Check Box** component in the insert menu for report components. By default, the property is set to **true**.
ShowSubReport	Enables displaying the **Sub Report** component in the insert menu for report components. By default, the property is set to **true**.
ShowZipCode	Enables displaying the **Zip Code** component in

	the insert menu for report components. By default, the property is set to **true**.
ShowTable	Enables displaying the **Table** component in the insert menu for report components. By default, the property is set to **true**.
ShowCrossTab	Enables displaying the **Cross-Tab** component in the insert menu for report components. By default, the property is set to **true**.
ShowChart	Enables displaying the **Chart** component in the insert menu for report components. It affects on all chart types. By default, the property is set to **true**.
ShowMap	Enables displaying the **Map** component in the insert menu for report components. By default, the property is set to **false**.
ShowGauge	Enables displaying the **Gauge** component in the insert menu for report components. By default, the property is set to **false**.
ShowSparkline	Enables displaying the **Sparkline** component in the insert menu for report components. By default, the property is set to **true**.
ShowMathFormula	Enables displaying the **Math Formula** component in the insert menu for report components. By default, the property is set to **true**.
ShowElectronicSignature	Enables displaying the **Electronic Signature** component in the insert menu for report components. By default, the property is set to **true**.
ShowPdfDigitalSignature	Enables displaying the **PDF Digital Signature** component in the insert menu for report components. By default, the property is set to **true**.

Bands

Name	Description
ShowReportTitleBands	Enables displaying the **Report Title** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowReportSummaryBand	Enables displaying the **Report Summary** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageHeaderBand	Enables displaying the **Page Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageFooterBand	Enables displaying the **Page Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowGroupHeaderBand	Enables displaying the **Group Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowGroupFooterBand	Enables displaying the **Group Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowHeaderBand	Enables displaying the **Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowFooterBand	Enables displaying the **Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowColumnHeaderBand	Enables displaying the **Column Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowColumnFooterBand	Enables displaying the **Column Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowDataBand	Enables displaying the **Data** item in the **Bands** menu of the designer. By default, the property is set to **true**.

ShowHierarchicalBand	Enables displaying the **Hierarchical** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowChildBand	Enables displaying the **Child** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowEmptyBand	Enables displaying the **Empty** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowOverlayBand	Enables displaying the **Overlay** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowTableOfContents	Enables displaying the **Table of Contents** item in the **Bands** menu of the designer. By default, the property is set to **true**.

DashboardElements

Name	Description
ShowTableElement	Enables displaying the **Table** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowCardsElement	Enables displaying the **Cards** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowChartElement	Enables displaying the **Chart** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowGaugeElement	Enables displaying the **Gauge** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowPivotTableElement	Enables displaying the **Pivot** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.

ShowIndicatorElement	Enables displaying the **Indicator** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowProgressElement	Enables displaying the **Progress** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowRegionMapElement	Enables displaying the **Region Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowOnlineMapElement	Enables displaying the **Online Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowImageElement	Enables displaying the **Image** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTextElement	Enables displaying the **Text** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowPanelElement	Enables displaying the **Panel** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowShapeElement	Enables displaying the **Shape** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowButtonElement	Enables displaying the **Button** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowListBoxElement	Enables displaying the **List Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowComboBoxElement	Enables displaying the **Combo Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTreeViewElement	Enables displaying the **Tree View** element in the Dashboard Elements menu of the designer.

	By default, the property is set to **true**.
ShowTreeViewBoxElement	Enables displaying the **Tree View Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowDatePickerElement	Enables displaying the **Date Picker** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.

CrossBands

Name	Description
ShowCrossGroupHeaderBand	Enables displaying the **Cross Group Header** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossGroupFooterBand	Enables displaying the **Cross Group Footer** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossHeaderBand	Enables displaying the **Cross Header** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossFooterBand	Enables displaying the **Cross Footer** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossDataBand	Enables displaying the **Cross Data** section in the insert menu for report components. By default, the property is set to **true**.

Dashboards

Name	Description
ShowNewDashboardButton	Sets a visibility of the **New Dashboard** button

	in the designer. By default, the property is set to **true**.

Pages

Name	Description
ShowNewPageButton	Sets a visibility of the **New Page** button in the designer. By default, the property is set to **true**.

When designing a report or dashboard in the report designer, you can also define **ExportOptions**, **EmailOptions**, and **PreviewToolbarOptions** on the **Preview** tab. These options are similar to the report viewer options.

7 Reports and Dashboards for Blazor

Blazor is a cross-platform technology for creating Web applications for Windows, Linux, and macOS. We offer tools for creating, displaying, converting reports using this technology. Two options of this technology are supported - **Blazor Server**, a distributed system, where all logic is executed on the server-side, and the visual part is displayed on the client; **Blazor WebAssembly**, where all product modules are loaded and run directly in the Web browser window.

Tools for creating and editing reports:

- Blazor Designer

Tools for viewing and converting reports:

- Blazor Viewer

7.1 Viewer

Samples

Get acquainted with the examples of working with the **Blazor Viewer** component. The Blazor Server and the Blazor WebAssembly technologies are available on GitHub. All the samples are separate projects, grouped into one solution for Visual Studio.

The **Blazor Viewer** (**StiBlazorViewer**) component is intended for report viewing in the browser window. At the same time, you don't need to install components or some special plugins on the client. All you need is a modern Web browser. With the help of the **Blazor Viewer**, you can view, print, export reports on any computer with any installed operating system.

The **Blazor Viewer** component is developed as a universal component, and it can work both with the use of the **Blazor Server** and the use of the **WebAssembly** technology. When making all interactive actions on reports, the component requests only the necessary data; it allows to get rid of reloading the entire page, save Web traffic, and increase work speed.

The **Blazor Viewer** supports many design themes, animated interface, bookmarks, interactive reports, editing report elements on the page, full-screen mode, search, and other necessary report viewing features.

To use the **Blazor Viewer** in Web-project, you should install the Nuget package Stimulsoft.Reports.Blazor or Stimulsoft.Dashboards.Blazor:

- Select the «Manage Nuget Packages...» in the context project menu.
- Specify the Stimulsoft.Reports.Blazor, in the search line on the Browse tab;
- Select the element, define the version of a package, and click on the **Install**. When updating a package, you should click on the **Update**.

If for some reason it is not possible, you should add the following assemblies to the project:
Stimulsoft.Base.dll
Stimulsoft.Blockly.dll
Stimulsoft.Data.dll
Stimulsoft.Drawing.dll
Stimulsoft.Map.dll
Stimulsoft.Report.dll
Stimulsoft.Report.Blazor.dll
Stimulsoft.Report.Check.dll
Stimulsoft.Report.Helper.dll
Stimulsoft.Report.Web.dll
Stimulsoft.Report.WebDesign.dll
Stimulsoft.System.dll
Stimulsoft.System.Web.dll

You should additionally add the following assemblies to the project for using dashboards:
Stimulsoft.Dashboard.dll
Stimulsoft.Dashboard.Drawing.dll
Stimulsoft.Dashboard.Export.dll

7.1.1 Activation

After acquiring a Stimulsoft product, you should activate the license for used components. You can do it by specifying your license key or loading a file with your license key. Below is an example of the **StiBlazorViewer** component activation.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer />

@code
{
   protected override void OnInitialized()
   {
      //Activation with using license code
```

```
        Stimulsoft.Base.StiLicense.Key = "Your activation code...";

        //Activation with using license file
        Stimulsoft.Base.StiLicense.LoadFromFile("Content/license.key");

        base.OnInitialized();
    }
}
```

You can get a license key or download a file with a license key in the user's account. To authorize in the user's personal account, you should use a username and password specified when buying a product.

7.1.2 Showing Reports

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `Report.Render()` method if you want to perform specific actions with the rendered report before it is displayed in the viewer. Likewise, when using the compilation mode, you need to call the `Report.Compile()` method only if you have actions to perform with the compiled report before it is built and shown in the viewer.

To display a report, you should add the **StiBlazorViewer** component on the Razor page, add the **StiReport** object, and assign it to the viewer using the Report property. After a report is loaded from a file, it will be displayed in the viewer automatically.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Report="@Report" />

@code
{
    //Report object to use in viewer
    private StiReport Report;

    protected override void OnInitialized()
    {
```

```
        base.OnInitialized();

        //Create empty report object
        var report = new StiReport();

        //Load report template
        report.Load("Reports/TwoSimpleLists.mrt");

        //Assing report object to viewer
        Report = report;
    }
}
```

If a report is not rendered before display, the **BlazorViewer** component will render it automatically. This way, to display a report, you can use various types of reports - report templates and already generated reports.

Loading fonts

The Blazor does not have access to the fonts installed on a computer, so to render a report correctly, you should specify the fonts you used in your report. You can do it with the help of the statistic **StiFontCollection** class by either specifying a file or the Base64 string, which has a necessary font.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Web

<StiBlazorViewer Report="@Report" />

@code
{
    //Report object to use in viewer
    private StiReport Report;

    protected override void OnInitialized()
    {
        base.OnInitialized();

        //Init base font as a file
        Stimulsoft.Base.StiFontCollection.AddFontFile("Fonts/Microsoft Sans
        Serif.ttf", "Segoe UI");

        //Init base font as a Base64 string
        var fontBase64 = "AAEAAAAVAQAABABQRFNJR/W/YxAAA814A...";
        Stimulsoft.Base.StiFontCollection.AddFontBase64(fontBase64, "Segoe
        UI");

        //Create empty report object
        report = new StiReport();
```

```
        //Load report template
        report.Load("Reports/TwoSimpleLists.mrt");

        //Assing report object to viewer
        Report = report;
    }
}
```

7.1.3 Connecting Data

Data for report rendering can be connected in various ways. The easiest way is to keep connection settings in a report template. In addition, data can be connected from a code. You can do it before a report is assigned to the viewer.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Report="@Report" />

@code
{
    //Report object to use in viewer
    private StiReport Report;

    protected override void OnInitialized()
    {
        base.OnInitialized();

        //Load new data from XML file
        var data = new System.Data.DataSet();
        data.ReadXml("Data/Demo.xml");

        //Create and load report template
        var report = new StiReport();
        report.Load("Reports/TwoSimpleLists.mrt");
        report.Dictionary.Databases.Clear();
        report.RegData("Demo", data);

        //Assing report object to viewer
        Report = report;
    }
}
```

Information

At this moment, the SQL data resources are supported only for Blazor Server components.

SQL data sources

The connection parameters to the SQL data source and any other ones can be stored in the report template. Suppose you want to set the connection parameters from the code before rendering the report (for example, for security reasons or depending on the authorized user). In that case, you can use the example below.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Report="@Report" />

@code
{
   //Report object to use in viewer
   private StiReport Report;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      OracleConnection connection = new OracleConnection("Data
      Source=Oracle8i;Integrated Security=yes");
      connection.Open();

      OracleDataAdapter adapter = new OracleDataAdapter();
      adapter.SelectCommand = new OracleCommand("SELECT * FROM Products",
      connection);

      var dataSet = new DataSet("productsDataSet");
      adapter.Fill(dataSet, "Products");

      var report = new StiReport();
      report.Load("Reports/TwoSimpleLists.mrt");
      report.Dictionary.Databases.Clear();
      report.RegData("Products", dataSet);

      //Assing report object to viewer
      Report = report;
   }
}
```

Also, for SQL data sources used in the report, you can specify the Query Timeout in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to change the connection string for MS SQL, adjust the query, set the query timeout for the already created connection and data sources in the report.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Report="@Report" />

@code
{
   //Report object to use in viewer
   private StiReport Report;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      var report = new StiReport();
      report.Load("Reports/Report.mrt");
      ((StiSqlDatabase)
      report.Dictionary.Databases["Connection"]).ConnectionString = @"Data
      Source=server;Integrated Security=True;Initial Catalog=DataBase";
      ((StiSqlSource)
      report.Dictionary.DataSources["DataSourceName"]).SqlCommand = "select
      * from Table where Column = 100";
      ((StiSqlSource)
      report.Dictionary.DataSources["DataSourceName"]).CommandTimeout =
      1000;

      //Assing report object to viewer
      Report = report;
   }
}
```

Information

For SQL data sources of other types, the connection is created similarly, and an adapter corresponding to the data source type is connected. For example, for the MS SQL data source, you should connect SqlDataAdapter. For OracleDataAdapter is required for Oracle. Also, you should specify a connection string that matches the connection type.

You can also use data for designing reports and dashboards obtained from OData storage. In this case, you can do the authorization using a username, user password, or using a token. Authorization parameters are specified in the connection string to the OData storage using the ";" separator.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Report="@Report" />

@code
{
   //Report object to use in viewer
   private StiReport Report;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      var report = new StiReport();

      //Authorization using a user account
      var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
      services.odata.org/V4/Northwind/
      Northwind.svc;AddressBearer=adress;UserName=UserName;Password=Passwor
      d;Client_Id=Your Client ID", false, null);

      //Authorization using a user token
      var oDataDatabase = new StiODataDatabase("OData", "OData", @"https://
      services.odata.org/V4/Northwind/Northwind.svc;Token=Enter your
      token", false, null);

      report.Dictionary.Databases.Add(oDataDatabase);
      oDataDatabase.Synchronize(report);

      //Query with data filter
      ((StiSqlSource)report.Dictionary.DataSources["Products"]).SqlCommand
      = "Products?$filter=ProductID eq 2";

      //Assing report object to viewer
      Report = report;
   }
}
```

The table below shows the connection string templates for different types of data sources.

Data Source	Connection String Template

MS SQL	Integrated Security=False; Data Source=myServerAddress;Initial Catalog=myDataBase; User ID=myUsername; Password=myPassword;
MySQL	Server=myServerAddress; Database=myDataBase;UserId=myUsername; Pwd=myPassword;
ODBC	Driver={SQL Server}; Server=myServerAddress;Database=myDataBase; Uid=myUsername; Pwd=myPassword;
OLE DB	Provider=SQLOLEDB.1; Integrated Security=SSPI;Persist Security Info=False; Initial Catalog=myDataBase;Data Source=myServerAddress
Oracle	Data Source=TORCL;User Id=myUsername;Password=myPassword;
MS Access	Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Password=pass;Data Source=C:\ \myAccessFile.accdb;
PostgreSQL	Server=myServerAddress; Port=5432; Database=myDataBase;User Id=myUsername; Password=myPassword;
Firebird	User=SYSDBA; Password=masterkey; Database=SampleDatabase.fdb;DataSource=myServerAddress; Port=3050; Dialect=3; Charset=NONE;Role=; Connection lifetime=15; Pooling=true; MinPoolSize=0;MaxPoolSize=50; Packet Size=8192; ServerType=0;
SQL CE	Data Source=c:\MyData.sdf; Persist Security Info=False;
SQLite	Data Source=c:\mydb.db; Version=3;
DB2	Server=myAddress:myPortNumber;Database=myDataBase;UID=myUsername;PWD=myPassword; Max Pool Size=100;Min Pool Size=10;
Infomix	Database=myDataBase;Host=192.168.10.10;Serv

	er=db_engine_tcp;Service=1492;Protocol=onsoctcp;UID=myUsername;Password=myPassword;
Sybase	Data Source=myASEserver;Port=5000;Database=myDataBase;Uid=myUsername;Pwd=myPassword;
Teradata	Data Source=myServerAddress;User ID=myUsername;Password=myPassword;
VistaDB	Data Source=D:\folder\myVistaDatabaseFile.vdb4;Open Mode=ExclusiveReadWrite;
Universal(dotConnect)	Provider=Oracle;direct=true;data source=192.168.0.1;port=1521;sid=sid;user=user;password=pass
MongoDB	mongodb://<user>:<password>@localhost/test
OData	http://services.odata.org/v3/odata/OData.svc/
Other...	The table shows the most commonly used templates for the connection string. You can view various connection string options at the special website.

Data from XML, JSON, Excel files

You can keep connections to the XML and the JSON data resources in a report template. If you need to specify data files from a code, you can use the following example.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Report="@Report" />

@code
{
   private StiReport Report;

   protected override void OnInitialized()
   {
```

```
        base.OnInitialized();

        //Load new data from XML file
        var dataSet = new System.Data.DataSet();
        dataSet.ReadXml("Data/Demo.xml");

        var report = new StiReport();
        report.Load("Reports/SimpleList.mrt");

        //Register data for the report
        report.RegData("Demo", dataSet);

        //Assing report object to viewer
        Report = report;
    }
}
```

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Report="@Report" />

@code
{
    private StiReport Report;

    protected override void OnInitialized()
    {
        base.OnInitialized();

        //Load new data from JSON file
        var dataSet = StiJsonToDataSetConverterV2.GetDataSetFromFile("Data/
        Demo.json");

        var report = new StiReport();
        report.Load("Reports/SimpleList.mrt");

        //Register data for the report
        report.RegData(dataSet);

        //Assing report object to viewer
        Report = report;
    }
}
```

Information

There is an option to retrieve data from an Excel file in the viewer. To do this you can use the following method:

```
var dataSet = StiExcelConnector.Get().GetDataSet(new StiExcelOptions(a
```

7.1.4 Localization

The Blazor Viewer component supports the localization of its interface. To localize the report viewer interface to the language you need, use the special **Localization** property. You should specify the path to the localization XML file (relative or absolute) as the value of this property (relative or absolute).

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Localization="Localization/en.xml" />
```

When loading the report viewer, the localization file will be loaded automatically.

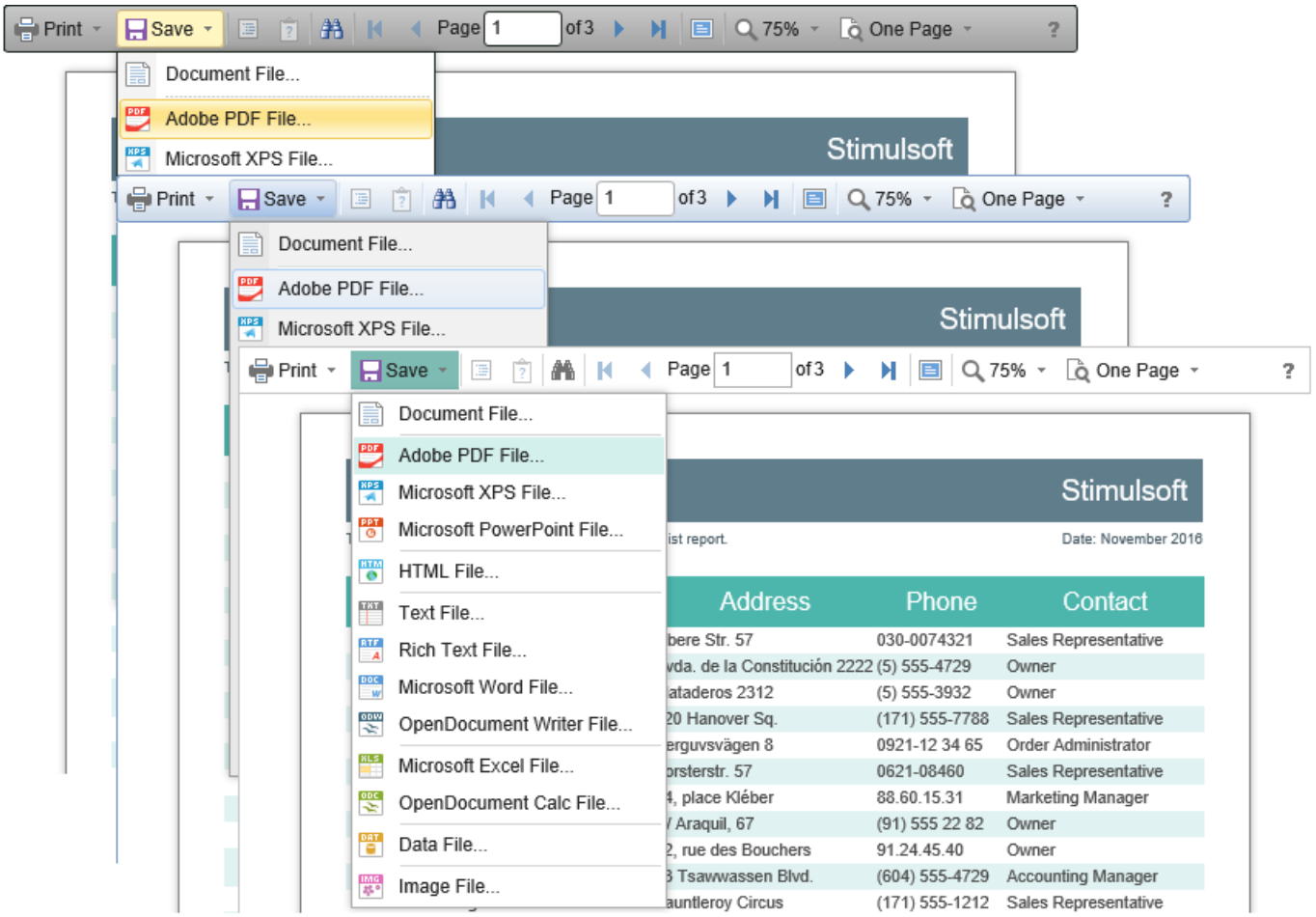
7.1.5 Using Themes

The **Blazor Viewer** component has an option to change the design themes of controls. The **Theme** property in the component options is used to change a theme. It may have one of the values of the **StiViewerTheme** enumeration.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Theme="StiViewerTheme.Office2022WhiteCarmine" />
```

There are currently **8 themes** available with different color accents. As a result, **more than 60** variants of the appearance are available. This allows you to customize the appearance of the viewer for almost any design of the Web project.

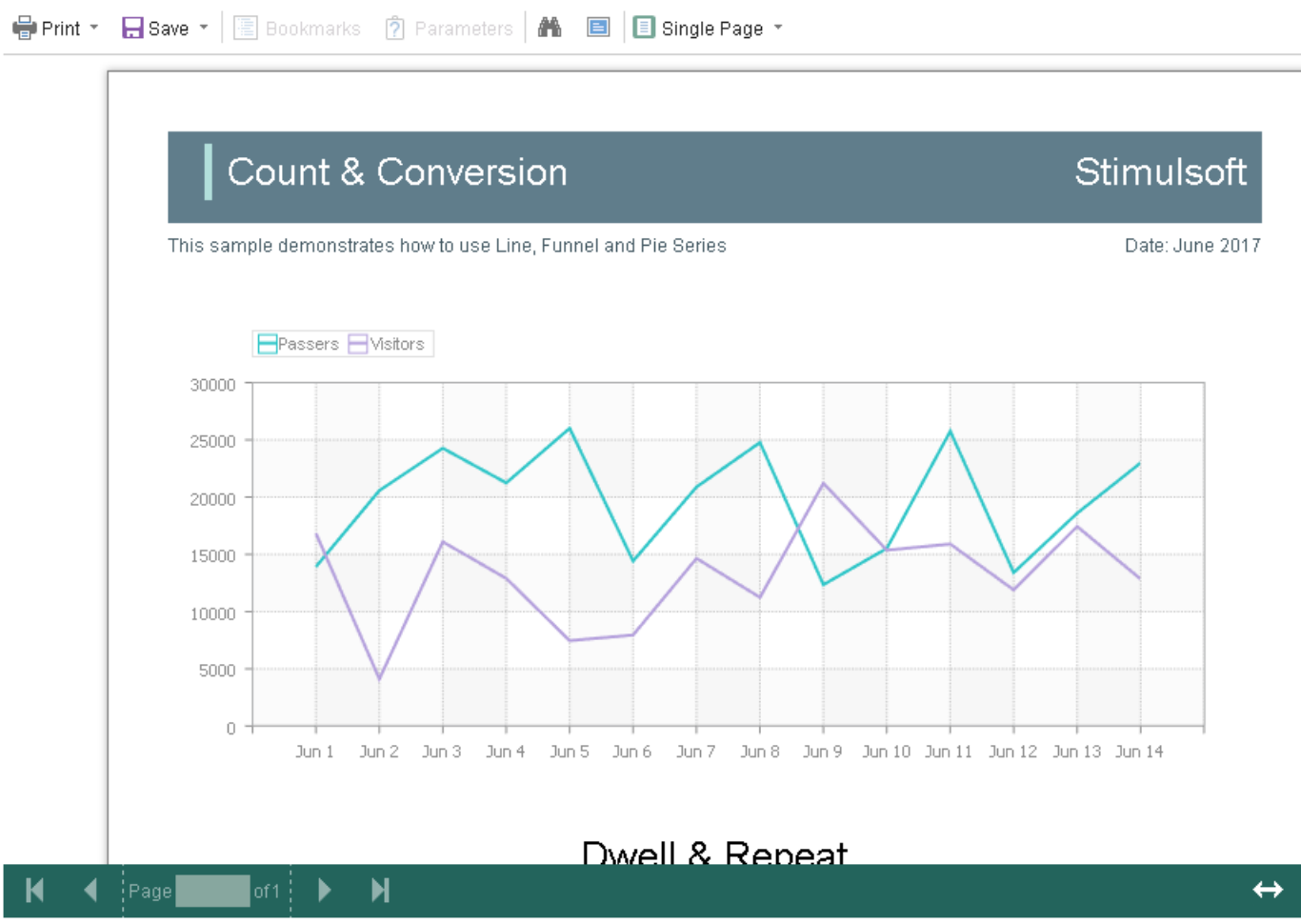


By default, the Viewer displays only the top toolbar, where there are all controls. If it's necessary, you can divide the toolbar into the top and the lower. The menu of printing and exporting reports will be on the top panel, also the buttons to work with parameters and bookmarks. The lower toolbar will contain elements to switch between report pages and the zoom control menu. The **DisplayMode** property is intended for enabling the specified mode, which can have the **Simple** value mode and the **Separated** mode.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
```

```
    //Options object
    private StiBlazorViewerOptions Options;

    protected override void OnInitialized()
    {
        base.OnInitialized();

        //Init options object
        Options = new StiBlazorViewerOptions();
        Options.Appearance.ScrollbarsMode = true;
        Options.Toolbar.DisplayMode = StiToolbarDisplayMode.Separated;
    }
}
```



Additionally, you may set the parameters of the main viewer elements. For example, you can change the font and color of the Viewer toolbar titles, set the background of the Viewer, color, etc. Below is the list of available properties, which change the Viewer design and their value by default.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web
@using System.Drawing

<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorViewerOptions();
      Options.Appearance.BackgroundColor = Color.White;
      Options.Appearance.PageBorderColor = Color.Blue;
      Options.Appearance.ShowPageShadow = true;
      Options.Toolbar.BackgroundColor = Color.White;
      Options.Toolbar.BorderColor = Color.Gray;
      Options.Toolbar.FontColor = Color.Black;
      Options.Toolbar.FontFamily = "Arial";
   }
}
```

7.1.6 Basic Features

The main options of the **Blazor Viewer** contain the following operations: report display, switching between report pages, zoom changing and report display mode. All specified operations are performed without reloading browser page. You don`t need to setting any special options or events to perform them.

The report viewer has a special event the **OnViewerEvent**, which will be invoked after any action of the viewer. At this event you can find out the type of action, which the viewer is performing at the moment and get all parameters of the viewer, transferred to the server side.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer OnViewerEvent="@OnViewerEvent" />

@code
{
   private void OnViewerEvent(StiReportDataEventArgs args)
```

```
    {
        var action = args.Action;
        var report = args.Report;
        var parameters = args.RequestParams;
    }
}
```

Information

The event won't be invoked for events that have their own processors - printing, export, interactive actions on a report etc. These events are described separately in the relevant sections of the documentation.

7.1.7 Printing Reports

Several options of report printing are envisaged in the **Blazor Viewer** component. Each of them has its features, advantages, and disadvantages.

Print to PDF
Printing will be performed with the help of report exporting to **PDF format**. Advantages include great accuracy of location and report elements printing in comparison with other printing options. Among the disadvantages, we can mention the obligatory presence of a plug-in installed in the browser for viewing PDF files (modern Web browsers have an embedded tool for viewing and printing PDF files).

Print with Preview
Report printing will be performed in the separate pop-up window of a browser in the **Blazor Viewer**. A report can be previewed and then sent to the printer or copied to another place as text or an HTML code. Its advantages contain cross-browser compatibility when printing, the absence of need for special plug-ins installation. However, there is one disadvantage; it is relatively low accuracy of report elements location, tied features of the implementation of HTML formatting.

Print without Preview
Report printing will be performed directly in the printer without preview. After selection of this, system print dialog displays. So as printing in this mode is done in HTML, print quality is analogous to report printing quality with the Preview.

Information

When printing to HTML, you should make sure a report page parameters correspond to the printer page parameters (the size of paper, orientation, fields, indents) and check such browser print settings as indents, headers, and footers, the printing of background images, color printing.

You don't need additional settings of the Viewer for print functions to work. If you need to make some actions before a report print, you can specify the special **OnPrintReport** event.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer OnPrintReport="@OnPrintReport" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   private void OnPrintReport(StiPrintReportEventArgs args)
   {
      // Some code before print
      // ...
   }
}
```

Report printing setting

The menu with print options is displayed when selecting a report print in the Viewer panel. The **Blazor Viewer** component has a feature to set the requested print mode forcibly. To use this option, you should set the **PrintDestination** property to one of the **StiPrintDestination** enum values specified below.

- **Default** – when selecting a printing, the menu (property value by default) will be displayed;
- **Pdf** – printing in PDF format;
- **Direct** – printing in an HTML format directly to the Printer, the systemic print dialog will be displayed;
- **WithPreview** – printing in HTML format with the Preview in a pop-up window.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;


   protected override void OnInitialized()
   {
      base.OnInitialized();

      Options = new StiBlazorViewerOptions();
      Options.Toolbar.PrintDestination = StiPrintDestination.Default;
   }
}
```

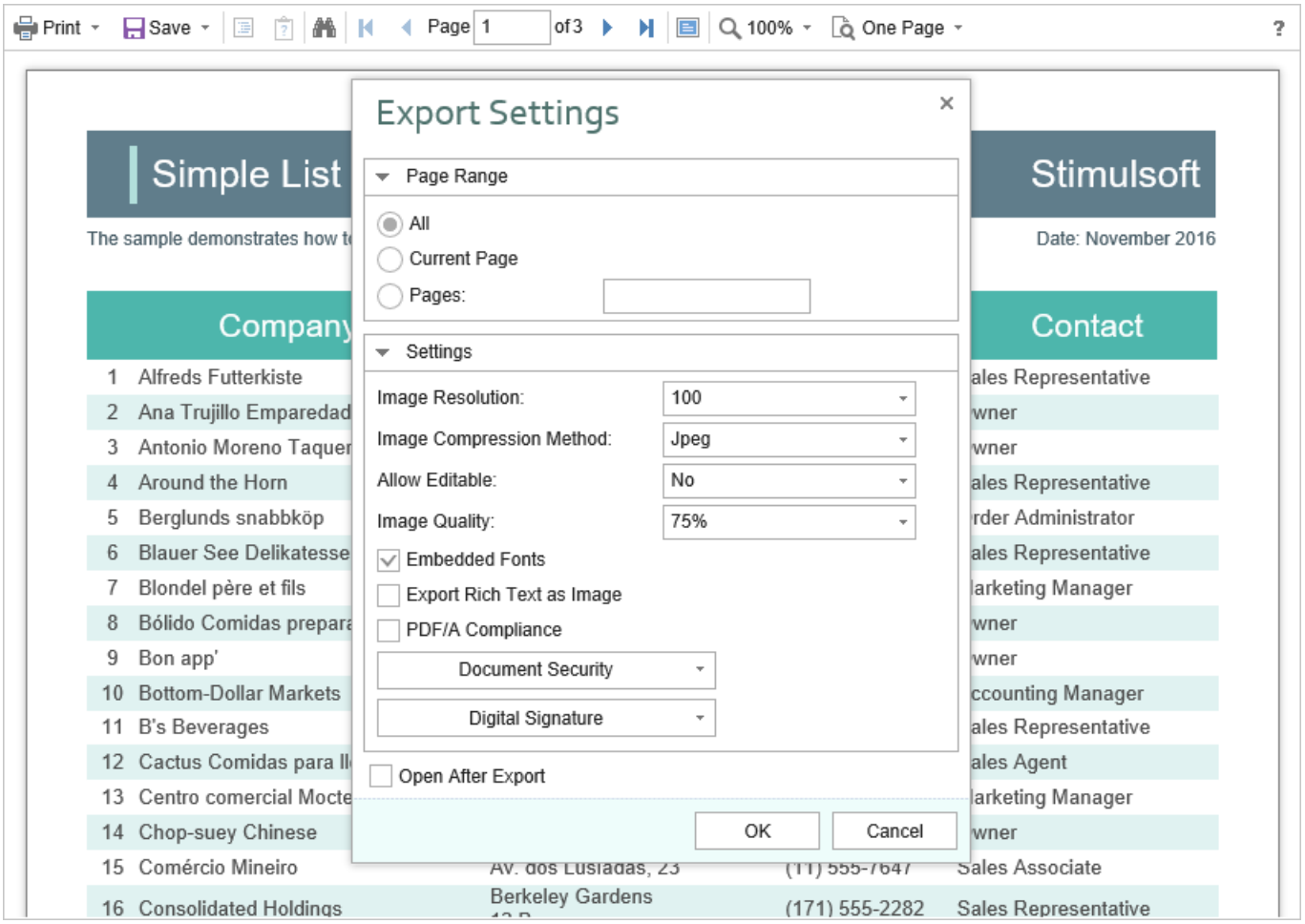
The HTML5 component has a feature, which allows you to disable report printing. To do this, you should set the **ShowPrintButton** property to **false**.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options"></StiBlazorViewer>

@code
{
   //Options object
   private StiBlazorViewerOptions Options;


   protected override void OnInitialized()
   {
      base.OnInitialized();

      Options = new StiBlazorViewerOptions();
      Options.Toolbar.ShowPrintButton = true;
   }
}
```

7.1.8 Exporting Reports

The **Blazor Viewer** component allows you to export a displayed report to various formats such as **PDF**, **HTML**, **Word**, **Excel**, text, etc. The export function doesn`t require additional settings in the viewer.



If you need to make any actions before exporting a report, you can set the special **OneExportReport** event.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer OnExportReport="@OnExportReport" />

@code
{
   private void OnExportReport(StiExportReportEventArgs args)
   {
      // Some code before export
      // ...
   }
}
```

Export Settings

Each report export format of the Blazor Viewer has a lot of settings, and each setting has its values by default. Sometimes you need other values by default. The special **DefaultSettings** property of the Viewer is used for this. You can find it in the export options. This property is the container of all export settings used by default.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorViewerOptions();

      //PDF default settings
      Options.Exports.DefaultSettings.ExportToPdf.ImageQuality = 0.75f;
      Options.Exports.DefaultSettings.ExportToPdf.ImageFormat =
      Stimulsoft.Report.Export.StiImageFormat.Color;

      //HTML default settings
      Options.Exports.DefaultSettings.ExportToHtml.UseEmbeddedImages =
      true;
      Options.Exports.DefaultSettings.ExportToHtml.ExportMode =
      Stimulsoft.Report.Export.StiHtmlExportMode.Div;
   }
}
```

If required, you can completely hide the display of the export dialog windows. The exporting will always be done with the settings by default. To do this, you just need to set the false value for the **ShowExportDialog** property.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
```

```
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorViewerOptions();
      Options.Exports.ShowExportDialog = false;
   }
}
```

The **Blazor Viewer** component contains about 20 various export formats, and sometimes you need to disable some of them. It allows you to load the interface and simplify the use of the Viewer. To disable not used export formats, just set the **false** value for corresponding viewer properties, given in the list below.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorViewerOptions();
      Options.Exports.ShowExportDialog = true;
      Options.Exports.ShowExportToDocument = true;
      Options.Exports.ShowExportToPdf = true;
      Options.Exports.ShowExportToXps = true;
      Options.Exports.ShowExportToPowerPoint = true;
      Options.Exports.ShowExportToHtml = true;
      Options.Exports.ShowExportToHtml5 = true;
      Options.Exports.ShowExportToMht = true;
      Options.Exports.ShowExportToText = true;
      Options.Exports.ShowExportToRtf = true;
```

```
        Options.Exports.ShowExportToWord2007 = true;
        Options.Exports.ShowExportToOpenDocumentWriter = true;
        Options.Exports.ShowExportToExcel = true;
        Options.Exports.ShowExportToExcelXml = true;
        Options.Exports.ShowExportToExcel2007 = true;
        Options.Exports.ShowExportToOpenDocumentCalc = true;
        Options.Exports.ShowExportToCsv = true;
        Options.Exports.ShowExportToDbf = true;
        Options.Exports.ShowExportToXml = true;
        Options.Exports.ShowExportToDif = true;
        Options.Exports.ShowExportToSylk = true;
    }
}
```

The **Blazor Viewer** component has a feature, which allows you to disable the report export menu. To do this, you should set the **ShowSaveButton** property to **false**.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
    //Options object
    private StiBlazorViewerOptions Options;

    protected override void OnInitialized()
    {
        base.OnInitialized();

        //Init options object
        Options = new StiBlazorViewerOptions();
        Options.Toolbar.ShowSaveButton = false;
    }
}
```

7.1.9 Vieweing Modes

There are two modes of report display - with scroll bars and without them. The mode to view reports without scroll bars is set by default. To disable the viewing mode with scroll bars, you should set the **ScrollbarsMode** property to **true**.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web
```

```
<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorViewerOptions();
      Options.Appearance.ScrollbarsMode = false;
   }
}
```

The Viewer displays a page or a report entirely, automatically stretching the viewer in the first mode (without scroll bars). If the sizes in width and heights are specified, the Viewer will crop the page that has gone beyond the margins. In the second mode, unlike the first one, the cropping won`t be made when a page goes beyond the Viewer size. Instead, scroll bars appear, with the help of them, you can view the entire page or a report.

Information

You should set the height of the Viewer in the report viewing mode; otherwise, the height is **650 pixels** will be set by default.

The mode of a report or a dashboard full-screen display is envisaged in the **Blazor Viewer** component. The standard viewing mode is enabled by default. The Viewer has specified sizes in settings. To enable the full-screen mode of viewing, just set the **true** value for the **FullScreenMode** property.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
```

```
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorViewerOptions();
      Options.Appearance.FullScreenMode = true;
   }
}
```

Also, to enable or disable the full-screen mode, you can use the corresponding button in the Viewer toolbar.

There are three report display modes in the **Blazor Viewer** - page display, full report as a ribbon, and tabular display of report pages. The **View Mode** property is used for this. This property accordingly takes one of the following values - **SinglePage**, **Continuous**, **MultiplePages**.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorViewerOptions();
      Options.Toolbar.ViewMode = StiWebViewMode.SinglePage;
   }
}
```

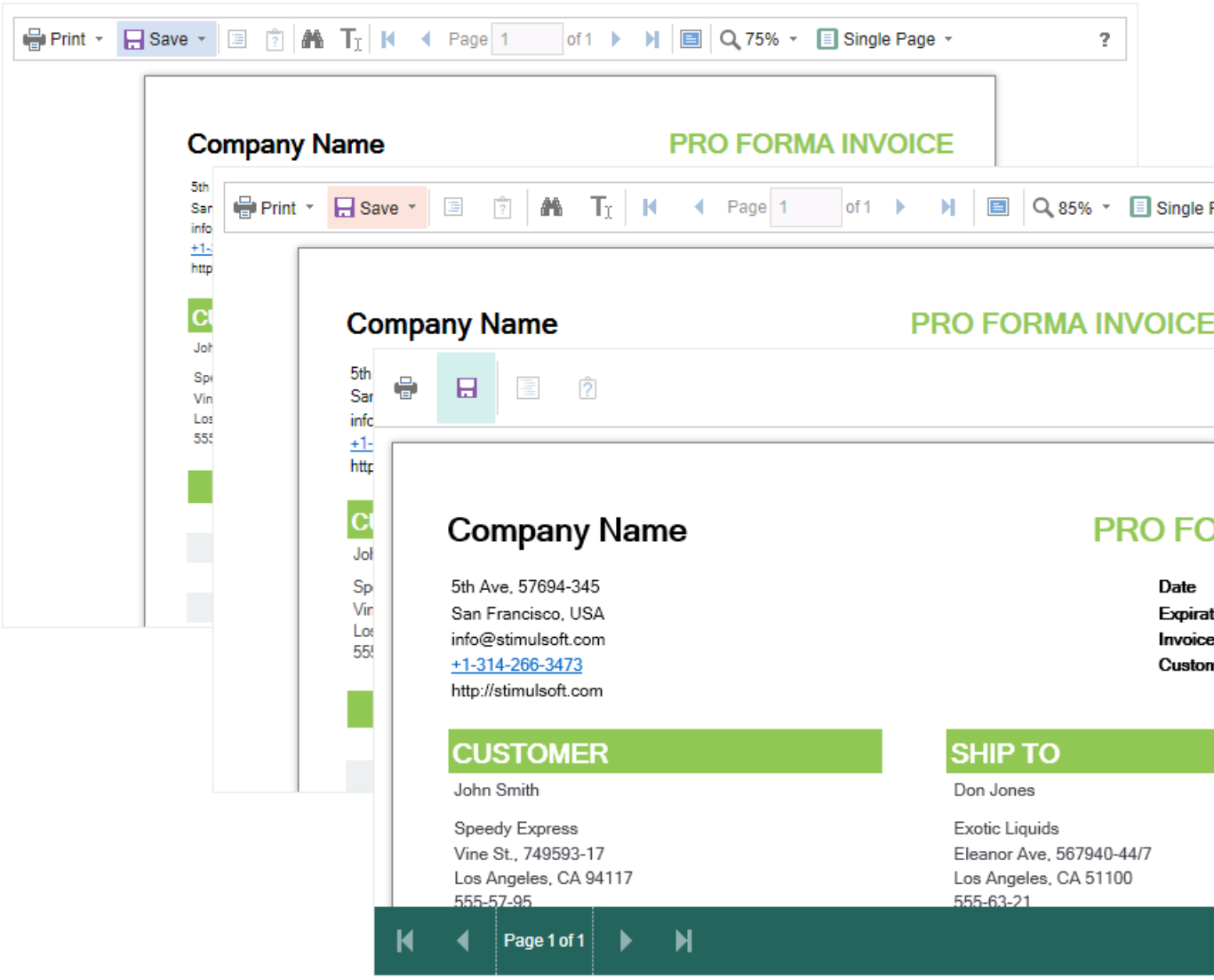
The support of work with a simple computer as well as with touchscreens and mobile devices is realized in the **Blazor Viewer** component. The Interface type property is intended for managing modes. This property takes one of the following

values:

- **Auto** - the Viewer interface type will be selected automatically depending on the device used (value by default).
- **Mouse** - forced using the standard interface for managing the viewer.
- **Touch** - forced using the Touch interface for managing the viewer with the help of the touchscreen monitor. In this mode, the viewer interface elements have increased sizes for comfortable managing.
- **Mobile** - forced use of Mobile interface for managing viewer with the help smartphone screen, in this mode the viewer interface has a simplified view and adapted for managing with the help of a mobile device.

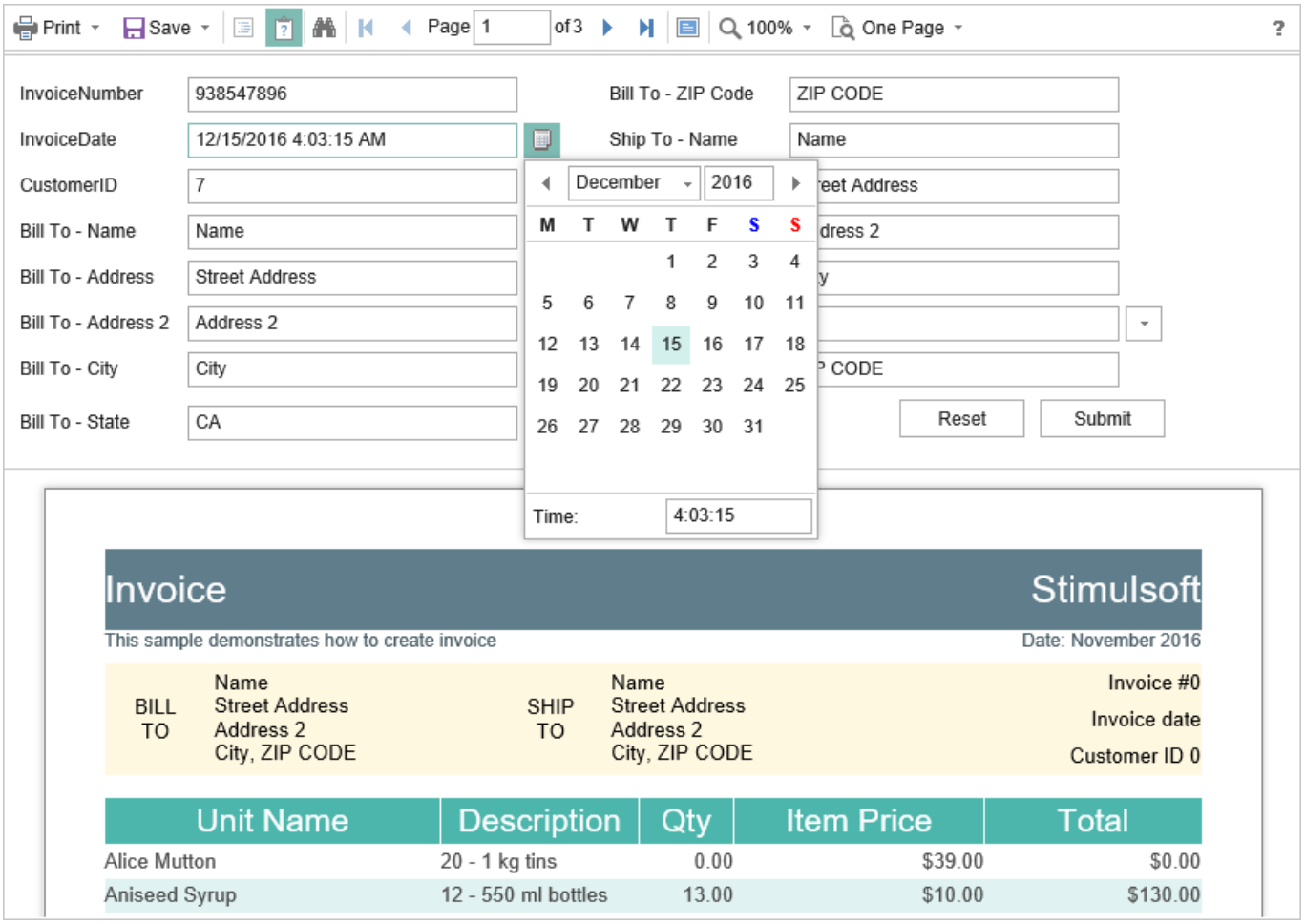
Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorViewerOptions();
      Options.Appearance.InterfaceType = StiInterfaceType.Auto;
   }
}
```



7.1.10 Work with Parameters

The support of a particular parameter panel for work with report parameters in the **Blazor Viewer** is realized. To add a parameter to the panel, you should define the variable requested from a user in a report. When viewing a report in the viewer, this variable will be added automatically to the parameter panel — all types of report variables (simple variables, date and time, borders, lists, etc.).



Additional viewer settings are not required for working reports with parameters. If you need to take some actions before using parameters, you can define the special **OnInteraction** event.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer OnInteraction="@OnInteraction" />

@code
{
   private void OnInteraction(StiReportDataEventArgs args)
   {
      // Some code before any interaction
      // ...
   }
}
```

This event is triggered for any interactive viewer actions. If you need to make some actions only when applying report parameters, you can use value parameters in argument events. The arguments contain all necessary data and conditions of the viewer client side. To define a type of viewer actions, you should use the **Action** property.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer OnInteraction="@OnInteraction" />

@code
{
   private void OnInteraction(StiReportDataEventArgs args)
   {
      if (args.Action == StiAction.Variables)
      {
         // Some code before apply parameters
      }
   }
}
```

If the work with parameters is requested, you can disable this feature. The **ShowParametersButton** property is intended for this. This property is located in the **Toolbar** properties section, and the **false** value must be set for it.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
```

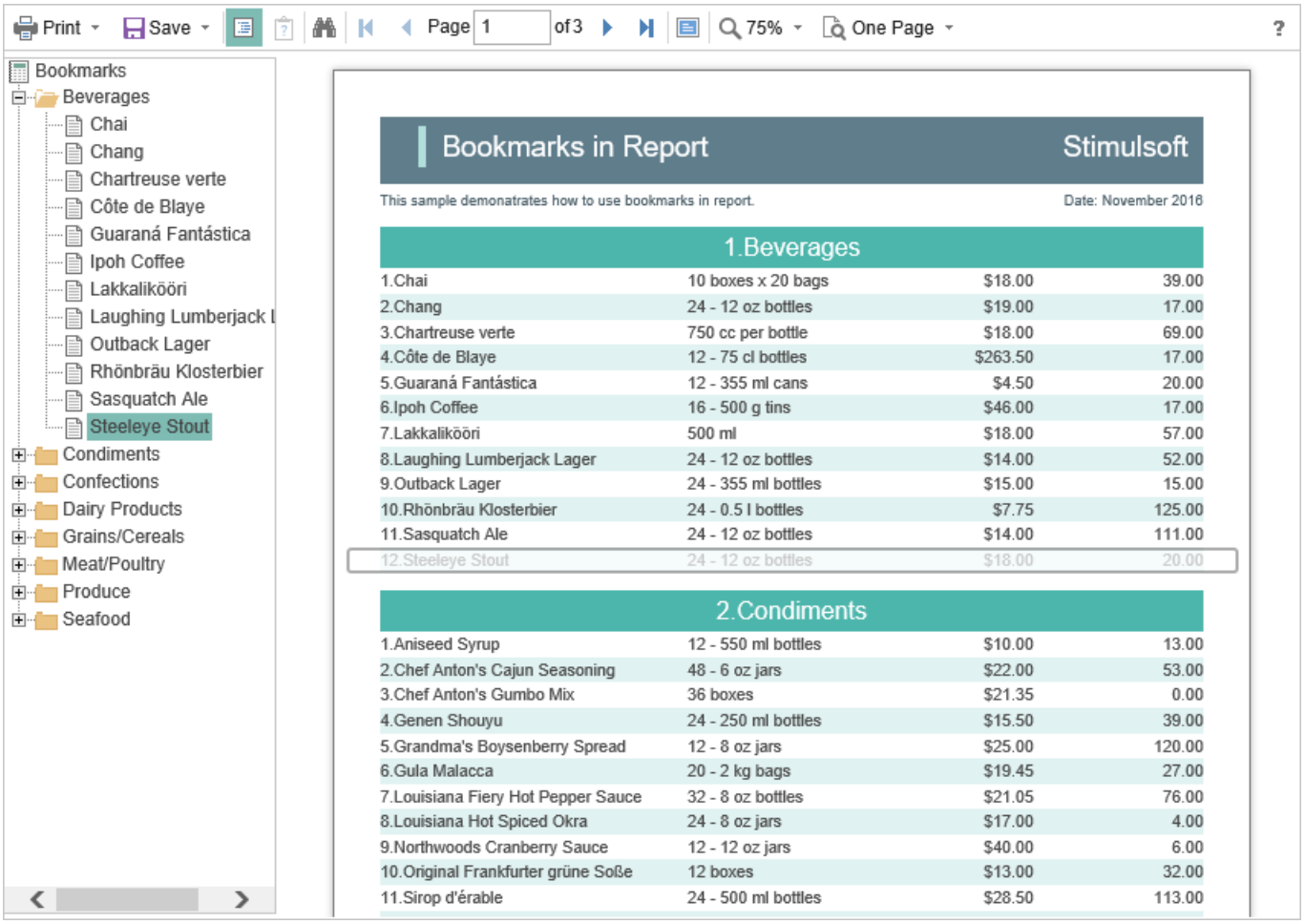
```
        Options = new StiBlazorViewerOptions();
        Options.Toolbar.ShowParametersButton = false;
    }
}
```

Information

Due to this viewer configuration, the parameter panel won't be shown, even if the parameters are present in a displayed report.

7.1.11 Work with Bookmarks

The support of report bookmarks is implemented in the **Blazor Viewer** component. When displaying a report, the panel with bookmarks will be displayed to the left of the page. When selecting a bookmark for a report, the viewer will automatically transit to the page you need, and the report element with a bookmark will be highlighted.



By default, the width of the bookmark panel is 180 pixels; the **Blazor Viewer** component allows you to change this value. The **BookmarksTreeWidth** is intended for this. Its value is specified in pixels.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();
```

```
    //Init options object
    Options = new StiBlazorViewerOptions();
    Options.Appearance.BookmarksTreeWidth = 200;
  }
}
```

If the work with report bookmarks is not requested, you can completely disable this feature. The **ShowBookmarksButton** property is used for this, and it should be set to **false**.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
{
  //Options object
  private StiBlazorViewerOptions Options;

  protected override void OnInitialized()
  {
    base.OnInitialized();

    //Init options object
    Options = new StiBlazorViewerOptions();
    Options.Toolbar.ShowBookmarksButton = false;
  }
}
```

Information

In this case, report bookmarks won't be shown, even if they are present in a displayed report. This feature does not exert influence over printing and exporting a report.

When printing a report with bookmarks, the tree of bookmarks will be hidden. If apart from a report you need to print and bookmarks too, you should set the **BookmarksPrint** property to **true**.

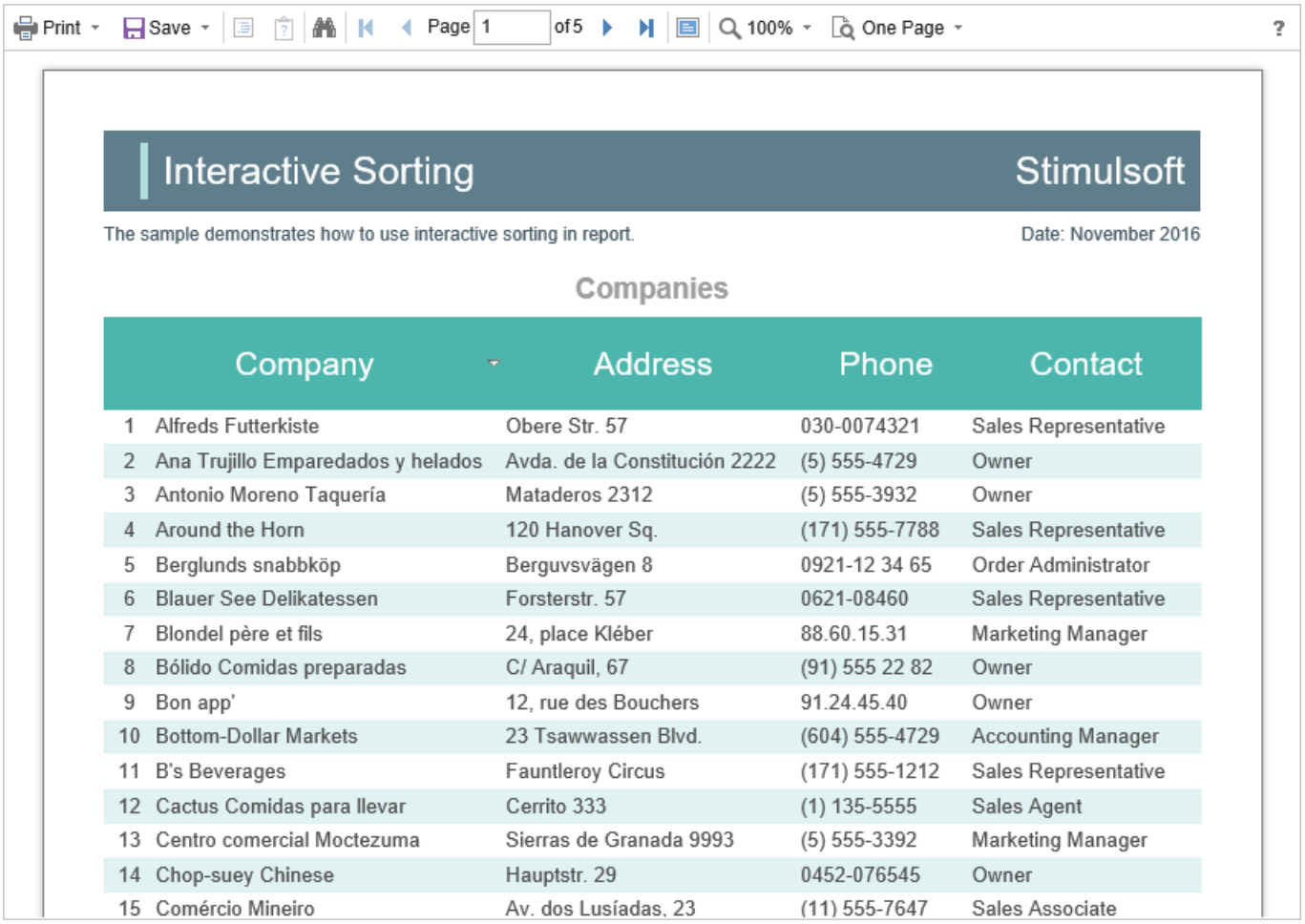
Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@options" />

@code
{
   //Options object
   private StiBlazorViewerOptions options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      Stimulsoft.Base.StiFontCollection.AddFontFile("Fonts/Microsoft Sans
      Serif.ttf", "Segoe UI");

      //Init options object
      options = new StiBlazorViewerOptions();
      options.Appearance.BookmarksPrint = true;
   }
}
```

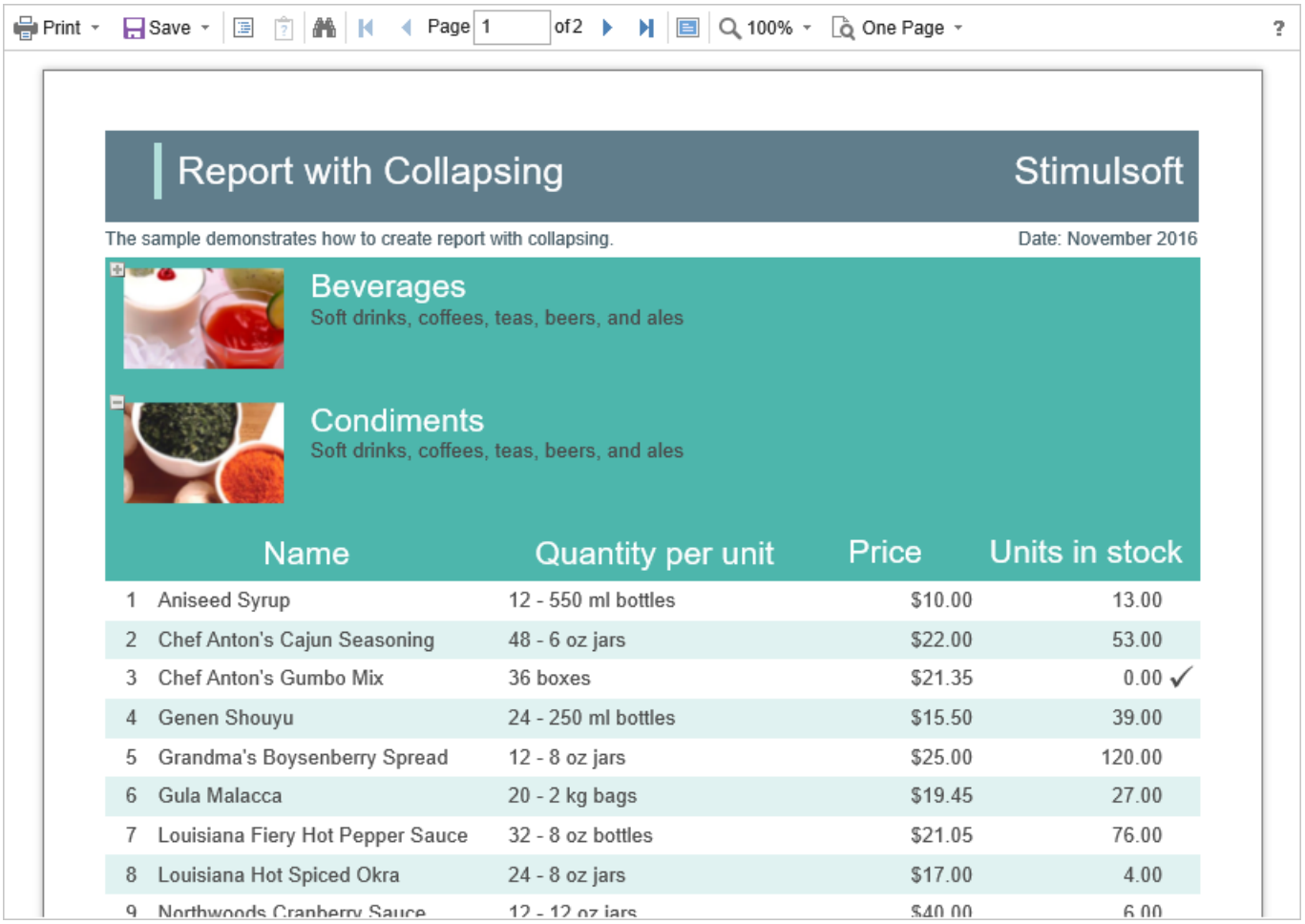
7.1.12 Dynamic Sorting, Collapsing, and Drill-Down

The **Blazor Viewer** supports dynamic sorting, collapsing, and drill down in reports. Dynamic sorting allows changing the sort direction in a rendered report. To do it, you should click on the component in which dynamic sorting was set. Dynamic sorting is carried out in the following direction: **Ascending** and **Descending**. Each time when you click on some component, the direction is reversed.

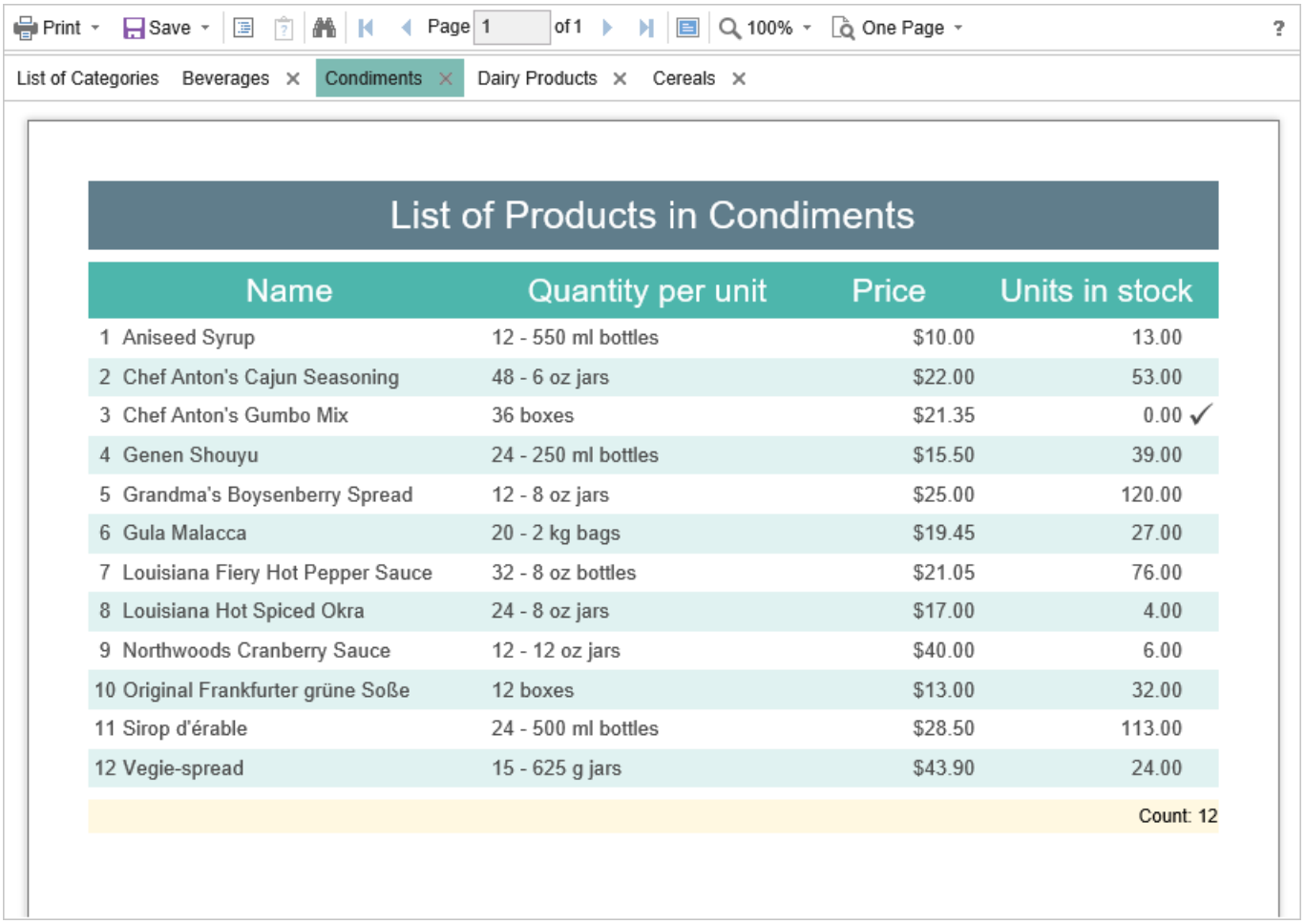
Multilevel sorting is allowed in a report. To do it, you should hold down the **Ctrl** and, step-by-step, click on sorted report components. To reset the sort, you can click on any sorted report without holding down the **Ctrl**.



A report with dynamic collapsing is an interactive report where collapsing blocks can collapse/expand their content when clicking on the header of the block. The report elements which you can collapse/expand are highlighted with a special icon with **[-]** or **[+]**.



When data is drilled down under the main panel, the drill down panel with the bookmarks of drill down reports will be displayed. A displayed report now will be highlighted.



Additional viewer settings are not required for the work with dynamic sorting, collapsing, and drilling down. If you need to make some actions before sorting or drilling down a report, you can define the special **OnInteraction** event.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer OnInteraction="@OnInteraction" />

@code
{
   private void OnInteraction(StiReportDataEventArgs args)
   {
      // Some code before any interaction
      // ...
   }
}
```

To get an action type, you can use an event argument. A definite type of action is envisaged for each kind of viewer interaction:

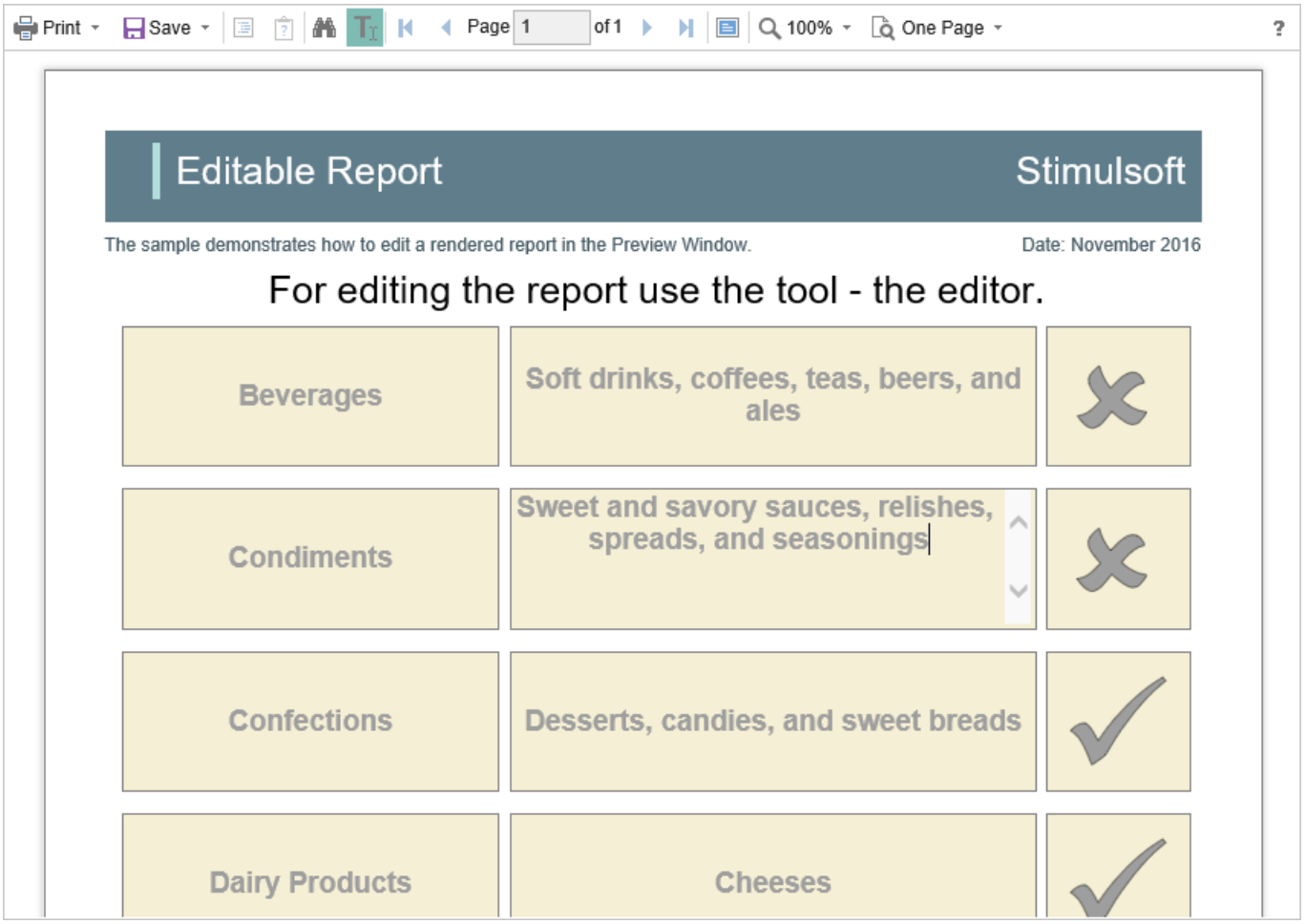
- The **Sorting** - when using sorting and columns.
- The **DrillDown** - when using a report drill down.
- The **Collapsing** - when using a report blocks collapsing.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer OnInteraction="@OnInteraction" />

@code
{
   private void OnInteraction(StiReportDataEventArgs args)
   {
      switch (args.Action)
      {
         case StiAction.Sorting:
            break;

         case StiAction.DrillDown:
            break;

         case StiAction.Collapsing:
            break;
      }
   }
}
```

7.1.13 Editing Report

The **Blazor Viewer** component has the ability to edit report items, such as text boxes and check boxes. In order the editing be possible, in the report template, you should mark the required components as editable. After displaying a report in the viewer, you need to click the corresponding button on the viewer panel to start editing. After editing, it is necessary to click the button once more, and all changes will be applied to the report.



For the report edit mode, no special settings of the viewer required.

Information

The edited settings will be applied when you print or export a report, and the original report remains unchanged. After restarting the viewer, all the values will be returned to the initial ones.

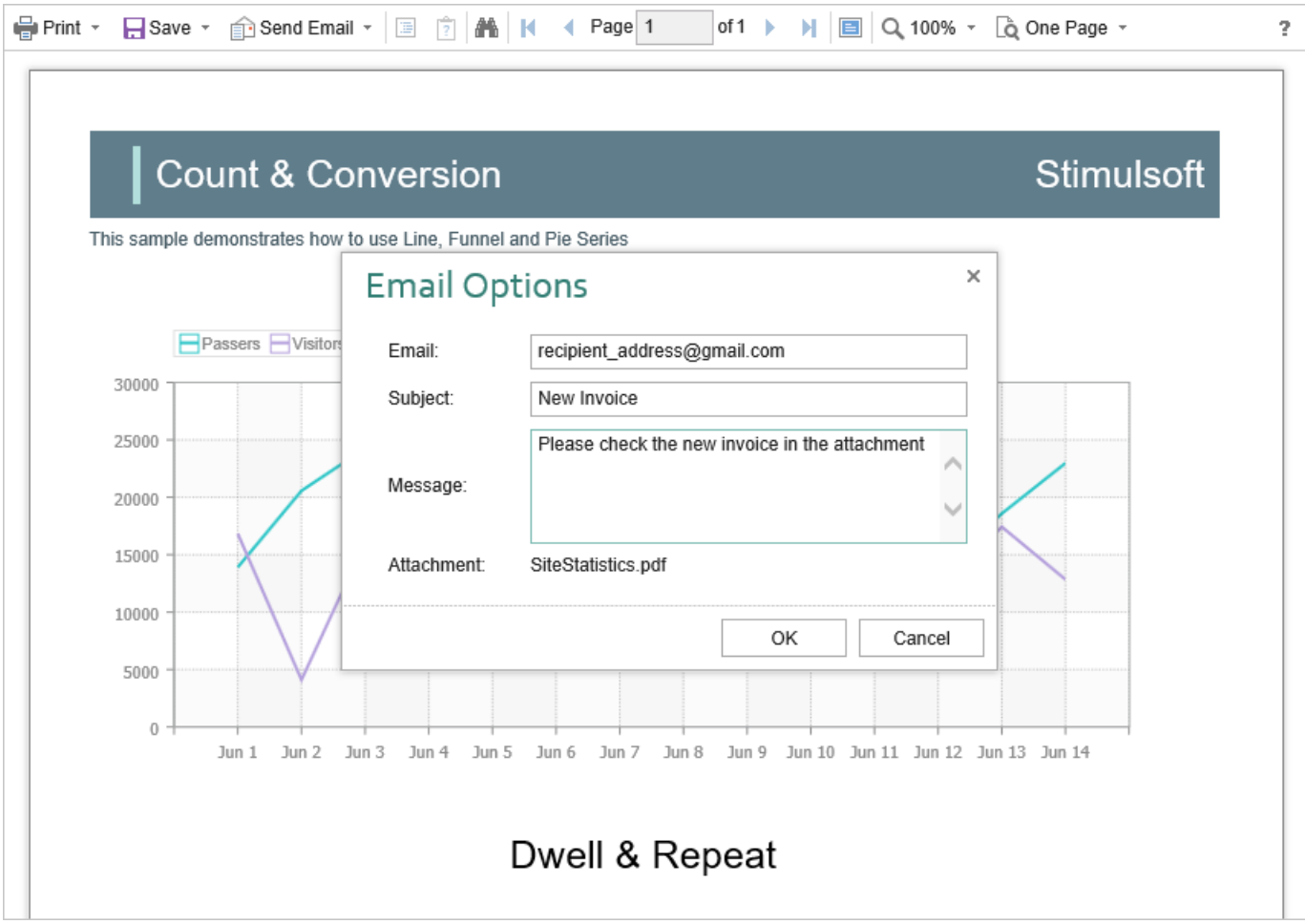
7.1.14 Sending Report by Email

The option to send a report by email is implemented in the **Blazor Viewer** component. To activate it, you should set the **ShowSendEmailButton** viewer property to true and define the **OnEmailReport** event.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" OnEmailReport="@OnEmailReport" />

@code
{
   //Options object
   private StiBlazorViewerOptions Options;

   private void OnEmailReport(StiEmailReportEventArgs args)
   {
      //args.Options.AddressTo = "";
      //args.Options.Subject = "";
      //args.Options.Body = "";

      // Should be filled here
      args.Options.AddressFrom = "admin_address@test.com";
      args.Options.Host = "smtp.test.com";
      args.Options.Port = 465;
      args.Options.UserName = "admin_address@test.com";
      args.Options.Password = "admin_password";

      //args.Options.CC.Add("email@test.com");
      //args.Options.BCC.Add("email@test.com");
      //args.Options.EnableSsl = true;
   }
}
```

The menu of attachment format selection displays when sending a report by email. This menu corresponds to the Export format menu. After a format is selected, the parameters dialog, such as recipient`s email, theme, and text of the letter, will display.



After confirming the submission, the **OnEmailReport** event described above will be triggered, where you can check and correct data typed in this form. The exported report file will be attached to an email automatically.

The **Blazor Viewer** component allows setting values by default for the Send Email form. The **DefaultEmailAddress**, the **DefaultEmailSubject**, and **DefaultEmailMessage** properties are used for this. These properties are empty by default.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer Options="@Options" />

@code
```

```
{
   //Options object
   private StiBlazorViewerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorViewerOptions();
      Options.Email.DefaultEmailAddress = "recipient_address@gmail.com";
      Options.Email.DefaultEmailSubject = "New Invoice";
      Options.Email.DefaultEmailMessage = "Please check the new invoice in
      the attachment";
   }
}
```

7.1.15 Calling Designer from Viewer

The **Blazor Viewer** component has the ability to call the report designer. To use this feature, you should define the **OnDesignReport** event handler.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web
@inject NavigationManager NavigationManager

<StiBlazorViewer Report="@Report" OnDesignReport="@OnDesignReport" />

@code
{
   //Report object to use in viewer
   private StiReport Report;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      var report = new StiReport();
      report.Load("Reports/TwoSimpleLists.mrt");
      Report = report;
   }

   protected void OnDesignReport(StiReportDataEventArgs args)
   {
      //Redirect to the Designer page
      NavigationManager.NavigateTo("Designer?report=" +
      args.Report.ReportName);
   }
}
```

Information

The viewer doesn't run the designer. It just calls the specified event in which you can get all necessary parameters. Later in the activity, you can redirect to another page that contains the report designer.

7.1.16 Export and Printing from Code

The **Blazor Viewer** provides the ability to print reports in various ways and export reports to various formats. These actions are performed using the viewer menu. If you want to print or export a report by using the code, for example, in the event of pressing the button, you can use the special **StiReportResponse** class. This class contains a set of static methods that allow you to print or export a report from the code, and the report viewer is not required.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<button @onclick="@OnClickPrintButton">Print PDF</button>
<button @onclick="@OnClickExportButton">Export PDF</button>

@code
{
   private StiReport LoadSimpleList()
   {
      var dataSet = new System.Data.DataSet();
      dataSet.ReadXml("Data/Demo.xml");

      var report = new StiReport();
      report.Load("Reports/SimpleList.mrt");
      report.RegData(dataSet);

      return report;
   }

   protected void OnClickPrintButton()
   {
      var report = LoadSimpleList();

      StiReportResponse.PrintAsPdf(report);
      //StiReportResponse.PrintAsHtml(report);
   }

   protected void OnClickExportButton()
   {
      var report = LoadSimpleList();

      StiReportResponse.ResponseAsPdf(report);
      //StiReportResponse.ResponseAsExcel2007(report);
```

```
        //StiReportResponse.ResponseAsPng(report);
        //StiNetCoreReportResponse.ResponseAsJson(report);
    }
}
```

If the **Razor** page does not have components for working with reports (viewer or designer), it is necessary to pre-initialize the reporting tool. You can do this by overriding the standard **OnInitializedAsync** event of a report by adding a special static method, **StiBlazorHelper.Initialize()**, to which you need to pass a **JSRuntime** object as input. After that, the actions for exporting and printing the report will work correctly.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@inject IJSRuntime JSRuntime;

protected override Task OnInitializedAsync()
{
    StiBlazorHelper.Initialize(JSRuntime);

    return base.OnInitializedAsync();
}
```

7.1.17 Viewer Events

The **Blazor Viewer** component supports events which allows you to execute necessary operations before certain actions, such as printing and exporting, sending reports by email, interactivity etc. Below is a sample of processing viewer events.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer OnViewerReport="@OnViewerEvent" />

@code
{
    private void OnViewerEvent(StiReportDataEventArgs args)
    {
        var action = args.Action;
        var report = args.Report;
        var parameters = args.RequestParams;
    }
}
```

Events list

Name	Description
OnOpenReport	The event occurs when opening a report.
OnPrintReport	The event occurs when printing a report.
OnExportReport	The event occurs when exporting a report.
OnEmailReport	The event occurs when sending a report by email.
OnInteraction	The event occurs when the viewer works with interactive operations, such as using parameters, dynamic sorting, collapsing, and drilling a report.
OnViewerEvents	The event occurs for any action in the report viewer.
OnDesignReport	The event occurs when pressing the Design button on the toolbar of the viewer.
OnViewerAfterRender	The event occurs when the HTML5 code of the viewer and all its controls have been completed.

7.1.18 Viewer Settings

The **Blazor Viewer** setting is configured with the help of component properties. The main settings are set in the Razor component; all additional settings are set using the **StiBlazorViewerOptions** class. Below is an example of the viewer properties setting.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorViewer ID="Viewer1" Width="500px" Height="500px"
Localization="Lozalization/en.xml" Options="@Options"
Theme="StiViewerTheme.Office2022WhiteCarmine" />

@code
{
   //Options object
```

```
    private StiBlazorViewerOptions Options;

    protected override void OnInitialized()
    {
        base.OnInitialized();

        //Init options object
        Options = new StiBlazorViewerOptions();
        Options.Appearance.ScrollbarsMode = true;
        Options.Appearance.ShowTooltips = false;
        Options.Toolbar.DisplayMode = StiToolbarDisplayMode.Separated;
        Options.Appearance.ReportDisplayMode =
        StiReportDisplayMode.FromReport;
        Options.Exports.ShowExportToDbf = false;
        Options.Exports.ShowExportToDif = false;
        Options.Exports.DefaultSettings.ExportToPdf.CreatorString = "Company
        Name";
        Options.Exports.DefaultSettings.ExportToPdf.ImageQuality = 0.75f;
    }
}
```

The Viewer properties

Name	Description
ID	Allows you to set a string unique identifier of a component, by default **StiBlazorViewer**.
Report	Allows you to set report object that will be displayed in the viewer.
Localization	Specifies the path to the XML localization file. The path can be absolute or relative. By default, the English localization is used, which is built into the viewer and does not require additional XML files.
Width	Sets the width of the component in the required units - pixels (default value), percentage, points. For example, “500”, “500px”, “100%”, “300pt”.
Height	Sets the height of the component in the required units - pixels (default value), percentage, points. For example, “500”, “500px”, “100%”, “300pt”. The automatic height is set by default, depending on the size of a report page or 650 pixels in the mode of the viewer display

	with scroll bars.
Options	It allows specifying the object of the **StiBlazorViewerOptions**, which contains the set of the component options.
Theme	Specifies the theme of the viewers layout. The list of available themes can be found in the **StiTheme** enumeration. The default value is **Office2022WhiteCarmine**.

Server

Name	Desription
UseRelativeUrls	Sets the viewer mode in which relative URLs are used for AJAX requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value which specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-1** removes the port number.
PassQueryParametersToReport	Enables using all the URL parameters of the request as the variable values. The variable names must match the parameters. The default value of the property is **false**.
PassQueryParametersForResources	Enables transferring all request URL parameters when generating links to the resources of the viewer. If **false**, only the necessary parameters are used to request the resources of the viewer. By default, the property is set to **true**.
PassFormValues	Enables passing the values of the POST form to the client side, if these values are required to be used in the actions of the viewer. If you enable this property, the additional **GetFormValues()** method will return a collection of form parameters. By default, the property is **false**.
AllowLoadingCustomFontsToClie	Allows you to pass custom fonts to the client

ntSide	side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
CustomCss	Sets the path to the CSS file of the viewer's styles. The standard styles of the chosen theme will not be loaded if this property has got a value. The default value of the property is an empty string.
BackgroundColor	Sets the background color of the viewer. By default it is set to **White**.
PageBorderColor	Sets the border color of the viewer. By default it is set to **Gray**.
RightToLeft	Sets the **Right to Left** mode for viewer controls. By default the property is set to **false**. By default, the property is set to **false**.
FullScreenMode	Sets the full-screen display mode of the viewer. By default, the property is set to **false**.
ScrollbarsMode	Sets the preview mode with scrollbars. By default, the property is set to **false**.
OpenLinksWindow	Sets the target window for opening links contained in the report. By default, the property is set to **Blank** (new window).
OpenExportedReportWindow	Sets the target window for opening the export file from the viewer. By default, the property is set to **Blank** (new window).
DesignWindow	Sets the destination window for launching the report designer. The default value of the property is **Self** (whish is the current window).
ShowTooltips	Enables showing tips for the viewer controls when the mouse hovers over. By default, the

	property is set to **true**.
ShowTooltipsHelp	Enables showing links to online documentation for the viewer controls. By default, the property is set to **true**.
ShowDialogsHelp	Sets a value which indicates that show or hide the help button in dialogs. By default, the property is set to **true**.
PageAlignment	Sets the position of the report page in the viewer window. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – the page will be aligned left; › **Center** – the page will be centered (default value); › **Right** – the page will be aligned right.
ShowPageShadow	Enables displaying shadow for report pages. By default the property is set to **true**.
BookmarksPrint	Enables printing of report bookmarks (besides the report itself). By default the property is set to **false**.
BookmarksTreeWidth	Sets the width of the bookmarks panel in pixels. By default, the width is 180 pixels.
ParametersPanelPosition	Specifies the position of the report parameters panel. It can take one of the following **StiParametersPanelPosition** enumeration values: › **Top** - the panel will be docked to the top margin (default value); › **Left** - the panel will be docked to the left margin.
ParametersPanelMaxHeight	Sets the maximum height of the parameters bar in pixels. By default, the maximum height is 300 pixels.
ParametersPanelColumnsCount	Sets the number of columns to display report

	parameters. By default, there are 2 columns.
ParametersPanelSortDataItems	Gets or sets a value which indicates that variable items will be sorted. By default the property is set to **true**.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
InterfaceType	Sets the type of interface used for the viewer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the viewer's interface is determined automatically depending of the device that is report is displayed on. That is the default value. › **Mouse** – the standard interface with a mouse control will be used for all the screen types. › **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability. › **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with the smartphones.
AllowMobileMode	Enables or disables displaying a report or dashboard in the mobile mode. If the option is set to **false**, then the mobile view will not be used. If the option is set to **true**, the mobile view mode will be used when opening the viewer on mobile devices. By default, the option is set to **true**.
ChartRenderType	Sets the displaying mode of charts on the report

	page. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages. It can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN markup.
DatePickerFirstDayOfWeek	Sets the first day of the week for the date picker. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Auto** – automatic detection of the first day of the week from the browser settings (default value); › **Monday** – the first day of the week is Monday; › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default the property is set to **false**.
AllowTouchZoom	Sets ability to change the scale of the report page by using the two-fingers gesture (Pinch to

	Zoom) for the touch-screens. The default value of the property is **true**.
ShowReportIsNotSpecifiedMessage	Sets a value which indicates that 'The report is not specified' message will be shown. The default value of the property is **true**.
PrintToPdfMode	Sets the Print to PDF mode. It has the following values: › **StiPrintToPdfMode.Hidden** - hidden print mode (default value); › **StiPrintToPdfMode.Popup** - the PDF document will be displayed before printing in a pop-up window.
ImagesQuality	Gets or sets the image quality that will be used on the viewer page. It has the following values: › **StiImagesQuality.Low** - low quality, used to speed up loading reports and saves memory; › **StiImagesQuality.Normal** - normal quality, suitable for most cases (default value); › **StiImagesQuality.High** - high quality, used for ultra high-definition displays, but may slow down the loading of pages.
CombineReportPages	Sets a value which indicates that if a report contains several pages, then they will be combined in preview. By default the property is set to **false**.

Toolbar

Name	Description
Visible	Enables displaying the viewer toolbar. By default, the property is set to **true**.
DisplayMode	Specifies the display mode of the toolbar of the viewer. It can take one of the following values of the **StiToolbarDisplayMode** enumeration: › **Simple** - all controls are located on the same

	control panel (default value); › **Separated** - the control panel is split into top and bottom panels.
BackgroundColor	Specifies the background color of the viewer toolbar. The default color of the selected theme is used.
BorderColor	Specifies the border color of the viewer toolbar. The default color of the selected theme is used.
FontColor	Specifies the text color for the toolbar and the viewer menu. The default color of the selected theme is used.
FontFamily	Specifies the font for the toolbar and the viewer menu. The default font of the selected theme is used.
Alignment	Sets the alignment mode for the controls on the viewer toolbar. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – elements will be aligned left; › **Center** – elements will be centered; › **Right** – elements will be aligned right; › **Default** – the alignment depends on the RightToLeft property (default value).
ShowButtonCaptions	Enables text of the buttons on the toolbar of the viewer. By default the property is set to **true**.
ShowPrintButton	Enables showing the button - **Print** - on the viewer toolbar. By default, the property is set to **true**.
ShowOpenButton	Enables displaying the **Open** button on the toolbar of the viewer when viewing reports. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the viewer when viewing reports. By default, the property is set to true.

ShowSendEmailButton	Enables showing the button - **Send Email** - on the viewer toolbar. By default, the property is set to **false**. Also, you should add the EmailReport action.
ShowFindButton	Enables showing the button - **Find** - on the viewer toolbar. By default, the property is set to **true**.
ShowBookmarksButton	Enables showing the button - **Bookmarks** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the bookmarks panel will not be displayed even if there are bookmarks in the report.
ShowParametersButton	Enables showing the button - **Parameters** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the parameters panel will not be displayed even if there are parameters in the report.
ShowResourcesButton	Enables showing the button - **Resources** - on the toolbar of the viewer. By default, the property is set to **true**. If the button is hidden, the resources panel will not be displayed even if there are resources in the report.
ShowEditorButton	Enables showing the button - **Editor** - on the viewer toolbar. By default, the property is set to **true**.
ShowFullScreenButton	Enables displaying the **Full Screen** button on the toolbar of the viewer when viewing reports. By default, the property is set to **true**.
ShowFirstPageButton	Enables showing the button - **First Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowPreviousPageButton	Enables showing the button - **Previous Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowCurrentPageControl	Enables showing the current report page indicator. By default, the property is set to **true**.

ShowNextPageButton	Enables showing the button - **Next Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowLastPageButton	Enables showing the button - **Last Page** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowZoomButton	Enables showing the button to select the report zoom. By default, the property is set to **true**.
ShowViewModeButton	Enables showing the button to select the view mode of the report page. By default, the property is set to **true**.
ShowDesignButton	Enables displaying the **Design** button on the toolbar of the viewer when viewing reports. By default, the property is set to **false**.
ShowAboutButton	Enables showing the button - **About** - on the viewer toolbar. By default, the property is set to **true**.
ShowRefreshButton	Sets a visibility of the **Refresh** button in the toolbar of the viewer. By default, the property is set to **true**.
ShowPinToolbarButton	Enables displaying of the **Pin Toolbar** button on the viewer's toolbar. The button is available only in the Mobile mode of the viewer's interface. The default value of the property is **true**.
PrintDestination	Sets the report printing mode. It can take one of the following values of the **StiPrintDestination** enumeration: › **Default** – a menu with a choice of printing modes will be displayed (default value); › **Pdf** – printing will be done in the PDF format; › **Direct** – printing will be done to the HTML format directly to the printer, the system print dialog will be displayed; › **PopupWindow** – printing will be done in the

	HTML format via the preview window of the report.
ViewMode	Sets the mode for displaying report pages. It can take one of the following **StiWebViewMode** enumeration values: › **SinglePage** - displays one page of the report selected in the toolbar of the viewer (default value); › **Continuous** - displays all pages of the report; › **MultiplePages** - displays all report pages as a table.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. The values are from 10 to 500 percent. You can also set one of the following values: › **StiZoomMode.PageWidth** – when the viewer runs, the zoom, necessary to display the report by the page width, will be set; › **StiZoomMode.PageHeight** – when the viewer runs, the zoom, necessary to display the report by the page height, will be set.
MenuAnimation	Enables animation when the viewer menu shows/hides. By default the property is set to **true**.
ShowMenuMode	Sets the display mode of the viewer menu. It can take one of the following values of the **StiShowMenuMode** enumeration: › **Click** – shows menu by mouse click (default value); › **Hover** – shows menu by hovering the mouse cursor.
AutoHide	Enables auto-hiding of the viewer's toolbar. The property will work only for the Mobile mode of the viewer's interface. The default value of the

	property is **false**.

Export

Name	Description
DefaultSettings	This group of properties provides the ability to specify the default export settings for each export type. These settings will be applied to the export dialogs when the viewer runs or to the report, if export dialogs are disabled.
StoreExportSettings	Enables saving selected settings in the export dialogs. Settings will be stored in browser cookies. By default the property is set to **true**.
ShowExportDialog	Enables showing the export options dialog box. If the property is set to **false**, the export will be done with the default settings. By default the property is set to **true**.
ShowExportToDocument	Enables the export menu item - **Document File**. By default, the property is set to **true**.
ShowExportToPdf	Enables displaying the **Adobe PDF file** export menu item when viewing reports, and the **Adobe PDF** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToXps	Enables the export menu item - **Microsoft XPS File**. By default, the property is set to **false**.
ShowExportToPowerPoint	Enables the export menu item - **Microsoft PowerPoint 2007/2010 File**. By default, the property is set to **true**.
ShowExportToHtml	Enables the export menu item - **HTML File**. By default, the property is set to **true**.
ShowExportToHtml5	Enables the export menu item - **HTML5 File**. By default, the property is set to **true**.
ShowExportToMht	Enables the export menu item - **MHT Web Archive**. By default, the property is set to **true**.

ShowExportToText	Enables the export menu item - **Text File**. By default, the property is set to **true**.
ShowExportToRtf	Enables the export menu item - **Rich Text File**. By default, the property is set to **true**.
ShowExportToWord2007	Enables the export menu item - **Microsoft Word 2007/2010 File**. By default, the property is set to **true**.
ShowExportToOpenDocumentWriter	Enables the export menu item - **OpenDocument Writer File**. By default, the property is set to **true**.
ShowExportToExcel	Enables the export menu item - **Microsoft Excel File**. By default, the property is set to **true**.
ShowExportToExcelXml	Enables the export menu item - **Microsoft Excel Xml File**. By default, the property is set to **true**.
ShowExportToExcel2007	Enables displaying the **Microsoft Excel 2007/2010 File** export menu item when viewing reports, and the **Microsoft Excel** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToOpenDocumentCalc	Enables the export menu item - **OpenDocument Calc File**. By default, the property is set to **true**.
ShowExportToCsv	Enables the export menu item - **CSV File**. By default, the property is set to **true**.
ShowExportToDbf	Enables the export menu item - **DBF File**. By default, the property is set to **true**.
ShowExportToXml	Enables the export menu item - **XML File**. By default, the property is set to **true**.
ShowExportToDif	Enables the export menu item - **Data Interchange Format (DIF) File**. By default, the property is set to **true**.
ShowExportToSylk	Enables the export menu item - **Symbolic Link (SYLK) File**. By default, the property is set to **true**.

ShowExportToJson	Enables the export menu item - **JSON File**. By default, the property is set to **true**.
ShowExportToImageBmp	Enables displaying the **BMP Image** export menu item when viewing reports, and the **BMP Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageGif	Enables displaying the **GIF Image** export menu item when viewing reports, and the **GIF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageJpeg	Enables displaying the **JPEG Image** export menu item when viewing reports, and the **JPEG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePcx	Enables displaying the **PCX Image** export menu item when viewing reports, and the **PCX Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePng	Enables displaying the **PNG Image** export menu item when viewing reports, and the **PNG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageTiff	Enables displaying the **TIFF Image** export menu item when viewing reports, and the **TIFF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvg	Enables displaying the **Scalable Vector Graphics (SVG) File** export menu item when viewing reports, and the **Scalable Vector Graphics (SVG) File** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvgz	Enables displaying the **Compressed SVG (SVGZ) File** export menu item when viewing reports, and the **Compressed SVG (SVGZ) File** item when viewing dashboards. By default, the

	property is set to **true**.
ShowOpenAfterExport	Enables displaying the **Open After Export** parameter in export settings menu. By default, the property is set to **true**.

Email

Name	Description
ShowEmailDialog	Enables displaying settings for sending the report via email. If the dialog box is disabled, the email will be sent with the settings set on the server side in the **EmailReport** action. By default the property is set to **true**.
ShowExportDialog	Enables displaying export options dialog box when sending email. If the property is set to **false**, the export will be done with the default settings. By default the property is set to **true**.
DefaultEmailAddress	Sets the default recipient email, i.e. the address to which the email with the attached report will be sent.
DefaultEmailSubject	Sets the default email subject (header).
DefaultEmailMessage	Sets the default email message (text).
DefaultEmailReplyTo	Sets the default text of the replyTo of the message created in the viewer.

7.2 Designer

Samples

On GitHub, you may find the samples which show how to work with the **Blazor Designer** component. We support Blazor Server and Blazor WebAssembly. All the examples are separate projects, grouped into one solution for the Visual Studio.

The **Blazor Designer** component is designed to create reports in the web browser. And you don't need to set components or any special plugins on the client. All you need is a modern web browser. With the help of the **Blazor Designer**, you can create, edit, save, view reports, and print them on any computer with any installed operating system.

The **Blazor Designer** component is developed as a unique component, and it can work both with the Blazor Server and WebAssembly. When performing all actions on reports, the component requests only necessary data, which allows you to get rid of reloading the entire page and save Web traffic and increase work speed.

To use the **Blazor Designer** in a Web project, you should use the NuGet package Stimulsoft.Reports.Blazor or Stimulsoft.Dashboards.Blazor:

- Select the Manage NuGet Packages item in the context menu of the project;
- Specify the Stimulsoft.Reports.Blazor on the Browse in the search line;
- Select the element, define the package version, and click **Install**. You should click the **Update** when updating the package.

If for some reason it is not possible, you should add the following assemblies to the project:
Stimulsoft.Base.dll
Stimulsoft.Blockly.dll
Stimulsoft.Data.dll
Stimulsoft.Drawing.dll
Stimulsoft.Map.dll
Stimulsoft.Report.dll
Stimulsoft.Report.Blazor.dll
Stimulsoft.Report.Check.dll
Stimulsoft.Report.Helper.dll
Stimulsoft.Report.Web.dll
Stimulsoft.Report.WebDesign.dll
Stimulsoft.System.dll
Stimulsoft.System.Web.dll

You should additionally add the following assemblies to the project for using dashboards:
Stimulsoft.Dashboard.dll
Stimulsoft.Dashboard.Drawing.dll
Stimulsoft.Dashboard.Export.dll

7.2.1 Activation

After acquiring a Stimulsoft product, you should activate the license. You can do it by specifying a license key or loading a file with a license key. Below is an example of the **Blazor Designer** component activation.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner />

@code
{
   protected override void OnInitialized()
   {
      //Activation with using license code
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";

      //Activation with using license file
      Stimulsoft.Base.StiLicense.LoadFromFile("Content/license.key");

      base.OnInitialized();
   }
}
```

You can get a license key or download a file with a license key in a personal account.

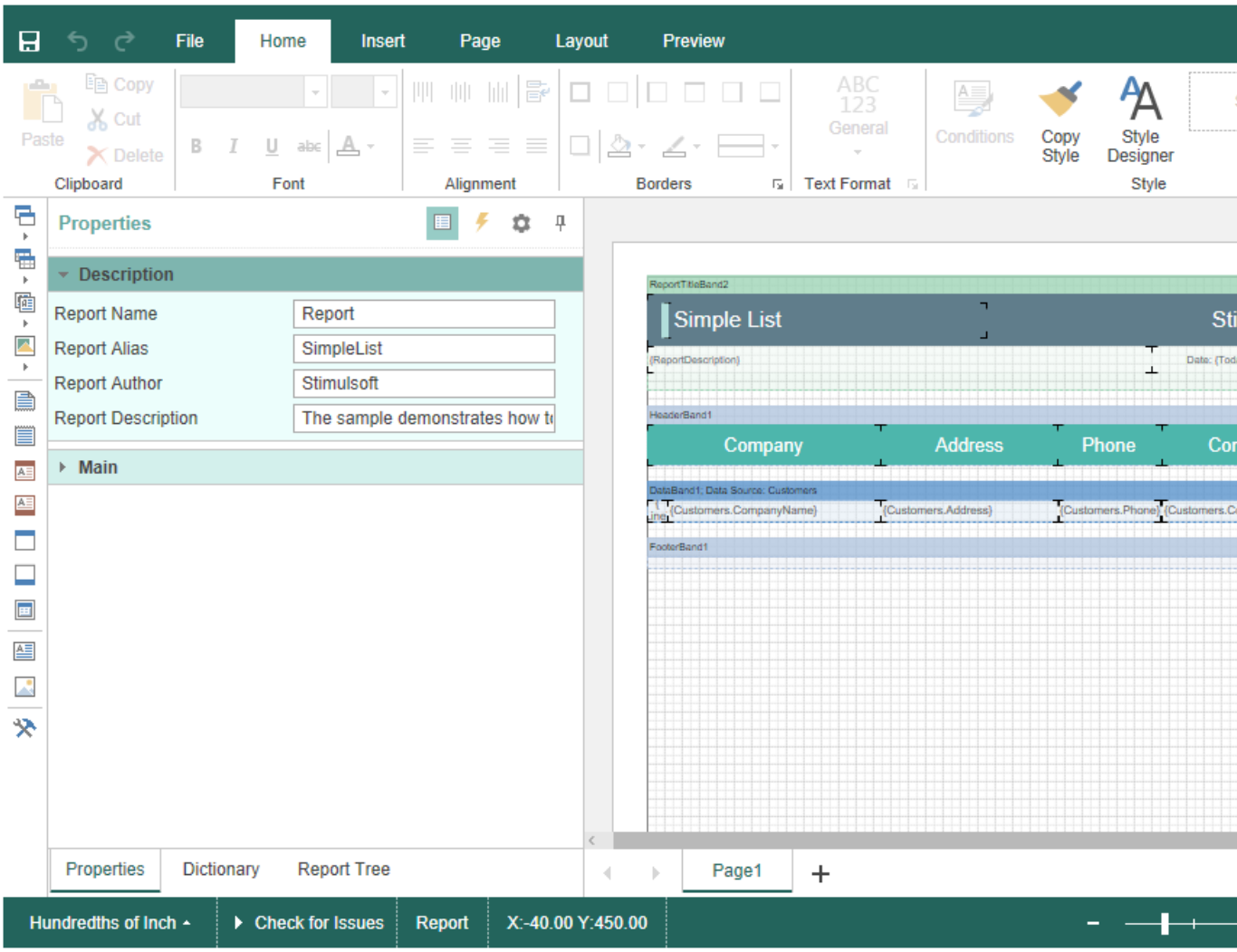
7.2.2 Editing Reports

To edit a report, you should add the **StiBlazorDesigner** component to the **Razor** page, add the **StiReport** object, and assign it to the designer, using the **Report** property. After loading a report from a file, it will be automatically transferred to the

designer.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner Report="@Report" />

@code
{
   //Report object to use in designer
   private StiReport Report;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Create empty report object
      var report = new StiReport();

      //Load report template
      report.Load("Reports/TwoSimpleLists.mrt");

      //Assing report object to viewer
      Report = report;
   }
}
```



7.2.3 Creating New Reports

It's enough to create a new object in the **OnInitialized** event to run the designer with a new report. If required, you can preload data for your report or perform some other necessary actions.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner Report="@Report" />

@code
{
   //Report object to use in designer
   private StiReport Report;
```

```
    protected override void OnInitialized()
    {
        base.OnInitialized();

        //Create empty report object
        var report = new StiReport();

        //Assing report object to designer
        Report = report;
    }
}
```

Also, you can create a new report using the main menu of the designer. You can use the **OnCreateReport** event to preload data for a new report or perform other necessary actions. This event will be processed when creating a new blank report from the main menu or when creating a report using the wizard.

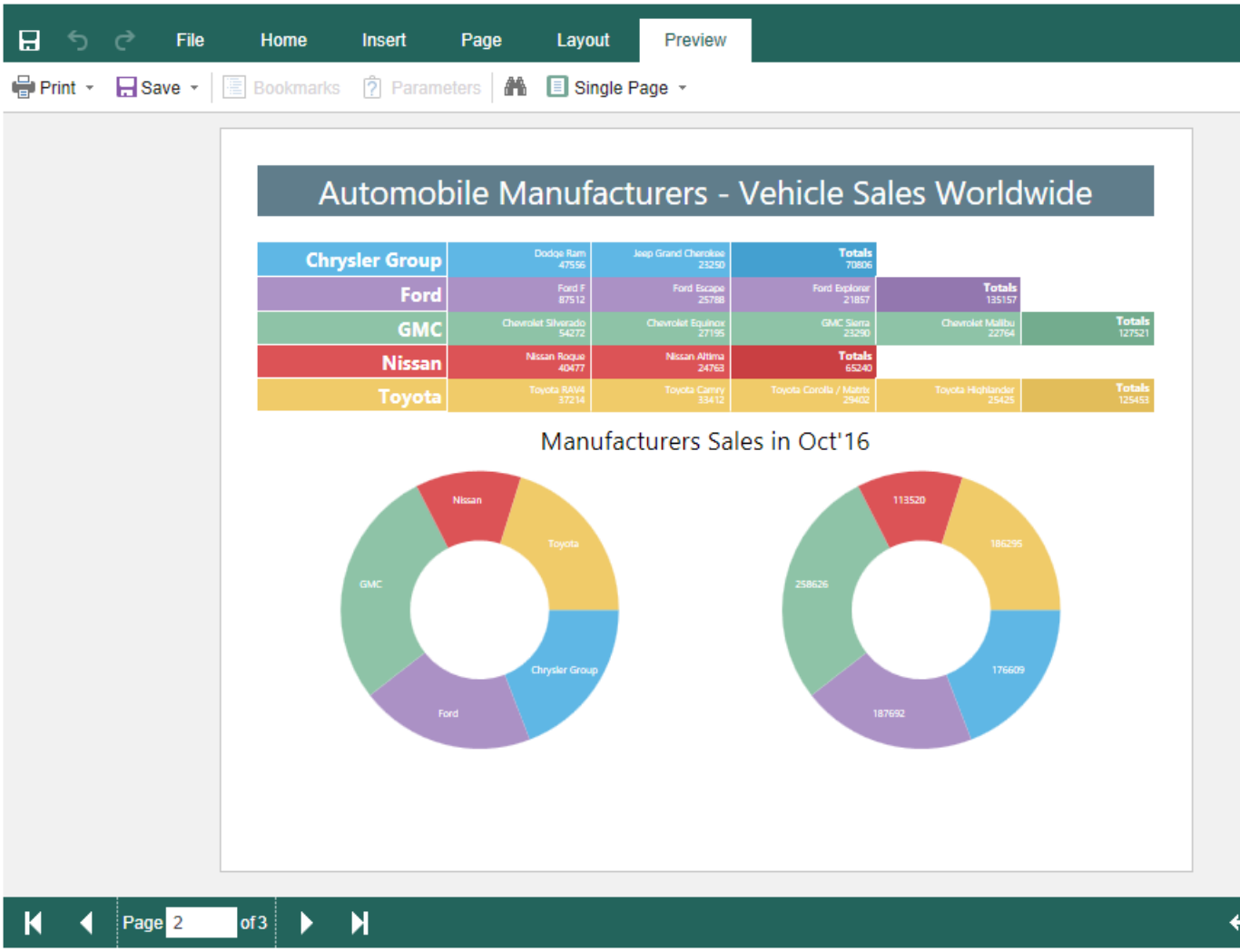
Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Web

<StiBlazorDesigner Report="@Report" OnCreateReport="@OnCreateReport"/>

@code
{
    //Report object to use in designer
    private StiReport Report;

    protected override void OnInitialized()
    {
        base.OnInitialized();

        //Create empty report object
        var report = new StiReport();

        //Load report template
        report.Load("Reports/Simple List.mrt");

        //Assing report object to designer
        Report = report;
    }

    private void OnCreateReport(StiReportDataEventArgs args)
    {
        //Delete connections in the report template
        args.Report.Dictionary.Databases.Clear();

        //Load new data from XML file
        var data = new System.Data.DataSet();
        data.ReadXml("Data/Demo1.xml");

        args.Report.RegData(data);
```

```
        args.Report.Dictionary.Synchronize();
    }
}
```

7.2.4 Preview

The **Blazor Designer** component has the mode of an editable report preview. To view a report, you should go to an appropriate bookmark in the designer window. The template will be rendered and displayed in the rendered viewer.



There is the ability to perform some necessary actions before previewing a report, for example: connect data for a report. To do it, you should define the **OnPreviewReport** event, which will be invoked before viewing a report.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner Report="@Report" OnPreviewReport="@OnPreviewReport"/>

@code
{
   //Report object to use in designer
   private StiReport Report;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Create empty report object
      var report = new StiReport();

      //Load report template
      report.Load("Reports/Simple List.mrt");

      //Assing report object to designer
      Report = report;
   }

   private void OnPreviewReport(StiReportDataEventArgs args)
   {
      //Load new data from XML file
      var data = new System.Data.DataSet();
      data.ReadXml("Data/Demo1.xml");

      args.Report.RegData(data);
   }
}
```

7.2.5 Additional Features of Preview

The window of report viewing of the **Blazor Designer** is a full-fledged interactive viewer, which can print and export a report. Besides, it supports the work with their parameters. Also, interactive actions are supported among them - dynamic sorting, drill-down, collapsing. To use these options, you don't need any additional settings of the report designer.

You can make manipulations with a report template in any of the above actions. For example, you can change properties and parameters, connect new data for rendering. When exporting a report, you can get the export format, read or change its settings.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
```

```
@using Stimulsoft.Report.Web

<StiBlazorDesigner Report="@Report" OnExportReport="@OnExportReport"/>

@code
{
   //Report object to use in designer
   private StiReport Report;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Create empty report object
      var report = new StiReport();

      //Load report template
      report.Load("Reports/Simple List.mrt");

      //Assing report object to designer
      Report = report;
   }

   private void OnExportReport(StiExportReportEventArgs args)
   {
      //Current export format
      var exportFormat = args.Format;

      //Current export settings
      var exportSettings = args.Settings;

      //Load new data from XML file
      var data = new System.Data.DataSet();
      data.ReadXml("Data/Demo1.xml");

      args.Report.RegData(data);
   }
}
```

Information

Suppose some of the specified additional options of the Report Preview are not required (for example, exporting or report printing). In that case, you can disable them using the appropriate options of the **Blazor Designer**.

7.2.6 Saving Reports

There are two options of saving reports in the **Blazor Designer** component, which are available in the main menu and the main panel of the designer - **Save** and **Save as**. At the same time, each of these options of saving has its modes and settings.

Saving reports on the server-side

To save an edited report on the server-side, you should define the **OnSaveReport** event, which will be invoked when selecting the **Save** item in the main menu or clicking the **Save** in the main panel of the designer.

Index.razor

```
@using Stimulsoft.Base
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner Report="@Report" OnSaveReport="@OnSaveReport"/>

@code
{
   //Report object to use in designer
   private StiReport Report;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Create empty report object
      var report = new StiReport();

      //Load report template
      report.Load("Reports/TwoSimpleLists.mrt");

      //Assing report object to designer
      Report = report;
   }

   private void OnSaveReport(StiSaveReportEventArgs args)
   {
      args.Report.Save("Reports/TwoSimpleLists.mrt");
   }
}
```

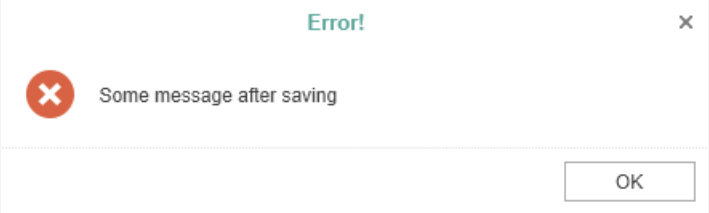
The **OnSaveReport** event will be invoked when clicking **Save**. The edited report will be passed in the event arguments. If required, you can use the option of displaying the dialog window with an error or a text message after saving a report. The **ErrorCode** and **ErrorString** properties in the arguments of the event are used for this.

Index.razor

```
...
private void OnSaveReport(StiSaveReportEventArgs args)
{
```

```
    args.Report.Save("Reports/TwoSimpleLists.mrt");

    args.ErrorString = "Some message after saving";
}
...
```

In this case, the dialog window with a specified text will be displayed. The text can contain either a save error or a warning message or any other message.



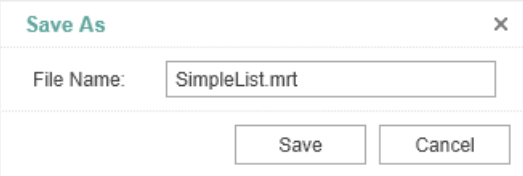
If required, you can access the original report name or report name from the save dialog.

Index.razor

```
...
private void OnSaveReport(StiSaveReportEventArgs args)
{
    //Report name from designer save dialog
    var savingReportName = args.FileName;

    //Original report name from properties
    var originalReportName = args.Report.ReportName;
}
...
```

Saving on the client-side

To save an edited report on the client-side as a file, you don't need additional designer settings. It's enough to select **Save as**, when clicking on it, the file saving dialog will be displayed. In this dialog, you can change the name of your report file, and after that, the file will be saved on the local disk of the computer.



The **Blazor Designer** component changes the behavior of the specified saving variant; there is the **OnSaveReportAs** event for this. When specifying this action, a report will be saved on the server-side; the work of this action will be analogous to the **OnSaveReportAs** event.

Index.razor

```
...
private void OnSaveReportAs(StiSaveReportAsEventArgs args)
{
   args.Report.Save("Reports/TwoSimpleLists.mrt");
}
...
```

Saving settings

A report is saved in the background mode, i.e., without reloading a page in the browser window. Suppose it is required to control the process of report saving somehow visually. In that case, you should change the **SaveReportMode** option (or the **SaveReportAsMode** option) to one of three values - **Hidden** (set by default), the **Visible**, or the **NewWindow**.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner Report="@Report" Options="@Options" />

@code
{
   //Report object to use in designer
   private StiReport Report;

   private StiBlazorDesignerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorDesignerOptions();
      Options.Behavior.SaveReportMode = StiSaveMode.Visible;
      Options.Behavior.SaveReportAsMode = StiSaveMode.Visible;
   }
```

```
}
```

If the **SaveReportMode** option is set to **Visible**, report saving will be invoked in the current window of the browser in a simple (visible) mode with the help of the POST request. If the **SaveReportMode** option is set to the **NewWindow** value, the save report event will be invoked in a new browser window. This option is set to the **Hidden** value by default, i.e., the report saving is invoked in the background with the help of the AJAX request, and it is not displayed in the browser window. The same values and behavior are available for the **SaveReportAsMode** option.

7.2.7 Localization

The **Blazor Designer** component supports the full localization of its interface. The **Localization** property is used to localize the interface of the report designer. You should specify the path to the XML localization file (relative or absolute).

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner Localization="Localization/en.xml" />
```

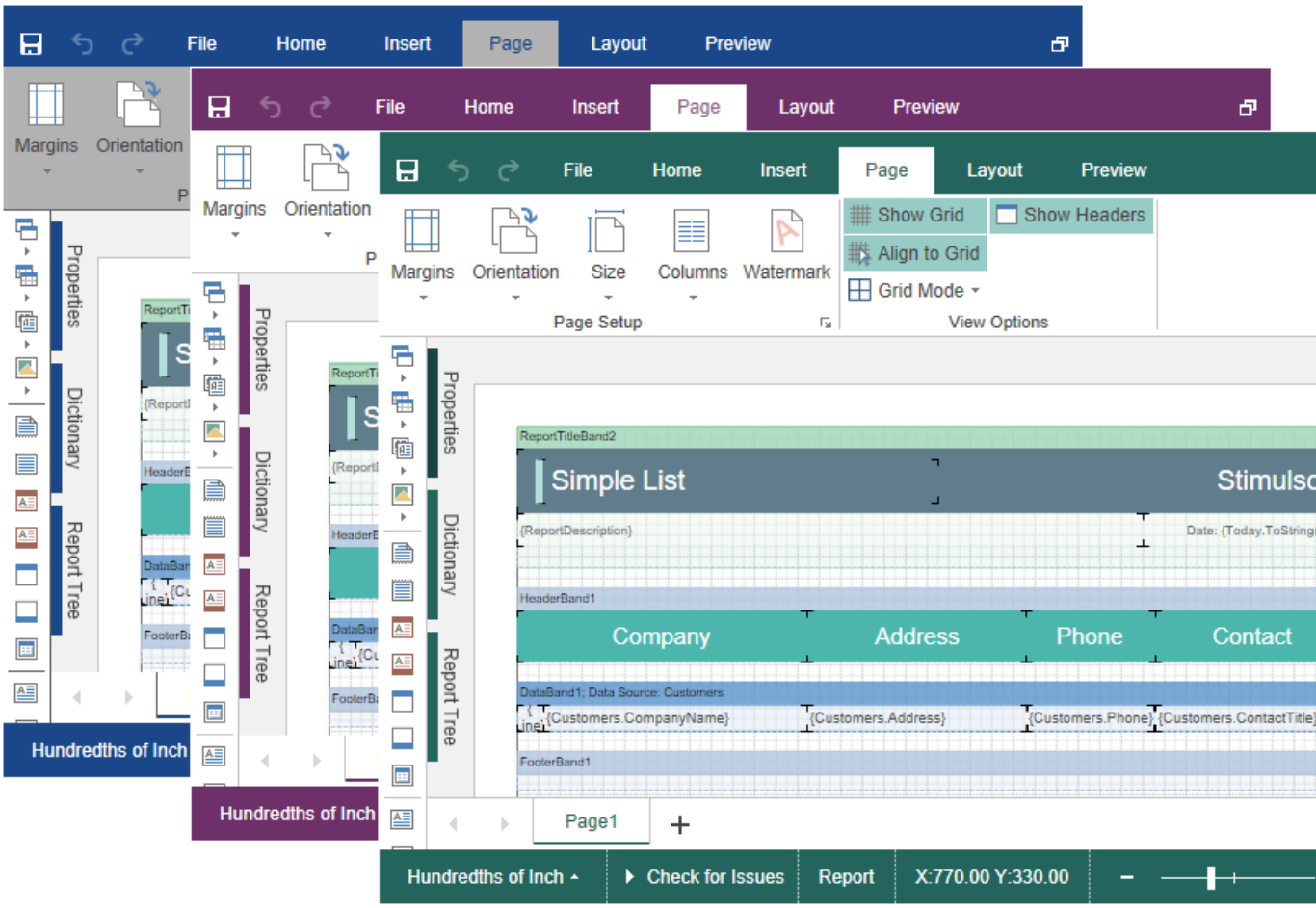
7.2.8 Using Themes

The **Blazor Designer** component has an option to change themes of visual controls. You should use the Theme property to change a theme.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner Theme="StiDesignerTheme.Office2022WhiteCarmine" />
```

There are currently **2 themes** available with different color accents. As a result, **more than 50** variants of the appearance are available. This allows you to customize the appearance of the designer for almost any design of the Web project.



7.2.9 Designer Events

The **Blazor Designer** component supports events which allows you to execute necessary operations before certain actions, such as creating and editing report templates, previewing, printing and exporting, interactivity and etc. Below is a sample for processing designer events.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner Report="@Report" OnPreviewReport="@OnPreviewReport"/>

@code
{
   //Report object to use in designer
   private StiReport Report;

   protected override void OnInitialized()
   {
```

```
        base.OnInitialized();

        //Create empty report object
        var report = new StiReport();

        //Load report template
        report.Load("Reports/TwoSimpleLists.mrt");

        //Assing report object to designer
        Report = report;
    }

    private void OnPreviewReport(StiReportDataEventArgs args)
    {
        //Load new data from XML file
        var data = new System.Data.DataSet();
        data.ReadXml("Data/Demo1.xml");

        args.Report.RegData(data);
    }
}
```

Events list

Name	Description
OnCreateReport	Occurs when creating new reports from the designer menu.
OnOpenReport	The event occurs when you open a report from the designer menu. In the arguments of the event, the downloaded report will be sent.
OnPreviewReport	Occurs when going to the preview tab, as well as when interactive activities such as using report variables, dynamic collapsing, drill-down, and sorting a report when previewing it.
OnSaveReport	Occurs when clicking the Save button on the panel or from the main menu of the designer.
OnSaveReportAs	Occurs when clicking the Save As button from the main menu of the designer. If the event is not specified, the report will be saved to the local

	disk.
OnExportReport	Occurs when expoting reports.
OnDesignerEvent	The event occurs for any action in the report designer.
OnPrintReport	The event occurs when printing a report from the preview.
OnExit	The event occurs when clicking the **Exit** button in the main menu of the designer.
OnDesignerAfterRender	The event occurs when the HTML5 code of the designer and all its controls have been completed.

7.2.10 Settings

The **Blazor Designer** is configured with the help of the component properties. Basic settings are set at the **Razor** component; all additional settings are set with the help of the special **StiBlazorDesignerOptions** options class. Below is an example of the designer properties setting.

Index.razor

```
@using Stimulsoft.Report
@using Stimulsoft.Report.Blazor
@using Stimulsoft.Report.Web

<StiBlazorDesigner ID="Viewer1" Width="500px" Height="500px"
Localization="Localization/en.xml" Options="@Options"
Theme="StiDesignerTheme.Office2022WhiteCarmine" />

@code
{
   //Options object
   private StiBlazorDesignerOptions Options;

   protected override void OnInitialized()
   {
      base.OnInitialized();

      //Init options object
      Options = new StiBlazorDesignerOptions();
      Options.FileMenu.Visible = false;
      Options.Dictionary.Visible = false;
      Options.Toolbar.ShowToolbar = false;
      Options.Components.ShowText = false;
      Options.Bands.ShowEmptyBand = false;
      Options.DashboardElements.ShowTableElement = false;
```

```
        Options.Behavior.SaveReportMode = StiSaveMode.Hidden;
    }
}
```

The Designer properties

Name	Description
ID	Allows you to set a string unique identifier of a component, by default **StiBlazorDesigner**.
Options	Allows you to set an object of the **StiBlazorDesignerOptions** class which contains a set of options (settings) for a component.
Report	Allows you to set report object that will be displayed in the designer.
Theme	It specifies the theme of the designer. The list of available themes is in the **StiDesignerTheme** enumeration. The **Office2022WhiteBlue** value is set by default.
Localization	It specifies the path to the XML localization file. The path can be absolute or relative. English localization is used by default.
Width	It specifies the width of a component in required units – pixels (by default), percent, and points. For example, «500», «500px», «100%», «300pt». The component expands to the entire area of the browser window by default.
Height	It specifies the height of a component in required units – pixels (by default), percent, and points. For example, «500, » «500px, » «100%, » «300pt. » The component expands to the entire area of the browser window by default.

Server

Name	Description

UseRelativeUrls	It sets the designer mode, where relative links are used for requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value that specifies the port number to use in a URL. The 0 value defines automatic detection (set by default). The -1 value removes the port number.
PassFormValues	Enables the passing of the POST-form values to the client side, if these values are required to use in the designer actions. When enabling the feature, the **GetFormValues()** helper method will return a collection of form parameters. By default, the property is set to **false**.
PassQueryParametersForResources	It enables the transferring of all parameters of the URL request when generating links to the designer sources. If it is set to **false**, only necessary parameters are used to request the designer sources. It helps the browser work more correctly with the cache. By default, the property is set to **true**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
CustomCss	Specifies the path to the CSS file of styles for the report designer. If this property is set, the standard styles of the selected theme will not be loaded. The default value is an empty value.
DefaultUnit	Sets the units for the size of the report and all its components. By default, centimeters are used.

Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. It can take one of the following values of the **StiZoomMode** enumeration: › **PageWidth** – when the designer runs, the zoom, necessary to display the report by the page width, will be set; › **PageHeight** – when the designer runs, the zoom, required to display the page height of the report, will be set.
ShowAnimation	Enables animation for various elements of the designer interface. By default, the property is set to **true**.
ShowOpenDialog	Allows to display the open dialog, or to open with the open event. By default, the property is set to **true**.
ShowTooltips	Enables displaying tooltips for designer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enable displaying links to online documentation in tooltips for designer controls. By default, the property is set to **true**.
ShowDialogsHelp	Enables displaying links to online documentation on the titles of dialog forms of the designer. By default, the property is set to **true**.
InterfaceType	Sets the type of interface used for the designer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the interface type of the designer will be selected automatically depending on the device used (default value); › **Mouse** – forced use of the interface to control the designer with the mouse; › **Touch** – forced use of the Touch interface to

	control the designer via the touch screen (mobile devices), also in this mode, the interface elements are increased.
DatePickerFirstDayOfWeek	Sets the first day of the week for the select date item. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Auto** – automatic detection of the first day of the week from the browser settings (default value); › **Monday** – the first day of the week is Monday; › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default, the property is set to **false**.
FormatForDateControls	This feature allows you to customize the format for date controls. By default, the current option does not have a specified value, and the date format is determined based on the browser's locale.
ShowReportTree	Enables displaying the tree of report components. By default, the property is set to **true**.
ReportDisplayMode	Sets the export mode for displaying report pages in the preview tab. Can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN

	markup.
ChartRenderType	Gets or sets the type of the chart in the preview. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
CloseDesignerWithoutAsking	Sets a value which indicates that the designer will be closed without asking. By default, the property is set to **true**.
ShowSystemFonts	Sets a visibility of the system fonts in the fonts list. By default, the property is set to **true**.
WizardTypeRunningAfterLoad	Calls the Report wizard after starting the report designer. It may have one of the following **StiWizardType** enumeration values: › **None** - runs the report designer without running the report wizard; › **StandardReport** - runs the Standard wizard; › **MasterDetailReport** - runs the Master-Detail wizard; › **LabelReport** - runs the Label wizard; › **InvoicesReport** - runs the Invoice wizard; › **OrdersReport** - runs the Order wizard; › **QuotationReport** - runs the Quote wizard.
AllowWordWrapTextEditors	Allows word wrap in the text editors. By default, the property is set to **true**.

Behavior

Name	Description
ShowSaveDialog	Enables displaying the dialog to insert a report name when it is saved. The name of the report will be transferred in the parameters of the report designer. By default, the property is set to **false**.
UndoMaxLevel	Sets the maximum number to cancel actions with the report (the Undo/Redo function). A big value of this property will consume memory on the server side to store the undo parameters. The default value is **6**.
AllowChangeWindowTitle	Allows using the title of the browser window to display the file name of the edited report. By default, the property is set to **true**.
SaveReportMode	Sets the mode for saving the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - saving of the report is called in the current web browser window in the visible mode using the POST request; › **NewWindow** - saving of the report is called in a new window (tab) of the web browser.
SaveReportAsMode	Sets the mode for saving the report. It has the three values of the **StiSaveMode** enumeration. › **Hidden** - saving of the report is called in the background mode using the AJAX request and is not shown in the browser window (default value); › **Visible** - saving of the report is called in the

	current web browser window in the visible mode using the POST request; › **NewWindow** - saving of the report is called in a new window (tab) of the web browser.
CheckReportBeforePreview	Sets the value that allows running the report checker before preview.

FileMenu

Name	Description
Visible	Enables displaying the main menu of the report designer. By default, the property is set to **true**.
ShowNew	Enables showing the main menu item - **New**. By default, the property is set to **true**.
ShowFileMenuNewReport	Sets a visibility of the new report button in the designer. By default, the property is set to **true**.
ShowFileMenuNewDashboard	Sets a visibility of the new dashboard button in the designer. By default, the property is set to **true**.
ShowOpen	Enables showing the main menu item - **Open**. By default, the property is set to **true**.
ShowSave	Enables showing the main menu item - **Save**. By default, the property is set to **true**.
ShowSaveAs	Enables showing the main menu item - **Save As**. By default, the property is set to **true**.
ShowClose	Enables showing the main menu item - **Close**. By default, the property is set to **true**.
ShowExit	Enables showing the main menu item - **Exit**. By default, the property is set to **false**.
ShowReportSetup	Enables showing the main menu item - **Report Setup**. By default, the property is set to **true**.
ShowOptions	Enables showing the main menu item - **Options**. By default, the property is set to **true**.

ShowInfo	Enables showing the main menu item - **Info**. By default, the property is set to **true**.
ShowAbout	Enables showing the main menu item - **About**. By default, the property is set to **true**.
ShowHelp	Enables showing the main menu item - **Help**. By default, the property is set to **true**.

Dictionary

Name	Description
Visible	Enables showing the data dictionary of the report. By default, the property is set to **true**.
UseAliases	Allows you to use aliases in the data dictionary. It has the three values of the **StiUseAliases** enumeration: › **Auto** - defines the mode of using aliases from a saved value in cookies (default value); › **True** - sets the mode of using aliases in the data dictionary; › **False** - disables the mode of using aliases in the data dictionary.
NewReportDictionary	It allows you to create a new data dictionary or join the existing one when creating a new report in the designer. It has the three values of the **StiNewReportDictionary** enumeration: › **Auto** - defines the mode to create or join the data dictionary from a saved value in cookies (default value); › **DictionaryNew** - sets the mode to create a new data dictionary when creating a new report; › **DictionaryMerge** - sets the mode to join the existing data dictionary with a new one when creating a new report in the designer.
ShowDictionaryContextMenuPro	Sets a visibility of the **Properties** item in the

perties	dictionary context menu. By default, the property is set to **true**.
ShowDictionaryActions	Sets a visibility of the **Actions** menu on the dictionary toolbar. By default, the property is set to **true**.
PermissionDataConnections	Sets the available actions to connect data to the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataSources	Sets available actions on report data sources. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataColumns	Sets the available actions on data columns in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionBusinessObjects	Sets the available actions on the business objects in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionDataRelations	Sets the available actions to linking data in the report. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionVariables	Sets available actions on report variables. It can take one or more values from the **StiDesignerPermissions** enumeration.
PermissionResources	Sets the available actions for the resources in the Report Dictionary. Takes one or several values from the **StiDesignerPermissions** enumeration.
PermissionSqlParameters	Sets the available actions for the parameters of the SQL queries for the Report DataSources. Takes one or several values from **StiDesignerPermissions** enumeration.
DataTransformationsPermissions	Sets the available actions on data transformation. It can take one or more values from the **StiDesignerPermissions** enumeration.

The table below shows all available values for the **StiDesignerPermissions** enumeration, which can be set for the dictionary elements of the report.

Value	Description
None	Disables any action on the item of the data dictionary.
All	Allows any action on the item of the data dictionary.
Create	Allows creating a specific data dictionary item.
Delete	Allows deleting a specific data dictionary item.
Modify	Allows modifying a specific data dictionary item.
View	Allows viewing a specific data dictionary item.
ModifyView	Allows modifying and viewing a specific data dictionary item.

Toolbar

Name	Description
ShowToolbar	Enables displaying the toolbar of the designer. By default, the property is set to **true**.
ShowSetupToolboxButton	Enables displaying the button to call the dialog box with settings for the side toolbar. By default, the property is set to **true**.
ShowInsertButton	Enables displaying the **Insert** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowLayoutButton	Enables displaying the tab **Layout** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowPageButton	Enables displaying the tab **Page** tab on the toolbar of the designer. By default, the property

	is set to **true**.
ShowPreviewButton	Enables displaying the tab **Preview** tab on the toolbar of the designer. By default, the property is set to **true**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the designer. By default, the property is set to **true**.
ShowAboutButton	Enables displaying the **About** on the toolbar of the designer. By default, the property is set to **false**.

PropertiesGrid

Name	Description
Visible	Enables displaying the property panel. By default, the property is set to **true**.
Width	Sets the width of the property panel. By default, the width is set to **370 px**.
LabelWidth	Specifies the width of the labels on the properties panel. By default, the width is set to **160 px**.
PropertiesGridPosition	Sets **Left** or **Right** position of the properties grid in the designer. It has the three values of the **StiPropertiesGridPosition** enumeration: › **Left**; › **Right**.
ShowPropertiesWhichUsedFromStyles	Sets a visibility of the properties which used from styles in the designer. By default, the property is set to **false**.

Components

Name	Description

ShowText	Enables displaying the **Text** component in the insert menu for report components. By default, the property is set to **true**.
ShowTextInCells	Enables displaying the **Text in Cells** component in the insert menu for report components. By default, the property is set to **true**.
ShowRichText	Enables displaying the **Rich Text** component in the insert menu for report components. By default, the property is set to **true**.
ShowImage	Enables displaying the **Image** component in the insert menu for report components. By default, the property is set to **true**.
ShowBarCode	Enables displaying the **Bar Code** component in the insert menu for report components. By default, the property is set to **true**.
ShowShape	Enables displaying the **Shape** component in the insert menu for report components. By default, the property is set to **true**.
ShowHorizontalLinePrimitive	Enables displaying the **Horizontal Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowVerticalLinePrimitive	Enables displaying the **Vertical Line** component in the insert menu for report components. By default, the property is set to **true**.
ShowRectanglePrimitive	Enables displaying the **Rectangle** component in the insert menu for report components. By default, the property is set to **true**.
ShowRoundedRectanglePrimitive	Enables displaying the **Rounded Rectangle** component in the insert menu for report components. By default, the property is set to **true**.
ShowPanel	Enables displaying the **Panel** component in the insert menu for report components. By default, the property is set to **true**.

ShowClone	Enables displaying the **Clone** component in the insert menu for report components. By default, the property is set to **true**.
ShowCheckBox	Enables displaying the **Check Box** component in the insert menu for report components. By default, the property is set to **true**.
ShowSubReport	Enables displaying the **Sub Report** component in the insert menu for report components. By default, the property is set to **true**.
ShowZipCode	Enables displaying the **Zip Code** component in the insert menu for report components. By default, the property is set to **true**.
ShowTable	Enables displaying the **Table** component in the insert menu for report components. By default, the property is set to **true**.
ShowCrossTab	Enables displaying the **Cross-Tab** component in the insert menu for report components. By default, the property is set to **true**.
ShowChart	Enables displaying the **Chart** component in the insert menu for report components. It affects on all chart types. By default, the property is set to **true**.
ShowMap	Enables displaying the **Map** component in the insert menu for report components. By default, the property is set to **false**.
ShowGauge	Enables displaying the **Gauge** component in the insert menu for report components. By default, the property is set to **false**.
ShowSparkline	Enables displaying the **Sparkline** component in the insert menu for report components. By default, the property is set to **true**.
ShowMathFormula	Enables displaying the **Math Formula** component in the insert menu for report components. By default, the property is set to **true**.

ShowElectronicSignature	Enables displaying the **Electronic Signature** component in the insert menu for report components. By default, the property is set to **true**.
ShowPdfDigitalSignature	Enables displaying the **PDF Digital Signature** component in the insert menu for report components. By default, the property is set to **true**.

Bands

Name	Description
ShowReportTitleBands	Enables displaying the **Report Title** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowReportSummaryBand	Enables displaying the **Report Summary** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageHeaderBand	Enables displaying the **Page Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowPageFooterBand	Enables displaying the **Page Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowGroupHeaderBand	Enables displaying the **Group Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowGroupFooterBand	Enables displaying the **Group Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowHeaderBand	Enables displaying the **Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowFooterBand	Enables displaying the **Footer** item in the **Bands**

	menu of the designer. By default, the property is set to **true**.
ShowColumnHeaderBand	Enables displaying the **Column Header** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowColumnFooterBand	Enables displaying the **Column Footer** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowDataBand	Enables displaying the **Data** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowHierarchicalBand	Enables displaying the **Hierarchical** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowChildBand	Enables displaying the **Child** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowEmptyBand	Enables displaying the **Empty** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowOverlayBand	Enables displaying the **Overlay** item in the **Bands** menu of the designer. By default, the property is set to **true**.
ShowTableOfContents	Enables displaying the **Table of Contents** item in the **Bands** menu of the designer. By default, the property is set to **true**.

DashboardElements

Name	Description
ShowTableElement	Enables displaying the **Table** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowCardsElement	Enables displaying the **Cards** element in the

	Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowChartElement	Enables displaying the **Chart** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowGaugeElement	Enables displaying the **Gauge** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowPivotTableElement	Enables displaying the **Pivot** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowIndicatorElement	Enables displaying the **Indicator** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowProgressElement	Enables displaying the **Progress** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowRegionMapElement	Enables displaying the **Region Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowOnlineMapElement	Enables displaying the **Online Map** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowImageElement	Enables displaying the **Image** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTextElement	Enables displaying the **Text** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowPanelElement	Enables displaying the **Panel** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowShapeElement	Enables displaying the **Shape** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.

ShowButtonElement	Enables displaying the **Button** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowListBoxElement	Enables displaying the **List Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowComboBoxElement	Enables displaying the **Combo Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTreeViewElement	Enables displaying the **Tree View** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowTreeViewBoxElement	Enables displaying the **Tree View Box** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.
ShowDatePickerElement	Enables displaying the **Date Picker** element in the Dashboard Elements menu of the designer. By default, the property is set to **true**.

CrossBands

Name	Description
ShowCrossGroupHeaderBand	Enables displaying the **Cross Group Header** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossGroupFooterBand	Enables displaying the **Cross Group Footer** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossHeaderBand	Enables displaying the **Cross Header** section in the insert menu for report components. By default, the property is set to **true**.
ShowCrossFooterBand	Enables displaying the **Cross Footer** section in

	the insert menu for report components. By default, the property is set to **true**.
ShowCrossDataBand	Enables displaying the **Cross Data** section in the insert menu for report components. By default, the property is set to **true**.

Dashboards

Name	Description
ShowNewDashboardButton	Sets a visibility of the **New Dashboard** button in the designer. By default, the property is set to **true**.

Pages

Name	Description
ShowNewPageButton	Sets a visibility of the **New Page** button in the designer. By default, the property is set to **true**.

When designing a report or dashboard in the report designer, you can also define **ExportOptions**, **EmailOptions**, and **PreviewToolbarOptions** on the **Preview** tab. These options are similar to the report viewer options.

8 Reports for Angular

.NET Core Angular is a cross-platform technology for creating Web applications for Windows, Linux and macOS. Stimulsoft provides tools for displaying reports using this technology.

8.1 Get Started

Step 1: Open Visual Studio;

Step 2: Click **File** menu, select **New** item, and choose **Project** item;

Step 3: Select **ASP.NET Core Web Application** command;

Step 4: Click **Next** button;

Step 5: Specify the name of this project, location and solution name. For example, **AngularViewer**;

Step 6: Click **Create** button;

Step 7: Select **Angular** type, set **.NET Core** as a framework and select **ASP.NET Core 3.1** and higher version;

Step 8: Uncheck **Configure for HTTPS** parameter;

Step 9: Click **Create** button;

Step 10: Install Stimulsoft.Reports.Angular.NetCore NuGet package:
- Select **Manage NuGet Packages ...** command in the context menu of the project;
- Specify **Stimulsoft.Reports.Angular.NetCore** in the search bar on the **Browse** tab;
- Select the item, define the version of the package, and click Install. When updating the package, click the **Update** button.

Step 11: Add **ViewerController** to Conntrollers folder:
- Call context menu of **Controllers** folder;
- Select **Add** item;
- Choose **Controller...** command;
- Set **MVC Controller - Empty** as controller type;
- Click **Add** button;
- Specify controller name. For example, **ViewerController**;
- Click the **Add** button.

Step 12: Create the **Reports** folder in the project, and add a report file to it. For example, add **MasterDetail.mrt** report.

Step 13: Add below code in **ViewerController.cs**.

ViewerController.cs

```
...
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Stimulsoft.Report;
using Stimulsoft.Report.Angular;
using Stimulsoft.Report.Web;

namespace AngularViewer.Controllers
   {
   [Controller]
   public class ViewerController : Controller
   {

      //Specify viewer options
      [HttpPost]
      public IActionResult InitViewer()
      {
         var requestParams = StiAngularViewer.GetRequestParams(this);
         var options = new StiAngularViewerOptions();
         options.Actions.ViewerEvent = "ViewerEvent";
         return StiAngularViewer.ViewerDataResult(requestParams, options);
      }

      //ViewerEvent() that will process viewer requests.
      [HttpPost]
      public IActionResult ViewerEvent()
      {
         var requestParams = StiAngularViewer.GetRequestParams(this);

         if (requestParams.Action == StiAction.GetReport)
         {
            var report = StiReport.CreateNewReport();
            var path = StiAngularHelper.MapPath(this, $"Reports/
            MasterDetail.mrt");
            report.Load(path);
            return StiAngularViewer.GetReportResult(this, report);
         }

         return StiAngularViewer.ProcessRequestResult(this);
      }
   }
}
...
```

Step 14: Open project folder in explorer;

Step 15: Install Angular Client Components from npm.

```
npm install stimulsoft-viewer-angular
```

```
```

Step 16: Close console;

Step 17: Delete the **ClientApp** folder;

Step 18: Go to the **AngularViewer** folder and open console;

Step 19: Enter **ng new ClientApp** command in console:

```
ng new ClientApp
```

Step 20: Enter '**N**' to not use routing;

Step 21: Select the **CSS** format and click Enter button;

Step 22: Close console and go to **ClientApp** folder;

Step 23: Open console and download **stimulsoft-viewer-angular**.

```
npm install stimulsoft-viewer-angular
```

Step 24: Close console;

Step 25: Open **app.module.ts** file in the editor from "**...ClientApp\src\app**" directory, and add **StimulsoftViewerModule**. Specify below code in **app.module.ts**:

app.module.ts

```
...
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
```

```
import { StimulsoftViewerModule } from 'stimulsoft-viewer-angular';
import { HttpClientModule } from '@angular/common/http';
import { BrowserAnimationsModule } from '@angular/platform-browser/
animations';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

@NgModule({
   declarations: [
      AppComponent
   ],
   imports: [
      BrowserModule,
      StimulsoftViewerModule
      HttpClientModule,
      BrowserAnimationsModule,
      FormsModule
   ],
   providers: [],
   bootstrap: [AppComponent]
})
export class AppModule { }
...
```

Step 26: Open **app.component.html** file in the editor from "**...ClientApp\src\app**" directory, and add **AppComponent**. Specify below code in **app.component.html**:

app.component.html

```
...
<stimulsoft-viewer-angular
   [requestUrl]="'http://localhost:60801/Viewer/{action}'"
   [action]="'InitViewer'"
   [height]="'600px'"
></stimulsoft-viewer-angular>
...
```

Where:

- **[requestUrl]** is URL to ViewerController with {action} template. You can find out the URL to **ViewerController** in launchSettings.json file from the "**...\Properties**" directory.
- **[action]** is an action name to initialize an angular viewer.
- **[height]** is viewer height.

Step 27: Go to the visual studio and run the project. After the project run, you can see the viewer with a specific report.

Information

If your project does not start, try to delete node_modules folder and package-lock.json file from the "**...ClientApp**" directory.

8.2 Activation

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiAngularViewer** component.

ViewerController.cs

```
...
//Activation with using license code
public class ViewerController : Controller
{
   public ViewerController()
   {
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";
   }
}

//Activation with using license file
public class ViewerController : Controller
{
   public ViewerController(IWebHostEnvironment hostEnvironment)
   {
      var path = Path.Combine(hostEnvironment.ContentRootPath, "Content\
      \license.key");
      Stimulsoft.Base.StiLicense.LoadFromFile(path);
   }
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

8.3 Angular Viewer

YouTube

Watch videos .NET Core Angular Viewer. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

Samples

See on GitHub examples for working with the Angular components. All examples are separate projects, grouped into one solution for Visual Studio.

The **Angular Viewer (StiAngularViewer)** component is designed to view reports in the web browser. You do not need to install the .NET Framework, ActiveX components or any special plug-ins on the client side. All that is needed is any modern Web browser.

With help of **Angular Viewer**, you can view, print and export reports on any computer with any operating system installed. Since the viewer only uses HTML and JavaScript technologies, it can be run on devices where there is no Flash or Silverlight support - tablets, smartphones. Also, the viewer supports the Touch interface, which is automatically enabled when using devices with a touch screen.

The **Angular Viewer** component uses the AJAX technology to perform all actions (uploading a report, paging, scaling, interactivity in reports, etc.), which allows you to get rid of reloading the entire page, and save Web traffic and speed up work. The report engine built using the .NET Core technology is used to render reports. This is a cross-platform technology. It allows you to deploy the application on servers that use the operating systems like Windows, macOS, and Linux.

Angular Viewer supports many themes, animated interface, bookmarks, interactive reports, editing of report elements on the page, full screen mode, search panel, and other necessary features for viewing reports.

To use the **Angular Viewer** in a your projects, you need to install:

NetCore

To use the **Angular Viewer** in a Web project, you need to install the NuGet package

of **Stimulsoft.Reports.Angular.NetCore**:
› Select "Manage NuGet Packages ..." in the context menu of the project;
› Specify Stimulsoft.Reports.Angular.NetCore in the search bar on the Browse tab;
› Select the item, define the version of the package, and click **Install**. When updating the package, click the **Update** button.

Angular
The easiest way is to install through npm:
```
npm install stimulsoft-viewer-angular
```


8.3.1 How this Works

To run the viewer, you need to place the **stimulsoft-viewer-angular** component on the page, set the necessary settings to it, and set the necessary actions in the view controller. When the report viewer runs, the following actions occur:
› When the component is output, the JavaScript method is launched. It requests the first page of the report on the server side or the entire report (depending on the selected mode) and the required report parameters;
› Each action in the viewer (for example, paging, printing or exporting a report, etc.) calls a certain action on the server side, in which you can perform the necessary manipulations with the report;

› To speed up the work, the viewer saves the report in cache or server session, which makes it impossible to re-build the report.

8.3.2 Showing Reports

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `report.render()` method if you want to perform specific actions with the rendered report before it is displayed in the viewer.

To show the report, you need to add the **stimulsoft-viewer-angular** component to the template and set it to the minimum required properties: request URL template & action name.

component.ts

```
...
<stimulsoft-viewer-angular [requestUrl]="'http://server.url:51528/Viewer/{action}'" [action]="'Post'" ></stimulsoft-viewer-angular>
...
```

And in the view controller, specify the necessary actions.

HomeController.cs

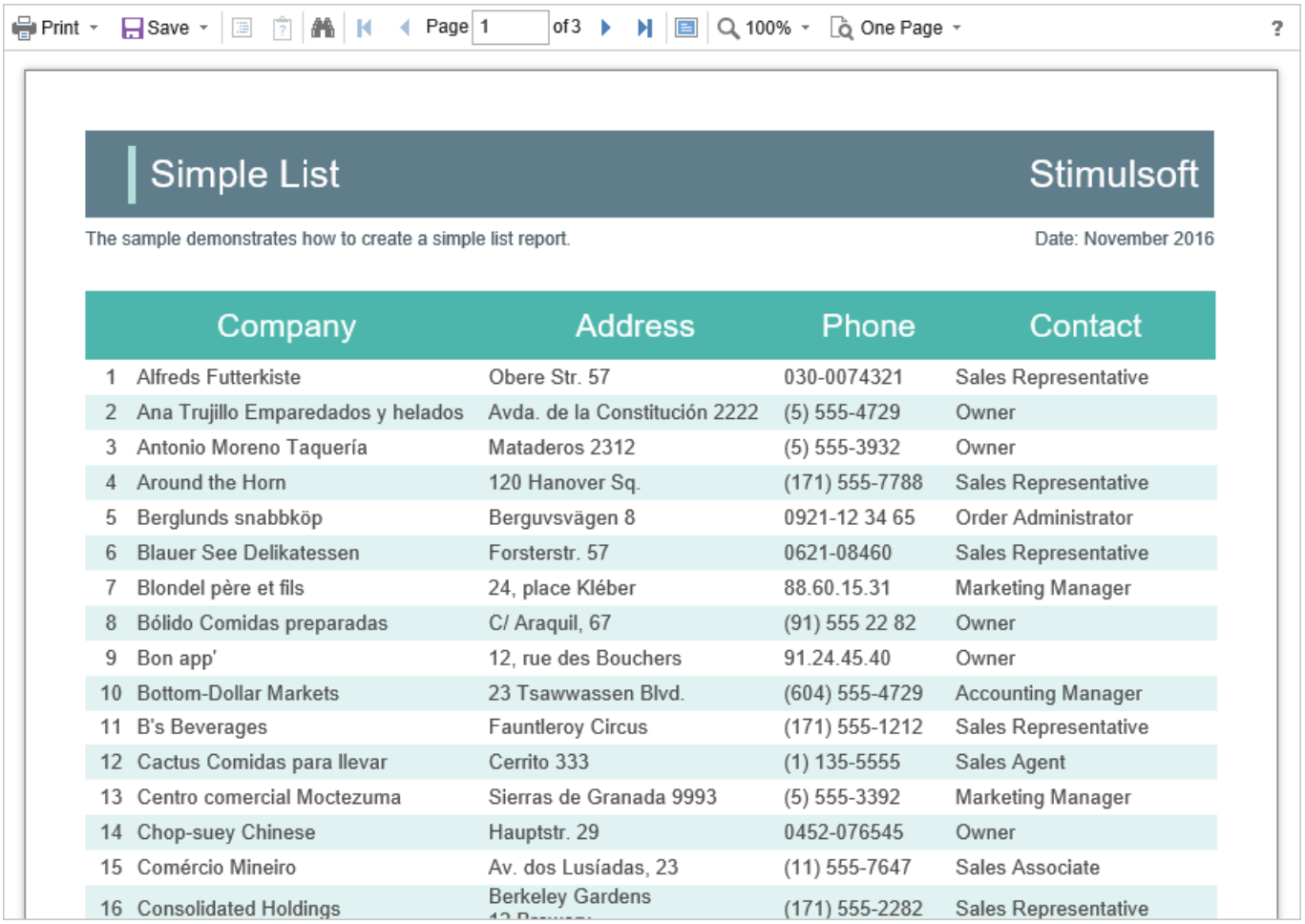
```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}

public IActionResult ViewerEvent()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);

   if (requestParams.Action == StiAction.GetReport)
   {
      return GetReport(reportName);
   }
   return StiAngularViewer.ProcessRequestResult(this);
```

```
}

public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(StiAngularHelper.MapPath(this, "Reports/SimpleList.mrt"));

   return StiAngularViewer.GetReportResult(this, report);
}
...
```

In the above example, processing of actions of the viewer is added. Angular viewer initialized by action configured in `[requestUrl] & [action]` here you need to specify **StiAngularViewerOptions** and **ViewerEvent** action. The **ViewerEvent** action handles the viewer events and if action is **GetReport** returns the report prepared for preview.

Information

The **ViewerEvent** action is mandatory. Without it, the correct operation of the viewer is not possible.



If the report was not rendered before showing, the **Angular Viewer** component will automatically render it. So you can use report templates and rendered reports to display reports.

HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.LoadDocument(StiAngularHelper.MapPath(this, "Reports/
   SimpleList.mdc"));

   return StiAngularViewer.GetReportResult(this, report);
}
...
```

8.3.3 Connecting Data

Data to a report can be connected in various ways. The easiest way is to store connection settings in the report template. You can also connect the data from the code, this can be done when the report is loaded in the **GetReport** action.

HomeController.cs

```
...
public IActionResult GetReport()
{
   DataSet ds = new DataSet();
   ds.ReadXml(StiAngularHelper.MapPath(this, "Data/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(StiAngularHelper.MapPath(this, "Reports/
   TwoSimpleLists.mrt"));
   report.Dictionary.Databases.Clear();
   report.RegData("Demo", ds);

   return StiAngularViewer.GetReportResult(this, report);
}
...
```

Data for the report can be connected not only when the report is loaded. For example, you can connect new data at the moment of interactive actions in the viewer (applying report parameters, sorting, drill-down, collapsing). To do this, you should set the **Interaction** action for the **Angular Viewer** component, and, in the action handler, connect the data for the current report. The same way you can connect data in other actions of the viewer.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Actions.Interaction = "ViewerInteraction";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

HomeController.cs

```
...
public IActionResult ViewerInteraction()
```

```
{
   DataSet data = new DataSet();
   data.ReadXml(StiAngularHelper.MapPath(this, "Data/Demo.xml"));

   StiReport report = StiAngularViewer.GetReportObject(this);
   report.RegData("Demo", data);

   return StiAngularViewer.InteractionResult(this, report);
}
...
```

If you want to connect new data only for a certain interactive action of the viewer, for example, only when you apply report parameters, you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class, they are passed to any server side on any request, and contain all necessary information and states of the client part of the viewer. To determine the type of the action of the viewer, it is enough to check the **Action** property of the viewer parameters.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   StiRequestParams requestParams =
   StiAngularViewer.GetRequestParams(this);
   if (requestParams.Action == StiAction.Variables)
   {
      DataSet data = new DataSet();
      data.ReadXml(StiAngularHelper.MapPath(this, "Data/Demo.xml"));

      StiReport report = StiAngularViewer.GetReportObject(this);
      report.RegData("Demo", data);

      return StiAngularViewer.InteractionResult(this, report);
   }

   return StiAngularViewer.InteractionResult(this);
}
...
```

SQL data sources

The connection parameters to the SQL data source, as well as to any other ones, can be stored in the report template. If you want to set the connection parameters from the code before rendering the report (for example, for security reasons or depending on the authorized user), you can use the example below.

HomeController.cs

```
...
public IActionResult GetReport()
{
   OracleConnection connection = new OracleConnection("Data
   Source=Oracle8i;Integrated Security=yes");
   connection.Open();
   OracleDataAdapter adapter = new OracleDataAdapter();
   adapter.SelectCommand = new OracleCommand("SELECT * FROM Products",
   connection);

   DataSet dataSet = new DataSet("productsDataSet");
   adapter.Fill(dataSet, "Products");

   StiReport report = new StiReport();
   report.Load(StiAngularHelper.MapPath(this, "Reports/
   SqlSampleReport.mrt"));
   report.RegData("Products", dataSet);

   return StiAngularViewer.GetReportResult(this, report);
}
...
```

Also, for SQL data sources used in the report, you can specify the Query Timeout in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to change the connection string for MS SQL, adjust the query, set the query timeout for the already created connection, and data sources in the report.

HomeController.cs

```
...
public IActionResult GetReport()
{

   StiReport report = new StiReport();
   report.Load(StiAngularHelper.MapPath("Report.mrt"));
   ((StiSqlDatabase)
   report.Dictionary.Databases["Connection"]).ConnectionString = @"Data
   Source=server;Integrated Security=True;Initial Catalog=DataBase";
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).SqlCommand = "select *
   from Table where Column = 100";
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   return StiAngularViewer.GetReportResult(this, report);
```

```
}
...
```

Information

For SQL data sources of other types, the connection is created similarly, and an adapter corresponding to the type of the data source is connected. For example, for the MS SQL data source, you should connect SqlDataAdapter, for OracleDataAdapter is required for Oracle. Also, you should specify a connection string that matches the connection type.

The table below shows the connection string templates for different types of data sources.

Data Source	Connection String Template
MS SQL	Integrated Security=False; Data Source=myServerAddress;Initial Catalog=myDataBase; User ID=myUsername; Password=myPassword;
MySQL	Server=myServerAddress; Database=myDataBase;UserId=myUsername; Pwd=myPassword;
ODBC	Driver={SQL Server}; Server=myServerAddress;Database=myDataBase ; Uid=myUsername; Pwd=myPassword;
OLE DB	Provider=SQLOLEDB.1; Integrated Security=SSPI;Persist Security Info=False; Initial Catalog=myDataBase;Data Source=myServerAddress
Oracle	Data Source=TORCL;User Id=myUsername;Password=myPassword;
MS Access	Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Password=pass;Data Source=C:\ \myAccessFile.accdb;

PostgreSQL	Server=myServerAddress; Port=5432; Database=myDataBase;User Id=myUsername; Password=myPassword;
Firebird	User=SYSDBA; Password=masterkey; Database=SampleDatabase.fdb;DataSource=myServerAddress; Port=3050; Dialect=3; Charset=NONE;Role=; Connection lifetime=15; Pooling=true; MinPoolSize=0;MaxPoolSize=50; Packet Size=8192; ServerType=0;
SQL CE	Data Source=c:\MyData.sdf; Persist Security Info=False;
SQLite	Data Source=c:\mydb.db; Version=3;
DB2	Server=myAddress:myPortNumber;Database=myDataBase;UID=myUsername;PWD=myPassword; Max Pool Size=100;Min Pool Size=10;
Infomix	Database=myDataBase;Host=192.168.10.10;Server=db_engine_tcp;Service=1492;Protocol=onsoctcp;UID=myUsername;Password=myPassword;
Sybase	Data Source=myASEserver;Port=5000;Database=myDataBase;Uid=myUsername;Pwd=myPassword;
Teradata	Data Source=myServerAddress;User ID=myUsername;Password=myPassword;
VistaDB	Data Source=D:\folder \myVistaDatabaseFile.vdb4;Open Mode=ExclusiveReadWrite;
Universal(dotConnect)	Provider=Oracle;direct=true;data source=192.168.0.1;port=1521;sid=sid;user=user; password=pass
MongoDB	mongodb://<user>:<password>@localhost/test
OData	http://services.odata.org/v3/odata/OData.svc/

Information

The table shows the most commonly used templates for the connection string. You can view various connection string options at the special website.

Data from XML, JSON, Excel files

Connecting to XML and JSON data sources can be stored in the report template. If you want to specify data files from the code, you can use the example below.

HomeController.cs

```
...
public IActionResult GetReport()
{
   DataSet data = new DataSet();
   data.ReadXml(StiAngularHelper.MapPath(this, "Data/Demo.xml"));

   StiReport report = new StiReport();
   report.Load(StiAngularHelper.MapPath(this, "Reports/SimpleList.mrt"));
   report.RegData(data);

   return StiAngularViewer.GetReportResult(this, report);
}
...
```

HomeController.cs

```
...
public IActionResult GetReport()
{
   DataSet data
   = StiJsonToDataSetConverterV2.GetDataSetFromFile(StiAngularHelper.MapPat
   h(this, "Data/Demo.json"));

   StiReport report = new StiReport();
   report.Load(StiAngularHelper.MapPath(this, "Reports/SimpleList.mrt"));
   report.RegData(data);

   return StiAngularViewer.GetReportResult(this, report);
}
...
```

Information

The viewer has the possibility of obtaining data from an Excel file. To do this, you can use the following method.

```
DataSet dataSet = StiExcelConnector.Get().GetDataSet(new StiExcelOptions(ar
```

8.3.4 Localization

The **Angular Viewer** component supports the complete localization of its interface. To localize the report viewer interface, use the special **Localization** property. The value of this property should specify the path to the localization XML file (relative or absolute).

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Localization = "Localization/en.xml";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

When you load the report viewer, the localization file will be loaded automatically.

8.3.5 Using Themes

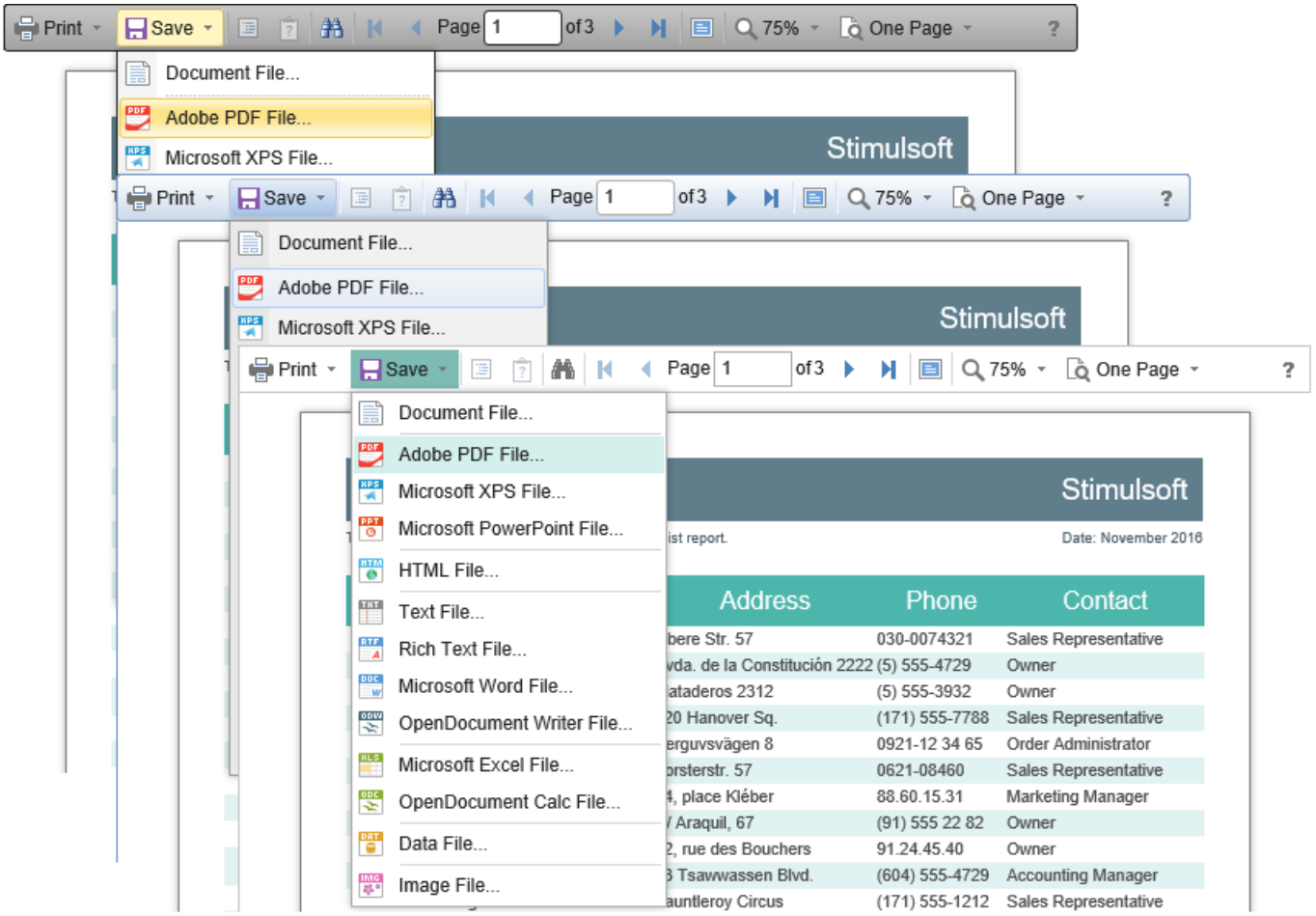
The **Angular Viewer** component can change the appearance of visual controls. To change the theme, use the **Theme** property, which can take one of the values of the **StiViewerTheme** enumeration.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Theme = StiViewerTheme.Office2022WhiteTeal;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

There are currently **8 themes** available with different color accents. As a result, **more**

than 60 variants of the appearance are available. This allows you to customize the appearance of the viewer for almost any design of the Web project.

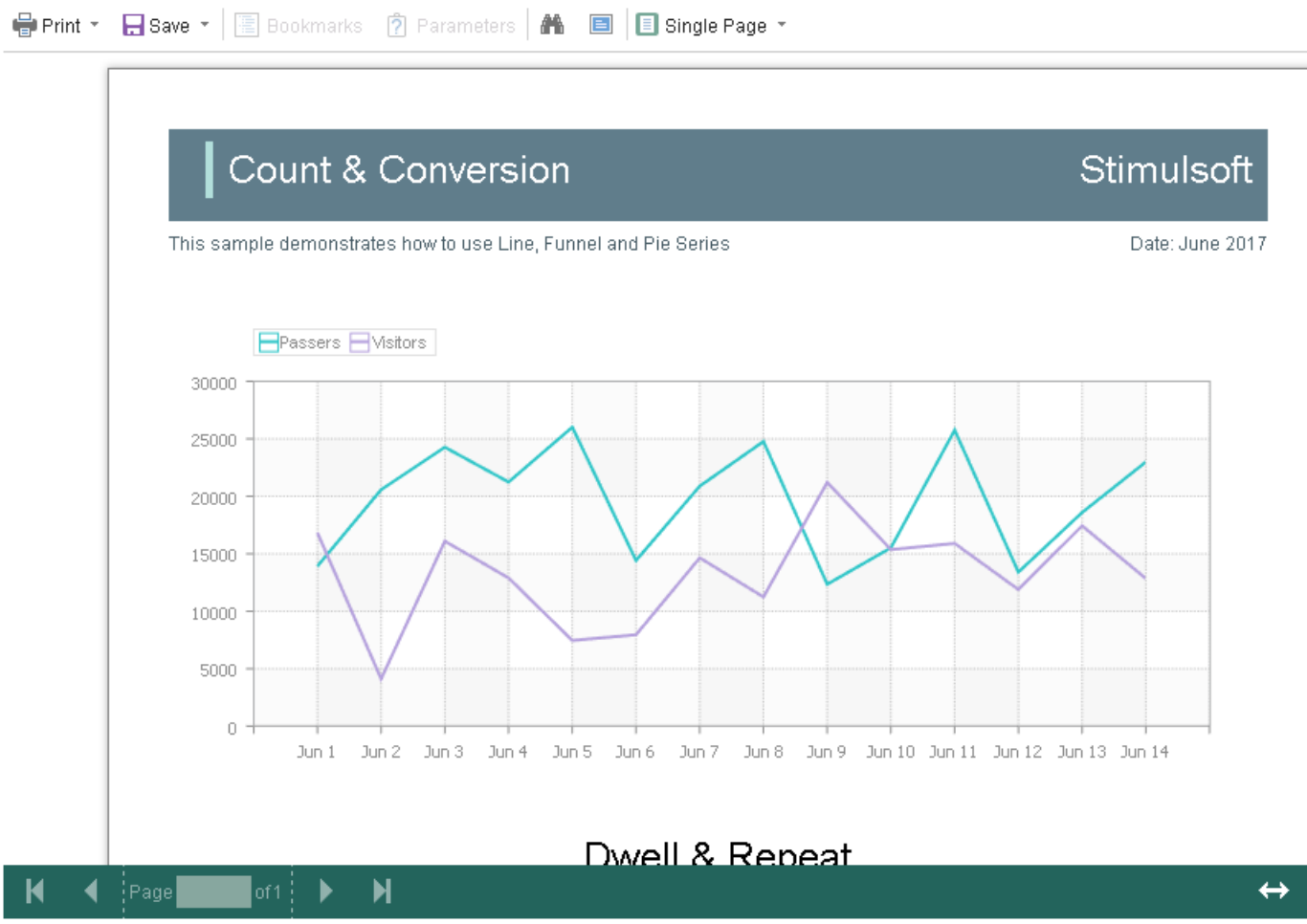


By default, the viewer has only the top toolbar on which all the report controls are located. If necessary, the toolbar can be split into top and bottom parts. The top panel will contain the menu for printing and exporting the report, as well as the buttons for working with parameters and bookmarks. The bottom panel will contain controls to switch between the report pages and setting zoom of pages. To enable this mode, enable the **ToolbarDisplayMode** property. It has values **Simple** and **Separated**.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
```

```
    var options = new StiAngularViewerOptions();
    options.Actions.ViewerEvent = "ViewerEvent";
    options.Appearance.ScrollbarsMode = true;
    options.Toolbar.DisplayMode = StiToolbarDisplayMode.Separated;

    return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```



In addition, it is possible to set the appearance parameters for the main elements of the viewer. For example, you can change the font and color of the control panel inscriptions of the viewer, set the background of the viewer, set the color of page borders, etc. Below is a list of available properties that change the appearance of the viewer, and their default values.

HomeController.cs

```
...
```

```
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Appearance.BackgroundColor = Color.White;
   options.Appearance.PageBorderColor = Color.Blue;
   options.Appearance.ShowPageShadow = true;
   options.Toolbar.BackgroundColor = Color.White;
   options.Toolbar.BorderColor = Color.Gray;
   options.Toolbar.FontColor = Color.Black;
   options.Toolbar.FontFamily = "Arial";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

8.3.6 Basic Features

The main features of the viewer include the following operations: displaying the report, switching between the report pages, changing the scale and displaying the preview mode. All specified operations are performed in the AJAX mode without restarting the browser page. For the correct work of these operations, you should define a special **ViewerEvent** action.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}

public IActionResult ViewerEvent()
{
   // Some code before viewer event
   // ...

   return StiAngularViewer.ViewerEventResult(this);
}
...
```

Information

This action is mandatory. Without it, the correct operation of the viewer is not possible.

The **ViewerEvent** action returns a prepared HTML page of the report (or set of pages), built taking into account the current state of the viewer. If necessary, you can change the parameters of the current report in the specified action, as well as update the report data in case of interactive actions of the viewer.

8.3.7 Printing Reports

The **Angular Viewer** component provides several options for printing a report. Each has its own advantages and disadvantages.

Print to PDF
Printing will be done by exporting the report to the **PDF format**. The advantages are greater accuracy of positioning and printing of the report elements in comparison with other printing options. Among the drawbacks, one can mention the mandatory presence of a plug-in installed in a web browser for viewing PDF files (modern browsers have embedded PDF viewer and printer).

Print with Preview
The report will be printed in a separate pop-up browser window in the **HTML format**. The report can be previewed, and then sent to the printer or copied to another location in as text or HTML code. Advantages - cross-browser compatibility when printing, no need to install special plug-ins. The disadvantage is the relatively low accuracy of the position of the report elements, due to the peculiarities of the implementation of HTML formatting.

Print without Preview
The report will be printed directly to the printer without preview. After selecting this menu item, the system print dialog is displayed. Since printing in this mode is carried out in the **HTML format**, then the print quality is similar to the quality of printing a report with a preview.

Information

When printing to the **HTML format**, you should check the compliance of report page settings and printer parameters (paper size, orientation, margins, indents), as well as check your browser print settings, such as margins, headers, footers, watermarks printing, color printing.

The print function does not require additional settings for the viewer. If you need to perform any actions before printing a report, you can define a special **PrintReport** action.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Actions.PrintReport = "PrintReport";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}

public IActionResult PrintReport()
{
   // Some code before print
   // ...

   return StiAngularViewer.PrintReportResult(this);
}
...
```

Print setup

If you choose printing a report in the viewer panel, a menu with printing options is displayed. The **Angular Viewer** component is able to force the required printing mode. To do this, set the **PrintDestination** property to one of the following values of the **StiPrintDestination** enumeration.

- **Default** – the menu will be displayed (the default property value);
- **Pdf** – print to the PDF format;
- **Direct** – printing to the HTML format directly to the printer, the system print dialog will be displayed;
- **WithPreview** – print to the HTML format with preview in a pop-up window.

Index.cshtml

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
```

```
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Toolbar.PrintDestination = StiPrintDestination.Default;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

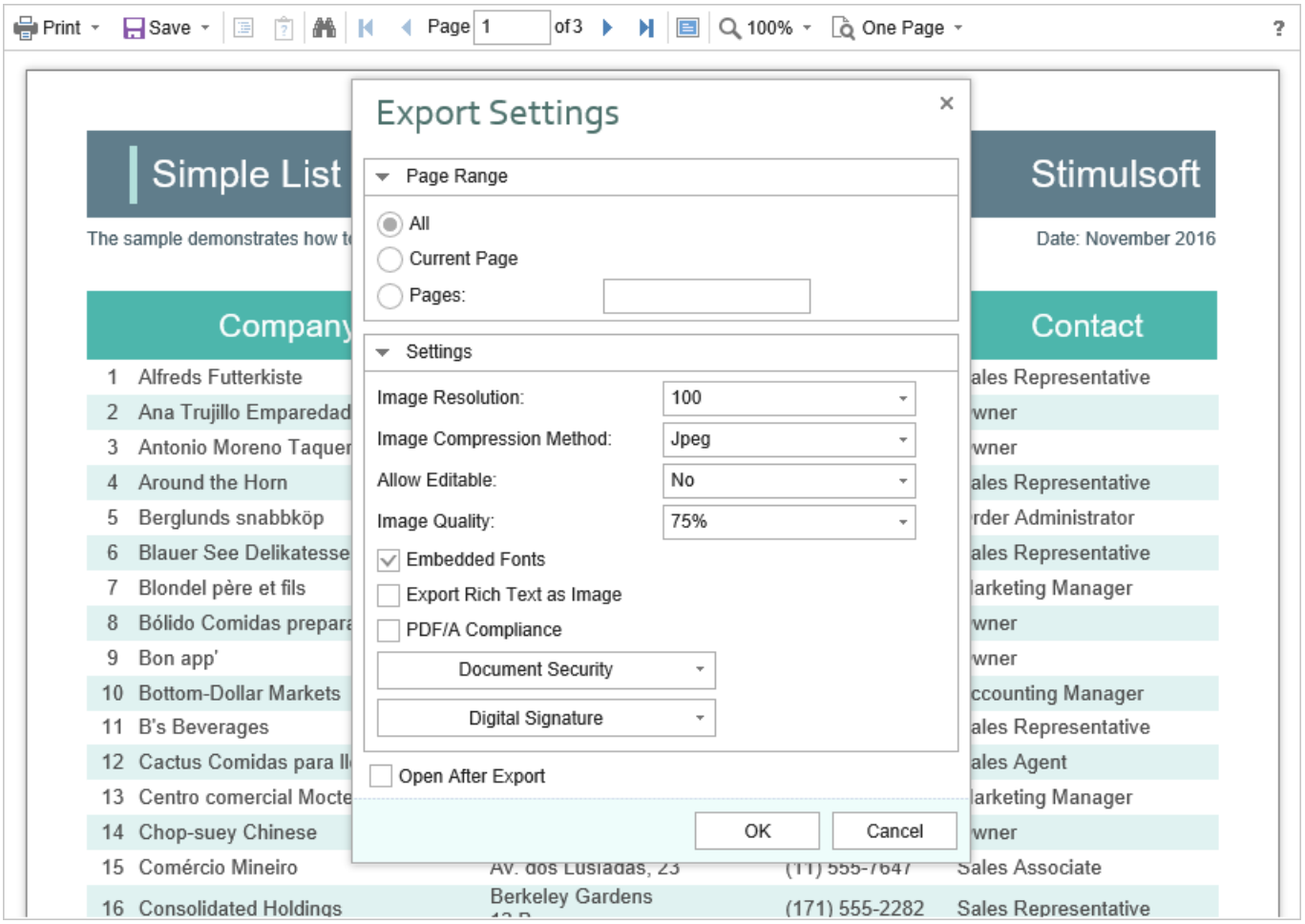
The **Angular Viewer** component is able to completely disable report printing. To do this, set the value of the **ShowPrintButton** property to **false**.

Index.cshtml

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Toolbar.ShowPrintButton = false;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

8.3.8 Exporting Reports

The **Angular Viewer** component allows you to export the displayed report to three dozen of various formats, such as **PDF**, **HTML**, **Word**, **Excel**, **XPS**, **RTF**, images, text and others.



The export function does not require additional settings for the viewer. If you need to perform any actions before exporting the report, you can define a special **ExportReport** action.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Actions.ExportReport = "ExportReport";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}

public IActionResult ExportReport()
{
   // Some code before export
```

```
    // ...

    return StiAngularViewer.ExportReportResult(this);
}
...
```

Export settings

Each report export format of the **Angular Viewer** component has a lot of settings, and each setting has its own default values. Sometimes you need to set other default values. For this purpose, a special **DefaultSettings** property of the viewer is used. It is a container for all the default export settings.

HomeController.cs

```
...
public IActionResult InitViewer()
{
    var requestParams = StiAngularViewer.GetRequestParams(this);
    var options = new StiAngularViewerOptions();
    options.Actions.ViewerEvent = "ViewerEvent";
    options.Exports.DefaultSettings.ExportToPdf.ImageQuality = 0.75f;
    options.Exports.DefaultSettings.ExportToPdf.ImageFormat =
    Stimulsoft.Report.Export.StiImageFormat.Color;
    options.Exports.DefaultSettings.ExportToHtml.ExportMode =
    Stimulsoft.Report.Export.StiHtmlExportMode.Div;
    options.Exports.DefaultSettings.ExportToHtml.UseEmbeddedImages = true;

    return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

If it is required, you can completely hide export dialogs. Exporting will always be done with default settings. For this, it is enough to set the value of the **ShowExportDialog** property to **false**.

HomeController.cs

```
...
public IActionResult InitViewer()
{
    var requestParams = StiAngularViewer.GetRequestParams(this);
    var options = new StiAngularViewerOptions();
    options.Actions.ViewerEvent = "ViewerEvent";
    options.Exports.ShowExportDialog = false;

    return StiAngularViewer.ViewerDataResult(requestParams, options);
}
```

```
...
```

The **Angular Viewer** component contains 30+ export formats, and sometimes you need to disable unwanted formats. This allows you to simplify UI and the use of the viewer. To disable unused export formats, it is enough to set the values for the corresponding properties of the viewer listed in the list below to **false**.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Exports.ShowExportToDocument = true;
   options.Exports.ShowExportToPdf = true;
   options.Exports.ShowExportToXps = true;
   options.Exports.ShowExportToPowerPoint = true;
   options.Exports.ShowExportToHtml = true;
   options.Exports.ShowExportToHtml5 = true;
   options.Exports.ShowExportToMht = true;
   options.Exports.ShowExportToText = true;
   options.Exports.ShowExportToRtf = true;
   options.Exports.ShowExportToWord2007 = true;
   options.Exports.ShowExportToOpenDocumentWriter = true;
   options.Exports.ShowExportToExcel = true;
   options.Exports.ShowExportToExcelXml = true;
   options.Exports.ShowExportToExcel2007 = true;
   options.Exports.ShowExportToOpenDocumentCalc = true;
   options.Exports.ShowExportToCsv = true;
   options.Exports.ShowExportToDbf = true;
   options.Exports.ShowExportToXml = true;
   options.Exports.ShowExportToDif = true;
   options.Exports.ShowExportToSylk = true;
   options.Exports.ShowExportToImageBmp = true;
   options.Exports.ShowExportToImageGif = true;
   options.Exports.ShowExportToImageJpeg = true;
   options.Exports.ShowExportToImagePcx = true;
   options.Exports.ShowExportToImagePng = true;
   options.Exports.ShowExportToImageTiff = true;
   options.Exports.ShowExportToImageMetafile = true;
   options.Exports.ShowExportToImageSvg = true;
   options.Exports.ShowExportToImageSvgz = true;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

The **Angular Viewer** component can completely disable the export menu. To do

this, set the value of the **ShowSaveButton** property to **false**.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Toolbar.ShowSaveButton = false;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

8.3.9 Viewing Modes

The **Angular Viewer** component has two modes for displaying reports - with and without scrollbars. By default, the view mode without scrollbars is set. To enable the scrollbar view mode, set the value of the **ScrollbarsMode** property to **true**.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Appearance.ScrollbarsMode = true;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

In the first mode (without scrollbars), the viewer displays a page or report as a whole, automatically stretching the preview space. If the width and height are specified, the viewer will truncate the page that is out of bounds. In the second mode, unlike the first one, when the page is out of bounds of the viewer's size, no truncation will be performed. Scrollbars will appear, using which you can view the entire page or report.

Information

In the report mode with scrollbars, you should set the height of the viewer,

otherwise the default height will be set to **650 pixels**.

```
<stimulsoft-viewer-angular [height]="'500px'" …
```

The **Angular Viewer** component provides the full-screen report mode. By default, the standard view mode is enabled, the viewer has the specified dimensions in the settings. To enable the full-screen mode, set the **FullScreenMode** property to **true**.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Appearance. FullScreenMode = true;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

Also, to enable or disable the full-screen mode, you can use the corresponding button on the control panel of the viewer.

The **Angular Viewer** component has three modes to display reports - page-by-page, entire report, and tabular display of report pages. To control the modes, the **ViewMode** property is used. It can have one of the specified values - **SinglePage**, **Continuous**, **MultiplePages**.

HomeController.cs

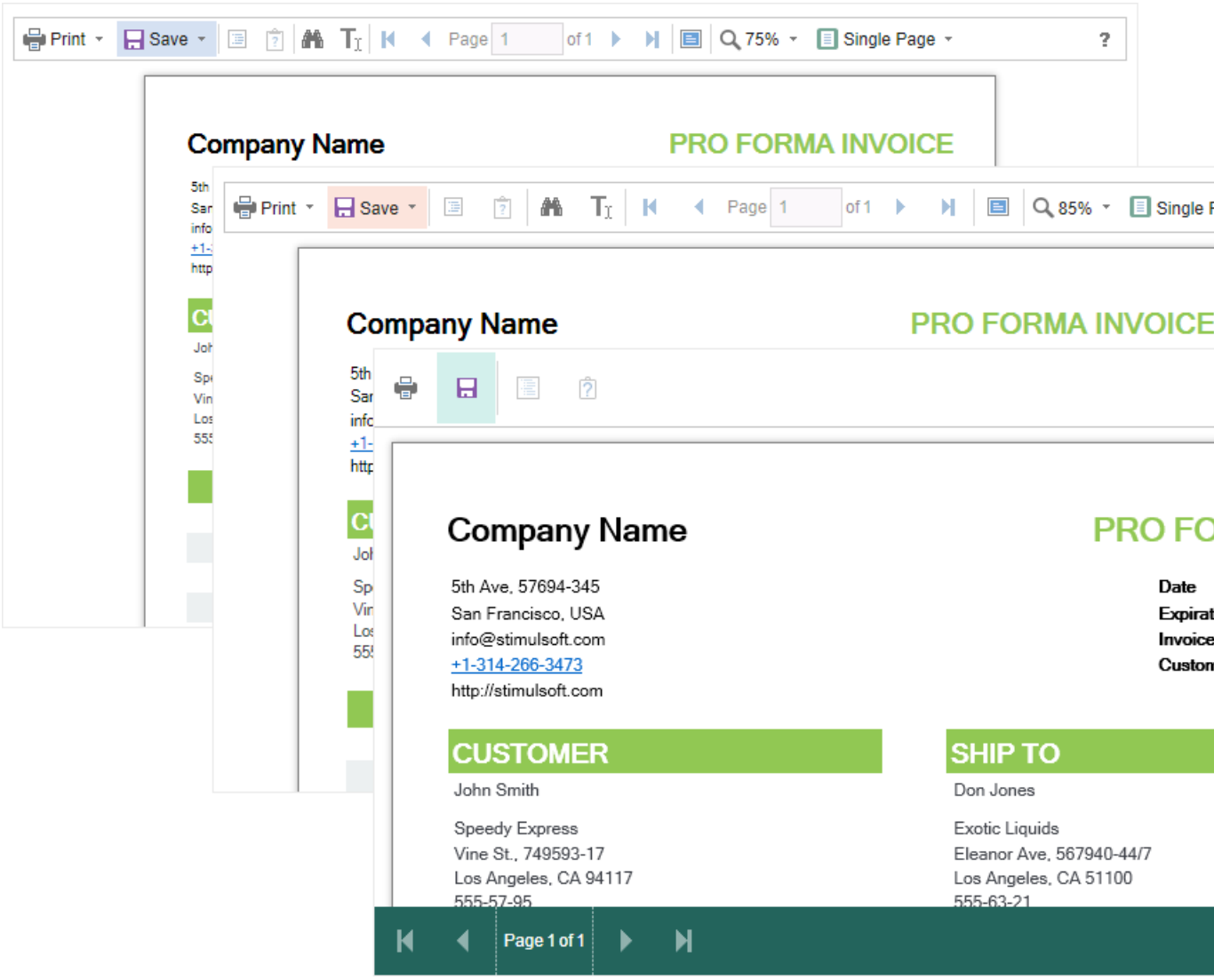
```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Toolbar.ViewMode = StiWebViewMode.SinglePage;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

Angular Viewer component supports interaction on a regular PC display and work with a touchscreen of screens and of the mobile devices. **InterfaceType** property allows to control the interface modes. The property can have one of the following values:

- **Auto** – the viewer's interface is determined automatically depending of the device that is report is displayed on. That is the default value.
- **Mouse** – the standard interface with a mouse control will be used for all the screen types.
- **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability.
- **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with the smartphones.

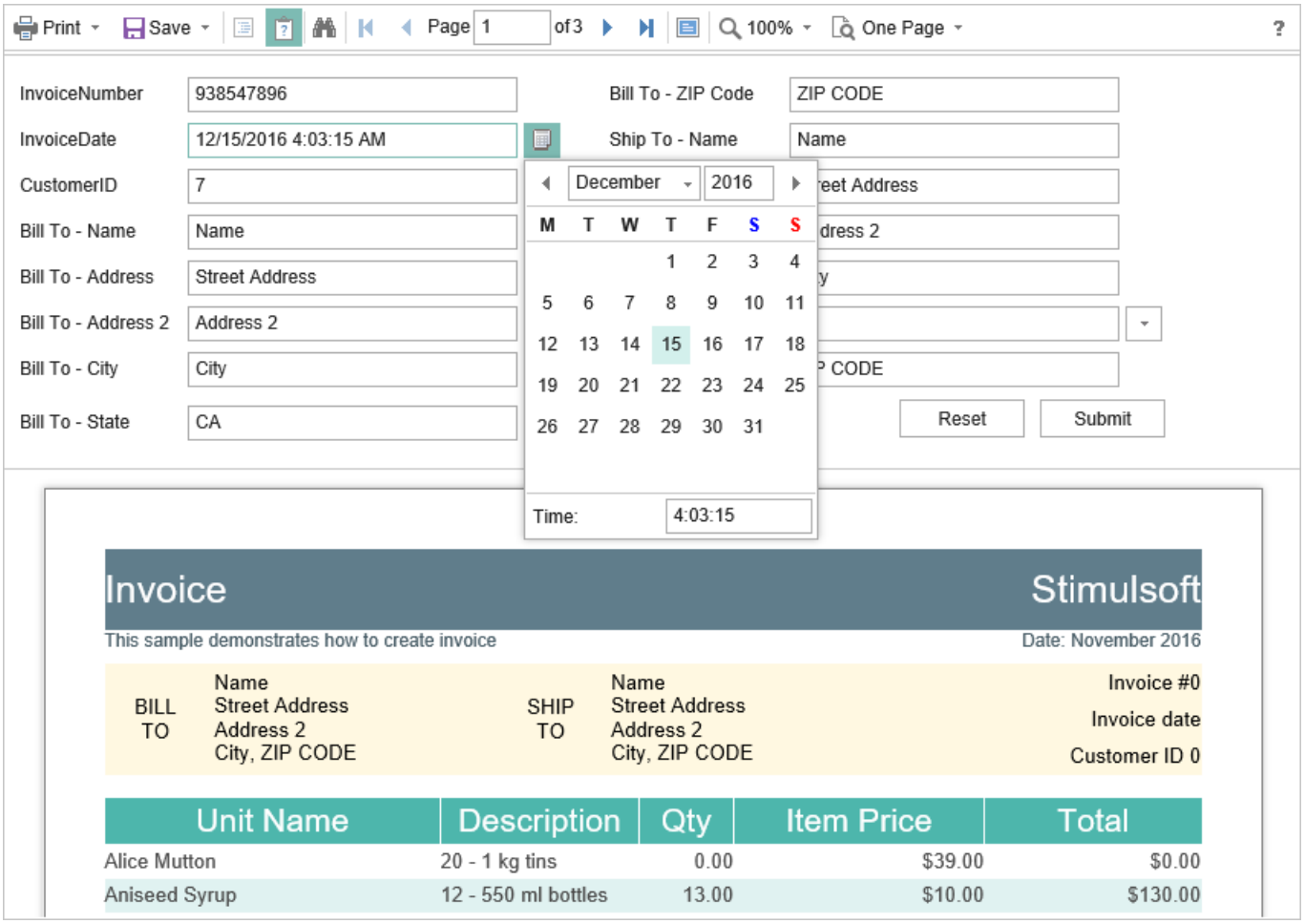
HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Appearance.InterfaceType = StiInterfaceType.Auto;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```



8.3.10 Work with Parameters

To work with report parameters in the **Angular Viewer**, there is a special settings panel. To add a parameter to the panel you need to define a variable in a report, requested by the user. When viewing a report in the viewer such a variable will be automatically added to the settings panel. It supports all types of report variables (normal variables, date and time, borders, lists, etc.).



To work with reports with parameters, no additional viewer settings are required. If you need to perform some actions before applying the parameters, you can define a special **Interaction** action.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Actions.Interaction = "ViewerInteraction";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}

public IActionResult ViewerInteraction()
{
   // Some code before any interaction
```

```
    // ...

    return StiAngularViewer.InteractionResult(this);
}
...
```

This action is called during any interactive actions of the viewer. If you need to perform any actions only when applying report parameters, you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class, they are passed to any server side on any request, and contain all necessary information and states of the client part of the viewer. To determine the type of the action of the viewer, it is enough to check the **Action** property of the viewer parameters.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
    StiRequestParams requestParams =
    StiAngularViewer.GetRequestParams(this);
    if (requestParams.Action == StiAction.Variables)
    {
        // Some code before apply parameters
    }

    return StiAngularViewer.InteractionResult(this);
}
...
```

If you do not need to work with parameters, you can completely disable this feature. To do this, use the **ShowParametersButton** property in the **Toolbar** section of properties, which should be set to **false**.

HomeController.cs

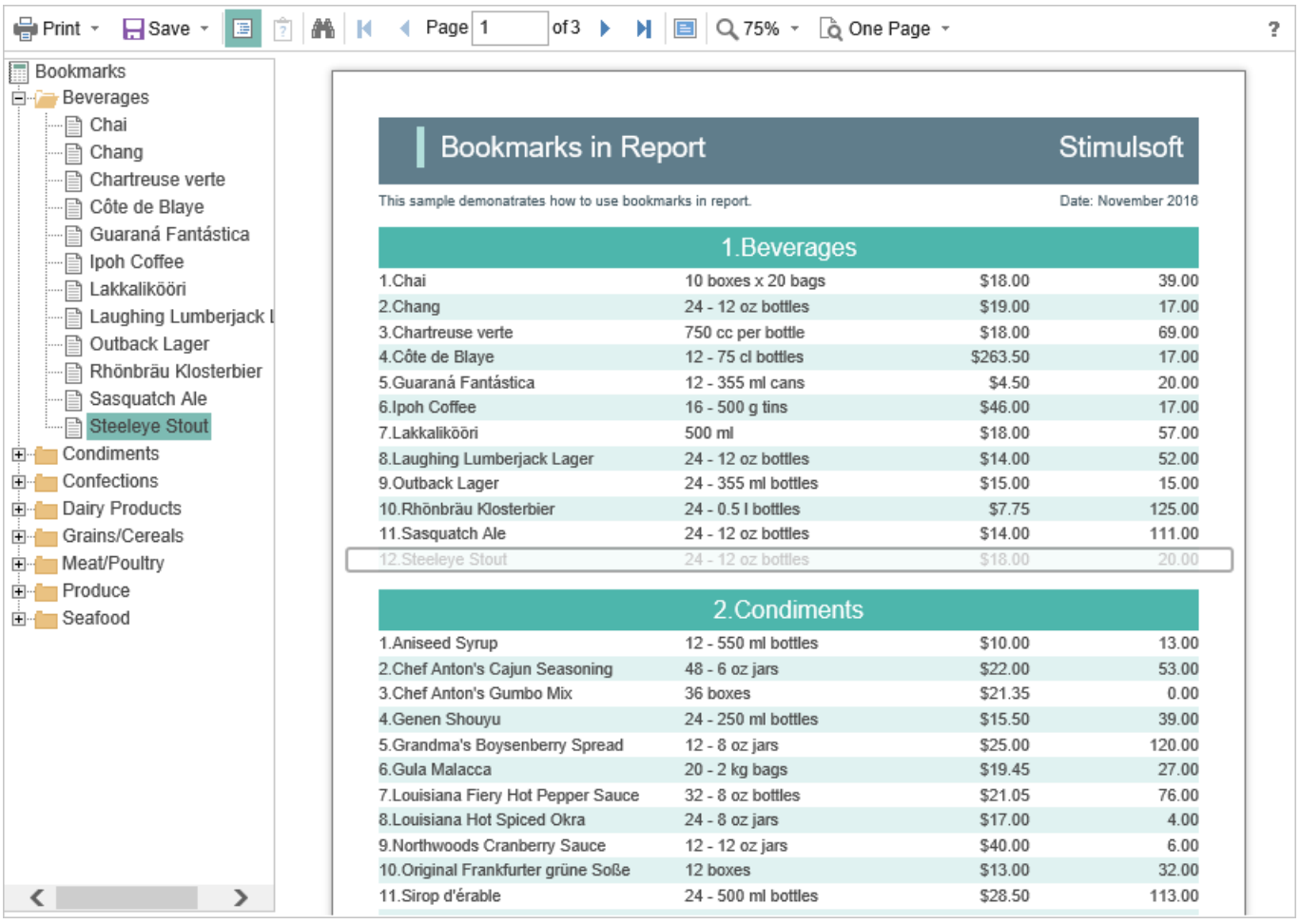
```
...
public IActionResult InitViewer()
{
    var requestParams = StiAngularViewer.GetRequestParams(this);
    var options = new StiAngularViewerOptions();
    options.Actions.ViewerEvent = "ViewerEvent";
    options.Toolbar.ShowParametersButton = false;

    return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

Information

With such a viewer configuration, the options panel will not be displayed, even if the parameters are present in the displayed report.

8.3.11 Work with Bookmarks

The **Angular Viewer** component supports report bookmarks. When displaying such a report on the left side of the page, a panel with bookmarks will be displayed. When you select a bookmark of the report, the viewer will carry out an automatic transition to the specified page, and the report item with a bookmark is highlighted.



By default, the bookmarks bar width is 180 pixels, the **Angular Viewer** component

allows you to change this value. For this, the **BookmarksTreeWidth** property, which value is specified in pixels, is used.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Appearance.BookmarksTreeWidth = 200;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

If work with report bookmarks is not required, you can disable this feature. For this, set the **ShowBookmarksButton** property to **false**.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Toolbar.ShowBookmarksButton = false;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

Information

In this case, report bookmarks will not be displayed, even if they are present in the displayed report. This property has no effect on printing and exporting reports.

When printing a report with bookmarks, the bookmark tree will be hidden. If you want to print bookmarks with the report, it is necessary to set the **BookmarksPrint** property to **true**.

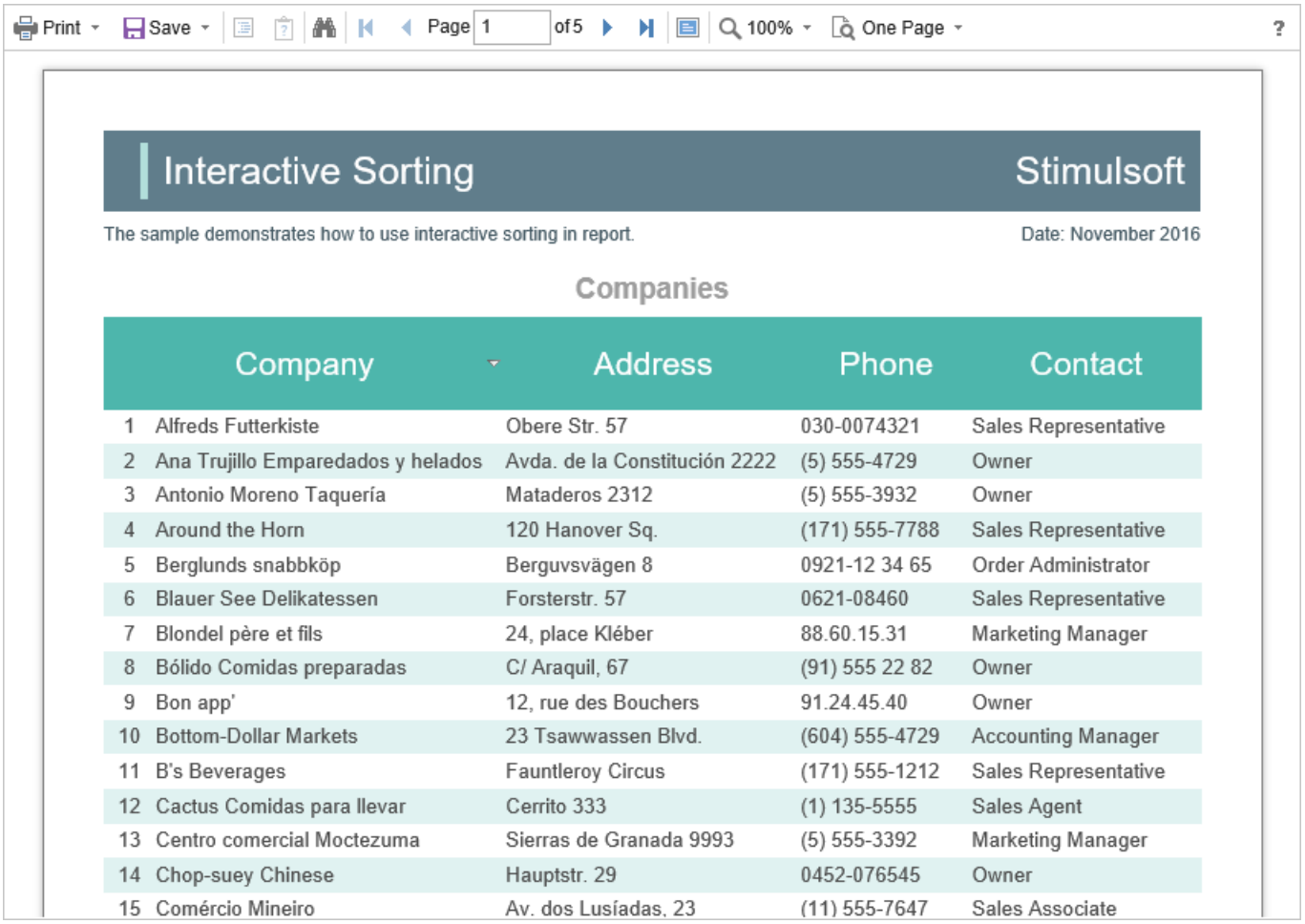
HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Appearance.BookmarksPrint = true;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

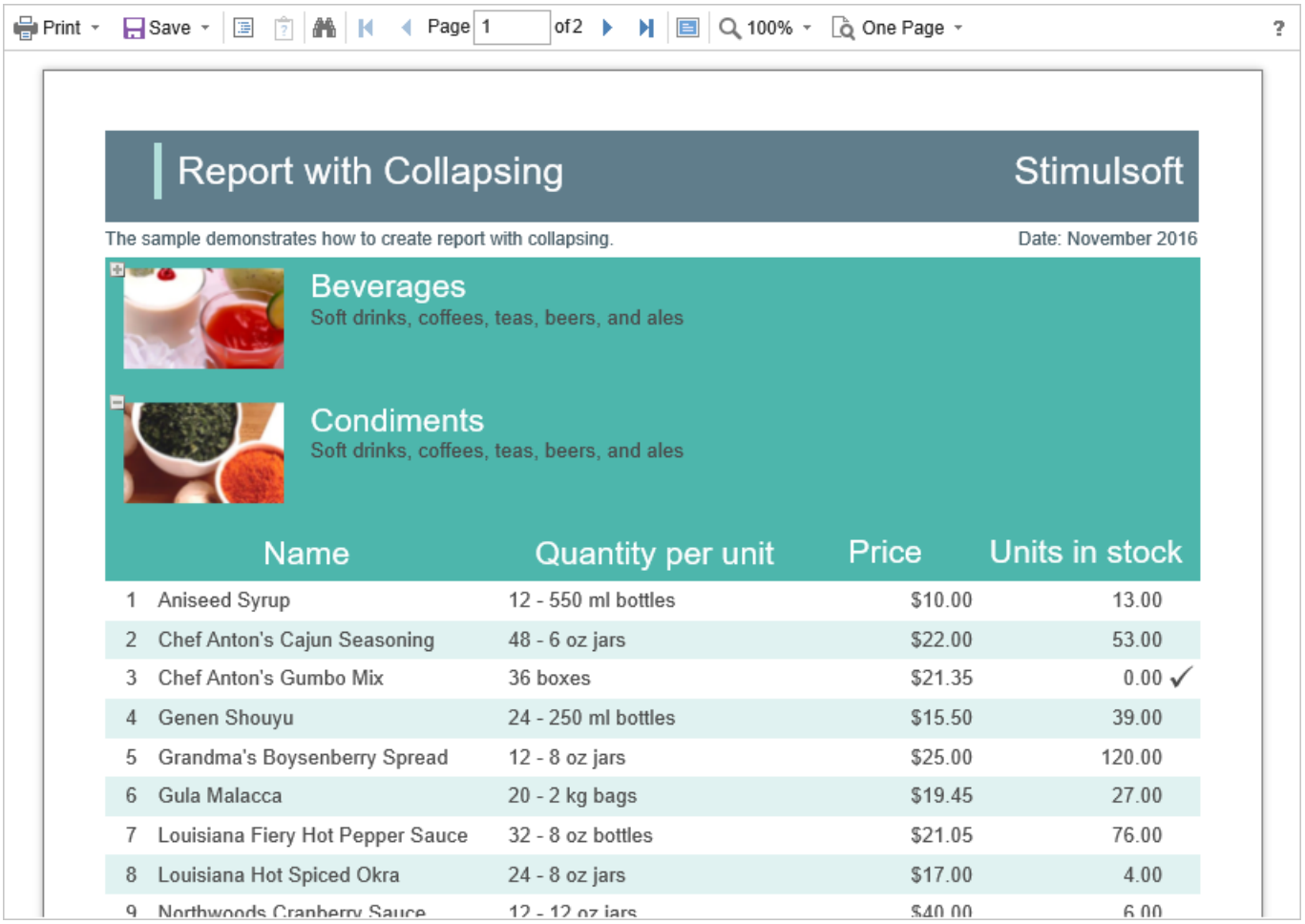
8.3.12 Dynamic Sorting, Collapsing, and Drill-Down

The **Angular Viewer** component supports dynamic sorting, collapsing, and drill-down of reports. Dynamic sorting provides the ability to change the direction of sorting in a rendered report. To do this, click on the component that has the dynamic sorting enabled. Dynamic sorting is carried out in the following directions - **Ascending** and **Descending**. Each time the component is clicked, the sorting direction is reversed.

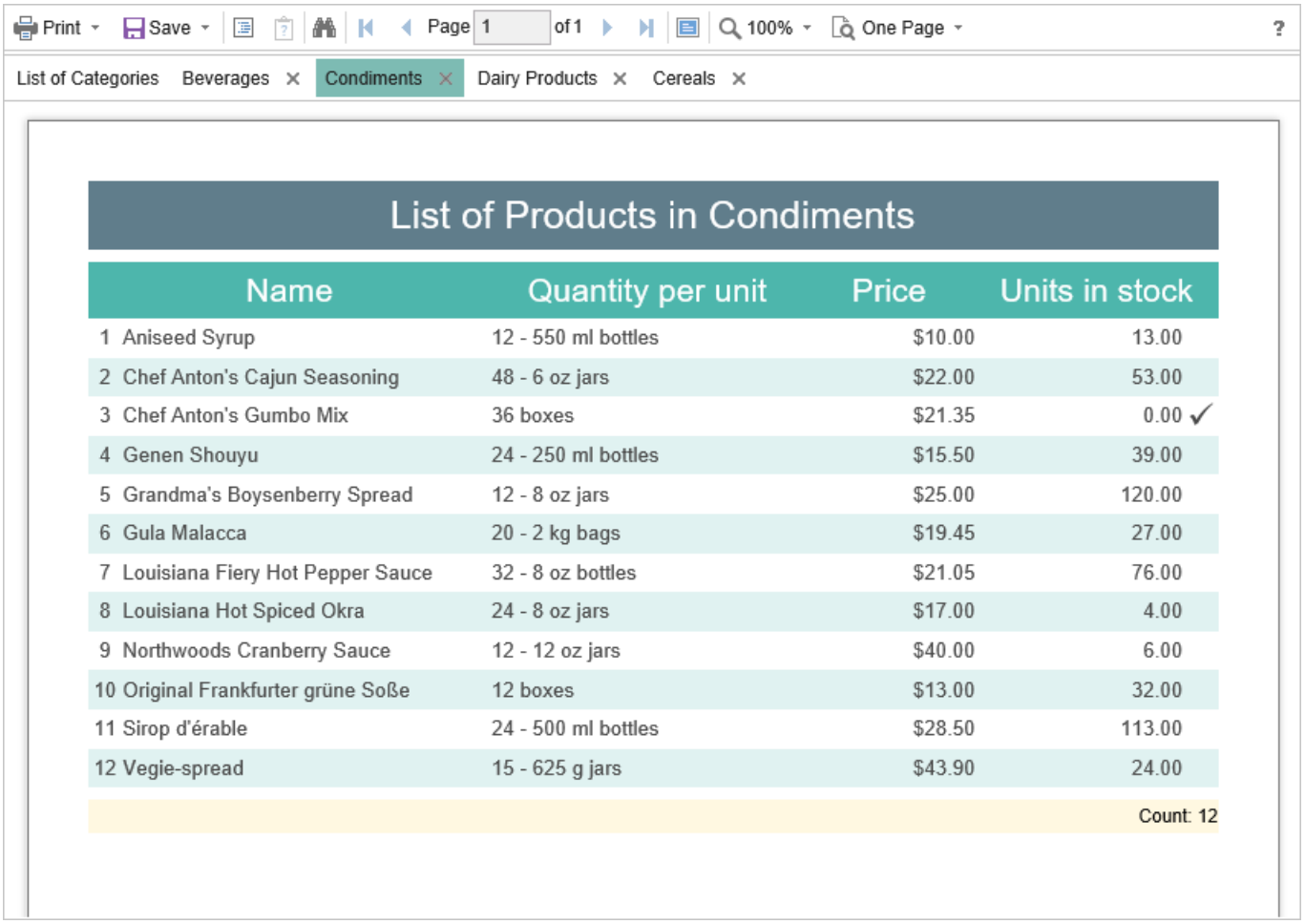
Multi-level sorting is allowed in the report. To do this, hold down the **Ctrl** key and sequentially click on the sorted components in the report. To reset sorting, you can click on any sorted component without holding down the **Ctrl** key.



A report with dynamic collapsing is an interactive report in which blocks can collapse/expand their content when you click on the block title. Report elements, which can be collapsed/expanded, are indicated by special icons - **[-]** or **[+]**.



When using drill-down, under the main panel of the viewer, the drill-down panel with tabs for drill-down reports will be displayed. The currently displayed report will be highlighted.



To work with dynamic sorting, collapsing and drill-down reports, no additional viewer settings are required. To perform any actions before sorting, collapsing or drill-down of the report, a special **Interaction** action is used. It will be called when interactive action of the viewer.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Actions.Interaction = "ViewerInteraction";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}

public IActionResult ViewerInteraction()
```

```
{
   // Some code before any interaction
   // ...

   return StiAngularViewer.InteractionResult(this);
}
...
```

To get the type of action, you can use the parameters of the viewer. The viewer parameters are represented as an object of the **StiRequestParams** class, they are passed to any server side by any request, and contain all necessary information and states of the client part of the viewer. For each type of interactivity, the viewer has a certain type of action:

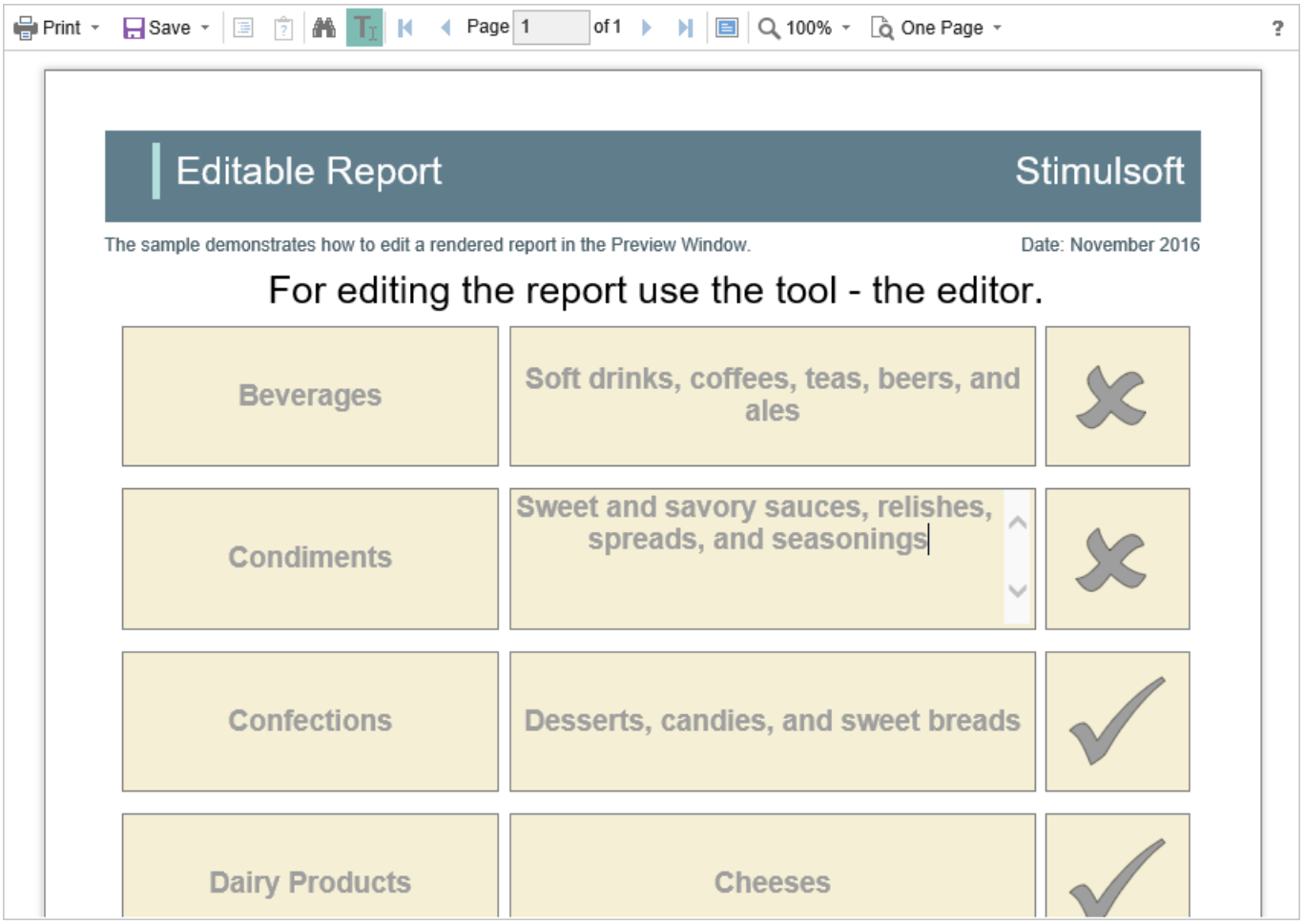
- **Sorting** – when using column sorting;
- **DrillDown** – when using drill-down in reports;
- **Collapsing** – when using collapsing report blocks.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   StiRequestParams requestParams =
   StiAngularViewer.GetRequestParams(this);
   switch (requestParams.Action)
   {
      case StiAction.Sorting:
        break;

      case StiAction.DrillDown:
        break;

      case StiAction.Collapsing:
        break;
   }

   return StiAngularViewer.InteractionResult(this);
}
...
```

8.3.13 Editing Report

The **Angular Viewer** component has the ability to edit report items, such as text boxes and check boxes. In order the editing be possible, in the report template, you should mark the required components as editable. After displaying a report in the viewer, you need to click the corresponding button on the viewer panel to start editing. After editing, it is necessary to click the button once more, and all changes will be applied to the report.



For the report edit mode, no special settings of the viewer required.

Information

The edited settings will be applied when you print or export a report, and the original report remains unchanged. After restarting the viewer, all the values will be returned to the initial ones.

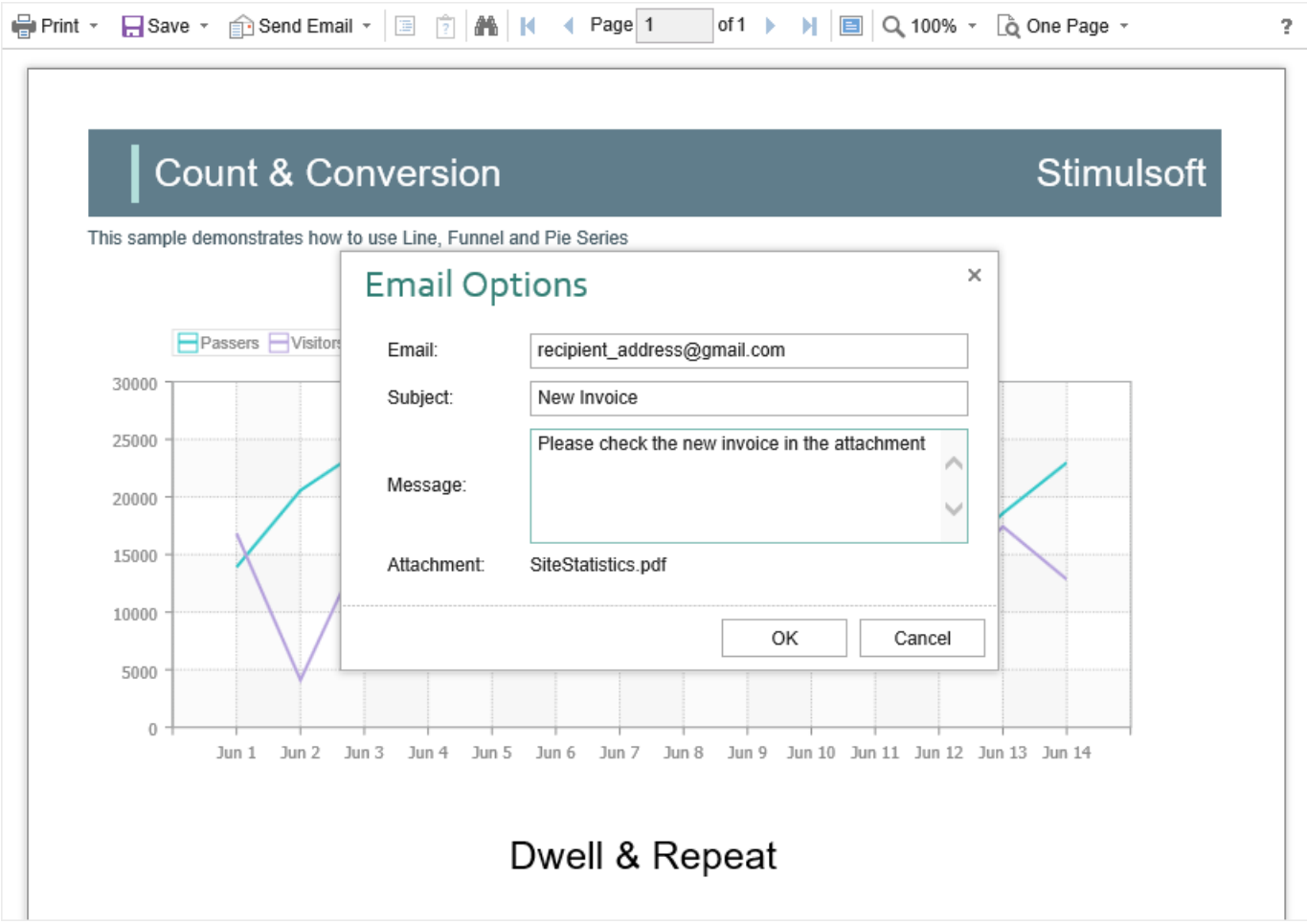
8.3.14 Sending Report by Email

The **Angular Viewer** component provides the ability to send reports by email. To activate this feature, you should set the **ShowSendEmailButton** property of the viewer to **true**, and define the **EmailReport** action.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Actions.EmailReport = "EmailReport";
   options.Toolbar.ShowSendEmailButton = true;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}

public IActionResult EmailReport()
{
   StiEmailOptions options = StiAngularViewer.GetEmailOptions(this);

   // Passed from the viewer, can be checked and changed
   // options.AddressTo = "";
   // options.Subject = "";
   // options.Body = "";

   // Should be filled here
   options.AddressFrom = "admin_address@test.com";
   options.Host = "smtp.test.com";
   options.Port = 465;
   options.UserName = "admin_address@test.com";
   options.Password = "admin_password";

   // options.CC.Add("email@test.com");
   // options.BCC.Add("email@test.com");
   // options.EnableSsl = true;

   return StiAngularViewer.EmailReportResult(this, options);
}
...
```

When sending a report by email, the menu to select the attachment format is displayed. It corresponds to the menu for selecting the format for exporting the report. After selecting the format, the dialog to enter send email parameters, such as the recipient's email, subject and text of the message is displayed.



After confirmation of sending the email the above described **EmailReport** event will be called. You can check and correct the data entered in this form. The exported report file will be attached to the email automatically.

The **Angular Viewer** component allows you to set default values for the send email form. The **DefaultEmailAddress**, **DefaultEmailSubject** and **DefaultEmailMessage** properties can be used for this. By default, these properties are empty.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Email.DefaultEmailAddress = "recipient_address@gmail.com";
   options.Email.DefaultEmailSubject = "New Invoice";
```

```
    options.Email.DefaultEmailMessage = "Please check the new invoice in the
    attachment";

    return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

8.3.15 Calling Designer from Viewer

The **Angular Viewer** component has the ability to call the report designer. The special **Design** button in the toolbar of the viewer (the button is disabled by default) should be used. To use this feature, you should set the **ShowDesignButton** property to **true**, and also to define the **DesignReport** event handler.

HomeController.cs

```
...
public IActionResult InitViewer()
{
    var requestParams = StiAngularViewer.GetRequestParams(this);
    var options = new StiAngularViewerOptions();
    options.Actions.ViewerEvent = "ViewerEvent";
    options.Actions.DesignReport = "DesignReport";
    options.Toolbar.ShowDesignButton = true;

    return StiAngularViewer.ViewerDataResult(requestParams, options);
}

public IActionResult DesignReport()
{
    StiReport report = StiAngularViewer.GetReportObject(this);
    ViewBag.ReportName = report.ReportName;

    return View("Designer");
}
...
```

Information

The viewer does not run the designer, it only calls the specified action, in which you can get all the necessary parameters. Then, in the action, you can implement redirection to another View, which contains the report designer.

8.3.16 Caching

The **Angular Viewer** component allows you to use the server cache to store rendered reports. If you do not use caching, then, every time you request a page, you should load the report, connect data, and render it again. If you use caching,

every time you refresh the page, the previously rendered report will be loaded from the cache.

When using caching, it should be taken into account that every report saved in the cache takes up server memory and, with a large number of requests to reports, this can become a critical issue. Therefore, you need to choose between two options - either low memory requirements but high in performance, or low performance requirements but high in memory.

To use caching, you need to connect modules to work with a session or cache on the server side. To do this, just add the following services to the project in the start file of a project.

Startup.cs

```
...
public void ConfigureServices(IServiceCollection services)
{
   services.AddMemoryCache();
   services.AddSession();
   services.AddMvc();
}
...
```

You can manage caching with the following properties.

The CacheMode property

This property of the viewer enabled caching and sets its type. It can take one of the following values, specified in the **StiServerCacheMode** enumeration:

- **None** – caching is disabled, each action of the viewer requires loading the report from the file and, if it is a report template, then render it;
- **ObjectCache** – for caching, the server cache is used. The report object is saved in this cach (set by default);
- **StringCache** – for caching, the server cache is used. The report is saved as a packed string in this cache;
- **ObjectSession** – the current session, in which the report object is saved, is used for caching;
- **StringSession** – for caching, the current session is used. The report is saved as a packed string in this cache.

The CacheItemPriority property

This property sets the priority of the report stored in the server's cache. It affects the automatic clearing of the server memory in case of lack of memory. The lower the priority is, the greater is the chance of removing information from memory.

The CacheTimeout property

This property specifies the amount of time in minutes for which you want to save the report in the server cache. If you use caching and the requested report is not found in the cache (the objects storage time has expired), then it will be requested again using a special **GetReport** event, then connect the report data and render it.

StiCacheHelper

The **Angular Viewer** component provides the ability to define your own methods of working with report caching. For this purpose, a special class **StiCacheHelper** is used. It contains methods for obtaining a report from the cache and saving the report to the cache. It is necessary to create a new class inherited from **StiCacheHelper** and reload the above methods, which respectively have the names - **GetReport** and **SaveReport**.

HomeController.cs

```
...
public class ViewerController : Controller
{
   public class StiMyCacheHelper : StiCacheHelper
   {
      public override StiReport GetReport(string guid)
      {
         string path =
         System.IO.Path.Combine(this.HttpContext.Server.MapPath("CacheFiles"
         ), guid);
         if (System.IO.File.Exists(path))
         {
            StiReport report = new StiReport();
            string packedReport = System.IO.File.ReadAllText(path);
            if (guid.EndsWith("template"))
            report.LoadPackedReportFromString(packedReport);
```

```
            else report.LoadPackedDocumentFromString(packedReport);

            return report;
        }
        return null;

        //return base.GetReport(guid);
    }

    public override void SaveReport(StiReport report, string guid)
    {
        string packedReport = guid.EndsWith("template") ?
        report.SavePackedReportToString() :
        report.SavePackedDocumentToString();
        string path =
        System.IO.Path.Combine(this.HttpContext.Server.MapPath("CacheFiles"
        ), guid);
        System.IO.File.WriteAllText(path, packedReport);

        //base.SaveReport(report, guid);
    }
  }

  static ViewerController()
  {
    StiAngularViewer.CacheHelper = new StiMyCacheHelper();
  }
}
...
```

To initialize the work with report caching using the created class, it is enough to set it as a value of the static **StiAngularViewer.CacheHelper** property in the controller constructor.

Information

If report caching is disabled (the **CacheMode** property of the viewer is set to **None**), the specified class will not be used.

8.3.17 Additional Methods

For **Angular Viewer**, there are several additional methods that are used to get the object of the currently viewed report, parameters of the current state of the viewer and other useful data. These methods can be used in any actions of the viewer.

The GetReportObject() method

Returns the report object with which the viewer is currently working. It is possible to perform the necessary actions with it - register new data sets, change report properties, assign parameters or load another report to the object. Then, the report can be returned to the viewer, specifying it as a parameter in the resulting action method.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   StiReport report = StiAngularViewer.GetReportObject(this);
   report.ReportName = "MyReportName";

   return StiAngularViewer.InteractionResult(this, report);
}
...
```

The GetRouteValues() method

Returns values for URLs with which the viewer page was opened. Thus, it is possible to get the initial collection of run page parameters in any viewer action and use these values for any checks and conditions.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   RouteValueDictionary routeValues =
   StiAngularViewer.GetRouteValues(this);

   return StiAngularViewer.InteractionResult(this);
}
...
```

You can also get values of URL parameters by parameter name, specifying it as the parameter of the called action of the viewer.

HomeController.cs

```
...
public IActionResult ViewerInteraction(string id)
{
   return StiAngularViewer.InteractionResult(this);
}
```

```
...
```

The GetFormValues() method

Returns the values of the form that initiated (opened by the POST request) a page of the viewer. Thus, it is possible to get a collection of form parameters in any action of the viewer.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   NameValueCollection formValues = StiAngularViewer.GetFormValues(this);

   return StiAngularViewer.InteractionResult(this);
}
...
```

By default, this feature is disabled in order to optimize requests of the client side of the viewer to the server. To enable it, set the **PassFormValues** property to **true**.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Server.PassFormValues = true;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

The GetRequestParams() method

Returns all parameters of the current state of the viewer passed to the server side. They can be useful for determining the type of action that the viewer is currently executing - for example, to determine the type of export, as well as all action parameters.

HomeController.cs

```
...
public IActionResult ExportReport()
{
   StiRequestParams requestParams =
   StiAngularViewer.GetRequestParams(this);
   if (requestParams.ExportFormat == StiExportFormat.Pdf)
   {
      StiReport report = StiAngularViewer.GetReportObject(this);

      // Some action with report for the PDF export
      // ...

      return StiAngularViewer.ExportReportResult(this, report);
   }

   return StiAngularViewer.ExportReportResult(this);
}
...
```

You can change the values of some parameters. After making changes, for correct operation of the viewer, you should transfer the changed parameter object to the input of thse resulting method.

HomeController.cs

```
...
public IActionResult ViewerInteraction()
{
   StiRequestParams requestParams =
   StiAngularViewer.GetRequestParams(this);
   if (requestParams.Action == StiAction.Variables)
   {
      requestParams.Interaction.Variables["Variable1"] = "MyValue";
      return StiAngularViewer.InteractionResult(this, requestParams);
   }

   return StiAngularViewer.InteractionResult(this);
}
...
```

The GetExportSettings() method

Returns all the parameters of the current report export. The type of the parameter object will correspond to the type of export selected in the viewer menu. Any export parameters can be changed and passed to the input of the resulting method. In this case, the report will be exported with the parameters transferred.

HomeController.cs

```
...
public IActionResult ExportReport()
{
   StiExportSettings settings = StiAngularViewer.GetExportSettings(this);
   if (settings.GetExportFormat() == StiExportFormat.Pdf)
   {
      StiPdfExportSettings pdfSettings = (StiPdfExportSettings)settings;
      pdfSettings.EmbeddedFonts = true;
      pdfSettings.AllowEditable = StiPdfAllowEditable.No;
      return StiAngularViewer.ExportReportResult(this, settings);
   }

   return StiAngularViewer.ExportReportResult(this);
}
...
```

The MapPath() and MapWebRootPath() methods

Returns the absolute path, respectively, to the application or wwwroot directory. You can use this to upload report templates files, data files, etc. These methods are located in the StiAngularHelper static class.

HomeController.cs

```
...
public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(StiAngularHelper.MapPath(this, "Reports/SimpleList.mrt"));

   return StiAngularViewer.GetReportResult(this, report);
}
...
```

8.3.18 Timeout

When working with the **StiNetCoreViewer** component, you can set the timeout for various operations — storing the report in the cache, server response, and query execution. The timeout setting is done using the component properties and report options.

CacheTimeout Property

Sets the time in minutes that the server will store the rendered report since the last action of the viewer. The default setting is 10 minutes.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Server.CacheTimeout = 10;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

Using cache will increase the speed of the report viewer. See the chapter Caching for more information

RequestTimeout Property

Sets the time to wait for a response from the server in seconds, after which an error will be generated. The default value is 30 seconds. For big reports, it is recommended to increase this value.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Server.RequestTimeout = 30;

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

CommandTimeout Option

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to set the query timeout for the already created connection, and data sources in the report.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Actions.GetReport = "GetReport";

   return StiAngularViewer.ViewerDataResult(requestParams, options);
}

public IActionResult GetReport()
{
   StiReport report = new StiReport();
   report.Load(Server.MapPath("Report.mrt"));
   ((StiSqlSource)
   report.Dictionary.DataSources["DataSourceName"]).CommandTimeout = 1000;

   return StiNetCoreViewer.GetReportResult(this, report);
}

public IActionResult ViewerEvent()
{
   return StiNetCoreViewer.ViewerEventResult(this);
}
...
```

8.3.19 Viewer Settings

The **Angular Viewer** is configured using properties that are located in the **StiNetCoreViewerOptions** class. All properties are divided into groups. Some of the groups contain subgroups for ease of use. The following is an example of setting the properties of the viewer.

HomeController.cs

```
...
public IActionResult InitViewer()
{
   var requestParams = StiAngularViewer.GetRequestParams(this);
   var options = new StiAngularViewerOptions();

   options.Theme = StiViewerTheme.Office2022WhiteTeal;
   options.Localization = "Localization/en.xml";
   options.Actions.GetReport = "GetReport";
   options.Actions.ViewerEvent = "ViewerEvent";
   options.Appearance.InterfaceType = StiInterfaceType.Auto;
   options.Appearance.ScrollbarsMode = true;
   options.Appearance.ShowTooltips = false;
   options.Exports.DefaultSettings.ExportToPdf.CreatorString = "Company
   Name";
```

```
    options.Exports.DefaultSettings.ExportToPdf.ImageQuality = 0.75f;
    options.Exports.ShowExportToDbf = false;
    options.Exports.ShowExportToDif = false;

    return StiAngularViewer.ViewerDataResult(requestParams, options);
}
...
```

Main settings (without groups)

Name	Description
Theme	Sets the viewer theme. The list of available themes can be found in the **StiViewerTheme** enumeration. The default value is **Office2022WhiteBlue**.
Localization	Sets the path to the XML localization file. The path can be absolute or relative. By default, English localization is used. It is built into the viewer and does not require additional XML files.
Width	Sets the width of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. The default width is 100%.
Height	Sets the height of the component in the required units that are specified in the **Unit** class. The value can be set in pixels - **Unit.Pixel()**, points - **Unit.Point()** and per cent - **Unit.Percentage()**. By default, the automatic height is set depending on the size of the report page, or 650 pixels in the view mode of the viewer with scrollbars.

Actions

Name	Description

GetReport	Specifies the name of the action method for preparing the rendered report. Specifies the name of the action method for preparing the constructed report. If report caching is enabled, this action will be called only once when the report is requested or if the requested report is not found in the server cache.
PrintReport	Specifies the name of the action method of report printing.
ExportReport	Specifies the name of the action method of the export the report to the specified format.
EmailReport	Specifies the name of the action method of sending the report by email.
Interaction	Specifies the name of the action method for the viewer to work with interactive operations, such as using parameters, dynamic sorting, collapsing and drill-down.
DesignReport	Specifies the name of the action method to go to the specified view by clicking the Design button on the viewer panel.
ViewerEvent	Specifies the name of the action method of basic viewer events and the processing actions of the viewer, such as printing and exporting a report, working with parameters, and interactivity, if these actions are not specified separately. In addition, this action is used to load scripts and styles of the viewer. This action is mandatory.

Server

Name	Description
Controller	Specifies the name of the report controller for the report viewer. If this property is not specified, then the current controller will be

	used to process requests.
RouteTemplate	Sets the route template that is returned when the report viewer actions are executed. If the property is not set, then the MVC project template will be used instead. If the `UseRelativeUrls` property is set to `true`, the `BasePath` will not be respected for this property.The default value of this property is null.
RequestTimeout	Sets the response timeout from the server in seconds, after which an error will be generated. The default value is 20 seconds. For big reports, it is recommended to increase this value.
CacheTimeout	Sets the time in minutes that the server will store the report since the last action of the viewer. The default value is 20 minutes.
CacheMode	Sets the report caching mode. It can take one of the following values of the **StiServerCacheMode** enumeration: › **None** – caching is disabled, the report will be reloaded each time using the **GetReport** event; › **ObjectCache** – the cache is used as the storage, the report is stored as an object (default value); › **ObjectSession** – the session is used as the storage, the report is stored as an object; › **StringCache** – the server cache is used as the storage, the report is serialized to a packed string; › **StringSession** – the session is used as a storage, the report is serialized into a packed string.

CacheItemPriority	Sets the priority of the report stored in the server cache. This property affects the automatic clearing of the server memory in case of lack of memory. The lower the priority is, the greater is the chance of removing information from memory.
AllowAutoUpdateCache	Sets the mode for automatic cache update. The report stored in the cache or the server session will be automatically re-saved after a certain period of time when the viewer is idle (every 3 minutes). By default, the property is set to **true**.
UseRelativeUrls	Sets the viewer mode in which relative URLs are used for AJAX requests to the server. By default, the property is set to **true**.
PortNumber	Gets or sets a value which specifies the port number to use in the URL. A value of **0** defines automatic detection (default value). A value of **-1** removes the port number.
PassQueryParametersForResources	Enables transferring all request URL parameters when generating links to the resources of the viewer. If **false**, only the necessary parameters are used to request the resources of the viewer. This corresponds to the more correct work of the browser cache. By default, the property is set to **true**.
PassQueryParametersToReport	Enables using all the URL parameters of the request as the variable values. The variables names must match the parameters. The default value of the property is false.
PassFormValues	Enables passing the values of the POST form to the client side, if these values are required to be used in the actions of the viewer. If you enable this property, the additional **GetFormValues()** method will return a collection of form parameters. By default, the property is **false**.
ShowServerErrorPage	Enables displaying an HTML page with the details of the error that occurred on the server

	side. When the property is enabled, the details of the error will be displayed in the viewer window. If the property is disabled, only the numeric error code and a short error text in the dialog box will be displayed. By default, the property is set to **true**.
UseCompression	Enables compression of the viewer requests into the GZip stream. That allows to decrease the amount of the internet-traffic but slows down slightly the viewer. The default value of the property is false.
UseCacheForResources	Enables caching of the component resources on the server side. The following resources are supported: scripts, styles and images. This option improves the load speed of the component and also reduces the server load in multi-client environments. The default value is **true**.
UseLocalizedCache	Sets a value which enables the use of a different cache depending on the selected localization. The default value of the property is **false**.
AllowLoadingCustomFontsToClientSide	Allows you to pass custom fonts to the client side and convert them to CSS style for the correct display of text as HTML with a specified font. By default, the property is set to **false**.

Appearance

Name	Description
CustomCss	Sets the path to the CSS file of the viewer's styles. The standard styles of the chosen theme will not be loaded if this property has got a value. The default value of the property is an empty string.
BackgroundColor	Sets the background color of the viewer. By default it is set to **White**.

PageBorderColor	Sets the border color of the viewer. By default it is set to **Gray**.
RightToLeft	Sets the **Right to Left** mode for viewer controls. By default the property is set to **false**. By default, the property is set to **false**.
FullScreenMode	Sets the full-screen display mode of the viewer. By default, the property is set to **false**.
ScrollbarsMode	Sets the preview mode with scrollbars. By default, the property is set to **false**.
OpenLinksWindow	Sets the target window for opening links contained in the report. By default, the property is set to **Blank** (new window).
OpenExportedReportWindow	Sets the target window for opening the export file from the viewer. By default, the property is set to **Blank** (new window).
DesignWindow	Sets the destination window for launching the report designer. The default value of the property is **Self** (whish is the current window).
ShowTooltips	Enables showing tips for the viewer controls when the mouse hovers over. By default, the property is set to **true**.
ShowTooltipsHelp	Enables showing links to online documentation for the viewer controls. By default, the property is set to **true**.
ShowDialogsHelp	Sets a value which indicates that show or hide the help button in dialogs. By default, the property is set to **true**.
PageAlignment	Sets the position of the report page in the viewer window. It can take one of the following values of the **StiContentAlignment** enumeration: › **Left** – the page will be aligned left; › **Center** – the page will be centered (default value);

	› **Right** – the page will be aligned right.
ShowPageShadow	Enables displaying shadow for report pages. By default the property is set to **true**.
BookmarksPrint	Enables printing of report bookmarks (besides the report itself). By default the property is set to **false**.
BookmarksTreeWidth	Sets the width of the bookmarks panel in pixels. By default, the width is 180 pixels.
ParametersPanelPosition	Specifies the position of the report parameters panel. It can take one of the following **StiParametersPanelPosition** enumeration values: › **Top** - the panel will be docked to the top margin (default value); › **Left** - the panel will be docked to the left margin.
ParametersPanelMaxHeight	Sets the maximum height of the parameters bar in pixels. By default, the maximum height is 300 pixels.
ParametersPanelColumnsCount	Sets the number of columns to display report parameters. By default, there are 2 columns.
ParametersPanelSortDataItems	Gets or sets a value which indicates that variable items will be sorted. By default the property is set to **true**.
ParametersPanelDateFormat	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the date and time format set by the browser is used.
InterfaceType	Sets the type of interface used for the viewer. It can take one of the following **StiInterfaceType** enumeration values: › **Auto** – the viewer's interface is determined automatically depending of the device that is report is displayed on. That is the default value.

	› **Mouse** – the standard interface with a mouse control will be used for all the screen types. › **Touch** – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability. › **Mobile** - the Mobile interface will be used to control the viewer for all the screen types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use with the smartphones.
AllowMobileMode	Enables or disables displaying a report or dashboard in the mobile mode. If the option is set to **false**, then the mobile view will not be used. If the option is set to **true**, the mobile view mode will be used when opening the viewer on mobile devices. By default, the option is set to **true**.
ChartRenderType	Sets the displaying mode of charts on the report page. It can take one of the following **StiChartRenderType** enumeration values: › **Image** – charts are displayed as static images; › **Vector** – charts are displayed in the vector mode as an SVG object; › **AnimatedVector** - charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (default value).
ReportDisplayMode	Sets the export mode for displaying report pages. It can take one of the following values of the **StiReportDisplayMode** enumeration: › **FromReport** - the export mode of the report elements is defined from report template

	settings - Div or Table; › **Table** – report elements are exported using HTML tables (default value); › **Div** – report elements are exported using DIV markup; › **Span** - report items are exported using SPAN markup.
DatePickerFirstDayOfWeek	Sets the first day of the week for the date picker. It can take one of the following values of the **StiFirstDayOfWeek** enumeration: › **Monday** – the first day of the week is Monday (default value); › **Sunday** – the first day of the week is Sunday.
DatePickerIncludeCurrentDayForRanges	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default the property is set to **false**.
AllowTouchZoom	Sets ability to change the scale of the report page by using the two-fingers gesture (Pinch to Zoom) for the touch-screens. The default value of the property is **true**.
ShowReportIsNotSpecifiedMessage	Sets a value which indicates that 'The report is not specified' message will be shown. The default value of the property is **true**.
PrintToPdfMode	Sets the Print to PDF mode. It has the following values: › **StiPrintToPdfMode.Hidden** - hidden print mode (default value); › **StiPrintToPdfMode.Popup** - the PDF document will be displayed before printing in a pop-up window.
ImagesQuality	Gets or sets the image quality that will be used on the viewer page. It has the following values: › **StiImagesQuality.Low** - low quality, used to speed up loading reports and saves memory; › **StiImagesQuality.Normal** - normal quality, suitable for most cases (default value);

	› **StiImagesQuality.High** - high quality, used for ultra high-definition displays, but may slow down the loading of pages.
CombineReportPages	Sets a value which indicates that if a report contains several pages, then they will be combined in preview. By default the property is set to **false**.

Toolbar

Name	Description
Visible	Enables displaying the viewer toolbar. By default, the property is set to **true**.
DisplayMode	Specifies the display mode of the toolbar of the viewer. It can take one of the following values of the **StiToolbarDisplayMode** enumeration: › **Simple** - all controls are located on the same control panel (default value); › **Separated** - the control panel is split into top and bottom panels.
BackgroundColor	Specifies the background color of the viewer toolbar. The default color of the selected theme is used.
BorderColor	Specifies the border color of the viewer toolbar. The default color of the selected theme is used.
FontColor	Specifies the text color for the toolbar and the viewer menu. The default color of the selected theme is used.
FontFamily	Specifies the font for the toolbar and the viewer menu. The default font of the selected theme is used.
Alignment	Sets the alignment mode for the controls on the viewer toolbar. It can take one of the following values of the **StiContentAlignment**

	enumeration: › **Left** – elements will be aligned left; › **Center** – elements will be centered; › **Right** – elements will be aligned right; › **Default** – the alignment depends on the RightToLeft property (default value).
ShowButtonCaptions	Enables text of the buttons on the toolbar of the viewer. By default the property is set to **true**.
ShowPrintButton	Enables showing the button - **Print** - on the viewer toolbar. By default, the property is set to **true**.
ShowOpenButton	Enables displaying the **Open** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to **false**.
ShowSaveButton	Enables displaying the **Save** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to true.
ShowSendEmailButton	Enables showing the button - **Send Email** - on the viewer toolbar. By default, the property is set to **false**. Also, you should add the EmailReport action.
ShowFindButton	Enables showing the button - **Find** - on the viewer toolbar. By default, the property is set to **true**.
ShowBookmarksButton	Enables showing the button - **Bookmarks** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the bookmarks panel will not be displayed even if there are bookmarks in the report.
ShowParametersButton	Enables showing the button - **Parameters** - on the viewer toolbar. By default, the property is set to **true**. If the button is hidden, the parameters panel will not be displayed even if there are

	parameters in the report.
ShowResourcesButton	Enables showing the button - **Resources** - on the toolbar of the viewer. By default, the property is set to **true**. If the button is hidden, the resources panel will not be displayed even if there are resources in the report.
ShowEditorButton	Enables showing the button - **Editor** - on the viewer toolbar. By default, the property is set to **true**.
ShowFullScreenButton	Enables displaying the **Full Screen** button on the toolbar of the viewer when viewing reports or dashboards. . By default, the property is set to **true**.
ShowFirstPageButton	Enables showing the button - **First Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowPreviousPageButton	Enables showing the button - **Previous Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowCurrentPageControl	Enables showing the current report page indicator. By default, the property is set to **true**.
ShowNextPageButton	Enables showing the button - **Next Page** - on the viewer toolbar. By default, the property is set to **true**.
ShowLastPageButton	Enables showing the button - **Last Page** - on the toolbar of the viewer. By default, the property is set to **true**.
ShowZoomButton	Enables showing the button to select the report zoom. By default, the property is set to **true**.
ShowViewModeButton	Enables showing the button to select the view mode of the report page. By default, the property is set to **true**.
ShowDesignButton	Enables displaying the **Design** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to

	false.
ShowAboutButton	Enables showing the button - **About** - on the viewer toolbar. By default, the property is set to **true**.
ShowRefreshButton	Sets a visibility of the **Refresh** button in the toolbar of the viewer. By default, the property is set to **true**.
ShowPinToolbarButton	Enables displaying of the **Pin Toolbar** button on the viewer's toolbar. The button is available only in the Mobile mode of the viewer's interface. The default value of the property is **true**.
PrintDestination	Sets the report printing mode. It can take one of the following values of the **StiPrintDestination** enumeration: › **Default** – a menu with a choice of printing modes will be displayed (default value); › **Pdf** – printing will be done in the PDF format; › **Direct** – printing will be done to the HTML format directly to the printer, the system print dialog will be displayed; › **PopupWindow** – printing will be done in the HTML format via the preview window of the report.
ViewMode	Sets the mode for displaying report pages. It can take one of the following **StiWebViewMode** enumeration values: › **SinglePage** - displays one page of the report selected in the toolbar of the viewer (default value); › **Continuous** - displays all pages of the report; › **MultiplePages** - displays all report pages as a table.
Zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. The values are

	from 10 to 500 percent. You can also set one of the following values: › **StiZoomMode.PageWidth** – when the viewer runs, the zoom, necessary to display the report by the page width, will be set; › **StiZoomMode.PageHeight** – when the viewer runs, the zoom, necessary to display the report by the page height, will be set.
MenuAnimation	Enables animation when the viewer menu shows/hides. By default the property is set to **true**.
ShowMenuMode	Sets the display mode of the viewer menu. It can take one of the following values of the **StiShowMenuMode** enumeration: › **Click** – shows menu by mouse click (default value); › **Hover** – shows menu by hovering the mouse cursor.
AutoHide	Enables auto-hiding of the viewer's toolbar. The property will work only for the Mobile mode of the viewer's interface. The default value of the property is **true**.

Export

Name	Description
DefaultSettings	This group of properties provides the ability to specify the default export settings for each export type. These settings will be applied to the export dialogs when the viewer runs or to the report, if export dialogs are disabled.
StoreExportSettings	Enables saving selected settings in the export dialogs. Settings will be stored in browser cookies. By default the property is set to **true**.

ShowExportDialog	Enables showing the export options dialog box. If the property is set to **false**, the export will be done with the default settings. By default the property is set to **true**.
ShowExportToDocument	Enables the export menu item - **Document File**. By default, the property is set to **true**.
ShowExportToPdf	Enables displaying the **Adobe PDF file** export menu item when viewing reports, and the **Adobe PDF** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToXps	Enables the export menu item - **Microsoft XPS File**. By default, the property is set to **false**.
ShowExportToPowerPoint	Enables the export menu item - **Microsoft PowerPoint 2007/2010 File**. By default, the property is set to **true**.
ShowExportToHtml	Enables the export menu item - **HTML File**. By default, the property is set to **true**.
ShowExportToHtml5	Enables the export menu item - **HTML5 File**. By default, the property is set to **true**.
ShowExportToMht	Enables the export menu item - **MHT Web Archive**. By default, the property is set to **true**.
ShowExportToText	Enables the export menu item - **Text File**. By default, the property is set to **true**.
ShowExportToRtf	Enables the export menu item - **Rich Text File**. By default, the property is set to **true**.
ShowExportToWord2007	Enables the export menu item - **Microsoft Word 2007/2010 File**. By default, the property is set to **true**.
ShowExportToOpenDocumentWriter	Enables the export menu item - **OpenDocument Writer File**. By default, the property is set to **true**.
ShowExportToExcel	Enables the export menu item - **Microsoft Excel File**. By default, the property is set to **true**.
ShowExportToExcelXml	Enables the export menu item - **Microsoft**

	Excel Xml File. By default, the property is set to **true**.
ShowExportToExcel2007	Enables displaying the **Microsoft Excel 2007/2010 File** export menu item when viewing reports, and the **Microsoft Excel** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToOpenDocumentCalc	Enables the export menu item - **OpenDocument Calc File**. By default, the property is set to **true**.
ShowExportToCsv	Enables the export menu item - **CSV File**. By default, the property is set to **true**.
ShowExportToDbf	Enables the export menu item - **DBF File**. By default, the property is set to **true**.
ShowExportToXml	Enables the export menu item - **XML File**. By default, the property is set to **true**.
ShowExportToDif	Enables the export menu item - **Data Interchange Format (DIF) File**. By default, the property is set to **true**.
ShowExportToSylk	Enables the export menu item - **Symbolic Link (SYLK) File**. By default, the property is set to **true**.
ShowExportToJson	Enables the export menu item - **JSON File**. By default, the property is set to **true**.
ShowExportToImageBmp	Enables displaying the **BMP Image** export menu item when viewing reports, and the **BMP Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageGif	Enables displaying the **GIF Image** export menu item when viewing reports, and the **GIF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageJpeg	Enables displaying the **JPEG Image** export menu item when viewing reports, and the **JPEG Image** item when viewing dashboards. By

	default, the property is set to **true**.
ShowExportToImagePcx	Enables displaying the **PCX Image** export menu item when viewing reports, and the **PCX Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImagePng	Enables displaying the **PNG Image** export menu item when viewing reports, and the **PNG Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageTiff	Enables displaying the **TIFF Image** export menu item when viewing reports, and the **TIFF Image** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvg	Enables displaying the **Scalable Vector Graphics (SVG) File** export menu item when viewing reports, and the **Scalable Vector Graphics (SVG) File** item when viewing dashboards. By default, the property is set to **true**.
ShowExportToImageSvgz	Enables displaying the **Compressed SVG (SVGZ) File** export menu item when viewing reports, and the **Compressed SVG (SVGZ) File** item when viewing dashboards. By default, the property is set to **true**.
ShowOpenAfterExport	Enables displaying the **Open After Export** parameter in export settings menu. By default the property is set to **true**.

Email

Name	Description
ShowEmailDialog	Enables displaying settings for sending the report via email. If the dialog box is disabled, the email will be sent with the settings set on the server side in the **EmailReport** action. By

	default the property is set to **true**.
ShowExportDialog	Enables displaying export options dialog box when sending email. If the property is set to **false**, the export will be done with the default settings. By default the property is set to **true**.
DefaultEmailAddress	Sets the default recipient email, i.e. the address to which the email with the attached report will be sent.
DefaultEmailSubject	Sets the default email subject (header).
DefaultEmailMessage	Sets the default email message (text).

8.3.20 API References

You can using Angular Viewer API. **StiAngularViewer** contains api object that allow to manipulate viewer or view states.

app.component.ts

```
...
export class AppComponent {
   @ViewChild('viewer') viewer: StimulsoftViewerComponent;
   ...
}
...
```

Enhance app.component.html, add reference to component, add display current page & zoom, add buttons that allow to zoom page to 50% & export report to PDF format with setting ImageQuality to 200:

app.component.html

```
...
Zoom is {{ viewer.api.zoom }}<br />
Current page {{ viewer.api.currentPage + 1 }}<br />
<input type="button" (click)="viewer.api.zoom = 50" value="Zomm to 50%" />
<input
   type="button"
   (click)="viewer.api.export('Pdf', { ImageResolution: 200 })"
   value="Export to PDF"
/>

<stimulsoft-viewer-angular
   #viewer
   [requestUrl]="'http://localhost:60801/Viewer/{action}'"
   [action]="'InitViewer'"
```

```
  [height]="'600px'"
></stimulsoft-viewer-angular>
...
```

Options

Angular **stimulsoft-viewer-angular** contains options.

Parameter	Description
requestUrl	Url to server instance, must contains placeholder {action} that will replace with action. Example: http://server.url:51528/Viewer/{action}
action	Controller action that handle viewer initial request.
properties	Properties that will transfer to controller action as JSON object.
width	Viewer width.
height	Viewer height.
backgroundColor	Viewer background color.
style	Style of viewer applied to main span as [style] ="style".

Events

Angular **stimulsoft-viewer-angular** contains events.

Parameter	Description
loaded	Occurs when report loaded.
error	Occurs on error, $event is ErrorMessage object contains error: string & type: any (if present).
export	Occurs on export, $event object contains exportFormat: string & exportSettings: {}.
email	Occurs on export & email, $event object contains exportFormat: string & exportSettings: {}.

print	Occurs on export & email, $event object contains format: string : 'PrintPdf' or 'PrintWithoutPreview' or 'PrintWithPreview'.

Methods

With API property of **stimulsoft-viewer-angular** (**StiAngularViewer**) you can perform different actions.

Parameter	Description
currentPage	Get or set the current page number.
pageCount	The total pages count.
viewMode	Get or set the view mode, can be 'SinglePage', 'Continuous' & 'MultiplePages'.
zoom	Get or set the page zoom in percent. From 1 to 1000.
zoomPageHeight()	Zoom page in height.
zoomPageWidth()	Zoom page in width.
printPdf()	Print current report to PDF.
printWithoutPreview()	Print current report without preview.
printWithPreview()	Print current report/dashboard with preview.
showExportForm(format: string)	Show export form. **format** The format to export, can be 'Document', 'Pdf', 'Xps', 'Ppt2007', 'Html', 'Html5', 'Mht', 'Text', 'Rtf', 'Word2007', 'Odt', 'Excel', 'ExcelBinary', 'ExcelXml', 'Excel2007', 'Ods', 'Csv', 'Dbf', 'Dif', 'Sylk', 'Json', 'Xml', 'ImageBmp', 'ImageGif', 'ImageJpeg', 'ImagePcx', 'ImagePng', 'ImageTiff', 'ImageEmf', 'ImageSvg', 'ImageSvgz'
showExportEmailForm(format: string)	Show export form & email. **format** The format to export, can be 'Document', 'Pdf',

	'Xps', 'Ppt2007', 'Html', 'Html5', 'Mht', 'Text', 'Rtf', 'Word2007', 'Odt', 'Excel', 'ExcelBinary', 'ExcelXml', 'Excel2007', 'Ods', 'Csv', 'Dbf', 'Dif', 'Sylk', 'Json', 'Xml', 'ImageBmp', 'ImageGif', 'ImageJpeg', 'ImagePcx', 'ImagePng', 'ImageTiff', 'ImageEmf', 'ImageSvg', 'ImageSvgz'
export(format: string, settings?: any)	Export report to selected format. Use default settings if not specified. **format** The format to export, can be 'Document', 'Pdf', 'Xps', 'Ppt2007', 'Html', 'Html5', 'Mht', 'Text', 'Rtf', 'Word2007', 'Odt', 'Excel', 'ExcelBinary', 'ExcelXml', 'Excel2007', 'Ods', 'Csv', 'Dbf', 'Dif', 'Sylk', 'Json', 'Xml', 'ImageBmp', 'ImageGif', 'ImageJpeg', 'ImagePcx', 'ImagePng', 'ImageTiff', 'ImageEmf', 'ImageSvg', 'ImageSvgz' **settings** The export settings
exportEmail(format: string, settings?: any, email?: string, subject?: string, message?: string)	Export report to selected format. Use default settings if not specified. Use default email settings if not specified. **format** The format to export, can be 'Document', 'Pdf', 'Xps', 'Ppt2007', 'Html', 'Html5', 'Mht', 'Text', 'Rtf', 'Word2007', 'Odt', 'Excel', 'ExcelBinary', 'ExcelXml', 'Excel2007', 'Ods', 'Csv', 'Dbf', 'Dif', 'Sylk', 'Json', 'Xml', 'ImageBmp', 'ImageGif', 'ImageJpeg', 'ImagePcx', 'ImagePng', 'ImageTiff', 'ImageEmf', 'ImageSvg', 'ImageSvgz' **settings** The export settings **email** The email

	message The email message **subject** The email subject

9 Reports and Dashboards for JS

JavaScript is a prototype-oriented scripting programming language. Reports.JS provides tools for designing, viewing, converting reports and dashboards for JavaScript.

9.1 Quick Start

In this section, you will find step-by-step guides for quickly deploying Stimulsoft components in JavaScript applications.

- Vanilla JavaScript;
- Angular JS;
- React JS;
- Vue JS;
- Node JS.

9.1.1 Vanilla JavaScript

This part describes an example of quickly deploying Stimulsoft in pure JavaScript applications. This application consists of an HTML page and JS scripts.

Create an index.html file
This can be any HTML file, but by default, the entry point is `index.html`.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
```

```
</body>
</html>
```

Install Stimulsoft components

First, you need to download the Stimulsoft package. If you need reporting tools, you should download the Stimulsoft Reports.JS package. If you need reporting tools and dashboards, you should download the Stimulsoft Dashboards.JS package. Then, you should connect the Stimulsoft scripts in the `index.html` file.

index.html

```
...
<script type="text/javascript" src="scripts/stimulsoft.reports.js"></
script>
<script type="text/javascript" src="scripts/stimulsoft.dashboards.js"></
script>
<script type="text/javascript" src="scripts/stimulsoft.designer.js"></
script>
<script type="text/javascript" src="scripts/stimulsoft.viewer.js"></
script>
<script type="text/javascript" src="scripts/
stimulsoft.blockly.editor.js"></script>
...
```

Create a start function

For example, create a function to open the report designer with an empty report.

index.html

```
...
<script type="text/javascript">
   function onLoad() {
      var report = new Stimulsoft.Report.StiReport();

      var designer = new Stimulsoft.Designer.StiDesigner();
      designer.renderHtml('content');
      designer.report = report;
   }
</script>

...

<body onload="onLoad()">
   <div id="content"></div>
</body>
...
```

Alternatively, create a function to open the report viewer with a previously created report template.

index.html

```
...
<script type="text/javascript">
   function onLoad() {
      var report = new Stimulsoft.Report.StiReport();
      report.loadFile('reports/Report.mrt');

      var viewer = new Stimulsoft.Viewer.StiViewer();
      viewer.renderHtml('content');
      viewer.report = report;
   }
</script>

...

<body onload="onLoad()">
   <div id="content"></div>
</body>
...
```

First Start

By default, the browser doesn't have access to the file system due to the browser's security policy. To ensure the local project runs correctly, you should use a web server. For example, you can globally install http-server or serve, and then start the web server from the command line in the project's root folder. In this case, `index.html` will be opened in the browser with the Stimulsoft designer or viewer.

9.1.2 Vanilla JavaScript and CDN Services

This chapter will cover an example of quickly deploying Stimulsoft in pure JavaScript applications by connecting script files via CDN services. Such an application consists of an HTML page and links to JS scripts.

Create an index.html file

It can be any HTML file, but by default, the entry point is considered to be `index.html`.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

Connect stimulsoft script files

Currently, the following services can be used:

- cdn.jsdelivr.net;
- unpkg.com.

These services provide access to script files via URLs from the npm packages stimulsoft-reports-js and stimulsoft-dashboards-js. This allows you to include script files in your `index.html` file via their URLs. For example, using the `cdn.jsdelivr.net` service:

index.html

```
...
<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/
stimulsoft-reports-js/Scripts/stimulsoft.reports.js"></script>
<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/
stimulsoft-reports-js/Scripts/stimulsoft.designer.js"></script>
<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/
stimulsoft-reports-js/Scripts/stimulsoft.viewer.js"></script>
<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/
stimulsoft-reports-js/Scripts/stimulsoft.blockly.editor.js"></script>
...
```

Or, using the `unpkg.com` service:

index.html

```
...
<script type="text/javascript" src="https://www.unpkg.com/stimulsoft-
dashboards-js/Scripts/stimulsoft.reports.js"></script>
<script type="text/javascript" src="https://www.unpkg.com/stimulsoft-
dashboards-js/Scripts/stimulsoft.dashboards.js"></script>
```

```
<script type="text/javascript" src="https://www.unpkg.com/stimulsoft-dashboards-js/Scripts/stimulsoft.designer.js"></script>
<script type="text/javascript" src="https://www.unpkg.com/stimulsoft-dashboards-js/Scripts/stimulsoft.viewer.js"></script>
<script type="text/javascript" src="https://www.unpkg.com/stimulsoft-dashboards-js/Scripts/stimulsoft.blockly.editor.js"></script>
...
```

Information

Please note that using the `cdn.jsdelivr.net` and `unpkg.com` services, you can include various scripts provided in the stimulsoft-reports-js and stimulsoft-dashboards-js packages. Additionally, you can include script files of a specific npm package version by specifying it in the URL with the `@` symbol. For example, https://cdn.jsdelivr.net/npm/stimulsoft-reports-js@2024.4.1/Scripts/stimulsoft.reports.js или https://www.unpkg.com/stimulsoft-reports-js@2024.4.1/Scripts/stimulsoft.reports.js.

If the version isn't specified in the URL, the script files will be loaded from the latest available version of the stimulsoft-reports-js or stimulsoft-dashboards-js package.

Create a launch function

For example, a function to launch the report designer with an empty report.

index.html

```
...
<script type="text/javascript">
   function onLoad() {
      var report = new Stimulsoft.Report.StiReport();

      var designer = new Stimulsoft.Designer.StiDesigner();
      designer.renderHtml('content');
      designer.report = report;
   }
</script>

...

<body onload="onLoad()">
   <div id="content"></div>
</body>
...
```

Or, a function to launch the report viewer with a previously created report template.

index.html

```
...
<script type="text/javascript">
   function onLoad() {
      var report = new Stimulsoft.Report.StiReport();
      report.loadFile('reports/Report.mrt');

      var viewer = new Stimulsoft.Viewer.StiViewer();
      viewer.renderHtml('content');
      viewer.report = report;
   }
</script>

...

<body onload="onLoad()">
   <div id="content"></div>
</body>
...
```

First launch

By default, browsers do not have access to the file system due to browser security policies. To ensure that a local project runs correctly, you need to use web servers. For example, you can globally install http-server or serve and then start a web server from the command line in the root folder of the project. In this case, the `index.html` file will open in the browser with the Stimulsoft designer or viewer.

9.1.3 Angular JS

This chapter describes an example of using Stimulsoft components in Angular JS applications.

Create an Angular project

In the terminal navigate to the directory where you want to place the new project. Then, run the command.

terminal

```
ng new sti-angular-js --no-standalone --routing --ssr=false --style css
```

Install Stimulsoft components

First, you need to download the Stimulsoft package. If you need reporting tools, you should download the Stimulsoft Reports.JS package. If you need reporting tools and dashboards, you should download the Stimulsoft Dashboards.JS package. Then, copy the Stimulsoft scripts into the project at the path `sti-angular-js./src/scripts`.

Configure the Angular project

To do this, in the `scripts` section of the **angular.json** file, you should specify an array of paths to the scripts files in the project.

angular.json

```
...
"scripts": [
   "src/scripts/stimulsoft.reports.engine.js",
   "src/scripts/stimulsoft.reports.export.js",
   "src/scripts/stimulsoft.reports.chart.js",
   "src/scripts/stimulsoft.reports.maps.js",
   "src/scripts/stimulsoft.reports.import.xlsx.js",
   "src/scripts/stimulsoft.viewer.js",
   "src/scripts/stimulsoft.designer.js",
   "src/scripts/stimulsoft.blockly.editor.js"
]
...
```

In the section `architect` for the parameter `builder`, set the type to `browser` instead of `application`. Additionally, rename the parameter in `options` from `browser` to `main`.

angular.json

```
...
"architect": {
   "build": {
      "builder": "@angular-devkit/build-angular:browser",
      "options": {
         "main": "src/main.ts"
      }
   }
}
...
```

If you plan to use pre-existing reports in the viewer or designer, you should add the path to them in the `assets` section.

angular.json

```
...
"assets": [
  "src/reports"
]
...
```

Add the HttpClientModule module

You need to import and then connect the `HttpClientModule` in the `app.module.ts` file.

app.module.ts

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
...
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  ...
})
```

Place the Stimulsoft component

In the **app.component.ts** file, you also need to import `HttpClientModule`. Then, you can import Stimulsoft using the directive `declare var Stimulsoft: any;`. In the `AppComponent` class, you should define the initialization of Stimulsoft components. For example, the designer with an empty report.

app.component.ts

```
import { Component } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

declare var Stimulsoft: any;

@Component({
```

```
  selector: 'app-root',
  template: `<div>
              <h2>Stimulsoft Reports.JS - Invoice.mrt - Designer</h2>
              <div id="content"></div>
            </div>`
})

export class AppComponent {
  designer: any = new Stimulsoft.Designer.StiDesigner(false,
  "StiDesigner", false);

  ngOnInit() {
    var report = new Stimulsoft.Report.StiReport();

    this.designer.report = report;
    this.designer.renderHtml("content");
  }

  constructor(private http: HttpClientModule) {

  }
}
```

Or open the viewer with a previously created report.

index.html

```
...
export class AppComponent {
  viewer: any = new Stimulsoft.Viewer.StiViewer(false, "StiViewer",
  false);

  ngOnInit() {
    var report = new Stimulsoft.Report.StiReport();
    report.loadFile("reports/Invoice.mrt");

    this.viewer.report = report;
    this.viewer.renderHtml("content");
  }

  constructor(private http: HttpClientModule) {

  }
}
...
```

First Start

An Angular application by default defines the start command in the **package.json** file. Therefore, to start the project, run the command from the terminal in the root

folder of the project.

console

```
npm start
```

9.1.4 React JS

This part describes an example of using Stimulsoft components in React applications.

Create a React project
In the terminal, navigate to the directory where you want to place the new project. Then, run the following command.

terminal

```
npx create-react-app my-app
```

Install Stimulsoft components
First, you need to download the Stimulsoft package. If you need reporting tools, you should download the Stimulsoft Reports.JS package. If you need reporting tools and dashboards, you should download the Stimulsoft Dashboards.JS package. For this, run the following command.

terminal

```
npm install stimulsoft-dashboards-js
```

Integrate Stimulsoft into the application
To do this, edit the **App.js** file in the project's **src** folder. First, you need to import the `Stimulsoft` module and the `React` class. Then, create an `App` class that extends `React.Component`, override the class constructor, the `render()` and the `componentDidMount()` methods. For example, the report designer will be opened with an empty report.

App.js

```
...
import React from 'react';
import { Stimulsoft } from 'stimulsoft-dashboards-js/Scripts/
stimulsoft.designer';

class App extends React.Component {
  constructor() {
    super();
    this.designer = new Stimulsoft.Designer.StiDesigner(false,
    "StiDesigner", false);
  }

  render() {
    return (
           <div className="App">
             <h2>Stimulsoft Designer</h2>
             <div id="content"></div>
           </div>
    );
  }

  componentDidMount() {
    var report = new Stimulsoft.Report.StiReport();

    this.designer.report = report;
    this.designer.renderHtml("content");
  }
}

export default App;
...
```

Alternatively, open the viewer with a previously created report. The report files should be copied beforehand into the **reports** folder in the **./public** directory of the project.

App.js

```
...
class App extends React.Component {
  constructor() {
    super();
    this.viewer = new Stimulsoft.Viewer.StiViewer(false, "StiViewer",
    false);
  }

  render() {
    return (
           <div className="App">
             <h2>Stimulsoft Viewer</h2>
```

```
                <div id="content"></div>
            </div>
        );
    }

    componentDidMount() {
        var report = new Stimulsoft.Report.StiReport();
        report.loadFile("reports/Invoice.mrt");

        this.viewer.report = report;
        this.viewer.renderHtml("content");
    }
}
...
```

First Start

A **React** application by default defines the start command in the **package.json** file. Therefore, to run the project, simply execute the command from the terminal in the project's root folder.

console

```
npm start
```

9.1.5 Vue JS

This part describes an example of using Stimulsoft components in Vue applications.

Create a Vue project

In the terminal navigate to the directory where you want to place the new project. Then, run the following command.

terminal

```
npm create vue@latest my-app
```

The application settings can be left as default or modified if necessary.

Install Stimulsoft components

First, you need to download the Stimulsoft package. If you need reporting tools, you should download the Stimulsoft Reports.JS package. If you need reporting tools and dashboards, you should download the Stimulsoft Dashboards.JS package. For this, run the following command.

terminal

```
npm install stimulsoft-dashboards-js
```

Integrate Stimulsoft into the application

To do this, edit the **App.vue** file in the project's **src** folder. First, import the Stimulsoft module and the `onMounted()` method. You can also modify the `template` and `styles` blocks here. Pass a `callback` function with the initialization of Stimulsoft components into the `onMounted()` method. For example, the report designer will open with a blank report.

App.vue

```
<script setup lang="ts">
   import { onMounted } from "vue";
   import { Stimulsoft } from "stimulsoft-dashboards-js/Scripts/
   stimulsoft.designer.js";

   onMounted(() => {
      let designer = new Stimulsoft.Designer.StiDesigner(false,
      "StiDesigner", false);
      let report = new Stimulsoft.Report.StiReport();

      designer.report = report;
      designer.renderHtml("content");
   });
</script>

<template>
   <div id="app">
      <div>
         <h2 id="app-title">Stimulsoft Vue JS</h2>
         <div id="content"></div>
      </div>
   </div>
</template>

<style>
   #app {
      font-family: Avenir, Helvetica, Arial, sans-serif;
      -webkit-font-smoothing: antialiased;
      -moz-osx-font-smoothing: grayscale;
```

```
    color: #2c3e50;
    margin-top: 60px;
  }

  #app-title {
    text-align: center;
  }
</style>
```

Alternatively, open the viewer with a previously created report. The report files should be copied beforehand into the **reports** folder in the **./public** directory of the project.

App.vue

```
...
<script setup lang="ts">
  import { onMounted } from "vue";
  import { Stimulsoft } from "stimulsoft-dashboards-js/Scripts/
  stimulsoft.viewer.js";

  onMounted(() => {
    let viewer = new Stimulsoft.Viewer.StiViewer(false, "StiViewer",
    false);
    let report = new Stimulsoft.Report.StiReport();
    report.loadFile("reports/SimpleList.mrt");

    viewer.report = report;
    viewer.renderHtml("content");
  });
</script>
...
```

First Start

By default, a Vue application defines the start command in the **package.js**on file. Therefore, to run the project, simply execute the command from the terminal in the project's root folder.

console

```
npm run dev
```

9.1.6 Node JS

This part describes an example of using Stimulsoft in Node.js applications. In this case, you can use only the report engine without the visual components, such as the report designer and viewer.

Create a Node.js project
In the terminal, navigate to the directory where you want to place the new project. Then, run the following command.

terminal

```
npm init
```

The application settings can be left at default or modified as needed.

Install Stimulsoft components
First, you need to download the Stimulsoft package. If you need reporting tools, you should download the Stimulsoft Reports.JS package. If you need reporting tools and dashboards, you should download the Stimulsoft Dashboards.JS package. For this, run the following command.

terminal

```
npm install stimulsoft-dashboards-js
```

Create a file project
This can be any js file, for example `index.js`.

Add reports to the project
Copy the folder with **reports** to the project. For example, the reports folder will contain the report file **Invoice.mrt**.

Integrate Stimulsoft into the application

To do this, edit the `index.js` file in the project's folder. First, you need to import the Stimulsoft module. Then, create a `StiReport()` object, load the report into it, build it and export it as a PDF file.

index.js

```
var Stimulsoft = require("stimulsoft-dashboards-js");

var report = new Stimulsoft.Report.StiReport();
report.loadFile("reports/Invoice.mrt");

report.renderAsync(function () {
   report.exportDocumentAsync(function (data) {
      var buffer = new Buffer.from(data, "utf-8");
      var fs = require("fs");
      fs.writeFileSync("Invoice.pdf", buffer);
   }, Stimulsoft.Report.StiExportFormat.Pdf);
});
```

First Start

To do this, run the project file in the Node.js

console

```
node index.js
```

9.1.7 Step by Step for Browser

This example shows how to install and run Stimulsoft JS for Browser. Also, you can find more samples Reports.JS and Dashboards.JS on the GitHub.

Step 1: Open your html file in editor. For example, **index.html**.

Step 2: Add reference to viewer **CSS file** in HTML head:

index.html

```
...
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <title>JS Project</title>
```

```
    <link href="https://unpkg.com/stimulsoft-dashboards-js/Css/
    stimulsoft.viewer.Office2022.whiteblue.css" rel="stylesheet"/>

    //Also, you must to specify designer css file link, if you need to use
    js designer
    <link href="https://unpkg.com/stimulsoft-dashboards-js/Css/
    stimulsoft.designer.Office2022.whiteblue.css" rel="stylesheet"/>
</head>
</html>
...
```

Step 3: Add the references to stimulsoft js scripts in HTML head.

index.html

```
...
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>JS Project</title>

    <link href="https://unpkg.com/stimulsoft-dashboards-js/Css/
    stimulsoft.viewer.Office2022.whiteblue.css" rel="stylesheet"/>

    //Also, you must to specify designer css file link, if you need to use
    js designer
    <link href="https://unpkg.com/stimulsoft-dashboards-js/Css/
    stimulsoft.designer.Office2022.whiteblue.css" rel="stylesheet"/>

    <script src="https://unpkg.com/stimulsoft-reports-js/Scripts/
    stimulsoft.reports.js" type="text/javascript"></script>

    //Also, you must to specify dashboard script link, if you need to use
    dashboards
    <script src="https://unpkg.com/stimulsoft-dashboards-js/Scripts/
    stimulsoft.dashboards.js" type="text/javascript"></script>
    <script src="https://unpkg.com/stimulsoft-reports-js/Scripts/
    stimulsoft.viewer.js" type="text/javascript"></script>

    //Also, you must to specify designer script link, if you need to use js
    designer
    <script src="https://unpkg.com/stimulsoft-reports-js/Scripts/
    stimulsoft.designer.js" type="text/javascript"></script>
</head>
</html>
...
```

Information

It is important to follow the order in which scripts are connected:

- The **stimulsoft.reports.js** script should be the first;
- The **stimulsoft.viewer.js** script must be earlier than **stimulsoft.designer.js** script.

Step 4: Create the **onLoad()** function and a new viewer, if you want to show report or dashboard.

index.html

```
...
<script type="text/javascript">
function onLoad(){
   // Create the report viewer with default options
   var viewer = new Stimulsoft.Viewer.StiViewer(null, "StiViewer", false);

   // Show the report viewer in div content
   viewer.renderHtml("content");
}
</script>
...
//Call function onLoad() and put viewer in <div>.
<body onLoad="onLoad()">
   <div id="content">Component should be here</div>
</body>
...
```

Step 5: Create a new report object, load a report template file in it, and assign report object to the report viewer.

index.html

```
...
<script type="text/javascript">
function onLoad(){
...
   // Create a new report object
   var report = new Stimulsoft.Report.StiReport();

   // Load report template in the report object
   report.loadFile("report url");

   // Assign report object to report viewer
   viewer.report = report;
...
}
</script>
```

```
...
```

Step 6: Create the **onLoad()** function and a new designer, if you want to create or edit report or dashboard.

index.html

```
...
<script type="text/javascript">
function onLoad(){
   // Create the report designer with default options
   var designer = new Stimulsoft.Designer.StiDesigner(null, "Designer",
   false);

   // Show the report designer in div content
   designer.renderHtml("content");

   // Create a new report object
   var report = new Stimulsoft.Report.StiReport();

   // Load report template in the report object
   report.loadFile("report url");

   // Assign report object to report designer
   designer.report = report;
}
</script>
...
//Call function onLoad() and put designer in <div>.
<body onLoad="onLoad()">
   <div id="content">Component should be here</div>
</body>
...
```

Step 7: Save changes and open **index.html** file in browser.

9.1.8 Step by Step for Node.js

This example shows how to install and run Stimulsoft JS for Node.js. Also, you can find more samples Reports for Node.js and Dashboards for Node.js on the GitHub.

Step 1: Create a folder;

Step 2: Add a report template in this folder;

Step 3: Create **index.js** file in this folder;

Step 4: Open console;

Step 5: Install the Reports.JS module in this folder:

console

```
npm install stimulsoft-reports-js
```

Or

Install the Dashboards.JS module in this folder:

console

```
npm install stimulsoft-dashboards-js
```

Step 6: Open **index.js** file in the editor;

Step 7: Add the required code.

index.js

```
...
// Stimulsoft reports module loading
var Stimulsoft = require('stimulsoft-reports-js');

// Loading fonts
Stimulsoft.Base.StiFontCollection.addOpentypeFontFile("Roboto-Black.ttf");

//Creating a new report object
var report = Stimulsoft.Report.StiReport.createNewReport();

// Load report template in the report object
report.loadFile("report1.mrt");

// Save report object in mrt file
report.saveFile("report2.mrt");
...
```

Step 8: Save changes in the **index.js** file;

Step 9: Open console and run **index.js**.

console

```
node index
```

Step 10: You can perform various actions on reports with using **Node.js**. For example, let's export the report to pdf. Open **index.js** in the editor, define this code and save changes.

index.js

```
...
// Stimulsoft Reports module
var Stimulsoft = require('stimulsoft-reports-js');

// Loading fonts
Stimulsoft.Base.StiFontCollection.addOpentypeFontFile("Roboto-Black.ttf");

// Creating new report
var report = new Stimulsoft.Report.StiReport();

// Loading report template
report.loadFile("report1.mrt");

// Renreding report
report.renderAsync(() => {
   console.log("Report rendered. Pages count: ",
   report.renderedPages.count);

   // Export to PDF
   report.exportDocumentAsync((pdfData) => {

      // Converting Array into buffer
      var buffer = Buffer.from(pdfData)

      // File System module
      var fs = require('fs');

      // Saving string with rendered report in PDF into a file
      fs.writeFileSync('./SimpleList.pdf', buffer);
      console.log("Rendered report saved into PDF-file.");
   },
   Stimulsoft.Report.StiExportFormat.Pdf);
});
...
```

Step 11: Open console and run **index.js**.

console

```
node index
```

9.2 HTML5 Viewer

YouTube

Watch videos for the JS HTML5 Viewer. Subscribe to the Stimulsoft channel to find new video lessons uploaded. . Leave your questions and suggestions in the comments to the video.

Samples

See GitHub examples for working with the JS HTML5 Viewer component. All examples are separate projects, grouped into one solution for Visual Studio.

The **HTML5 Viewer** (**StiViewer**) component is designed to view reports in the web browser. You do not need to install the .NET Framework, ActiveX components or any special plug-ins on the client side. All that is needed is any modern Web browser.

With help of **HTML5 Viewer**, you can view, print and export reports on any computer with any operating system installed. Since the viewer only uses HTML and JavaScript technologies, it can be run on devices where there is no Flash or Silverlight support - tablets, smartphones. Also, the viewer supports Mobile and Touch interfaces, which automatically enable when using mobile devices and touch screen monitors.

The **HTML5 Viewer** component uses the **JavaScript** technology to perform all actions (uploading a report, paging, scaling, interactivity in reports, etc.), which allows you to get rid of reloading the entire page and speed up work.

HTML5 Viewer supports many themes, animated interface, bookmarks, interactive reports, editing of report elements on the page, full screen mode, search panel, and other necessary functionality for viewing reports.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use the **HTML5 Viewer** in project, you need to install the npm package of stimulsoft-reports-js:

```
npm install stimulsoft-reports-js
```

If this is not possible, you should add the following scripts to the project.

viewer.html

```
...
<script src="scripts/stimulsoft.report.js"></script>
<script scr="scripts/stimulsoft.viewer.js"></script>
...
```

Install the npm package to use the HTML5 Viewer in a project to view reports and dashboards stimulsoft-dashboards-js:

```
npm install stimulsoft-dashboards-js
```

If this is not possible, you should add the following scripts to the project.

viewer.html

```
...
<script src="Scripts/stimulsoft.reports.js"></script>
<script src="Scripts/stimulsoft.dashboards.js"></script>
<script src="Scripts/stimulsoft.viewer.js"></script>
...
```

Information

The script variants that you can use in your projects can be found in the Scripts of Reports.JS Package and Scripts of Dashboards.JS Package chapters.

9.2.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **StiViewer** component is designed to use only HTML5 and JavaScript technology, and does not require a server for its work (it is necessary only for hosting project files). When you run the report viewer, the following actions occur:

- JavaScript component adds code of the viewer interface to the current HTML page;
- If a report object has been assigned, the report will be rendered, then, the first page of the report will be displayed;
- Each action in the viewer (for example, paging, printing or exporting a report, etc.) calls a specific JavaScript event in which you can perform the necessary manipulations with the server report, which eliminates the re-build of the report.

9.2.2 Showing Reports

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `report.render()` method if you want to perform specific actions with the rendered report before it is displayed in the viewer.

To show the report, you should add the scripts and styles required for the **StiViewer** component to the HTML page of a project.

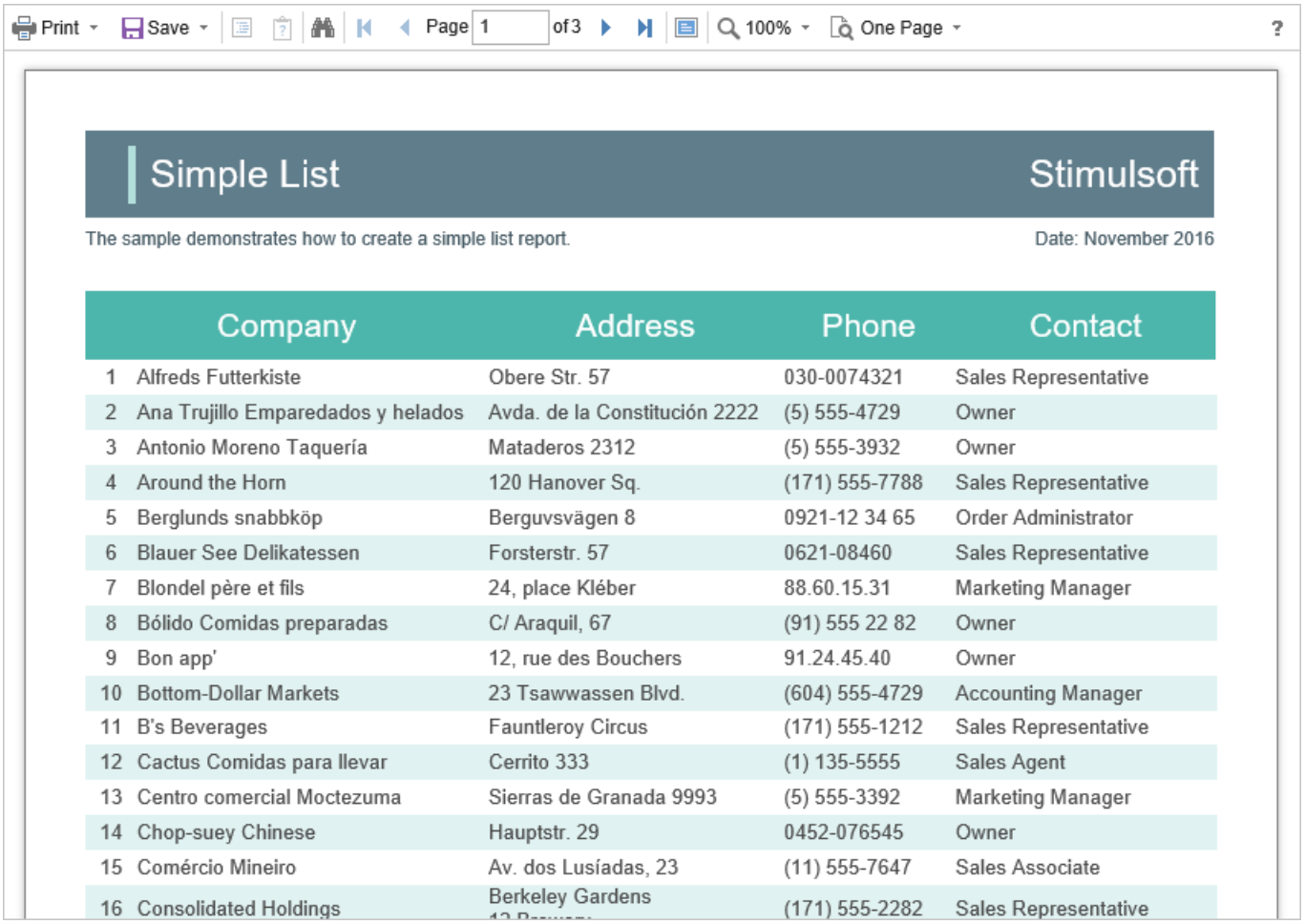
viewer.html

```
...
<script src="scripts/stimulsoft.reports.js" type="text/javascript"></
script>
<script src="scripts/stimulsoft.dashboards.js"></script>
<script src="scripts/stimulsoft.viewer.js" type="text/javascript"></
script>
...
```

Then you should add JavaScript code of report loading to the HTML page, and assign the resulting object to the viewer. In this case, the viewer will be deployed in the current DOM element at the place where the script is located.

viewer.html

```
...
<script type="text/javascript">
   var report = new Stimulsoft.Report.StiReport();
   report.loadFile("SimpleList.mrt");
   //report.loadFile("Dashboard.mrt");

   var viewer = new Stimulsoft.Viewer.StiViewer();
   viewer.report = report;
</script>
...
```



You can create a **StiViewer** object using the **Stimulsoft.Viewer.StiViewer()** constructor, which can accept non-required arguments as input:

› **options** is a set of options that can be found in the **Stimulsoft.Viewer.StiViewerOptions** class. All options are split into categories. A detailed description of the categories and options can be found in the Viewer Settings topic.

› **viewerId** - viewer identification is used when deploying a component as a DOM object, the default value is "StiViewer".

› **renderAfterCreate** - defines the viewer location. If it is set to **true**, the viewer will be displayed in the same place in the DOM tree where code to create an object is placed. If it is set to **false**, the viewer will be located at the place where the **renderHtml()** method is called. For example, this can be the initialization of the viewer in the page header.

viewer.html

```
...
<script type="text/javascript">
   var viewer = new Stimulsoft.Viewer.StiViewer(null, "StiViewer", false);
</script>
...
```

And the subsequent output of the viewer in the current DIV element.

viewer.html

```
...
<div>Page content</div>
<div>
   <script type="text/javascript">
   // Render the report viewer in this place
      viewer.renderHtml();
   </script>
</div>
...
```

As an argument of the **renderHtml(id)** output method of the viewer, it is allowed to specify the element identifier of the HTML page in which the viewer should be displayed.

viewer.html

```
...
<script type="text/javascript" >
   var viewer = new Stimulsoft.Viewer.StiViewer(null, "StiViewer", false);
   viewer.renderHtml("content");
</script>
...
```

The specified element must be located on the HTML page on which the report viewer is used.

viewer.html

```
...
<div id="content"></div>
...
```

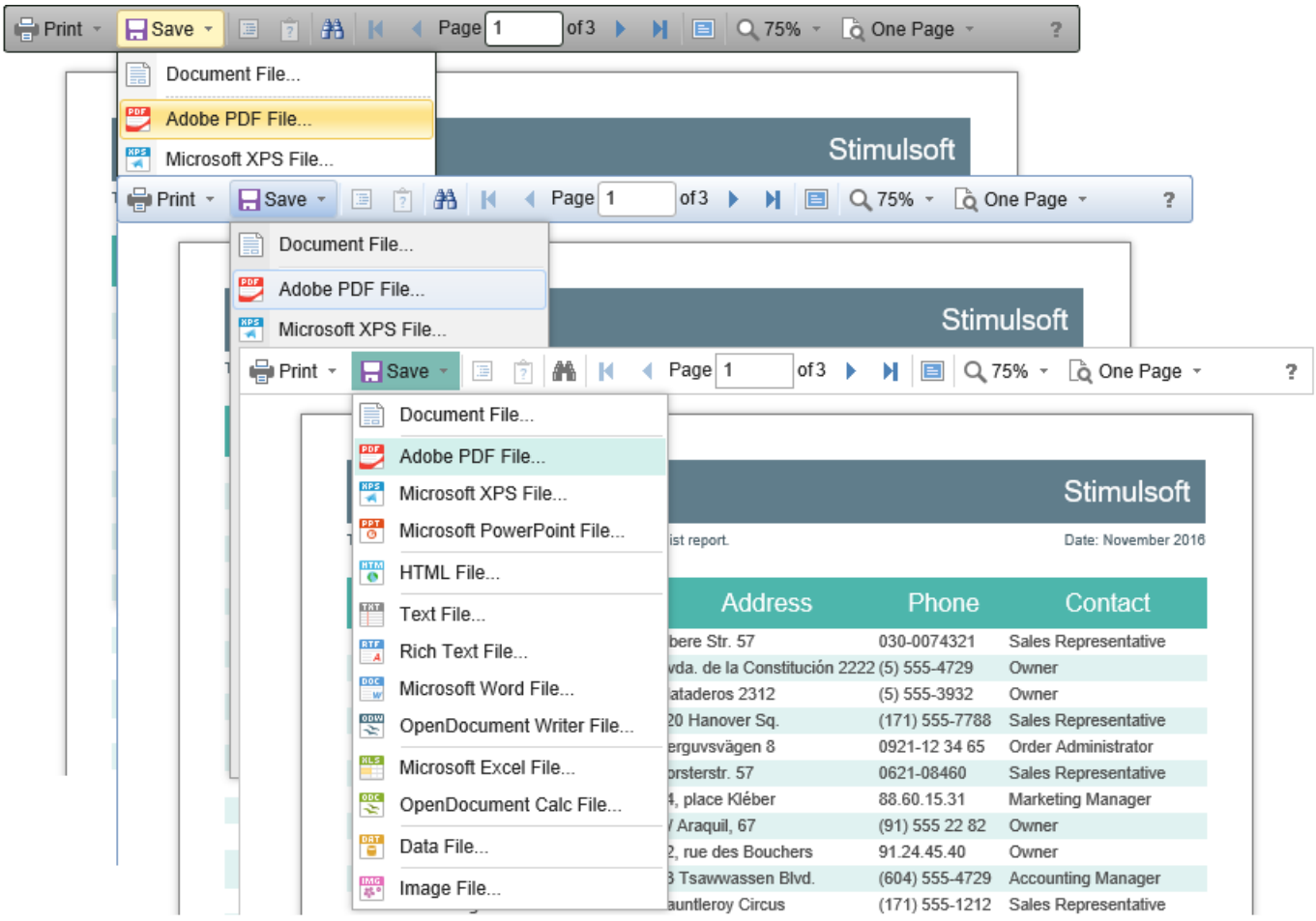
9.2.3 Using Themes

The **HTML5 Viewer** component has the ability to change themes for visual controls. You can use the theme component option or the **setTheme()** method for this.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.appearance.theme =
Stimulsoft.Viewer.StiViewerTheme.Office2022WhiteBlue;
...
viewer.setTheme(Stimulsoft.Viewer.StiViewerTheme.Office2022WhiteBlue);
...
```

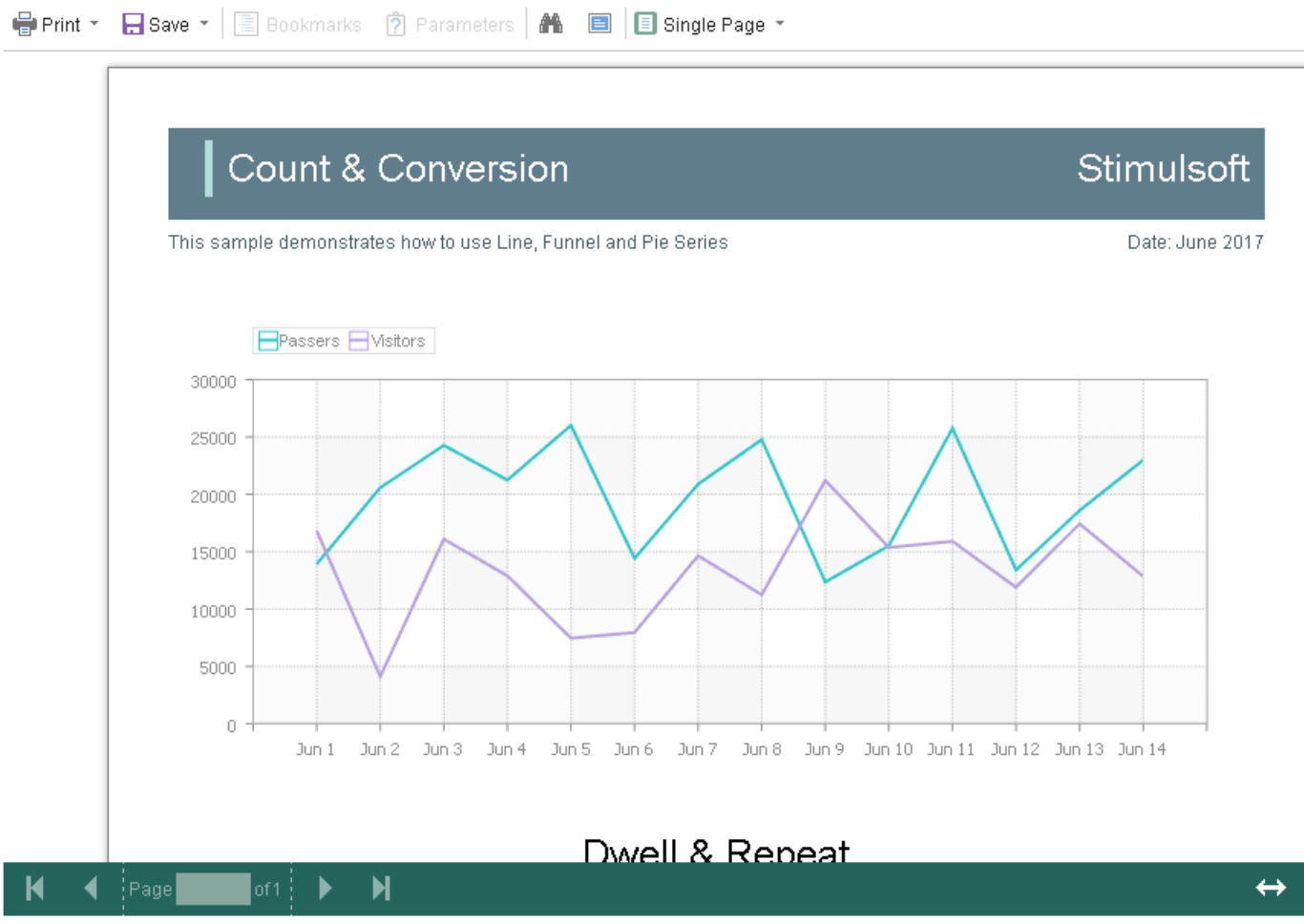
There are currently **8 themes** available with different color accents. As a result, **more than 60** variants of the appearance are available. This allows you to customize the appearance of the viewer for almost any design of the Web project.



By default, the viewer has only the top toolbar on which all the report controls are located. If necessary, the toolbar can be split into top and bottom parts. The top panel will contain the menu for printing and exporting the report, as well as the buttons for working with parameters and bookmarks. The bottom panel will contain controls to switch between the report pages and the menu to zoom pages. Use the **displayMode** property to enable this mode. The property has values **Simple** and **Separated**.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.toolbar.displayMode =
Stimulsoft.Viewer.StiToolbarDisplayMode.Simple;
options.appearance.scrollbarsMode = true;
...
```



In addition, it is possible to set the parameters of appearance for the main items of the viewer. For example, you can change the font and color for text of the control panel of the viewer, set the background of the viewer, set the color of page borders, etc. Below is a list of available properties that change the appearance of the viewer, and their default values.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();

options.appearance.backgroundColor =
Stimulsoft.System.Drawing.Color.white;
options.appearance.pageBorderColor = Stimulsoft.System.Drawing.Color.red;
options.appearance.showPageShadow = false;

options.toolbar.backgroundColor = Stimulsoft.System.Drawing.Color.aqua;
options.toolbar.borderColor = Stimulsoft.System.Drawing.Color.darkGreen;
options.toolbar.fontColor = Stimulsoft.System.Drawing.Color.white;
options.toolbar.fontFamily = "Arial";
...
```

9.2.4 Printing Reports

Information

Please note that the print option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component has several options for printing a report. Each has its own advantages and disadvantages.

Print to PDF

Printing will be done by exporting the report to the **PDF format**. The advantages are greater accuracy of positioning and printing of the report elements in comparison with other printing options. Among the drawbacks one can mention the mandatory presence of a plug-in installed in a web browser for viewing PDF files (modern browsers have embedded PDF viewer and printer).

Information

Internet Explorer and Edge browsers do not support direct output of PDF content from JavaSctipt code, so when printing as PDF, you will be prompted to save the file, and only then it can be printed.

Print with Preview

The report will be printed in a separate pop-up browser window in the **HTML format**. The report can be previewed, and then sent to the printer or copied to another location as text or HTML code. The advantages are cross-browser compatibility when printing, no need to install special plug-ins. The disadvantage is the relatively low accuracy of the position of the report elements, due to the issues with implementation of HTML formatting.

Print without Preview

The report will be printed directly to the printer without preview. After selecting this menu item, the system print dialog is displayed. Since printing in this mode is carried out in the **HTML format**, then the print quality is similar to the quality of printing a report with a preview.

Information

The report is printed using the built-in methods of the current browser, so the presentation of the dialog box may differ in operating systems and browsers. Also, the browser does not allow managing print settings from JavaScript code, so the required settings will need to be performed in the dialog box.

Print setup

If you choose printing a report in the viewer panel, a menu with printing options is displayed. The **HTML5 Viewer** component is able to force the required printing mode. To do this, set the **printDestination** property to one of the following values of the **StiPrintDestination** enumeration.

- **Default** – the menu will be displayed (the default property value);
- **Pdf** – print to the PDF format;

› **Direct** – printing to the HTML format directly to the printer, the system print dialog will be displayed;
› **WithPreview** – print to the HTML format with preview in a pop-up window.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.toolbar.printDestination =
Stimulsoft.Viewer.StiPrintDestination.Default;
...
```

The **HTML5 Viewer** component is able to completely disable report printing. To do this, set the value of the **showPrintButton** property to **false**.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.toolbar.showPrintButton = false;
...
```

Print Report from Code

Also it is possible to print a report using the code. To do this, you can use the special **print()** method on the report object.

viewer.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.loadFile("SimpleList.mrt");
report.renderAsync(function(){
   report.print();
});
...
```

When printing a report, it is possible to specify a print range. The special **StiPagesRange** class is used for this, it is allowed to set the following parameters as the arguments of the constructor.
› Range type (the following values are available

Stimulsoft.Report.StiRangeType.All, **Stimulsoft.Report.StiRangeType.Pages**, **Stimulsoft.Report.StiRangeType.CurrentPage**);

- Range as a string representation (page numbers are separated with commas or hyphenated);
- Current page number.

viewer.html

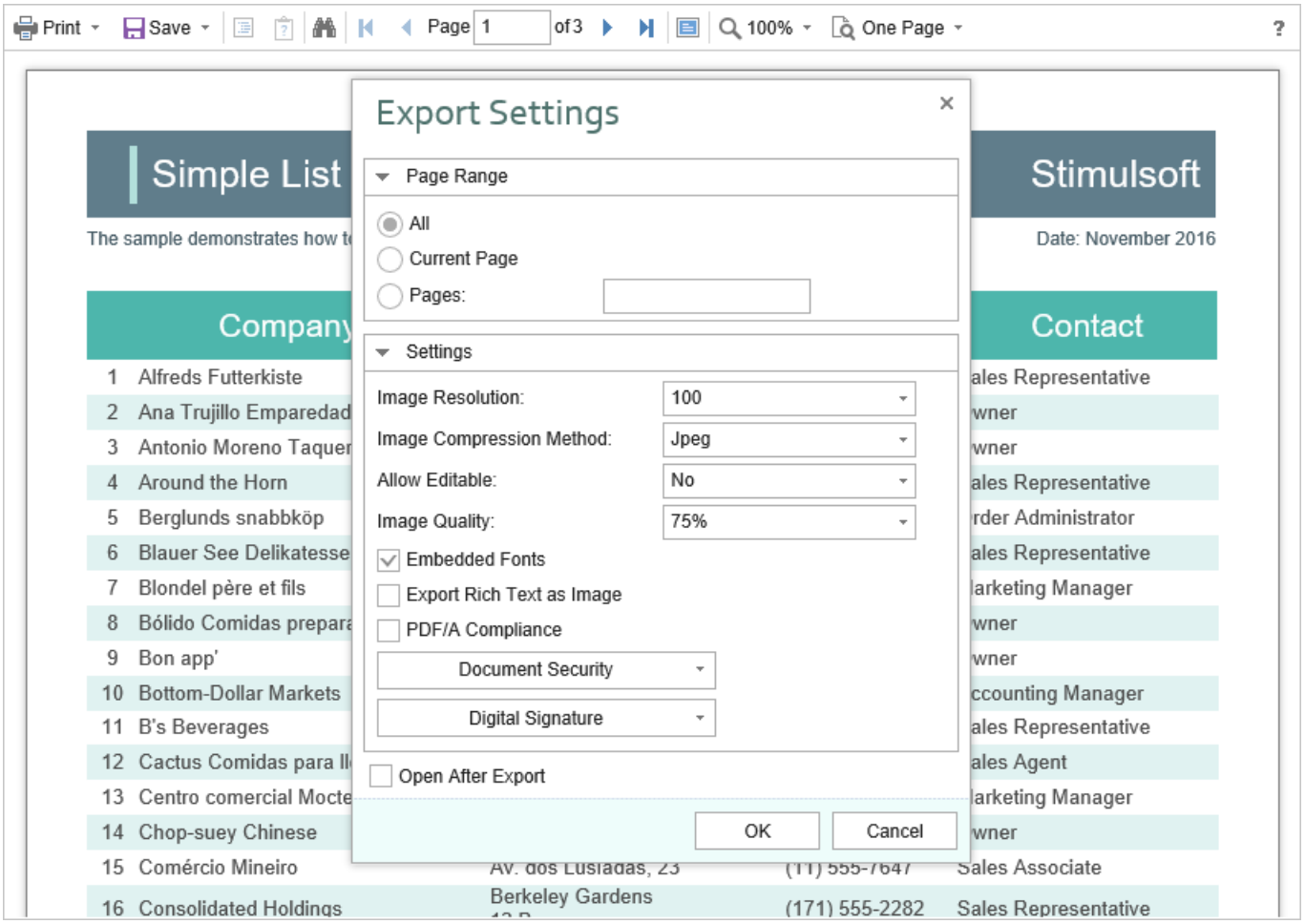
```
...
var pageRange = new
Stimulsoft.Report.StiPagesRange(Stimulsoft.Report.StiRangeType.CurrentPage
, "1,3-8", 5);
report.print(pageRange);
...
```

9.2.5 Exporting Reports

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Viewer** component allows you to export the displayed report to around three dozen of various formats, such as **PDF**, **HTML**, **Word**, **Excel**, **CSV**. The export function does not require additional settings of the viewer. You may export dashboard to PDF, Excel files.



Export events

To perform any actions before exporting a report, a special **onBeginExportReport** event is used. In this event, you can find out the type of report export, get the report, as well as get the report export settings and, if necessary, change them.

viewer.html

```
...
viewer.onBeginExportReport = function (event) {
   switch (event.format) {
      case Stimulsoft.Report.StiExportFormat.Html:
         event.settings.zoom = 2;  // Set zoom to 200%
         break;
   }
}
...
```

Read more about viewer events.

Export Settings

The **HTML5 Viewer** component contains 7 export formats, and sometimes you need to disable unused formats. This allows you to simplify UI and the use of the viewer. To disable unused export formats, it is enough to set the values for the corresponding properties of the viewer listed in the list below to **false**.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.exports.showExportToDocument = false;
options.exports.showExportToPdf = false;
options.exports.showExportToHtml = false;
options.exports.showExportToHtml5 = false;
options.exports.showExportToWord2007 = false;
options.exports.showExportToExcel2007 = false;
options.exports.showExportToCsv = false;
...
```

Also, if required, you can completely remove displaying dialog boxes of export, the export will always be carried out with the default settings. You should to set the value of the **showExportDialog** property to **false**.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.exports.showExportDialog = false;
...
```

Export Report from Code

Also you can export the report using code. You can use the special **exportDocument()** method on the report object.

viewer.html

```
...
```

```
// Create a new report instance
var report = new Stimulsoft.Report.StiReport();
// Load report from url
report.loadFile("../reports/SimpleList.mrt");
// Render report
report.renderAsync(function(){
   report.exportDocument(Stimulsoft.Report.StiExportFormat.Pdf); // Export
   report to PDF format
});
...
```

9.2.6 Viewing Modes

The **HTML5 Viewer** component has two modes for displaying reports - with and without scrollbars. By default, the view mode without scrollbars is set. To enable the scrollbar view mode, set the value of the **scrollbarsMode** property to **true**.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.appearance.scrollbarsMode = true;
...
```

In the first mode (without scrollbars), the viewer displays a page or report entirely, automatically stretching the preview space. If the width and height are specified, the viewer will truncate the page that is out of bounds. In the second mode, unlike the first one, when the page is out of bounds of the viewer's size, no truncation will be performed. Scrollbars will appear, using which you can view the entire page or report.

Information

In the report mode with scrollbars, you should set the height of the viewer; otherwise the default height will be set to **650 pixels**.

The **HTML5 Viewer** component has the full-screen report and dashboard mode. By default, the standard view mode is enabled; the viewer has the specified size in the settings. To enable the full-screen mode, set the **fullScreenMode** property to **true**.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.appearance.fullScreenMode = true;
...
```

Also, to enable or disable the full-screen mode, you can use the corresponding button on the control panel of the viewer.

The **HTML5 Viewer** component has three modes to display reports - page-by-page, entire report, and tabular display of report pages. To control the modes, the **viewMode** property is used. It can have one of the specified values - **SinglePage**, **Continuous**, **MultiplePages**.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.toolbar.viewMode = Stimulsoft.Viewer.StiWebViewMode.Continuous;
...
```

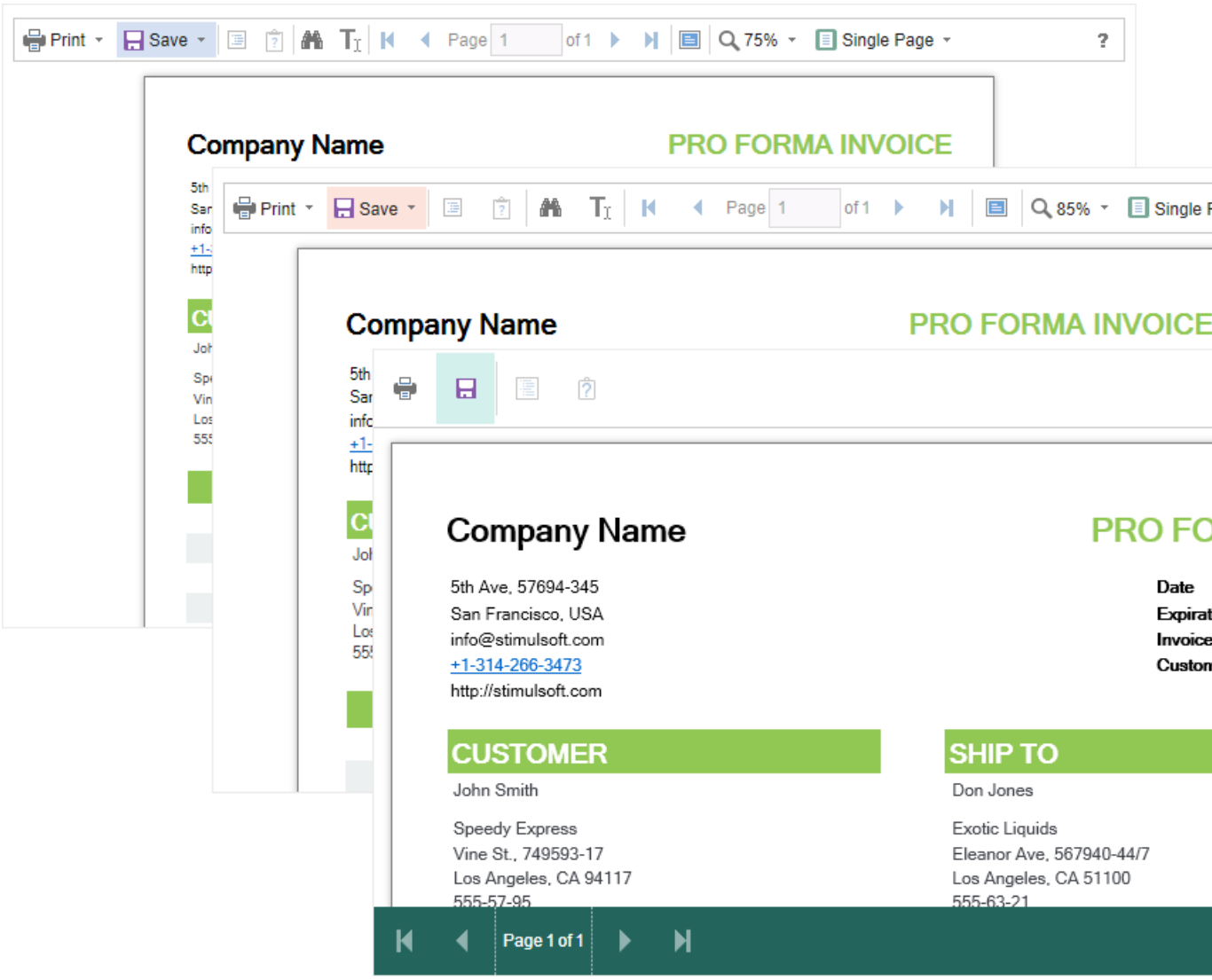
The **HTML5 Viewer** component supports interaction on a regular PC and touch screen displays and on mobile devices. The **interfaceType** property allows you to control the interface modes. The property can have one of the following values:

- **Auto** – the interface of the viewer is determined automatically depending of the device that the report is displayed on. This is the default value.
- **Mouse** – the standard interface with a mouse control will be used for all the screen types.
- **Touch** – the touch interface will be used to control the viewer. The interface design was optimized for the touch screen displays. The elements of the viewer interface are increased in size to simplify the control of the viewer and to improve its usability.
- **Mobile** - the mobile interface will be used to control the viewer for all the screen types. The mobile interface was designed to control the viewer using mobile smartphone displays. This interface design was simplified and adapted to use with the smartphones.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.appearance.interfaceType =
Stimulsoft.Viewer.StiInterfaceType.Auto;
```

```
. . .
```



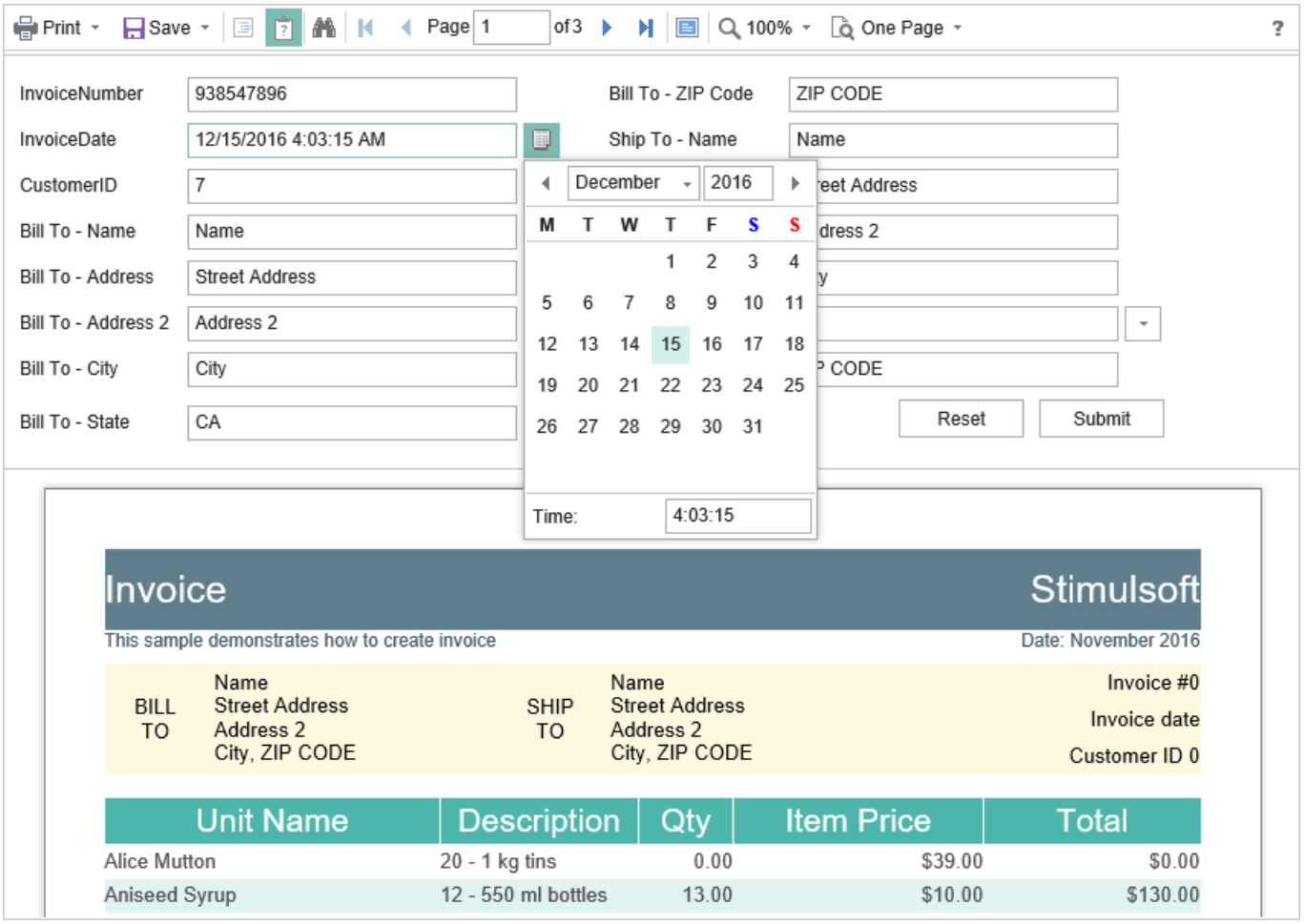
9.2.7 Work with Parameters

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

We have a special settings panel to work with report parameters in the **HTML5**

Viewer. To add a parameter to the panel you need to define a variable in a report, requested by the user. When viewing a report in the viewer such a variable will be automatically added to the settings panel. It supports all types of report variables (normal variables, date and time, borders, lists, etc.).



To work with reports with parameters, no additional viewer settings are required. If you need to perform some actions before applying the parameters, you can define a special **onInteraction** event. When using the parameters panel, the action type will be set to the **Variable** value.

viewer.html

```
...
viewer.onInteraction = function (args) {
   if (args.action == "Variables") {
      var variables = args.variables;
   }
```

```
}
...
```

If you do not need to work with parameters, you can completely disable this feature. To do this, use the **showParametersButton** property in the **Toolbar** section of properties, which should be set to **false**.

viewer.html

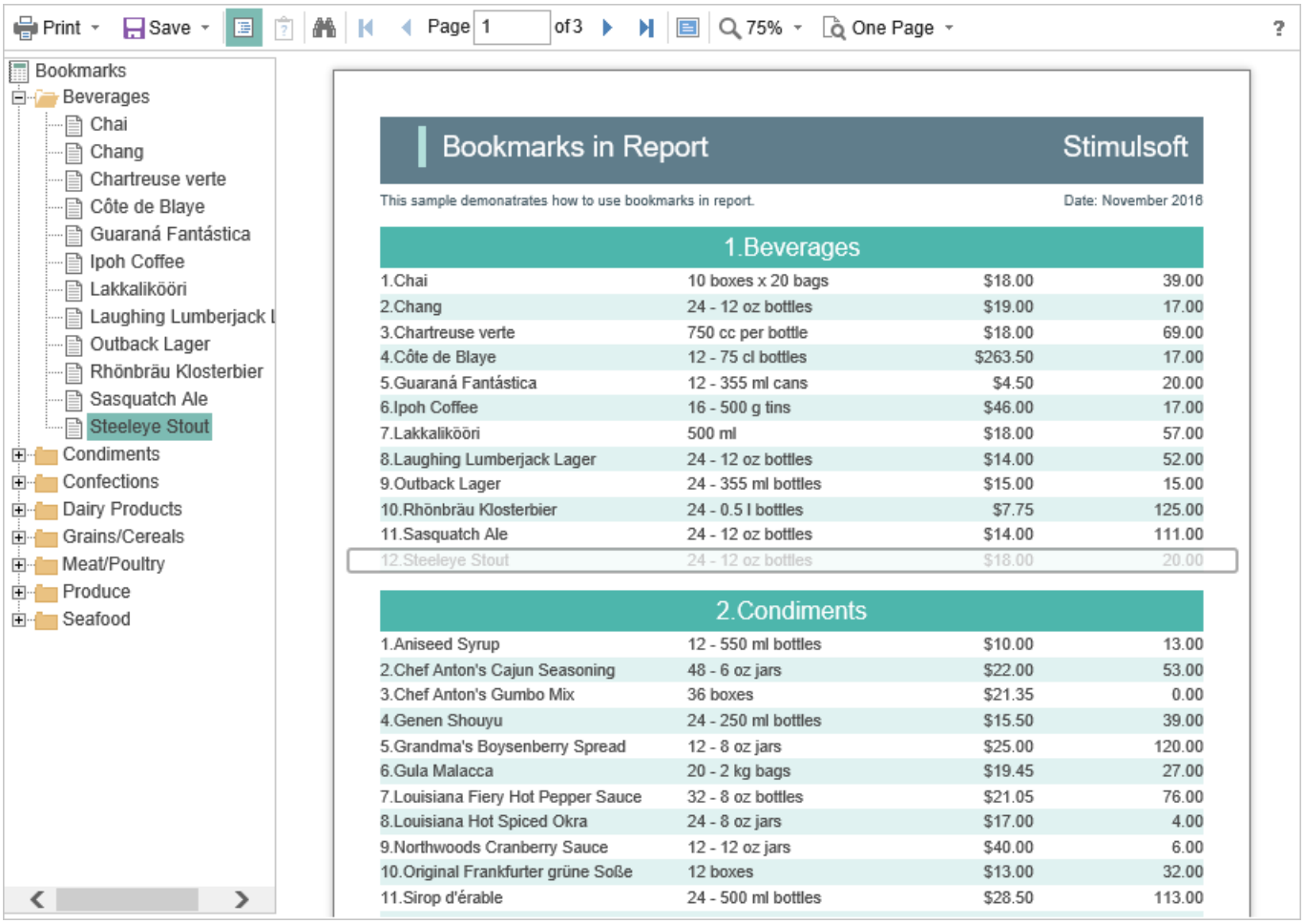
```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.toolbar.showParametersButton = false;
...
```

Information

With such a viewer configuration, the options panel will not be displayed, even if the parameters are present in the displayed report.

9.2.8 Work with Bookmarks

The **HTML5 Viewer** component supports report bookmarks. When displaying such a report on the left side of the page, a panel with bookmarks will be displayed. When you select a bookmark of the report, the viewer will carry out an automatic transition to the specified page, and the report item with a bookmark is highlighted.



By default, the bookmarks bar width is 180 pixels. The **HTML5 Viewer** component allows you to change this value. The **bookmarksTreeWidth** property, which value is specified in pixels, is used.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.appearance.bookmarksTreeWidth = 200;
...
```

If work with report bookmarks is not required, you can disable this functionality. For this, you should set the **showBookmarksButton** property to **false**.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.toolbar.showBookmarksButton = true;
...
```

Information

In this case, report bookmarks will not be displayed, even if they are present in the displayed report. This property has no effect on printing and exporting reports.

When printing a report with bookmarks, the bookmark tree will be hidden. If you want to print bookmarks with the report, it is necessary to set the **bookmarksPrint** property to **true**.

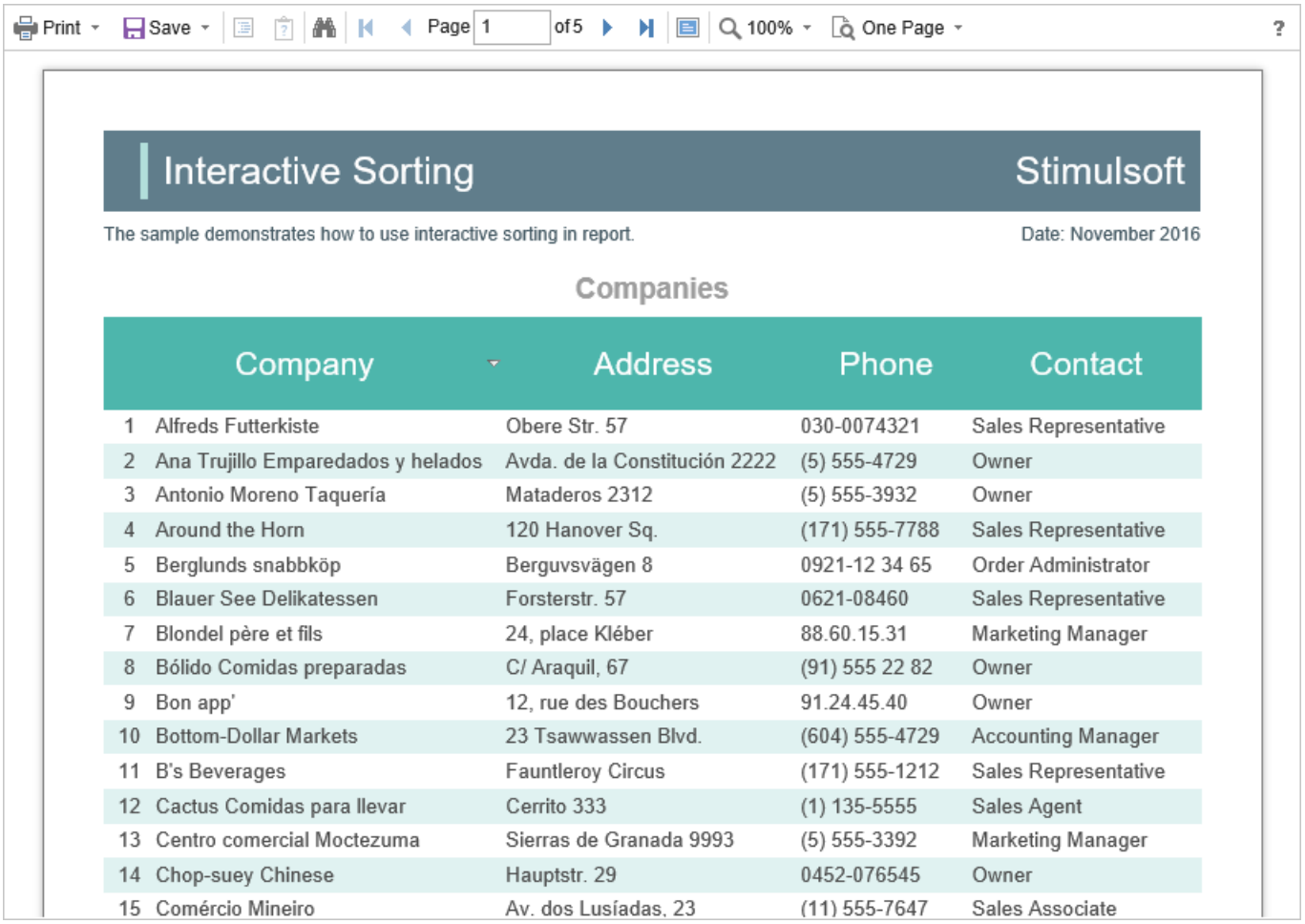
viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.appearance.bookmarksPrint = true;
...
```

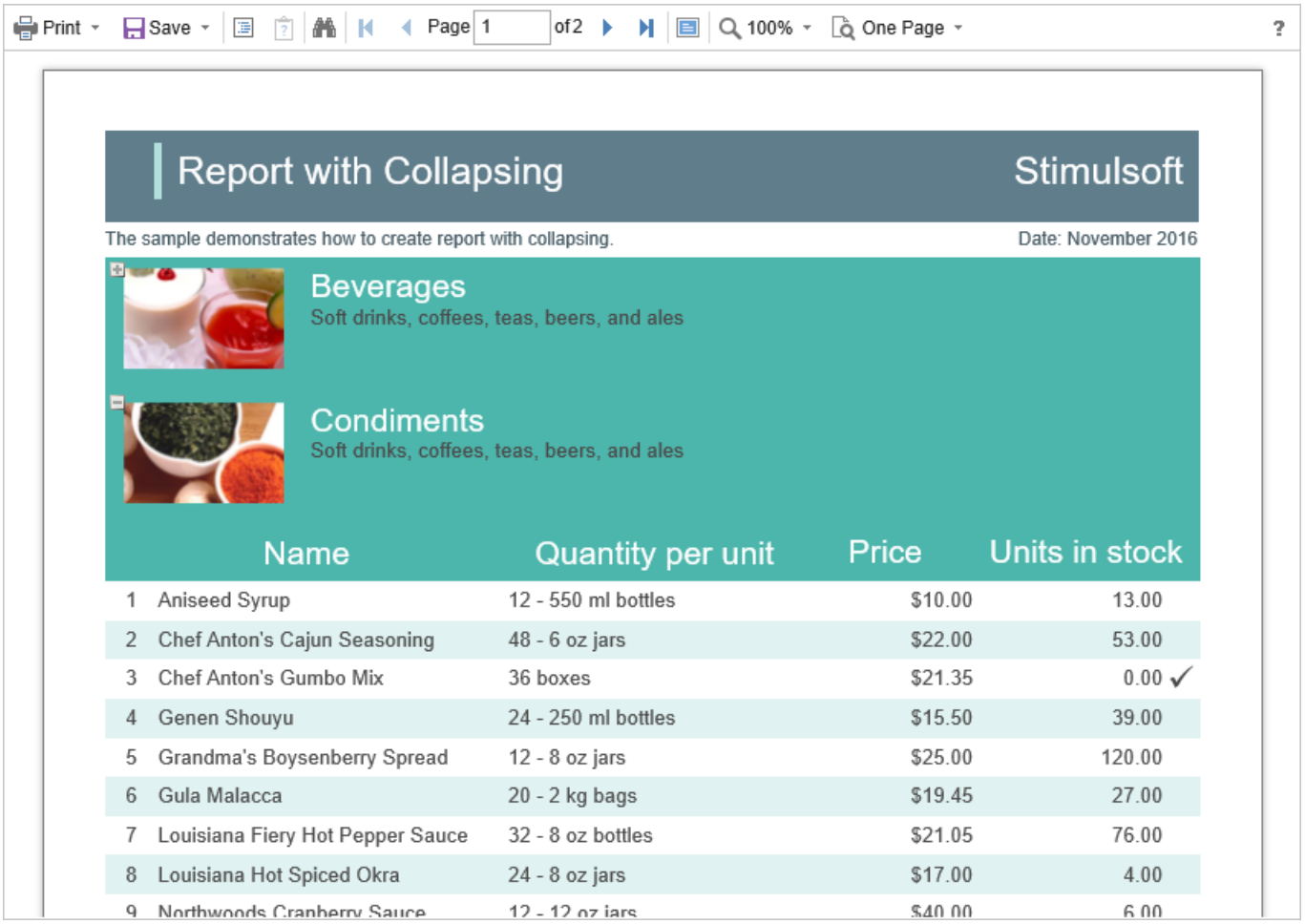
9.2.9 Dynamic Sorting, Collapsing, and Drill-Down

The **HTML5 Viewer** component supports dynamic sorting, collapsing, and drill-down reports. Dynamic sorting provides the ability to change the direction of sorting in a rendered report. To do this, click on the component that has the dynamic sorting enabled. Dynamic sorting is carried out in the following directions - **Ascending** and **Descending**. Each time the component is clicked, the sorting direction is reversed.

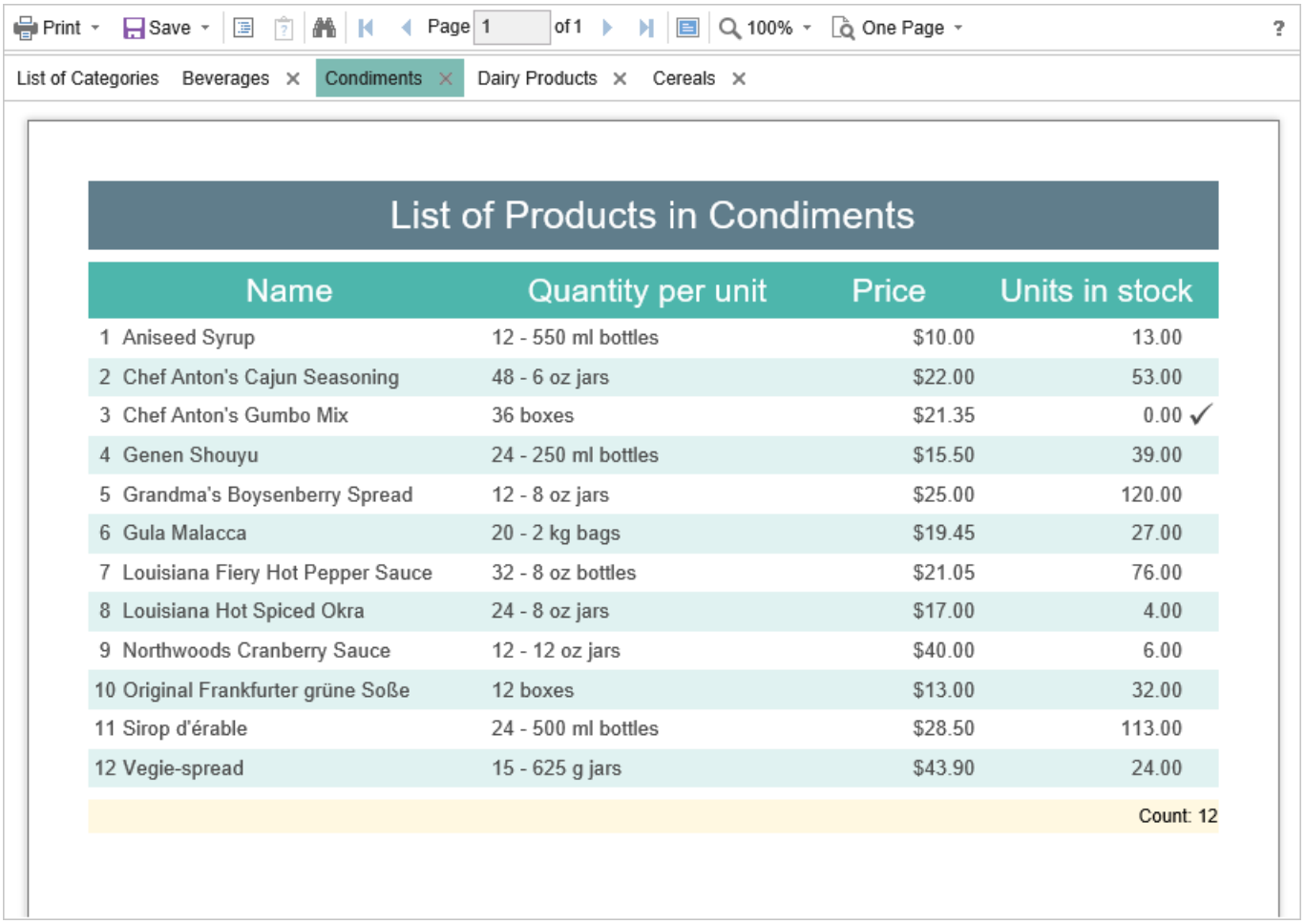
Multi-level sorting is allowed in the report. To do this, hold down the **Ctrl** key and sequentially click on the sorted components in the report. To reset sorting, you can click on any sorted component without holding down the **Ctrl** key.



A report with dynamic collapsing is an interactive report in which blocks can collapse/expand their content when you click on the block title. Report elements, which can be collapsed/expanded, are indicated by special icons - **[-]** or **[+]**.



When using drill-down under the main panel of the viewer, the drill-down panel with tabs for drill-down reports will be displayed. The currently displayed report will be highlighted.



For the work with dynamic sorting, folding and detailing of reports, no additional settings of the viewer are required. The special **onInteraction** event is used to perform any actions before sorting, minimizing or detailing a report. It will be triggered by interactive actions of the viewer. For each type of interactivity, the viewer provides a certain type of action:

- **Sorting** – when using column sorting;
- **DrillDown** – when using drill-down in reports;
- **Collapsing** – when using collapsing report blocks.

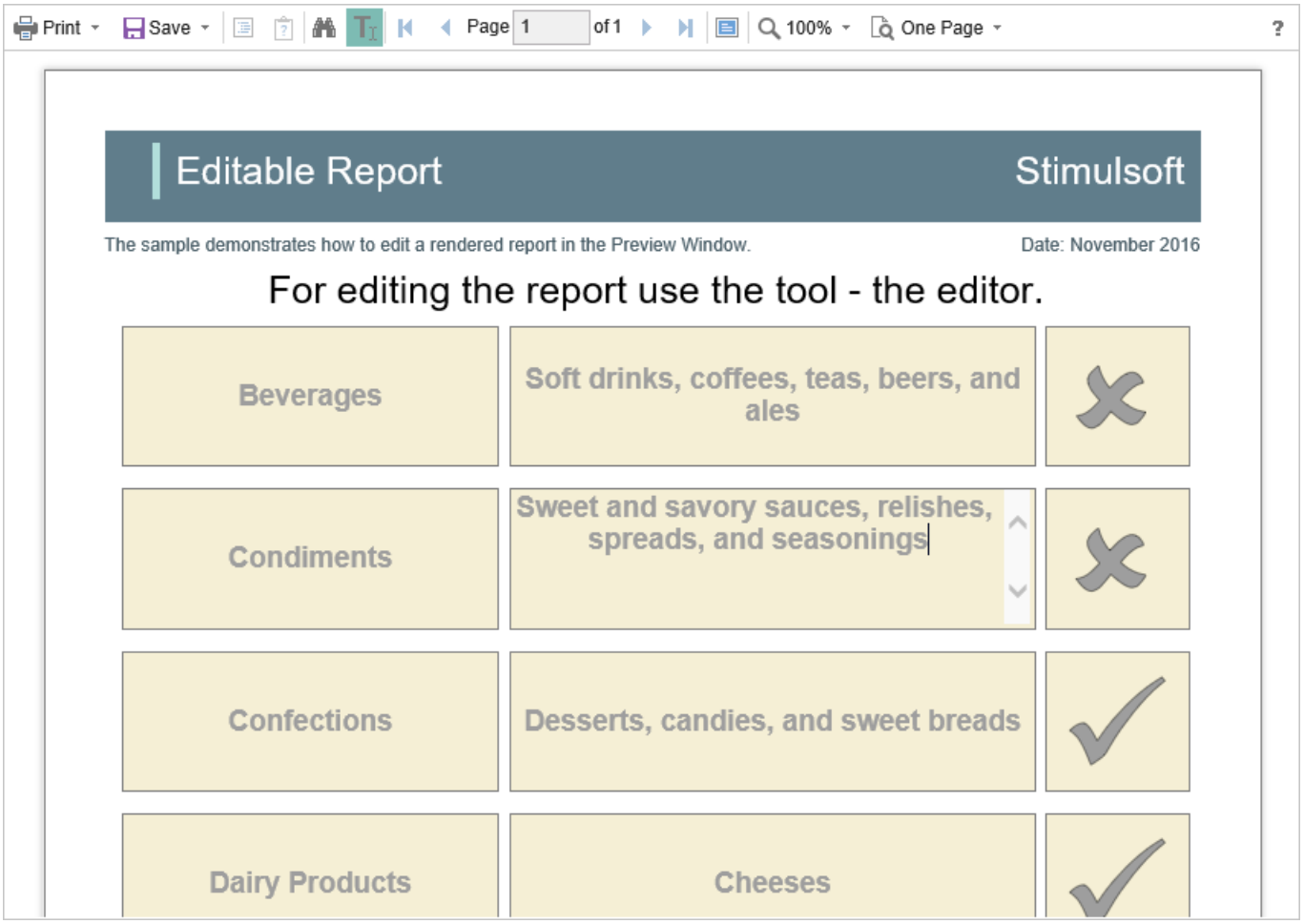
viewer.html

```
...
viewer.onInteraction = function (args) {
   switch (args.action) {
      case "Sorting":
        break;

      case "DrillDown":
```

```
        break;

      case "Collapsing":
        break;
    }
}
...
```

9.2.10 Editing Report

The **HTML5 Viewer** component has the ability to edit report items, such as text boxes and check boxes. In order the editing be possible, in the report template, you should mark the required components as editable. After displaying a report in the viewer, you need to click the corresponding button on the viewer panel to start editing. After editing, it is necessary to click the button once more, and all changes will be applied to the report.



For the report edit mode, no special settings of the viewer required.

Information

The edited settings will be applied when you print or export a report, and the original report remains unchanged. After restarting the viewer, all the values will be returned to the initial ones.

9.2.11 Sending Report by Email

Information

Please note that the Send Report by Email option is available only for reports, and not for dashboards.

The **HTML5 Viewer** component provides the ability to send reports by email. To activate this feature, you should set the **showSendEmailButton** property of the viewer to **true**, and add the **onEmailReport** event handler.

viewer.html

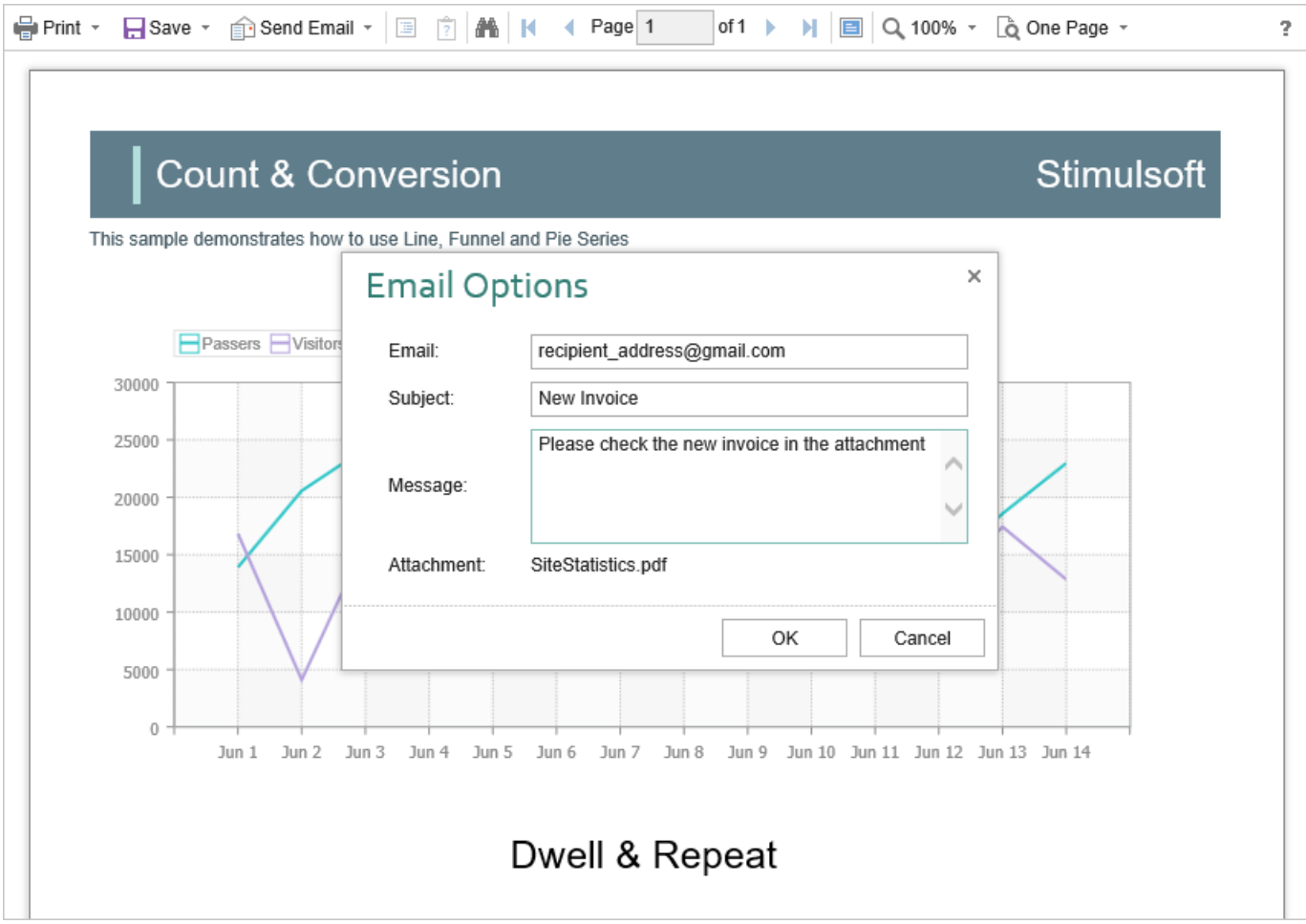
```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.toolbar.showSendEmailButton = true;
...
```

viewer.html

```
...
viewer.onEmailReport = function (args) {
   // args.settings -  send email form
   // args.settings.email  -  email adress
   // args.settings.subject  -  email subject
   // args.settings.message  -  email message

   // args.format  -  export format - PDF, HTML, HTML 5, Excel2007,
   Word2007, CSV
   // args.fileName - report file name (name of attachement)
   // args.data  -  byte array with exported report file
}
...
```

In the **onEmailReport** event you can find the type of the export, get the report, and also get the report export settings and change them, if necessary. Also in this event, you should set parameters of sending e-mail, such as sender address, server name, port number, user name and password - all these parameters will be used to send email.

When you send a report by email the menu to select the attachment format is displayed. This matches the menu to select an export format. After choosing the format, the dialog to insert send e-mail parameters such as email recipient, subject and message will pop-up.



After the confirmation of sending, the **onEmailReport** event described above will be triggered. You can check and correct the data entered in that form.

Information

Pure JavaScript does not have built-in e-mail methods, so the **StiViewer** component provides only an interface for sending email. To work with e-mails, you should use any server that supports work with e-mail. To the server side, the data for sending e-mails can be sent using an AJAX request or another convenient method.

The **HTML5 Viewer** component allows you to set default values for the send email form. The **defaultEmailAddress**, **defaultEmailSubject** and **defaultEmailMessage** properties can be used for this. By default, these properties are empty.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.toolbar.showSendEmailButton = true;

options.email.defaultEmailAddress = "recipient_address@gmail.com";
options.email.defaultEmailSubject = "New Invoice";
options.email.defaultEmailMessage = "Please check the new invoice in the
attachment";
...
```

9.2.12 Calling Designer from Viewer

The **HTML5 Viewer** component has the ability to call the report designer. The special **Design** button in the toolbar of the viewer (the button is disabled by default) should be used. To use this feature, you should set the **showDesignButton** property to **true**, and also to define the **onDesignReport** event handler.

viewer.html

```
...
var options = new Stimulsoft.Viewer.StiViewerOptions();
options.toolbar.showDesignButton = true;
...
```

viewer.html

```
...
viewer.onDesignReport = function (args) {
   window.open("https://www.stimulsoft.com?reportName=" +
   args.report.reportName);
}
...
```

Information

The viewer does not run the designer itself, it only calls the specified event and passes the previewed report as arguments. In the event, you can set redirect to the HTML page, on which the report designer is placed.

9.2.13 Viewer Events

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

In order to write interactive applications it is needed to respond to changes in the application. The event system which is represented in Stimulsoft Report Viewer can be used for this. The following events exist:

onPrepareVariables

Asynchronous event is called before filling in the variables in the report at the beginning of the report rendering. The event occurs immediately after execution `onPrepareVariables` event of the `StiReport` object. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **PrepareVariables**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.

async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

viewer.html

```
...
viewer.onPrepareVariables = (args, callback) => {
  args.variables[0].value = "Replace value";
}
...
```

onBeginProcessData

Asynchronous event is called before requesting the data for the report. The event occurs immediately after execution `onBeginProcessData` event of the `StiReport` object. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **BeginProcessData**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
command	a string ID for the current command. Can have values "TestConnection" and "ExecuteQuery" to call the command to test a connection and data acquisition respectively.
database	a string name of the current database.
connection	the name of the current connection to a data

	source, specified in report template.
headers	an array of the query headers.
withCredentials	an argument is used to provide the ability to set a cookies in the query.
connectionString	a connection string to the data source for a report.
dataSource	the name of the current data source, specified in the report template. It is set only for SQL data sources.
queryString	a string with the SQL query to the database for retrieving data. Used only with the command = "ExecuteQuery".
timeout	an argument is used to provide the ability to define the seconds of the query timeout.
parameters	a list of the query parameters.
escapeQueryParameters	a flag is used to provide the ability to use the escaped part of the request. By default, the value is set to **true**.
pathData	an argument is used to provide the ability to define a path to data file.
tryParseDateTime	a flag is trying to parse data as DateTime.
relationDirection	an argument is used to provide the ability to change the direction of the relation between data sources.
pathSchema	an argument is used to provide the ability to define a path to XSD files.
firstRowIsHeader	a flag is used to provide the ability to use a first row as data header in Excel data source.
collectionName	a string collection name of the OData data source.
separator	a string separator of the CSV data source.
dataType	an argument is used to provide the ability to set the data type of the GIS data source.

codePage	an argument is used to provide the ability to set a code page of the CSV or DBF data source.

viewer.html

```
...
//Replace connection string
viewer.onBeginProcessData = (args) => {
   if (args.database == "MySQL")
      args.connectionString = "new connection string";
}
...

//Add a some data
viewer.onBeginProcessData = (args, callback) => {
   if (args.database == "MySQL"){
      args.preventDefault = true;
      var result = {
         success: true,
         rows: [
            ["value1", 1, false],
            ["value2", 1, true],
            ["value3", 2, false]
         ],
         columns: [
            "Column1_name",
            "Column2_name",
            "Column3_name"
         ],
         types:[
            "string",
            "int",
            "boolean"
         ]
      }

      // https://github.com/stimulsoft/DataAdapters.JS/
      callback(result);
   }
}
...
```

onEndProcessData

Is called after retrieving data for the report. The event occurs immediately after execution `onEndProcessData` event of the `StiReport` object. The event handler argument “event” is an object with the next fields:

Name	Description

sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **EndProcessData**.
report	a report object **StiReport**.
command	a string ID for the current command. Can have values "TestConnection" and "ExecuteQuery" to call the command to test a connection and data acquisition respectively.
dataSource	the name of the current data source, specified in the report template. It is set only for SQL data sources.
connection	the name of the current connection to a data source, specified in report template.
database	a string name of the current database.
result	a result dataset in a specific JSON format. It has two collections – columns and rows, with descriptions of columns and rows of data sources.

viewer.html

```
...
viewer.onEndProcessData = (args) => {
   if (args.command == "ExecuteQuery" && args.dataSource == "Categories")
      args.result.rows.push(rowData) ;
      // https://github.com/stimulsoft/DataAdapters.JS/
}
...
```

onPrintReport

Asynchronous event is called before printing the report. This is not relevant when viewing dashboards. The event handler argument “event” is an object with the next fields:

Name	Description

sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **PrintReport**.
report	a report for saving.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
printAction	a sting method name for printing.

viewer.html

```
...
//Remove image before report print
viewer.onPrintReport = (args) => {
   var page = args.report.renderedPages.getByIndex(0);
   var image = page.components.getByName("Image1");
   if (image)
      page.components.remove(image);
}
...
```

onBeginExportReport

Asynchronous event is called before export but after applying the export options. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **BeginExportReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.

async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
action	the action, which initiated export event - **StiExportAction**.
settings	export report settings.
format	export report format. Can have the following values: › Stimulsoft.Report.StiExportFormat.Pdf, › Stimulsoft.Report.StiExportFormat.Excel2007, › Stimulsoft.Report.StiExportFormat.Word2007, › Stimulsoft.Report.StiExportFormat.Html, › Stimulsoft.Report.StiExportFormat.Html5, › Stimulsoft.Report.StiExportFormat.Document. › Stimulsoft.Report.StiExportFormat.Text, › Stimulsoft.Report.StiExportFormat.Csv, › Stimulsoft.Report.StiExportFormat.ImageSvg, › Stimulsoft.Report.StiExportFormat.Ppt2007, › Stimulsoft.Report.StiExportFormat.Odt, › Stimulsoft.Report.StiExportFormat.Ods.
formatName	name of a selected report export format corresponds to the name of the constants in format list.
fileName	an exporting file name.
openAfterExport	The flag indicates that a report will be exported in a new browser tab (the `true` value) or after export completed file saving will be invoked (the `false` value).

viewer.html

```
...
viewer.onBeginExportReport = function (args) {
  switch (event.format) {
    case Stimulsoft.Report.StiExportFormat.Html:
    args.settings.zoom = 2;  // Set zoom to 200%
```

```
      break;
    }
    console.log("exporting");
}
...
```

onEndExportReport

Asynchronous event is called after export report. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, it is set to **EndExportReport**.
fileName	an exporting file name.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
action	the action, which initiated export event - **StiExportAction**.
format	export report format. Can have the following values: › Stimulsoft.Report.StiExportFormat.Pdf, › Stimulsoft.Report.StiExportFormat.Excel2007, › Stimulsoft.Report.StiExportFormat.Word2007, › Stimulsoft.Report.StiExportFormat.Html, › Stimulsoft.Report.StiExportFormat.Html5, › Stimulsoft.Report.StiExportFormat.Document. › Stimulsoft.Report.StiExportFormat.Text,

	Stimulsoft.Report.StiExportFormat.Csv, Stimulsoft.Report.StiExportFormat.ImageSvg, Stimulsoft.Report.StiExportFormat.Ppt2007, Stimulsoft.Report.StiExportFormat.Odt, Stimulsoft.Report.StiExportFormat.Ods.
formatName	name of a selected report export format corresponds to the name of the constants in format list.
fileName	an exporting file name.
openAfterExport	The flag indicates that a report will be exported in a new browser tab (the `true` value) or after export completed file saving will be invoked (the `false` value).
data	export data to the string or byte array representation.

viewer.html

```
...
viewer.onEndExportReport = (args) => {
   args.fileName = "SampleFileName.txt";
}
...
```

onInteraction

Asynchronous event is called while interactive action of the viewer (dynamic sorting, collapsing, drill-down, applying of parameters) until processing values by the report generator. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, it is set to **Interaction**.
report	a report object **StiReport**.

preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
action	The identifier of the current interactive action can take the following values: • `Sorting` - This action happens when using sorting columns; • `DrillDown` - This action happens when using sorting columns; • `Collapsing` - This action happens when using of collapsing report blocks.

viewer.html

```
...
viewer.onInteraction = (args) => {
   if (args.action == "Variables")
      args.variables["Variable1"] = "New Value";
}
...
```

onEmailReport

It is called before sending the report by email. This is not relevant when viewing dashboards. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **EmailReport**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

report	a report object **StiReport**.
settings	export email settings.: › email - address of the recipient; › subject - email subject; › message - email message.
format	export report format. Can have the following values: › Stimulsoft.Report.StiExportFormat.Pdf, › Stimulsoft.Report.StiExportFormat.Excel2007, › Stimulsoft.Report.StiExportFormat.Word2007, › Stimulsoft.Report.StiExportFormat.Html, › Stimulsoft.Report.StiExportFormat.Html5, › Stimulsoft.Report.StiExportFormat.Document. › Stimulsoft.Report.StiExportFormat.Text, › Stimulsoft.Report.StiExportFormat.Csv, › Stimulsoft.Report.StiExportFormat.ImageSvg, › Stimulsoft.Report.StiExportFormat.Ppt2007, › Stimulsoft.Report.StiExportFormat.Odt, › Stimulsoft.Report.StiExportFormat.Ods.
formatName	name of a selected report export format corresponds to the name of the constants in format list.
fileName	an exporting file name.
data	export data to the string or byte array representation.

viewer.html

```
...
viewer.onEmailReport = (args, callback) => {
   args.async = true;

   var emailAddress = args.settings.email;
   var emailMessage = args.settings.message;
   var emailSubject = args.settings.subject;
   var emailAttachmentFileName = args.fileName;
   var emailAttachment = args.data;
   sendEmail(emailAddress, emailMessage, emailSubject,
   emailAttachmentFileName, emailAttachment);
```

```
   setTimeout(() => {
      callback();
   }, 5000);
}
...
```

onDesignReport

It is called by pressing the Design button. The event handler argument “event” is an object with the following fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event, by default, it is set to **DesignReport**.
report	a report for designing.

viewer.html

```
...
var viewerOptions = new Stimulsoft.Viewer.StiViewerOptions();
viewerOptions.toolbar.showDesignButton = true;
var viewer = new Stimulsoft.Viewer.StiViewer(viewerOptions, "StiViewer",
false);
viewer.renderHtml("content");

viewer.onDesignReport = (args) => {
   var viewerDiv = document.getElementById("content");
   viewerDiv.innerHTML = "";
   var designerOptions = new Stimulsoft.Designer.StiDesignerOptions();
   designerOptions.appearance.fullScreenMode = true;
   var designer = new Stimulsoft.Designer.StiDesigner(designerOptions,
   "StiDesigner", false);
   designer.renderHtml("content");

   designer.report = args.report;
}
...
```

onShowReport

Asynchronous event is called after a report is rendered before its displaying in the

viewer. The event handler argument “event” is an object with the following fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event, by default, it is set to **ShowReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

viewer.html

```
...
viewer.onShowReport = function (args) {
   console.log("showing");
}
...
```

onOpenReport

Asynchronous event that provides the ability to use custom method of opening templates. The event is called before the dialog box is opened and before report is assigned to the viewer. The event handler argument “event” is an object with the following fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event, by default, it is set to **OpenReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the

	event. By default, the value is set to **true**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

viewer.html

```
...
viewer.onOpenReport = (args) => {
   args.async = true;
   args.report = anotherReport;
   callback();
}
...
```

onOpenedReport

Asynchronous event that provides the ability to change of the report before it is assigned to the viewer. The event is called after report has been opened and before report is assigned to the viewer. The event handler argument “event” is an object with the following fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event, by default, it is set to **OpenedReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

viewer.html

```
...
viewer.onOpenedReport = (args) => {
   if (args.report.reportAuthor != "Stimulsoft") {
      args.preventDefault = true;
      window.alert("report.reportAuthor == " + args.report.reportAuthor);
   }
}
...
```

9.2.14 Viewer Settings

The **HTML5 Viewer** is configured using properties that are located in the **Stimulsoft.Viewer.StiViewerOptions** class. All properties are divided into groups. Some of the groups contain subgroups for ease of use. The following is an example of setting the properties of the viewer.

viewer.html

```
...
<script type="text/javascript">
   var report = new Stimulsoft.Report.StiReport();
   report.loadFile("SimpleList.mrt");

   var options = new Stimulsoft.Viewer.StiViewerOptions();

   options.width = "1000px";
   options.height = "1000px";

   options.appearance.theme =
   Stimulsoft.Viewer.StiViewerTheme.Office2022WhiteBlue;
   options.appearance.reportDisplayMode =
   Stimulsoft.Report.Export.StiHtmlExportMode.Div;
   options.appearance.scrollbarsMode = true;
   options.appearance.backgroundColor =
   Stimulsoft.System.Drawing.Color.dodgerBlue;
   options.appearance.showTooltips = false;

   options.toolbar.showPrintButton = false;
   options.toolbar.showDesignButton = false;
   options.toolbar.showAboutButton = false;

   options.exports.showExportToPdf = true;
   options.exports.ShowExportToWord2007 = true;

   var viewer = new Stimulsoft.Viewer.StiViewer(options);
   viewer.report = report;
</script>
...
```

Please note that all dashboard elements have their own save options and full-screen buttons for preview. There are no special options to control displaying them, but

they can be disabled through the properties of the element. The code below should be added after loading the report before passing it to the viewer.

Default.aspx.cs

```
...
var dbsElementInteraction =
report.getComponentByName("RegionMap1").dashboardInteraction;
  dbsElementInteraction.showFullScreenButton = false;
  dbsElementInteraction.showSaveButton = false;
...
```

Main settings (without groups)

Name	Description
`width`	Sets the width of the component in “px” or “%”.
`height`	Sets the height of the component in “px” or “%”. Works only in the mode if the `options.appearance.scrollbarsMode` property is set to `true`.

Appearance

Name	Description
`options.appearance.theme`	Specifies the theme of the viewers' layout. The list of available themes can be found in the `StiViewerTheme` enumeration. The default value is `Office2022WhiteBlue`.
`appearance.backgroundColor`	Sets the background color of the viewer. By default. it is set to `Stimulsoft.System.Drawing.Color.white`.
`appearance.pageBorderColor`	Sets the border color of the viewer.

	By default, it is set to `Stimulsoft.System.Drawing.Color.gray`.
`appearance.rightToLeft`	Sets the **Right to Left** mode for viewer controls. By default, the property is set to `false`.
`appearance.fullScreenMode`	Sets the full-screen display mode of the viewer. By default, the property is set to `false`.
`appearance.scrollbarsMode`	Sets the preview mode with scrollbars. By default, the property is set to `false`.
`appearance.openLinksWindow`	Sets the target window to open links contained in the report. By default, it is set to '`_blank`' (new window). It can have one of the standard values '`_blank`', '`_self`', '`_top`', as well as the name of the window or frame.
`appearance.showTooltips`	Enables displaying tips for the viewer controls when the mouse hovers over. By default, the property is set to `true`.
`appearance.showTooltipsHelp`	Sets a value which indicates that show or hide the help link in tooltips. By default, the property is set to `true`.
`appearance.showDialogsHelp`	Sets a value which indicates that show or hide the help button in dialogs. By default, the property is set to `true`.
`appearance.pageAlignment`	Sets the position of the report page in the viewer window. • `Stimulsoft.Viewer.StiContentAlignment.DefaultValue` - page alignment is determined from the template

	settings (default value); • `Stimulsoft.Viewer.StiContentAlignment.Left` – the page will be aligned left; • `Stimulsoft.Viewer.StiContentAlignment.Center` – the page will be centered; • `Stimulsoft.Viewer.StiContentAlignment.Right` – the page will be aligned right.
`appearance.showPageShadow`	Enables displaying shadow for report pages. By default, the property is set to `true`.
`appearance.bookmarksPrint`	Enables printing of report bookmarks (besides the report itself). By default, the property is set to `false`.
`appearance.bookmarksTreeWidth`	Sets the width of the bookmarks panel in pixels. By default, the width is 180 pixels.
`appearance.parametersPanelMaxHeight`	Sets the maximum height of the parameters bar in pixels. By default, the maximum height is 300 pixels.
`appearance.parametersPanelColumnsCount`	Sets the number of columns to display report parameters. By default, it is set to 2 columns.
`appearance.parametersPanelSortDataItems`	Gets or sets a value which indicates that variable items will be sorted. By default the property is set to `true`.
`appearance.parametersPanelDateFormat`	Sets the date and time format for variables of the corresponding type in the parameters panel. By default, the property is set to `String.empty`.
`appearance.reportDisplayMode`	Sets the export mode for displaying report pages. It can take one of the following values of the

	`reportDisplayMode` enumeration: • `FromReport` - the export mode of the report elements is defined from report template settings - Div or Table (default value); • `Table` – report elements are exported using HTML tables; • `Div` – report elements are exported using DIV markup.
`appearance.interfaceType`	Sets the type of interface used for the viewer. It can take one of the following values: • `Stimulsoft.Viewer.StiInterfaceType.Auto` – the viewer's interface is determined automatically depending of the device that is report is displayed on (default value); • `Stimulsoft.Viewer.StiInterfaceType.Mouse` – the standard interface with a mouse control will be used for all the screen types; • `Stimulsoft.Viewer.StiInterfaceType.Touch` – the Touch interface will be used to control the viewer. The interface design was optimized for the 'touchscreen' display types. The viewer interface elements have been increased in size to simplify the control of the viewer and to improve its usability; • `Stimulsoft.Viewer.StiInterfaceType.Mobile` – the Mobile interface will be used to control the viewer for all screen

	types. The Mobile interface was designed to control the viewer using the mobile smartphone display. This interface design was simplified and adapted to use on smartphones.
`appearance.allowMobileMode`	Enables or disables displaying a report or dashboard in the mobile mode. If the option is set to `false`, then the mobile view will not be used. If the option is set to `true`, the mobile view mode will be used when opening the viewer on mobile devices. By default, the option is set to `true`.
`appearance.chartRenderType`	Sets the chart displaying mode on the report page. • `Stimulsoft.Viewer.StiChartRenderType.AnimatedVector` – charts are displayed in the vector mode as an SVG object, the chart elements are displayed with animation (set by default). • `Stimulsoft.Viewer.StiChartRenderType.Vector` – charts are displayed in the vector mode as an SVG object;
`appearance.datePickerFirstDayOfWeek`	Sets the first day of week in the date picker. • `Stimulsoft.Viewer.StiFirstDayOfWeek.Auto` - Sets **Monday** or **Sunday** as the first day depending on the browser culture default value); • `Stimulsoft.Viewer.StiFirstDayOfWeek.Monday` - Sets **Monday** as the first day of the

	week; • `Stimulsoft.Viewer.StiFirstDayOfWeek.Sunday` - Sets **Sunday** as the first day of the week.
`appearance.datePickerIncludeCurrentDayForRanges`	Sets a value, which indicates that the current day will be included in the ranges of the date picker. By default the property is set to `false`.
`appearance.allowTouchZoom`	Sets a value which allows touch zoom in the viewer. By default the property is set to `true`.
`appearance.combineReportPages`	Sets a value which indicates that if a report contains several pages, then they will be combined in preview. By default the property is set to `false`.

Toolbar

Name	Description
`toolbar.visible`	Enables displaying the viewer toolbar. By default, the property is set to `true`.
`toolbar.displayMode`	Specifies the display mode of the toolbar of the viewer. It can take one of the following values of the `displayMode` enumeration: • `Simple` - all controls are located on the same control panel (default value); • `Separated` - the control panel is split into top and bottom panels.
`toolbar.backgroundColor`	Sets the color of the viewer toolbar. By default, the property is set to `Stimulsoft.System.Drawing.Co`

	lor.empty.
toolbar.borderColor	Sets the color of the borders of the Viewer toolbar. By default, the property is set to Stimulsoft.System.Drawing.Color.empty.
toolbar.fontColor	Sets the text color for the toolbar and the viewer menu. By default, the property is set to Stimulsoft.System.Drawing.Color.empty.
toolbar.fontFamily	Sets the font for the toolbar and the viewer menu. By default, the property is set to 'Arial'.
toolbar.alignment	Sets the alignment mode for the controls on the viewer toolbar. • Stimulsoft.Viewer.StiContentAlignment.Default – the alignment depends on the RightToLeft property (set by default). • Stimulsoft.Viewer.StiContentAlignment.Left – elements will be aligned left; • Stimulsoft.Viewer.StiContentAlignment.Center – elements will be centered; • Stimulsoft.Viewer.StiContentAlignment.Right – elements will be aligned right.
toolbar.showButtonCaptions	Enables text of the buttons on the toolbar of the viewer. By default, the property is set to true.
toolbar.showOpenButton	Enables displaying the **Open** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to true.

`toolbar.showPrintButton`	Enables showing the button - **Print** - on the toolbar of the viewer. By default, the property is set to `true`.
`toolbar.showSaveButton`	Enables displaying the **Save** button on the toolbar of the viewer when viewing reports or dashboards.. By default, the property is set to `true`.
`toolbar.showSendEmailButton`	Enables showing the button - **Send Email** - on the toolbar of the viewer. By default, the property is set to `false`. Also, you should add the onEmailReport event handler.
`toolbar.showFindButton`	Sets a visibility of the **Find** button in the toolbar of the viewer. By default, the property is set to `true`.
`toolbar.showBookmarksButton`	Enables showing the button - **Bookmarks** - on the toolbar of the viewer. By default, the property is set to `true`. If the button is hidden, the bookmarks panel will not be displayed even if there are bookmarks in the report.
`toolbar.showParametersButton`	Enables showing the button - **Parameters** - on the toolbar of the viewer. By default, the property is set to `true`. If the button is hidden, the parameters panel will not be displayed even if there are parameters in the report.
`toolbar.showResourcesButton`	Enables showing the button - **Resources** - on the toolbar of the viewer. By default, the property is set to `true`. If the button is hidden, the resources panel will not be displayed even if there are resources in the report.

`toolbar.showEditorButton`	Enables showing the button - **Editor** - on the toolbar of the viewer. By default, the property is set to `true`.
`toolbar.showFullScreenButton`	Enables displaying the **Full Screen** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to `true`.
`toolbar.showRefreshButton`	Sets a visibility of the **Refresh** button in the toolbar of the viewer. By default, the property is set to `true`.
`toolbar.showFirstPageButton`	Enables showing the button - **First Page** - on the toolbar of the viewer. By default, the property is set to `true`.
`toolbar.showPreviousPageButton`	Enables showing the button - **Previous Page** - on the toolbar of the viewer. By default, the property is set to `true`.
`toolbar.showCurrentPageControl`	Enables showing the current report page indicator. By default, the property is set to `true`.
`toolbar.showNextPageButton`	Enables showing the button - **Next Page** - on the toolbar of the viewer. By default, the property is set to `true`.
`toolbar.showLastPageButton`	Enables showing the button - **Last Page** - on the toolbar of the viewer. By default, the property is set to `true`.
`toolbar.showZoomButton`	Enables showing the Zoom button. By default, the property is set to `true`.
`toolbar.showViewModeButton`	Enables showing the button for selecting the display mode of report pages. By default, the property is set

	to `true`.
`toolbar.showDesignButton`	Enables displaying the **Design** button on the toolbar of the viewer when viewing reports or dashboards. By default, the property is set to `true`.
`toolbar.showAboutButton`	Enables showing the button - **About** - on the toolbar of the viewer. By default, the property is set to `true`.
`toolbar.showPinToolbarButton`	Sets a visibility of the **Pin** button in the toolbar of the viewer in mobile mode. By default, the property is set to `true`.
`toolbar.printDestination`	Sets the report printing mode. It can take one of the following values of the `StiPrintDestination` enumeration: • `Default` – a menu with a choice of printing modes will be displayed (default value); • `Pdf` – printing will be done to the PDF format; • `Direct` – printing will be done to the HTML format directly to the printer, the system print dialog will be displayed; • `PopupWindow` – printing will be done in the HTML format via the preview window of the report.
`toolbar.viewMode`	Sets the mode for displaying report pages. • `Stimulsoft.Viewer.StiWebViewMode.OnePage` – displays one page of the report selected in the toolbar of the viewer (the default value);

	• `Stimulsoft.Viewer.StiWebViewMode.Continuous` – displays all pages of the report; • `Stimulsoft.Viewer.StiWebViewMode.MultiplePages` – displays all report pages as a table.
`toolbar.zoom`	Sets the zoom for displaying report pages. The default setting is 100 percent. The values are from 10 to 500 percent. You can also set one of the following values: • `Stimulsoft.Viewer.StiZoomMode.PageWidth` – when the viewer runs, the zoom, necessary to display the report by the page width, will be set; • `Stimulsoft.Viewer.StiZoomMode.PageHeight` – when the viewer runs, the zoom, required to display the page height of the report, will be set.
`toolbar.menuAnimation`	Enables animation when the viewer menu shows/hides. By default, the property is set to `true`.
`toolbar.showMenuMode`	Sets the display mode of the viewer menu. It can take one of the following values of the `StiShowMenuMode` enumeration: • `Stimulsoft.Viewer.StiShowMenuMode.Click` – shows menu by mouse click (default value); • `Stimulsoft.Viewer.StiShowMenuMode.Hover` – shows menu by hovering the mouse cursor.

`toolbar.autoHide`	Sets a value which allows automatically hide the viewer toolbar in mobile mode. By default, the property is set to `false`.

Exports

Name	Description
`export.storeExportSettings`	Sets a value which allows store the export settings in the cookies. By default, the property is set to `true`.
`exports.showExportDialog`	Enables showing export options dialog box. If the property is set to `false`, the export will be done with the default settings. By default, the property is set to `true`.
`exports.showExportToDocument`	Enables the export menu item - **Document File**. By default, the property is set to `true`.
`exports.showExportToPdf`	Enables displaying the **Adobe PDF file** export menu item when viewing reports, and the **Adobe PDF** item when viewing dashboards. By default, the property is set to `true`.
`exports.showExportToXps`	Enables the export menu item - **XPS File**. By default, the property is set to `false`.
`exports.showExportToHtml`	Enables the export menu item - **HTML File**. By default, the property is set to `true`.
`exports.showExportToHtml5`	Enables the export menu item - **HTML5 File**. By default, the property is set to `true`.
`exports.showExportToWord2007`	Enables the export menu item - **Microsoft Word 2007/2010 File**. By

	default, the property is set to `true`.
`exports.showExportToExcel2007`	Enables displaying the **Microsoft Excel 2007/2010 File** export menu item when viewing reports, and the **Microsoft Excel** item when viewing dashboards. By default, the property is set to `true`.
`exports.showExportToCsv`	Enables the export menu item - **CSV**. By default, the property is set to `true`.
`exports.showExportToJson`	Enables the export menu item - **JSON**. By default, the property is set to `false`.
`exports.showExportToText`	Enables the export menu item - **Text File**. By default, the property is set to `true`.
`exports.showExportToOpenDocumentWriter`	Enables the export menu item - **OpenDocument Writer File**. By default, the property is set to `true`.
`exports.showExportToOpenDocumentCalc`	Enables the export menu item - **OpenDocument Calc File**. By default, the property is set to `true`.
`exports.showExportToPowerPoint`	Enables the export menu item - **Microsoft PowerPoint File**. By default, the property is set to `true`.
`exports.showExportToImageSvg`	Enables displaying the **Image** export menu item, with the ability to export the report to an SVG file. By default, the property is set to `true`.

Email

Name	Description
`email.showEmailDialog`	Enables displaying settings for

	sending the report via email. If the dialog box is disabled, the email will be sent with the settings set on the server side in the `onEmailReport` event. By default, the property is set to `true`.
`email.showExportDialog`	Enables displaying export options dialog box when sending email. If the property is set to `false`, the export will be done with the default settings. By default, the property is set to `true`.
`email.defaultEmailAddress`	Sets the default recipient email, i.e. the address to which the email with the attached report will be sent.
`email.defaultEmailSubject`	Sets the default email subject (header).
`email.defaultEmailMessage`	Sets the default email message (text).

9.3 HTML5 Designer

YouTube

Watch videos for working with JS HTML5 Designer. Subscribe to the Stimulsoft channel to watch new video lessons. Leave your questions and suggestions in the comments to the video.

Samples

See on GitHub examples of working with the JS HTML5 Designer component. All samples are separate projects, grouped into one solution.

The **HTML5 Designer** (**StiDesigner**) component is designed to create reports in the web browser. You do not need to install the .NET Framework, ActiveX components or

any special plug-ins on the client side. All that is needed is any modern Web browser.

With help of **HTML5 Designer** you can create, edit, save and preview reports on any computer with any operating system installed. Since the designer uses only HTML and JavaScript technologies, it can be run on devices where there is no Flash or Silverlight support - tablets, smartphones. Also, the designer supports the Touch interface, which is automatically enabled when using devices with a touch screen.

The **HTML5 Designer** component uses the **JavaScript** technology to perform all actions on reports, which allows you to get rid of reloading the entire page and speed up work.

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To use the **HTML5 Designer** in projects, you need to install the npm package of stimulsoft-reports-js:

```
npm install stimulsoft-reports-js
```

If this is not possible, you should add the following scripts to the project.

designer.html

```
...
<script src="scripts/stimulsoft.report.js"></script>
<script scr="scripts/stimulsoft.viewer.js"></script>
<script src="scripts/stimulsoft.designer.js"></script>

<!-- Stimulsoft Blockly editor for JS Designer -->
<script src="scripts/stimulsoft.blockly.editor.js"></script>
...
```

Install the npm package to use the HTML5 Viewer in a project to view reports and dashboards stimulsoft-dashboards-js:

```
npm install stimulsoft-dashboards-js
```

If this is not possible, you should add the following scripts to the project.

viewer.html

```
...
<script src="scripts/stimulsoft.report.js"></script>
<script src="scripts/stimulsoft.dashboards.js"></script>
<script scr="scripts/stimulsoft.viewer.js"></script>
<script src="scripts/stimulsoft.designer.js"></script>
...
```

Information

The script variants that you can use in your projects can be found in the Scripts of Reports.JS Package and Scripts of Dashboards.JS Package chapters.

9.3.1 How this Works

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **StiDesigner** component is developed using only HTML5 and JavaScript technology, and does not require a server for its work (it is necessary only for hosting project files). When you run the report viewer, the following actions occur:

› The JavaScript component adds designer interface code to the current HTML page;

› If a report object is assigned, the report will be uploaded to the designer page to edit it;

› Each action in the designer (for example, report preview, saving a report template, exporting a report, applying parameters, sorting and detailing a report) calls a specific JavaScript event in which you can perform the necessary manipulations with the report.

9.3.2 Editing Reports

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **StiDesigner** component is a tool for creating and editing reports. To start working with the report designer, you should add the scripts and styles required for the component to the HTML page of the project.

designer.html

```
...
<script src="scripts/stimulsoft.reports.js" type="text/javascript"></
script>
<script src="scripts/stimulsoft.dashboards.js"></script>
<script scr="scripts/stimulsoft.viewer.js" type="text/javascript"></
script>
<script src="scripts/stimulsoft.designer.js" type="text/javascript"></
script>
...
```

To edit report event scripts using the Blockly tool in the designer, additionally you should add an appropriate script file, which contains a visual part of editor:

designer.html

```
...
<script src="scripts/stimulsoft.reports.js" type="text/javascript"></
script>
<script src="scripts/stimulsoft.dashboards.js"></script>
<script scr="scripts/stimulsoft.viewer.js" type="text/javascript"></
script>
<script src="scripts/stimulsoft.designer.js" type="text/javascript"></
script>
<script src="scripts/stimulsoft.blockly.editor.js" type="text/
javascript"></script>
...
```

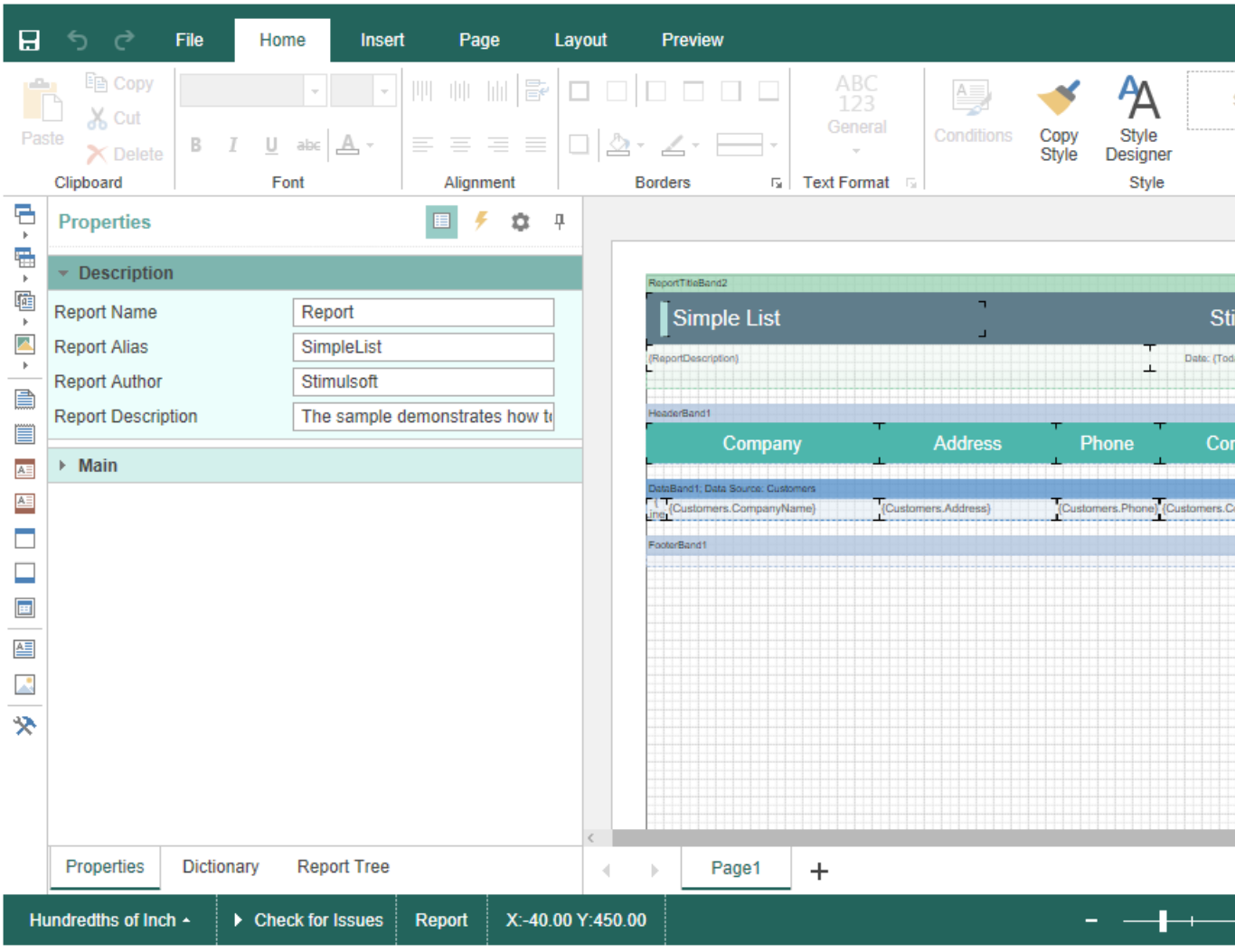
Information

Also, you need to add the scripts and styles for the report viewer, because this component is used to preview reports on the corresponding designer tab.

Then you should add the JavaScript code of report loading to the HTML page and assign the resulting object to the designer.

designer.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.loadFile("SimpleList.mrt");
//report.loadFile("Dashboard.mrt");

var designer = new Stimulsoft.Designer.StiDesigner(null, "Designer", false);
designer.report = report;
...
```



You can create a **StiDesigner** object using the **Stimulsoft.Designer.StiDesigner()** constructor, which can take the following optional arguments as input:

- **options** – is a set of options that can be found in the **Stimulsoft.Designer.StiDesignerOptions** class. All options are divided into categories. A detailed description of the categories and options can be found in the chapter Designer Settings.
- **designerId** – designer ID, used when deploying a component as a DOM object, the default value is **StiDesigner**.
- **renderAfterCreate** – defines the designer location. If it is set to **true**, the designer will be displayed in the same place in the DOM tree where the code to create an object is located. If it is set to **false**, the designer will be located at the place where the **renderHtml()** method is called. For example, this can be the initialization of the designer in the page header.

designer.html

```
...
<script type="text/javascript">
   var designer = new Stimulsoft.Designer.StiDesigner(null, "StiDesigner",
   false);
</script>
...
```

And the subsequent output of the designer in the current DIV element.

designer.html

```
...
<div>Page content</div>
<div>
   <script type="text/javascript">
      // Render the report designer in this place
      designer.renderHtml();
   </script>
</div>
...
```

As an argument of the **renderHtml(id)** designer output method, it is allowed to specify the element identifier of the HTML page in which the designer should be displayed.

designer.html

```
...
<script type="text/javascript" >
   var designer = new Stimulsoft.Designer.StiDesigner(null, "StiDesigner",
   false);
   designer.renderHtml("content");
</script>
...
```

The specified element must be located on the HTML page on which the report designer is used.

designer.html

```
...
<div id="content"></div>
...
```

Two methods of the **StiReport** object are used to load a report – **loadFile()** and **load()**. They are used as follows:

- loadFile(filePath) – loads a report from the **MRT** file which path is specified in the filePath;
- load(str) – loads a report from the string str, that contains **XML** or **JSON**;
- load(data) – loads a report from the **array data** of the number[] type;
- load(xml) – loads a report from the XML of the **XMLDocument** type;
- load(json) – loads a report from the **JS** object.

For example, use the code below to load a report from file:

designer.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.loadFile("SimpleList.mrt");
...
```

The **MRT** format of Stimulsoft Reports is the **JSON** based description of reports. You can use **MRT** files, created in other Stimulsoft Reports designers in the **JSON** based description. Use the code below to save a report to a string:

designer.html

```
...
```

```
var report = new Stimulsoft.Report.StiReport();
var jsonString = report.saveToJsonString();
...
```

Use the code below to load a report from this string:

designer.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.load(jsonString);
...
```

9.3.3 Creating New Report

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

No action is required to run the designer with a new (empty) report. When the component is loaded, the new report will be created automatically. If you need, you can create a new report object and preload the data for it, or perform some other necessary actions.

designer.html

```
...
var report = new Stimulsoft.Report.StiReport();
//var  newDashboard = Stimulsoft.Report.StiReport.createNewDashboard();

report.reportName = "MyNewReport";
//newDashboard.reportName = "MyDashboard";
...
```

A special **DataSet** object is used for data storage, which has a set of methods for loading data from various formats. The **regData()** method is used to connect the data to the report. The arguments of the method indicate the prepared **DataSet** object.

designer.html

```
...
report.regData(dataSet);
...
```

Data is added to a special collection of the report object, and is used to build it. Data structure synchronization is used to display the structure of the registered data in the report dictionary. The **synchronize()** method is used for this.

designer.html

```
...
report.dictionary.synchronize();
...
```

You can also create a new report using the main menu of the designer. The **onCreateReport** event is used to load data for a new report, or to perform some other necessary actions. This event will be called when you create a new empty report, or when you create a report using the wizard.

designer.html

```
...
designer.onCreateReport = function (args) {
   var dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
   dataSet.readJsonFile("Data/Demo.json");

   args.report.regData(dataSet.dataSetName, "", dataSet);
   args.report.dictionary.synchronize();
}
...
```

Also, you can pre-create a connection to the data source of the selected type, and add it to the collection of connections in the report template. This method is similar to creating a connection in the report dictionary using the designer interface.

designer.html

```
...
var report = new Stimulsoft.Report.StiReport();
//var  newDashboard = Stimulsoft.Report.StiReport.createNewDashboard();

var xmlDataBase = new Stimulsoft.Report.Dictionary.StiXmlDatabase("Demo",
```

```
"Demo.xsd", "Demo.xml");

report.loadFile("SimpleList.mrt");
//newDashboard.loadFile("Dashboard.mrt");
report.dictionary.databases.clear();
//newDashboard.dictionary.databases.clear();
report.dictionary.databases.add(xmlDataBase);
//newDashboard.dictionary.databases.add(xmlDataBase);
...
```

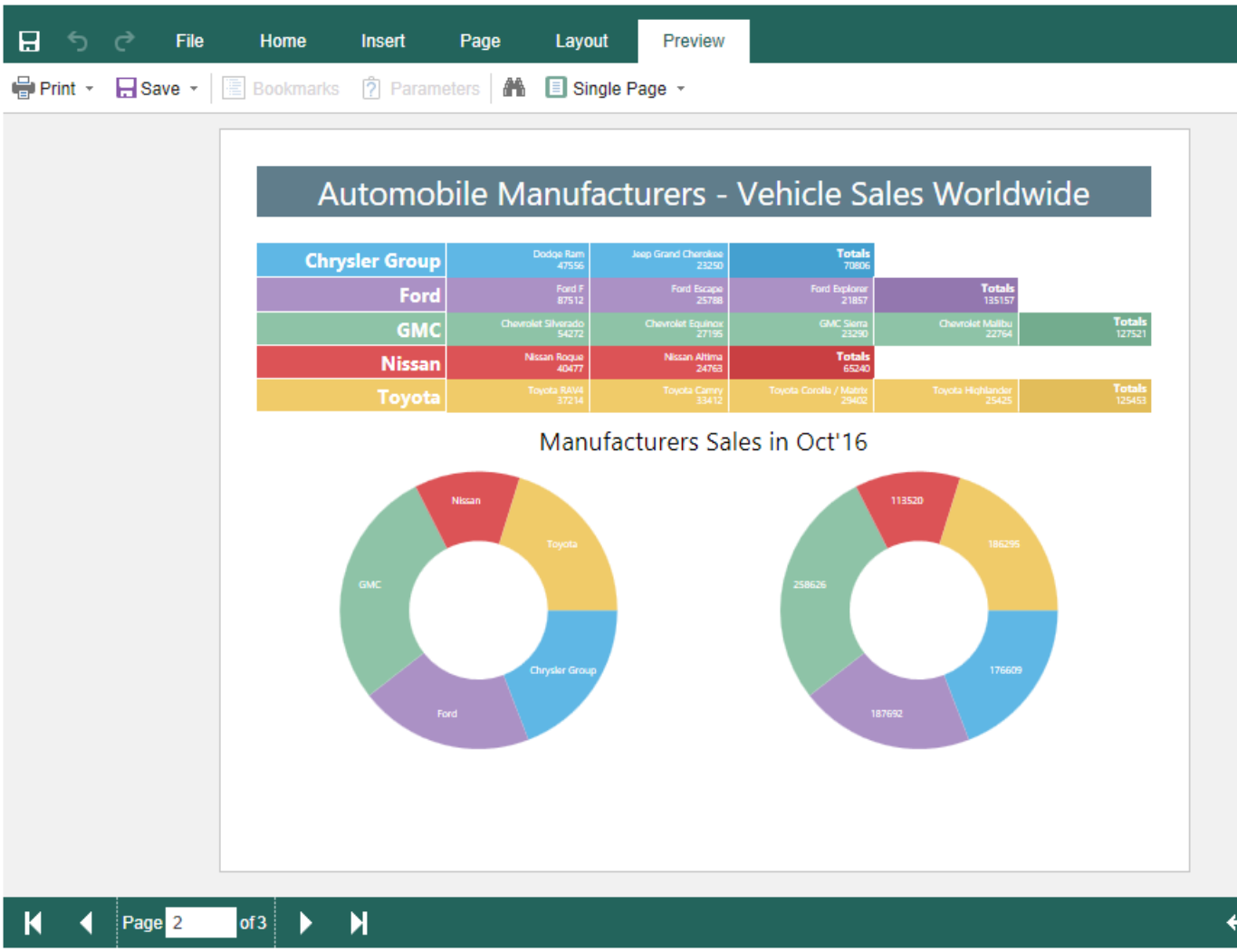
In this way, you can add the required number of data sources of different types.

9.3.4 Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component provides the ability to preview reports. To preview the report, just go to the appropriate tab in the designer window. The report template will be transferred to the server side, rendered and displayed in the embedded viewer.



Before previewing the report, it is possible to perform any necessary actions, for example, connect data for the report. To do this, you can use a special **onPreviewReport** event which will be called before previewing the report. The arguments of the event will contain a report to be previewed.

designer.html

```
...
designer.onPreviewReport = function (args) {
   var dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
   dataSet.readJsonFile("Data/Demo.json");

   args.report.regData(dataSet.dataSetName, "", dataSet);
}
...
```

9.3.5 Additional Features of Preview

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The preview window of the **HTML5 Designer** component has a fully functional interactive **HTML5 Viewer** that can print and export reports, supports working with report parameters, dynamic sorting, interactive reports, collapsing and etc. You do not need any additional settings for the report designer to use these features.

designer.html

```
...
designer.viewer.onBeginExportReport = function (event) {
   switch (event.format) {
      case Stimulsoft.Report.StiExportFormat.Html:
         event.settings.zoom = 2;  // Set zoom to 200%
         break;
   }
   console.log("exporting");
}
...
```

Information

If you do not need any of these additional options to preview the report (for example, exporting or printing a report), you can disable them using the appropriate properties of the **HTML5 Designer** component.

9.3.6 Saving Reports

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

The **HTML5 Designer** component has two ways of saving the report which are available in the main menu and in the main panel of the designer - **Save Report** and **Save As**. In turn, each of these ways has its own modes and settings.

Saving a report and dashboard on the server side

To save the edited report on the server side, you should set the **onSaveReport** special event which will be called when you select the **Save Report** menu item or click the **Save** button on the main panel of the designer.
An editable report will be passed in the arguments of the event. That report can be saved, for example, in a JSON string and then transferred to the server side.

designer.html

```
...
designer.onSaveReport = function (args) {
   args.preventDefault = false;
   var jsonReport = args.report.saveToJsonString();
}
...
```

Information

Additional details about the `onSaveReport` event handler argument are provided in the Designer Events.

By default, after saving the report, the designer continues working without displaying any messages. If necessary, after saving the report, it is possible to display a dialog box with an error or a text message. The special static function **showError()** is intended for this.

designer.html

```
...
designer.onSaveReport = function (args) {
   args.preventDefault = false;
   Stimulsoft.System.StiError.showError("Some message after saving", true);
}
```

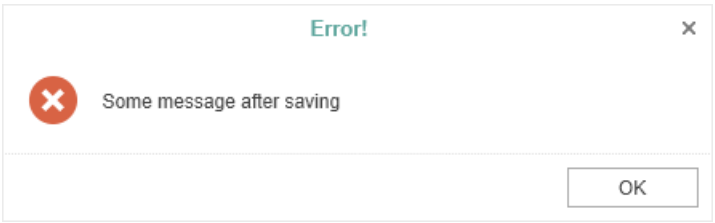
```
...
```

You can get a report name from the designer save dialog or an original report name.

designer.html

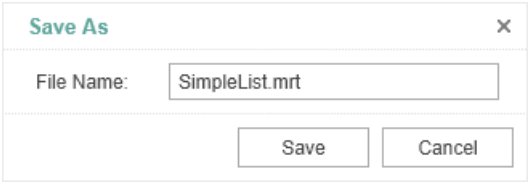
```
...
var designer = new Stimulsoft.Designer.StiDesigner(designerOptions,
"StiDesigner", false);
designer.renderHtml("content");

designer.onSaveReport = function (args) {

   //a flag to prevent further processing of the event
   args.preventDefault = false;

   //Report name from the designer save dialog
   var reportName = args.fileName;

   //Original report name from properties
   var reportName = args.report.reportName;
}
...
```

The function takes the error text and a flag as arguments that define the type of the window. The text can contain either a save error message or a warning, or any other message. If **true** is set as the second argument of the function, then an error window will be displayed. If **false** is set, the pop-up window with the error message will be displayed.



Saving reports and dashboards on the client side

No additional designer settings are required to save the edited report on the client side as a file. It is enough to click the **Save As** main menu item. The dialog box will be displayed. In this dialog you can change the name of the report file. The file will be saved to the local disk of the computer.



The **HTML5 Designer** component provides the ability to change the behavior of the specified save option. The special **onSaveReportAs** designer event is used for this. If you use this event, the report will be saved on the server side. It works similar to the **onSaveReport** event.

designer.html

```
...
designer.onSaveAsReport = function (args) {
   args.report.repotName = "Report";

   // Save the report template
   var jsonReport = args.report.saveToJsonString();
}
...
```

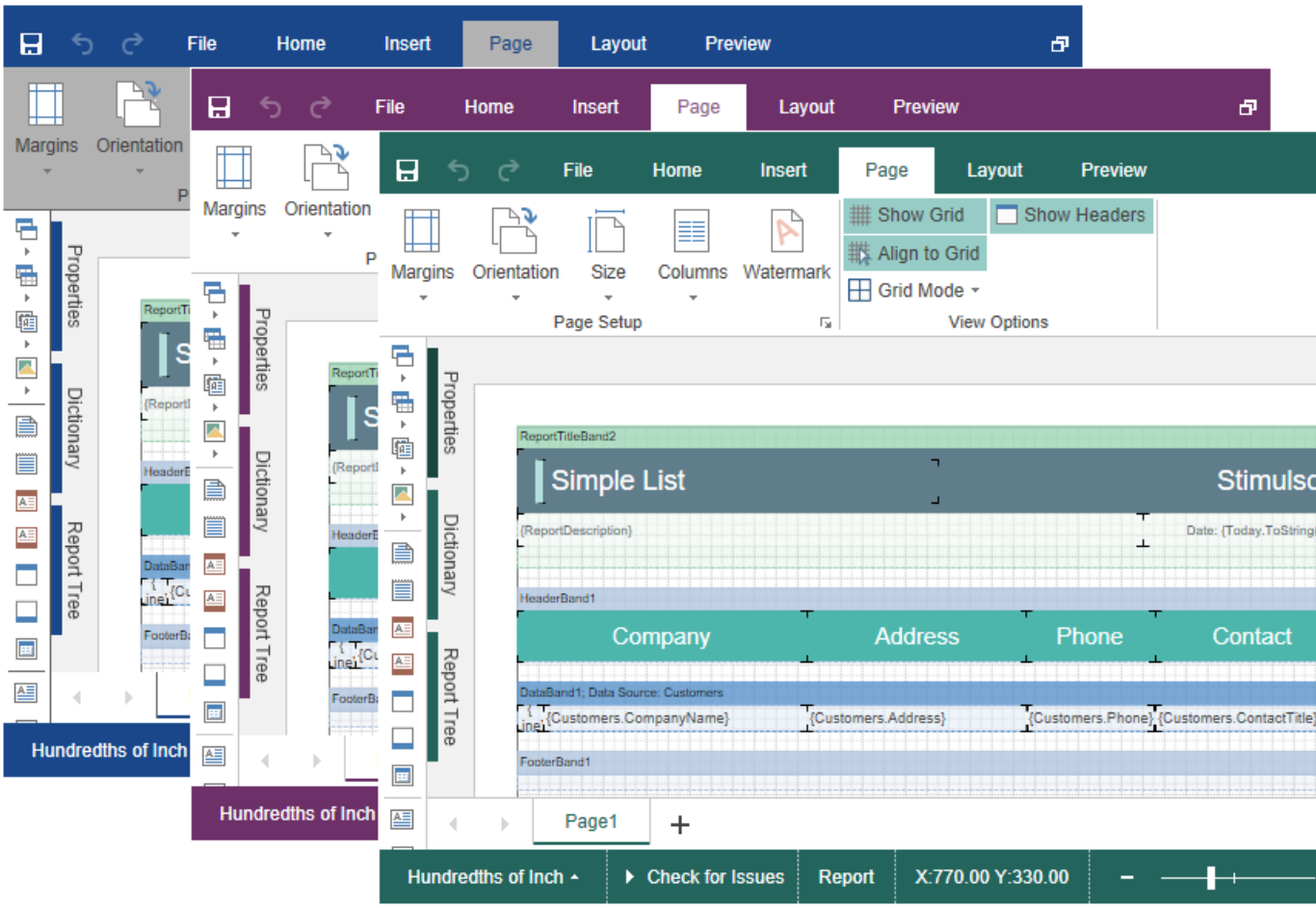
9.3.7 Using Themes

You can change the appearance of visual controls in the **HTML5 Designer** component. You can use the theme component option or the **setTheme()** method for this.

designer.html

```
...
var options = new Stimulsoft.Designer.StiDesignerOptions();
options.appearance.theme =
Stimulsoft.Designer.StiDesignerTheme.Office2022WhiteBlue;
...
designer.setTheme(Stimulsoft.Designer.StiDesignerTheme.Office2022WhiteBlue
);
...
```

There are currently **2 themes** available with different color accents. As a result, **more than 50** variants of the appearance are available. This allows you to customize the appearance of the designer for almost any design of the Web project.



9.3.8 Designer Events

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

To write interactive applications need to respond to changes in the application. The event system is used for this. It is represented in Stimulsoft Report Designer with the following events:

onPrepareVariables

Asynchronous event is called before filling in the variables in the report at the

beginning of the report rendering. The event occurs immediately after execution `onPrepareVariables` event of the `StiReport` object. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **PrepareVariables**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

designer.html

```
...
designer.onPrepareVariables = (args, callback) => {
   args.variables[0].value = "Replace value";
}
...
```

onBeginProcessData

Asynchronous event is called before requesting the data for the report. The event occurs immediately after execution `onBeginProcessData` event of the `StiReport` object. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **BeginProcessData**.

report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
command	a string ID for the current command. Can have values "TestConnection" and "ExecuteQuery" to call the command to test a connection and data acquisition respectively.
database	a string name of the current database.
connection	the name of the current connection to a data source, specified in report template.
headers	an array of the query headers.
withCredentials	an argument is used to provide the ability to set a cookies in the query.
connectionString	a connection string to the data source for a report.
dataSource	the name of the current data source, specified in the report template. It is set only for SQL data sources.
queryString	a string with the SQL query to the database for retrieving data. Used only with the command = "ExecuteQuery".
timeout	an argument is used to provide the ability to define the seconds of the query timeout.
parameters	a list of the query parameters.
escapeQueryParameters	a flag is used to provide the ability to use the escaped part of the request. By default, the value is set to **true**.
pathData	an argument is used to provide the ability to define a path to data file.
tryParseDateTime	a flag is trying to parse data as DateTime.

relationDirection	an argument is used to provide the ability to change the direction of the relation between data sources.
pathSchema	an argument is used to provide the ability to define a path to XSD files.
firstRowIsHeader	a flag is used to provide the ability to use a first row as data header in Excel data source.
collectionName	a string collection name of the OData data source.
separator	a string separator of the CSV data source.
dataType	an argument is used to provide the ability to set the data type of the GIS data source.
codePage	an argument is used to provide the ability to set a code page of the CSV or DBF data source.

designer.html

```
...
//Replace connection string
designer.onBeginProcessData = (args) => {
   if (args.database == "MySQL")
      args.connectionString = "new connection string";
}
...

//Add a some data
designer.onBeginProcessData = (args, callback) => {
   if (args.database == "MySQL"){
      args.preventDefault = true;
      var result = {
         success: true,
         rows: [
            ["value1", 1, false],
            ["value2", 1, true],
            ["value3", 2, false]
         ],
         columns: [
            "Column1_name",
            "Column2_name",
            "Column3_name"
         ],
         types:[
            "string",
            "int",
            "boolean"
         ]
      }
```

```
    // https://github.com/stimulsoft/DataAdapters.JS/
    callback(result);
  }
}
...
```

onEndProcessData

Is called after retrieving data for the report. The event occurs immediately after execution `onEndProcessData` event of the `StiReport` object. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **EndProcessData**.
report	a report object **StiReport**.
command	a string ID for the current command. Can have values "TestConnection" and "ExecuteQuery" to call the command to test a connection and data acquisition respectively.
dataSource	the name of the current data source, specified in the report template. It is set only for SQL data sources.
connection	the name of the current connection to a data source, specified in report template.
database	a string name of the current database.
result	a result dataset in a specific JSON format. It has two collections – columns and rows, with descriptions of columns and rows of data sources.

designer.html

```
...
```

```
designer.onEndProcessData = (args) => {
   if (args.command == "ExecuteQuery" && args.dataSource == "Categories")
      args.result.rows.push(rowData) ;
      // https://github.com/stimulsoft/DataAdapters.JS/
}
...
```

onCreateReport

Asynchronous event Is called after a new report is created. The event handler argument “event” it is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **CreateReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. The default value is **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
isWizardUsed	The flag indicates that a new report is created with the help of the master (the `true` value) or a blank report is created (the `false` value).

designer.html

```
...
designer.onCreateReport = function (args) {
   var report = args.report;

   var database = new
   Stimulsoft.Report.Dictionary.StiJsonDatabase("DemoData", "http://
   localhost/Demo.json");
   report.dictionary.databases.add(database);
   report.dictionary.synchronize();
}
...
```

onOpenReport

Asynchronous event is called before user click button for opening a report. The event handler argument event is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **OpenReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. The default value is **true**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

designer.html

```
...
//Call custom callback() function for loading template to designer
designer.onOpenReport = (args, callback) => {
   args.async = true;
   args.report = anotherReport;
   callback();
}
...
```

onOpenedReport

Asynchronous event is called before user click button for opening a report before it is assigned to the designer. The event handler argument event is an object with the next fields:

Name	Description

sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **OpenedReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. The default value is **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

designer.html

```
...
//Add image to report resource when it has been opening
designer.onOpenedReport = (args, callback) => {
   args.async = true;
   var xhr = new XMLHttpRequest();
   xhr.open('GET', "Url to image");
   xhr.onload = function () {
      var imageData = xhr.response;

      var resource = new
      Stimulsoft.Report.Dictionary.StiResource("ImageName");
      resource.content = imageData;
      args.report.dictionary.resources.add(resource);
      callback();
   };
   xhr.send();
}
...
```

onAssignedReport

The event is called after the report is assigned to the designer. The event handler argument event is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.

event	a string ID for the current event. By default, the value is set to **AssignedReport**.
report	a report object **StiReport**.

designer.html

```
...
designer.onAssignedReport = (args) => {
   console.log("The report was assigned to the designer")
}
...
```

onSaveReport

Asynchronous event is called before saving the report. The event handler argument event is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **SaveReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. The default value is **true**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
fileName	a saving file name.
autoSave	a flag is used to use autosave mode. The default value is **false**.

designer.html

```
...
//Remove report resources before saving
designer.onSaveReport = (args, callback) => {
```

```
    var report = args.report.clone();
    report.dictionary.resources.clear();
    args.report = report;
}
...
```

onSaveAsReport

Asynchronous event is called before saving the report if user click **Save As** button. The event handler argument event is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **SaveAsReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. The default value is **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
fileName	a saving file name.
autoSave	a flag is used to use autosave mode. The default value is **false**.

designer.html

```
...
//Stop and redefinition the save method
designer.onSaveAsReport = (args, callback) => {
    args.preventDefault = true;
    var jsonString = args.report.saveToJsonString();
    // save report
}
...
```

onPreviewReport

Asynchronous event is called when going to the report preview tab. The event handler argument event is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default it is **PreviewReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. The default value is **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
viewer	a viewer object **StiViewer**.

designer.html

```
...
designer.onPreviewReport = function (args) {
   var dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
   dataSet.readJsonFile("Data/Demo.json");

   args.report.regData(dataSet.dataSetName, "", dataSet);
}
...
```

onCloseReport

Asynchronous event is called after a report is closed, before the report has been unassigned from the report designer. The event handler argument “event” it is an object with the next fields:

Name	Description

sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **CloseReport**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. The default value is **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

designer.html

```
...
designer.onCloseReport = function (args) {
   console.log("The report was closed")
}
...
```

onExit

Is called before closing the designer. The event handler argument event is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default it is **Exit**.

designer.html

```
...
designer.onExit = function (args) {
   console.log(args.event);
}
...
```

9.3.9 Customizations in Designer

You can add custom controls to the report designer. In this chapter you may find several examples of customizing the report designer.

Adding a button to the **Home** tab.

designer.html

```
...
var designer = new Stimulsoft.Designer.StiDesigner(designerOptions,
"StiDesigner", false);
designer.renderHtml("content");

//Example add custom button to home toolbar panel
var homePanel = designer.jsObject.options.homePanel;

//Add buttons group to insert panel. Parameters of GroupBlock(groupName,
groupText) method.
var buttonsGroup = designer.jsObject.GroupBlock("buttonsGroup1",
"Group1");
var buttonsGroupTable = designer.jsObject.GroupBlockInnerTable();
buttonsGroup.container.appendChild(buttonsGroupTable);

//Add big button to buttons group. Parameters of BigButton(name,
groupName, caption, imageName, toolTip, arrow, styles) method.
var customBigButton = designer.jsObject.BigButton("customButton1", null,
"Custom Button", " ", "Tooltip for customButton1");
customBigButton.image.src = "https://www.stimulsoft.com/images/blocks/
ultimate-buttons/logo.png";

buttonsGroupTable.addCell(customBigButton).style.padding = "2px";

//customBigButton onclick event
customBigButton.action = function () {
   alert("customButton was pressed!");
}

//Add buttonsGroup and separator to customPanel
homePanel.firstChild.addCell(buttonsGroup);
homePanel.firstChild.addCell(designer.jsObject.GroupBlockSeparator());
...
```

Also, you can add a button to the top panel, near the **File** menu.

designer.html

```
...
var designer = new Stimulsoft.Designer.StiDesigner(designerOptions,
"StiDesigner", false);
designer.renderHtml("content");
```

```
var toolBarRow = designer.jsObject.options.toolBar.firstChild.tr[0];

var customButton = designer.jsObject.ToolButton("customButton1", "Custom
Button");

var buttonCell = document.createElement("td");
buttonCell.className = "stiDesignerToolButtonCell";
buttonCell.appendChild(customButton);

//For example insert button to position 3
toolBarRow.insertBefore(buttonCell, toolBarRow.childNodes[3]);

customButton.action = function () {
   alert("Button clicked!");
}
...
```

Below is an example of adding a custom panel in the report designer.

designer.html

```
...
var designer = new Stimulsoft.Designer.StiDesigner(designerOptions,
"StiDesigner", false);
designer.renderHtml("content");

var propertiesPanel = designer.jsObject.options.propertiesPanel;

var customPanel = document.createElement("div");
customPanel.jsObject = designer.jsObject;
customPanel.className = "stiDesignerPropertiesPanelInnerContent";
customPanel.style.top = "35px";
customPanel.style.display = "none";
customPanel.style.background = "gray";

propertiesPanel.containers["Custom"] = customPanel;
propertiesPanel.appendChild(customPanel);

var footerTable = propertiesPanel.footer.firstChild;

var tabButton = designer.jsObject.TabButton("CustomTabButton",
"PropertiesGridTabs", "Custom");
tabButton.style.margin = "0 0 0 3px";

tabButton.action = function () {
   if (!this.isSelected) propertiesPanel.showContainer("Custom");
}

designer.jsObject.loc.Panels.Custom = "Custom Panel Name";
...
```

9.3.10 Designer Settings

The HTML5 Designer is configured using properties that are located in the **Stimulsoft.Designer.StiDesignerOptions** class. All properties are split into groups. Some of the groups contain subgroups for ease of use. The following is an example of setting the properties of the viewer.

designer.html

```
...
<script type="text/javascript">
   var report = new Stimulsoft.Report.StiReport();
   report.loadFile("SimpleList.mrt");

   var options = new Stimulsoft.Designer.StiDesignerOptions();
   options.appearance.theme =
   Stimulsoft.Designer.StiDesignerTheme.Office2022WhiteBlue;
   options.viewerOptions.appearance.reportDisplayMode =
   Stimulsoft.Report.Export.StiHtmlExportMode.Auto;
   options.toolbar.showFileMenuExit = false;
   options.toolbar.showFileMenuOptions = false;
   options.bands.showChildBand = false;
   options.components.showPanel = false;
   options.appearance.showReportTree = false;
   options.appearance.showTooltips = false;

   var designer = new Stimulsoft.Designer.StiDesigner(options);
   designer.report = report;
</script>
...
```

Main settings (without groups)

Name	Description
Width	Sets the width of the component in “px” or “%”.
Height	Sets the height of the component in “px” or “%”.

Appearance

Name	Description
theme	Specifies the theme of the designer

	layout. The list of available themes can be found in the `StiDesignerTheme` enumeration. The default value is `Office2022WhiteBlue`.
`defaultUnit`	Sets the units for the size of the report and all its components. • `Stimulsoft.Report.StiReportUnitType.Centimeters` (default value); • `Stimulsoft.Report.StiReportUnitType.HundredthsOfInch;` • `Stimulsoft.Report.StiReportUnitType.Inches;` • `Stimulsoft.Report.StiReportUnitType.Millimeters.`
`zoom`	Sets the zoom for displaying report pages. The default setting is 100 percent. It can take one of the following values of the `StiZoomMode` enumeration: • `PageWidth` – when the designer runs, the zoom, necessary to display the report by the page width, will be set; • `PageHeight` – when the designer runs, the zoom, required to display the page height of the report, will be set.
`interfaceType`	Sets the type of interface used for the designer. It can take one of the following values: • `Stimulsoft.Designer.StiI`

	nterfaceType.Auto – the interface type of the designer will be selected automatically depending on the device used (default value); • Stimulsoft.Designer.StiInterfaceType.Mouse – forced use of the interface to control the designer with the mouse device; • Stimulsoft.Designer.StiInterfaceType.Touch – forced use of the touch interface to control the designer via the touch screen (mobile devices), also in this mode, the interface elements are increased.
showAnimation	Enables animation for various elements of the designer interface. By default, the property is set to true.
showSaveDialog	Enables displaying the dialog to insert a report name when it is saved. The name of the report will be transferred in the parameters of the report designer. By default, the property is set to true.
showTooltips	Enables displaying tooltips for designer controls when the mouse hovers over. By default, the property is set to true.
showTooltipsHelp	Enables displaying links to online documentation in tooltips for designer controls. By default, the property is set to true.
showDialogsHelp	Sets a value which indicates that show or hide the help button in

	dialogs. By default, the property is set to `true`.
`fullScreenMode`	Sets the full screen display mode of the designer. If the property is set to `true`, the values of the width and height properties are ignored. By default, the property is set to `false`.
`maximizeAfterCreating`	Sets a value which indicates that the designer will be maximized after creation. By default, the property is set to `false`.
`showLocalization`	Sets a visibility of the localization control of the designer. By default, the property is set to `true`.
`allowChangeWindowTitle`	Allows using a title of the browser window to display the file name of the edited report. By default, the property is set to `true`.
`showPropertiesGrid`	Enables displaying the Property panel of the report designer. By default, the property is set to `true`.
`showReportTree`	Enables displaying the tree of report components. By default, the property is set to `true`.
`propertiesGridPosition`	Sets **Left** or **Right** position of the properties grid in the designer.
`showSystemFonts`	Sets a visibility of the system fonts in the fonts list. By default, the property is set to `true`.
`datePickerFirstDayOfWeek`	Sets the first day of week in the date picker. • `Stimulsoft.Designer.StiFirstDayOfWeek.Auto` - Sets **Monday** or **Sunday** as the first day depending on the browser

	culture (default value); • `Stimulsoft.Designer.StiFirstDayOfWeek.Monday` - Sets **Monday** as the first day of the week. • `Stimulsoft.Designer.StiFirstDayOfWeek.Sunday` - Sets **Sunday** as the first day of the week.
`formatForDateControls`	This feature allows you to customize the format for date controls. By default, the current option does not have a specified value, and the date format is determined based on the browser's locale.
`wizardTypeRunningAfterLoad`	Calls the Report wizard after starting the report designer. It may have one of the following `StiWizardType` enumeration values: • `None` - runs the report designer without running the report wizard (default value); • `StandardReport` - runs the Standard wizard; • `MasterDetailReport` - runs the Master-Detail wizard; • `LabelReport` - runs the Label wizard; • `InvoicesReport` - runs the Invoice wizard; • `OrdersReport` - runs the Order wizard; • `QuotationReport` - runs the Quote wizard.

Toolbar

Name	Description
`visible`	It enables the display of toolbar in the report designer. The property has the `true` value by default.
`showPreviewButton`	It enables or disables the display of the **Preview** button in the designer toolbar. The property has the `true` value by default.
`showSaveButton`	It enables the display of the **Save** button in the designer toolbar. The property has the `true` value by default.
`showAboutButton`	It enables the display of the **About** button in the designer toolbar. The property has the `false` value by default.
`showFileMenu`	It enables the display of the main menu of the report designer. The property has the `true` value by default.
`showFileMenuNew`	It enables the display of the **New** main menu item. The property has the `true` value by default.
`showFileMenuOpen`	It enables the display of the **Open** main menu item. The property has the `true` value by default.
`showFileMenuSave`	It enables the display of the **Save** main menu item. The property has the `true` value by default.
`showFileMenuSaveAs`	It enables the display of the **Save as** main menu item. The property has the `true` value by default.

showFileMenuClose	It enables the display of the **Close** main menu item. The property has the `true` value by default.
showFileMenuExit	It enables the display of the **Exit** main menu item. The property has the `false` value by default.
showFileMenuReportSetup	It enables the display of the **Report Setup** main menu item. The property has the `true` value by default.
showFileMenuOptions	It enables the display of the **Options** main menu item. The property has the `true` value by default.
showFileMenuInfo	It enables the display of the **Info** main menu item. The property has the `true` value by default.
showFileMenuAbout	It enables the display of the **About** main menu item. The property has the `true` value by default.
showFileMenuNewReport	It enables or disables the display of the **New Page** main menu item. The property has the `true` value by default.
showFileMenuNewDashboard	It enables or disables the display of the **New Dashboard** main menu item. The property has the `true` value by default.
showSetupToolboxButton	It enables or disables the display of the report components side panel settings invoke button. The property has the `true` value by default.
showNewPageButton	It enables or disables the display of the **New Page** button in the toolbar. The property has the `true` value by default.
showNewDashboardButton	It enables or disables the display of

	the **New Dashboard** button in the toolbar. The property has the `true` value by default.

Bands

Name	Description
`showReportTitleBand`	It enables the display of the **Report Title** band in the designer components insert menu. The property has the `true` value by default.
`showReportSummaryBand`	It enables the display of the **Report Summary** band in the designer components insert menu. The property has the `true` value by default.
`showPageHeaderBand`	It enables the display of the **Page Header** band in the designer components insert menu. The property has the `true` value by default.
`showPageFooterBand`	It enables the display of the **Page Footer** band in the designer components insert menu. The property has the `true` value by default.
`showGroupHeaderBand`	It enables the display of the **Group Header** band in the designer components insert menu. The property has the `true` value by default.
`showGroupFooterBand`	It enables the display of the **Group Footer** band in the designer components insert menu. The

	property has the `true` value by default.
`showHeaderBand`	It enables the display of the **Header** band in the designer components insert menu. The property has the `true` value by default.
`showFooterBand`	It enables the display of the **Footer** band in the designer components insert menu. The property has the `true` value by default.
`showColumnHeaderBand`	It enables the display of the **Column Header** band in the designer components insert menu. The property has the `true` value by default.
`showColumnFooterBand`	It enables the display of the **Column Footer** band in the designer components insert menu. The property has the `true` value by default.
`showDataBand`	It enables the display of the **Data** band in the designer components insert menu. The property has the `true` value by default.
`showHierarchicalBand`	It enables the display of the **Hierarchical** band in the designer components insert menu. The property has the `true` value by default.
`showChildBand`	It enables the display of the **Child** band in the designer components insert menu. The property has the `true` value by default.
`showEmptyBand`	It enables the display of the **Empty** band in the designer components insert menu. The property has the

	`true` value by default.
`showOverlayBand`	It enables the display of the **Overlay** band in the designer components insert menu. The property has the `true` value by default.
`showTable`	It enables the display of the **Table** component in the designer components insert menu. The property has the `true` value by default.
`showTableOfContents`	It enables the display of the **Table of Contents** band in the designer components insert menu. The property has the `true` value by default.

Cross-Bands

Name	Description
`showCrossTab`	It enables the display of the **Cross-Tab** component in the designer components insert menu. The property has the `true` value by default.
`showCrossGroupHeaderBand`	It enables the display of the **Cross-Group Header** band in the designer components insert menu. The property has the `true` value by default.
`showCrossGroupFooterBand`	It enables the display of the **Cross-Group Footer** band in the designer components insert menu. The property has the `true` value by default.
`showCrossHeaderBand`	It enables the display of the **Cross-**

	Header band in the designer components insert menu. The property has the `true` value by default.
`showCrossFooterBand`	It enables the display of the **Cross-Footer** band in the designer components insert menu. The property has the `true` value by default.
`showCrossDataBand`	It enables the display of the **Cross-Data** band in the designer components insert menu. The property has the `true` value by default.

dashboardElements

Name	Description
`showTableElement`	It enables the display of the **Table** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showCardsElement`	It enables the display of the **Cards** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showChartElement`	It enables the display of the **Chart** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showGaugeElement`	It enables the display of the **Gauge** dashboard element in the toolbox or

	the **Insert** tab in the designer. The property has the `true` value by default.
`showPivotTableElement`	It enables the display of the **Pivot** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showIndicatorElement`	It enables the display of the **Indicator** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showProgressElement`	It enables the display of the **Progress** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showRegionMapElement`	It enables the display of the **Region Map** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showOnlineMapElement`	It enables the display of the **Online Map** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showImageElement`	It enables the display of the **Image** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showTextElement`	It enables the display of the **Text** dashboard element in the toolbox or the **Insert** tab in the designer. The

	property has the `true` value by default.
`showPanelElement`	It enables the display of the **Panel** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showShapeElement`	It enables the display of the **Shape** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showButtonElement`	It enables the display of the **Button** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showListBoxElement`	It enables the display of the **List Box** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showComboBoxElement`	It enables the display of the **Combo Box** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showTreeViewElement`	It enables the display of the **Tree View** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showTreeViewBoxElement`	It enables the display of the **Tree View Box** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true`

	value by default.
showDatePickerElement	It enables the display of the **Date Picker** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the true value by default.

Components

Name	Description
showText	It enables the display of the **Text** component in the designer components insert menu. The property has the true value by default.
showTextInCells	It enables the display of the **Text in Cells** component in the designer components insert menu. The property has the true value by default.
showRichText	It enables the display of the **Rich Text** component in the designer components insert menu. The property has the true value by default.
showImage	It enables the display of the **Image** component in the designer components insert menu. The property has the true value by default.
showBarCode	It enables the display of the **Bar Code** component in the designer components insert menu. The property has the true value by default.

showShape	It enables the display of the **Shape** component in the designer components insert menu. The property has the `true` value by default.
showPanel	It enables the display of the **Panel** component in the designer components insert menu. The property has the `true` value by default.
showClone	It enables the display of the **Clone** component in the designer components insert menu. The property has the `true` value by default.
showCheckBox	It enables the display of the **Check Box** component in the designer components insert menu. The property has the `true` value by default.
showSubReport	It enables the display of the **Text** component in the designer components insert menu. The property has the `true` value by default.
showZipCode	It enables the display of the **Sub-Report** component in the designer components insert menu. The property has the `true` value by default.
showChart	It enables the display of the **Chart** component in the designer components insert menu. The property has the `true` value by default.
showGauge	It enables the display of the **Gauge**

	component in the designer components insert menu. The property has the `true` value by default.
`showSparkline`	It enables the display of the **Sparkline** component in the designer components insert menu. The property has the `true` value by default.
`showMathFormula`	It enables the display of the **Math Formula** component in the designer components insert menu. The property has the `true` value by default.
`showMap`	It enables the display of the **Map** component in the designer components insert menu. The property has the `true` value by default.

Dictionary

Name	Description
`showAdaptersInNewConnectionForm`	It enables the display of the **Object** category in the new connection creation window. The property has the `true` value by default.
`showDictionary`	It enables the display of the report dictionary. The property has the `true` value by default.
`newReportDictionary`	It allows you to create a new data dictionary or join the existing one when creating a new report in the designer. It can take one of the specified below enumeration values:

	• `StiNewReportDictionary.Auto` - defines the mode to create or join the data dictionary from a saved value in cookies (default value); • `StiNewReportDictionary.DictionaryNew` - sets the mode to create a new data dictionary when creating a new report; • `StiNewReportDictionary.DictionaryMerge` - sets the mode to join the existing data dictionary with a new one when creating a new report in the designer.
`useAliases`	Allows you to use aliases in the data dictionary. It can take one of the specified below enumeration values: • `StiUseAliases.Auto` - defines the mode of using aliases from a saved value in cookies (default value); • `StiUseAliases.True` - sets the mode of using aliases in the data dictionary; • `StiUseAliases.False` - disables the mode of using aliases in the data dictionary.
`showDictionaryContextMenuProperties`	Sets a visibility of the **Properties** item in the dictionary context menu. By default, the property is set to `true`.
`showDictionaryActions`	Sets a visibility of the **Actions** menu on the dictionary toolbar. By default, the property is set to **true**.

dataSourcesPermissions	It sets available actions on report data sources. It can take one or several values from the StiDesignerPermissions enumeration.
dataConnectionsPermissions	It sets available actions on connections to report data. It can take one or several values from the StiDesignerPermissions enumeration.
dataColumnsPermissions	It sets available actions on report data columns. It can take one or several values from the StiDesignerPermissions enumeration.
dataRelationsPermissions	It sets available actions on report data connections. It can take one or several values from the StiDesignerPermissions enumeration.
businessObjectsPermissions	It sets available actions on report business objects. It can take one or several values from the StiDesignerPermissions enumeration.
variablesPermissions	It sets available actions on report variables. It can take one or several values from the StiDesignerPermissions enumeration.
resourcesPermissions	It sets available actions on sources in report dictionary. It can take one or several values from the StiDesignerPermissions enumeration.
dataTransformationsPermissions	Sets the available actions on data transformation. It can take one or more values from the

	StiDesignerPermissions enumeration.

In the table below you can see all available values for the `StiDesignerPermissions` enumeration. They can be set for report dictionary elements.

Name	Description
`None`	Disables any action on the item of the data dictionary.
`Create`	It allows you to create definite element of the dictionary.
`Delete`	It allows you to delete a definite element of the dictionary.
`Modify`	It allows you to edit a definite element of the dictionary.
`View`	It allows you to view a definite element of the dictionary.
`ModifyView`	It allows you to edit and view a definite element of the dictionary.
`All`	It allows you to make any actions on the dictionary element.

You can configure the built-in `StiViewer` component used to preview the report. To get access to all of its settings, you should use the `viewerOptions` property, which is an object of the viewer options. All its properties are described in the Viewer settings section.

9.4 Engine

This chapter contains a description of JS Engine functionality.

- Activation
- Loading and Saving Report

- Connecting Data File
- Connecting SQL Databases
- Localization
- Saving Rendered Report
- Getting Access to Pages
- Add custom functions

9.4.1 Activation

YouTube

Watch videos which show how to activate the JS components. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiDesigner** or **StiViewer** components.

index.html

```
...
function Start() {

   //Activation with using license code
   Stimulsoft.Base.StiLicense.Key = "Your activation code...";

   //Activation with using license file
   Stimulsoft.Base.StiLicense.loadFromFile("license.key");
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

Information

Please note that, due to security policies of the web browser, downloading the license file from the local storage will not be possible.

9.4.2 Connecting Data Files

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Connection parameters to data sources are usually stored in the report template itself. But if necessary, you can use other ways to connect data.

Data Sources from Files

The **DataSet** object is used to store data. It has methods for loading data from various file formats. The **regData()** method is used to connect data in the report, in the arguments of which the prepared **DataSet** object is specified.

The data can be loaded from XML files using XSD schema.

index.html

```
...
var dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
dataSet.readXmlSchemaFile("Demo.xsd");
dataSet.readXmlFile("Demo.xml");

var report = new Stimulsoft.Report.StiReport();
report.regData(dataSet.dataSetName, "", dataSet);
report.dictionary.synchronize();
...
```

And from JSON files:

index.html

```
...
var dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
dataSet.readJsonFile("Demo.json");

var report = new Stimulsoft.Report.StiReport();
report.regData(dataSet.dataSetName, "", dataSet);
report.dictionary.synchronize();
```

```
. . .
```

Additional methods for loading data from files

The **DataSet** object has a wide range of methods for loading data:

- readJsonFile(fileName) – loading a JSON file at the specified path;
- readJson(string) – loading JSON data as a string;
- readJson(data) – loading JSON data as a byte array;
- readJson(obj) – using a JavaScript object as data;

- readXmlFile(fileName) – loading an XML file at the specified path;
- readXml(string) – loading XML data as a string;
- readXml(data) – loading XML data as a byte array;

- readXmlSchemaFile(fileName) – loading an XSD file at the specified path;
- readXmlSchema(string) – loading XSD data as a string;
- readXmlSchema(data) – loading XML data as a byte array;

Information

Loading data schema is optional. If you want to use a data schema, you should add it before loading the XML data.

9.4.3 Connecting SQL Databases

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Samples

See examples on GitHub how to connect to databases.

Since pure JavaScript does not have built-in methods for working with remote databases, this functionality is implemented using server-side code. Therefore, Stimulsoft Reports.JS product contains server data adapters implemented using Node.js, PHP, .NET, .NET Core, Python, Java technologies. The database adapter is a software layer between the DBMS and the client script. The adapter connects to the DBMS and retrieves the necessary data, converting it into JSON. The script running on the server (using the adapter) provides for the exchange of JSON data between the client-side JavaScript application and the server side. To use this mechanism on the client side, you should specify the URL address of the host adapter, which processes requests to a required adapter

Links to examples with ready data adapters, implemented for various platforms: Node.js, PHP, .NET, .NET Core, Python, Java.

It`s easy to use an adapter. You should run an adapter and specify the its address:

index.html

```
...
StiOptions.WebServer.url = "http://localhost:9615";
...
```

When requesting data from SQL data sources, the Stimulsoft.Report.Engine sends a POST request to the URL, specified in the option:

index.html

```
...
StiOptions.WebServer.url = "https://localhost/handler.php";
...
```

A JSON object with parameters is passed in the body of the request that use the following structure:

- **command**: two variants are possible - "TestConnection" and "ExecuteQuery";
- **connectionString**: database connection string;
- **queryString**: query string;
- **database**: database type;
- **timeout**: the time of request waiting, specified in the data source;
- **parameters**: an array of parameters as a JSON object {name, value};

- **escapeQueryParameters**: a flag of parameters shielding before requesting.

In response, the Stimulsoft.Report.Engine expects a JSON object with data in the form of the following structure:

- **success**: a flag of successful command execution;
- **notice**: if the flag of command execution has the **false** value, this parameter will contain an error description;
- **rows**: strings array, each element is the array of values, the index is the column number;
- **columns**: an array of column names, index is the column number;
- **types**: an array of column types, the index is the column number. It can take the values "string", "number", "int", "boolean", "array", "datetime".

Request and response sample:

index.html

```
...
request = {
   command: "ExecuteQuery",
   connectionString: "Server=myServerAddress;Database=myDataBase;User
   Id=myUsername;Password=myPassword;",
   queryString: "select * from table1",
   database: "MS SQL"
}

response = {
   success: true,

   rows: [
      ["value1", 1, false],
      ["value2", 1, true]
      ["value3", 2, false]
   ],
   columns: [
      "Column1_name",
      "Column2_name",
      "Column3_name"
   ],
   types:[
      "string",
      "int",
      "boolean"
   ]
}
...
```

Custom Data Base

Also, you may register custom data base. To do it you should invoke the following option:

index.html

```
...
Stimulsoft.Report.Dictionary.StiCustomDatabase.registerCustomDatabase(opti
ons);
...
```

The options are the set of properties and the process() function, which will be invoked when requesting data:

- **serviceName**: adapter name which will be displayed in the designer when creating a new connection
- **designerCategory**: the category name to which the adapter will be added in the **New Data Source** menu. By default, the adapter can be found in the `Favorites` category, but you can change this by setting the option to one of the following values: `"Files"`, `"SQL"`, `"NoSQL"`, `"Azure"`, `"Google"`, `"OnlineServices"`, `"REST"`.
- **sampleConnectionString**: the sample of a connection string that is inserted in the form of setting up a new connection
- **process**: the function, which will be invoked to prepare and transmit data to the Stimulsoft.Report.Engine

Two arguments are transmitted to the input of the **process()** function: **command** and **callback**.

The **command** argument is the JSON object, where the Stimulsoft.Report.Engine will transfer the following parameters:

- **command**: action, which is being invoked at the moment. Possible values: "**TestConnection**": test the database connection from the new connection creation form "**RetrieveSchema**": retrieving data schema is needed to optimize a request and not only to transfer necessary data set. It is invokes after connection creation "**RetrieveData**": data request.
- **connectionString**: connection string;
- **queryString**: query string;
- **database**: database type;
- **timeout**: the time of request waiting, specified in the data source.

The **callback** argument is the function, which should be invoked to transmit prepared data to the Stimulsoft.Report.Engine. As the callback argument to the functions you must pass a JSON object with the following parameters:
- **success**: the flag of successful command execution;
- **notice**: if the flag of command execution has the **false** value, this parameter should contain an error description;
- **rows**: strings array, each element is the array from values, the index is the column number;
- **columns**: columns name array, the index is the column number;
- **types**: the object where field name is column name and the value is the type of the column {Column_Name : "string"}. The type can take the following values "string", "number", "int", "boolean", "array", "datetime". If the **columns** array will be transmitted, you will be able to transmit types array to the **types**, the index should be column number. It doesn't work for the "**RetrieveSchema**".

If the command = "**RetrieveSchema**", then in addition types you should transmit table names to the **types**.

The sample of a request and response when receiving a schema:

index.html

```
...
request = {
   command: "RetrieveSchema"
}

response = {
   success: true,

   types:{
      Table1: {
         Column1: "string",
         Column2: "number"
      },
      Table2: {
         Column1: "string"
      }
   }
}
...
```

The sample of a request and response when getting data:

index.html

```
...
request = {
   command: "RetrieveData",
   queryString: "Table1"
}
response = {
   success: true,

   rows: [
      ["value1", 1],
      ["value2", 1]
      ["value3", 2]
   ],
   columns:[
      "Column1",
      "Column2"
   ],
   types:[
      "string",
      "number"
   ]
}
...
```

The example of adapter registration.

Query Timeout

Also, for SQL data sources used in the report, you can specify the **Query Timeout** in seconds. The value of this property is stored in the report template for each SQL connection separately.

Below is an example of code that you may use to change the connection string for MS SQL, adjust the query, set the query timeout for the already created connection, and data sources in the report.

index.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.loadFile("Report.mrt");
report.dictionary.databases.getByName("Connection").connectionString =
"Data Source=server;Integrated Security=True;Initial Catalog=DataBase";
report.dictionary.dataSources.getByName("DataSourceName").sqlCommand =
"select * from Table where Column = 100";
report.dictionary.dataSources.getByName("DataSourceName").commandTimeout =
```

```
1000;
...
```

Information

The address to the data adapter must be set before the creation code of the component or the report object, since the value of this option must be known to the report engine before it is initialized.

The following SQL data sources are currently supported - MySQL, MS SQL, PostgreSQL, Firebird, and Oracle. The data adapters are open source and can be modified the way you need.

OData storages

You can also use data for designing reports and dashboards obtained from OData storages. In this case, you can do the authorization using a username, user password, or using a token. Authorization parameters are specified in the connection string to the OData storage using the ";" separator.

viewer.html

```
...
//Authorization using a user account
var oDataDatabase = new
Stimulsoft.Report.Dictionary.StiODataDatabase("OData", "OData", "https://
services.odata.org/V4/Northwind/
Northwind.svc;AddressBearer=adress;UserName=UserName;Password=Password;Cli
ent_Id=Your Client ID", false, null);

//Authorization using a user token
var oDataDatabase = new
Stimulsoft.Report.Dictionary.StiODataDatabase("OData", "OData", "https://
services.odata.org/V4/Northwind/Northwind.svc;Token=Enter your token",
false, null);

report.dictionary.databases.add(oDataDatabase);
report.dictionary.synchronize();

//Query with data filter
var productsDataSource =
report.dictionary.dataSources.getByName("Products");
if (productsDataSource != null) productsDataSource.sqlCommand =
"Products?$filter=ProductID eq 2";
...
```

Also, you can specify user HTTP headers for data sources. It can be done in the `onBeginProcessData` event handler. A complete sample is available on our website:

index.html

```
...
// In `onBeginProcessData` event handler add custom HTTP headers
report.onBeginProcessData = function (args) {
   if (
      args.database === "JSON" &&
      args.command === "GetData" &&
      args.pathData && args.pathData.include("/reports/ProtectedDemo.json")
   ) {
      // Add custom header to pass through backend server protection
      args.headers.push({key: "X-Auth-Token", value: "*YOUR TOKEN*"});
   }
};
...
```

9.4.4 Localization

The **HTML5 Viewer** and **HTML5 Designer** components support full localization of the user interface. The special static method - **addLocalizationFile()** - is used to localize the report viewer interface to the required language. As the arguments of the method, you should specify the path to the localization XML file, and specify whether the localization will automatically load together with the component.

The specified function adds localization to the designer menu:

index.html

```
...
Stimulsoft.Base.Localization.StiLocalization.addLocalizationFile("../
localization/de.xml", true);
...
```

The specified function adds localization to the designer menu, but doesn`t load it automatically:

index.html

```
...
Stimulsoft.Base.Localization.StiLocalization.addLocalizationFile("../
localization/de.xml", false, "Deutsch");
...
```

After the method is called, a link to a localization file will be added to the internal collection. The file will be loaded the first time you select the specified localization in the designer menu. It saves memory and time of components run. In this variant, you should specify localization name for displaying in the designer menu as the third argument. After localization loaded this value will be taken from a localization file.

The specified function adds localization to the designer menu and sets it by default:

index.html

```
...
Stimulsoft.Base.Localization.StiLocalization.setLocalizationFile("../
localization/de.xml");
...
```

When each the designer loading, this localization will be selected even if earlier another localization was selected.

Information

For the **HTML5 Designer**, when creating a project with pure JavaScript, the necessary localization files should be added using the above code. If the project uses the Node.js technology, then it is enough to specify a folder with localization files. In this case all the files in it will be added automatically:

```
Stimulsoft.System.NodeJs.localizationPath = "locales";
```

9.4.5 Loading and Saving Report

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

Two methods of the **StiReport** object are used to load a report – **loadFile()** and **load()**. They are used as follows:

- loadFile(filePath) – loads a report from the **MRT** file which path is specified in the filePath;
- load(str) – loads a report from the string str, that contains **XML** or **JSON**;
- load(data) – loads a report from the **array data** of the number[] type;
- load(xml) – loads a report from the XML of the **XMLDocument** type;
- load(json) – loads a report from the **JS** object.

For example, use the code below to load a report from file:

index.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.loadFile("SimpleList.mrt");
...
```

The **MRT** format of Stimulsoft Reports is the **JSON** based description of reports. You can use **MRT** files, created in other Stimulsoft Reports designers in the **JSON** based description. Use the code below to save a report to a string:

index.html

```
...
var report = new Stimulsoft.Report.StiReport();
var jsonString = report.saveToJsonString();
...
```

Use the code below to load a report from this string:

index.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.load(jsonString);
...
```

9.4.6 Saving Rendered Report

Rendered report can be saved into a **JSON** document for later viewing (data stored within a document):

index.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.loadFile("SimpleList.mrt");
report.renderAsync(function(){
   var jsonString = report.saveDocumentToJsonString();
});
...
```

9.4.7 Getting Access to Pages

Getting access to pages of a report

The report has a collection of pages, which can be accessed by the following way:

index.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.loadFile("SimpleList.mrt");

for (var index = 0; index < report.pages.count; index++) {
   alert(report.pages.getByIndex(index).name);
}
...
```

Getting access to pages of a rendered report

After rendering the report, a collection of pages is created. It can be accessed by the following way:

index.html

```
...
var report = new Stimulsoft.Report.StiReport();
report.loadFile("SimpleList.mrt");
      K      ^     E         EF
   for (var index = 0; index < report.renderedPages.count; index++) {
      alert(report.renderedPages.getByIndex(index).name);
   }
 FX
...
```

9.4.8 Report Events

Information

Since dashboards and reports use the same unified template format - MRT, methods for loading the template and working with data, the word “report” will be used in the documentation text.

In order to write interactive applications it is needed to respond to changes in the application. The event system which is represented in **StiReport** object can be used for this. The following events exist:

onPrepareVariables

Asynchronous event is called before filling in the variables in the report at the beginning of the report rendering. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **PrepareVariables**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.

viewer.html

```
...
report.onPrepareVariables = (args, callback) => {
   args.variables[0].value = "Replace value";
}
...
```

onBeginProcessData

Asynchronous event is called before requesting the data for the report. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **BeginProcessData**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
command	a string ID for the current command. Can have values "TestConnection" and "ExecuteQuery" to call the command to test a connection and data acquisition respectively.
database	a string name of the current database.
connection	the name of the current connection to a data source, specified in report template.
headers	an array of the query headers.
withCredentials	an argument is used to provide the ability to set a cookies in the query.
connectionString	a connection string to the data source for a report.
dataSource	the name of the current data source, specified in the report template. It is set only for SQL data sources.
queryString	a string with the SQL query to the database for retrieving data. Used only with the command = "ExecuteQuery".
timeout	an argument is used to provide the ability to define the seconds of the query timeout.
parameters	a list of the query parameters.

escapeQueryParameters	a flag is used to provide the ability to use the escaped part of the request. By default, the value is set to **true**.
pathData	an argument is used to provide the ability to define a path to data file.
tryParseDateTime	a flag is trying to parse data as DateTime.
relationDirection	an argument is used to provide the ability to change the direction of the relation between data sources.
pathSchema	an argument is used to provide the ability to define a path to XSD files.
firstRowIsHeader	a flag is used to provide the ability to use a first row as data header in Excel data source.
collectionName	a string collection name of the OData data source.
separator	a string separator of the CSV data source.
dataType	an argument is used to provide the ability to set the data type of the GIS data source.
codePage	an argument is used to provide the ability to set a code page of the CSV or DBF data source.

viewer.html

```
...
//Replace connection string
report.onBeginProcessData = (args) => {
   if (args.database == "MySQL")
      args.connectionString = "new connection string";
}
...

//Add a some data
report.onBeginProcessData = (args, callback) => {
   if (args.database == "MySQL"){
      args.preventDefault = true;
      var result = {
         success: true,
         rows: [
            ["value1", 1, false],
            ["value2", 1, true],
            ["value3", 2, false]
         ],
```

```
        columns: [
           "Column1_name",
           "Column2_name",
           "Column3_name"
        ],
        types:[
           "string",
           "int",
           "boolean"
        ]
     }

     // https://github.com/stimulsoft/DataAdapters.JS/
     callback(result);
   }
}
...
```

onEndProcessData

Is called after retrieving data for the report. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	a string ID for the current event. By default, the value is set to **EndProcessData**.
report	a report object **StiReport**.
command	a string ID for the current command. Can have values "TestConnection" and "ExecuteQuery" to call the command to test a connection and data acquisition respectively.
dataSource	the name of the current data source, specified in the report template. It is set only for SQL data sources.
connection	the name of the current connection to a data source, specified in report template.
database	a string name of the current database.
result	a result dataset in a specific JSON format. It has

	two collections – columns and rows, with descriptions of columns and rows of data sources.

viewer.html

```
...
report.onEndProcessData = (args) => {
   if (args.command == "ExecuteQuery" && args.dataSource == "Categories")
      args.result.rows.push(rowData) ;
      // https://github.com/stimulsoft/DataAdapters.JS/
}
...
```

onBeginRender

Is called at the beginning of the report rendering. This is not relevant for the dashboards.

onRendering

Is called at the report is rendering when each report page is created. This is not relevant for the dashboards.

onEndRender

Is called at the ending of the report rendering. This is not relevant for the dashboards.

onExportingRender

Is called before a report export is started.

onExportedRender

Is called after a report export is ended.

onPrinting

Is called when `report.print()` or `report.printToPdf()` method is invoked. This is not relevant for the dashboards.

viewer.html

```
...
//Remove image before report printing
report.onPrinting = () => {
   var page = report.renderedPages.getByIndex(0);
   var image = page.components.getByName("Image1");
   if (image)
      page.components.remove(image);
}
...
```

onPrinted

Asynchronous event is called when `report.print()` or `report.printToPdf()` method is invoked after a report is exported to Html or PDF file (depends on method type). This is not relevant for the dashboards. The event handler argument “event” is an object with the next fields:

Name	Description
sender	The identifier of the component, which initiated this event.
event	string ID for the current event. By default, the value is set to **Printed**.
report	a report object **StiReport**.
preventDefault	a flag to prevent further processing of the event. By default, the value is set to **false**.
async	a flag is used to provide the ability to stop the execution of the event before the callback function is executed. By default, the value is set to **false**.
data	export data to the string or byte array

	representation.

viewer.html

```
...
//Stop to report print and define a custom print method
report.onPrinted = (args) => {
   args.preventDefault = true;
   var printData = args.data;
   myPrintingMethod(printData);
}
...
```

onRefreshing

Is called after a report is rendered if **report.refreshTime** property is set more than zero. Also, this event is called when **Refresh** button is clicked in the viewer.

9.4.9 Add custom functions

You can add a custom function to the Dictionary in the report designer when you integrate it into your application. After adding the custom function, you can use this in creating reports and dashboards. Below is the example of adding a function for calculating the sum total.

index.html

```
...
var myFunc = function (value) {
   if (!Stimulsoft.Data.Extensions.ListExt.isList(value))
   return Stimulsoft.Base.Helpers.StiValueHelper.tryToNumber(value);

   return
   Stimulsoft.Data.Functions.Funcs.skipNulls(Stimulsoft.Data.Extensions.Lis
   tExt.toList(value))
   .tryCastToNumber()
   .sum();
};

Stimulsoft.Report.Dictionary.StiFunctions.addFunction("MyCategory",
"MySum", "MySum", "MySum", "", Number, "Return Description", [Object],
["value"], ["Descriptions"], myFunc);
...
```

10 Reports and Dashboards for PHP

The products Stimulsoft Reports.PHP and Stimulsoft Dashboards.PHP are a combination of JavaScript and PHP scripts, along with all the necessary functionality for their interaction. Report generation and exporting are performed using a JavaScript core on the client side in a web browser window, or on the server side utilizing the universal Node.js platform. The PHP server side includes everything needed to work with report files and connect to various SQL data sources. The client-server communication is carried out via AJAX requests that send and receive JSON data in a specific format. For ease of use, special events and functions have been developed both on the client-side JavaScript, including the Node.js platform, and on the PHP server side.

The product contains the report generator and the data analysis engine, as well as report viewer and designer components.

10.1 Engine

The Stimulsoft Reports.PHP report generator allows loading, building, and exporting a report to various formats without deploying the viewer and designer. This eliminates the need to load large scripts on the client side.

The report building and exporting is done using the JavaScript core on the client side in the web browser window, or on the server side using the universal Node.js platform. The PHP server-side contains everything needed to work with report files and to connect to various SQL data sources. The client-server communication is done via AJAX requests, transmitting and receiving JSON data in a specific format. For ease of use, special events and functions have been developed for both the client-side JavaScript and the PHP server-side.

Dashboards

Stimulsoft Dashboards.PHP allows loading, analyzing data, and exporting the dashboard to various formats without deploying the viewer and designer.

All data analysis, except for certain SQL operations, is performed using the JavaScript data analyzer on the client side in the web browser window, or on the server side using the universal Node.js platform. The PHP server side is universal for both dashboards and reports, utilizing the same events and functions.

10.1.1 Usage

To use the product, simply download the ZIP archive from the Downloads page on our website, extract it, and copy the contents of the **/PHP** folder to your web server. This folder represents a web project containing all the necessary files and resources for the product to function, as well as examples for working with the viewer and designer.

To install the product into an existing project, simply copy the `/vendor` folder from the **/PHP** directory to the root directory of your project, or use the Composer dependency manager by running the following console command:

console

```
composer require stimulsoft/reports-php
```

To work with dashboards, you will need to include the following package:

console

```
composer require stimulsoft/dashboards-php
```

When using the product, in most cases, it is sufficient to use only PHP code, which

provides the functionality for all the basic features. For more detailed configuration and to utilize all the features, JavaScript code needs to be used.

To use the components in a web project, simply include the script autoloader at the beginning of the PHP file. After that, you can use all available PHP classes and functions to work with reports and dashboards:

index.php

```
<?php
   require_once 'vendor/autoload.php';
   ...
?>
```

The class `StiReport` is designed for working with the report generator. Using this class, you can load a report template or document, perform report building and exporting, handle requests, and manage report generator events. For example, to load a report from a file, build it, and display a message in the browser window:

index.php

```
<?php
   require_once 'vendor/autoload.php';

   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onAfterRender = 'afterRender';
   $report->process();
   $report->loadFile('reports/SimpleList.mrt');
   $report->render();
?>

<html>
   <head>
      <?php
         $report->javascript->renderHtml();
      ?>
      <script type="text/javascript">
         function afterRender() {
            alert('Done!');
         }
      </script>
   </head>
   <body>
      <?php
```

```
        $report->renderHtml();
    ?>
  </body>
</html>
```

The complete code sample can be found on GitHub.

In this example, the following steps are performed sequentially:

- An instance of the `StiReport` object is created;
- Necessary event handlers are added;
- The current request is processed;
- The report template is loaded from the file `SimpleList.mrt`;
- The report building command is called;
- In the HTML file template, JavaScript event code is added, and the necessary JavaScript and HTML code for the component is rendered.

The `$report->process()` method processes the current request and, upon success, automatically returns the result to the client-side. This is explained in more detail in the Event Handler section.

The `$report->javascript->renderHtml()` method outputs the code for including the necessary component scripts. The `$report->renderHtml()` method outputs the JavaScript and HTML code for the component itself.

Our products, Stimulsoft Reports.PHP and Stimulsoft Dashboards.PHP, do not have a native report generator core in PHP. Report building and exporting are performed using JavaScript on the client side or on the server side with the Node.js platform. Therefore, when using PHP code to work with the components, you need to call one of the special methods, which will add the corresponding JavaScript code to the web page for performing all necessary actions, as well as the HTML code for the visual part of the component.

Name	Description
`getHtml($mode = StiHtmlMode::HtmlScripts)`	Returns the JavaScript and HTML code necessary for the component to function, including all required actions on the report. The `$mode` parameter allows you to set different

	options for the returned code: • `StiHtmlMode::Scripts` – only the necessary JavaScript code for insertion into the `<script></script>` block on an HTML page; • `StiHtmlMode::HtmlScripts` – the necessary JavaScript and HTML code for insertion into an HTML element on the page; • `StiHtmlMode::HtmlPage` – a fully prepared HTML page. **Note:** For component elements, such as the `$report->javascript` object, the `getHtml()` method does not accept parameters since it has only one output option.
`renderHtml($elementId = null)`	This method outputs the JavaScript and HTML code required for the component to function, including all necessary actions on the report. The `$elementId` parameter allows you to specify the ID of the HTML element on the page where the component will be rendered. By default, the output is rendered in the current location of the page.
`printHtml()`	The method can output a fully prepared HTML page with all the necessary scripts for the component to work. The current HTML page template is completely ignored. This mode is ideal for full-screen viewing or editing of the report.

Thus, the methods described above allow you to display the component in different

ways depending on the requirements. Here's an example of simplified component rendering without using an HTML page template:

index.php

```
<?php
   require_once 'vendor/autoload.php';

   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onAfterRender = "alert('Done!');";
   $report->process();
   $report->loadFile('reports/SimpleList.mrt');
   $report->render();
   $report->printHtml();
?>
```

Information

When using the Node.js platform for report generation on the PHP server side, the specified methods will be called automatically within the handler, and their explicit use is not required.

Managing URLs for Loading Report Generator JavaScript Files

By default, all product JavaScript files are loaded via URLs relative to the location of the current PHP script. In some cases, this behavior needs to be changed, and several options are provided for this. To use an absolute path for loading all product scripts, set the `useRelativeUrls` option to `false`:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->javascript->useRelativeUrls = false;
?>
```

To adjust the relative path, the `relativePath` option is provided. You need to assign a string value that will be used when forming the URL for loading the scripts. In this case, the `useRelativeUrls` option must be enabled (default value):

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->javascript->relativePath = '../../';
?>
```

By default, the scripts are loaded as static files. To enable dynamic script loading using a PHP handler, set the `useStaticUrls` option to `false`:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->javascript->useStaticUrls = false;
?>
```

Information

The report viewer and report designer components also have a `javascript` property, allowing you to manage script settings in the same way as described above.

Various alternatives of deployment and optimization are considered in the Optimization of script loading chapter.

10.1.2 Optimization of Scripts Loading

Due to the extensive functionality of the product, the scripts have a relatively large size. During the initial load of a web application, or if browser caching is disabled, loading may take some time, especially with slow internet connections. We offer two

solutions to this problem: using packed scripts or utilizing partial functionality and loading only the required components. It is possible to combine these options.

Packed scripts

Packed scripts have the same structure as regular scripts but end with `*.pack.js` in the file name. These scripts contain a block of packed data in the form of a JavaScript variable and a compact unpacker. When all scripts are loaded, the unpacker automatically unpacks all the loaded data and executes the prepared script. Unpacking takes some time, but under certain circumstances—such as with a slow internet connection—this time is significantly shorter than the time it would take to load regular scripts.

To use packed scripts you should set the `$js->packed` to `true`. For example:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->javascript->usePacked = true;
?>
```

Partial loading of scripts

The `stimulsoft.reports.js` script contains all the functionality for report building and exporting. If you only need specific features for generating reports, partial script loading is available, allowing you to load only the necessary parts of the generator that contain specific features. For example, if your reports do not use maps, you can skip loading that functionality, which will speed up the loading of your web project and reduce browser memory consumption.

Information

This option is only available for the report generator core. The viewer and designer cannot be split into parts; their scripts will always be loaded as a single block.

To use partial script loading, simply configure the appropriate options in the `javascript` property of the report object.

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->javascript->reportsChart = true;
   $report->javascript->reportsExport = true;
   $report->javascript->reportsImportXlsx = true;
   $report->javascript->reportsMaps = false;
   $report->javascript->blocklyEditor = false;
?>
```

Each PHP option of the StiJavaScript object controls the loading of a script containing specific functionality. The table contains the whole set of scripts, which you can load separately.

Name	Description
`javascript->reportsExport`	Contains algorithms for exporting generated reports into various formats such as PDF, HTML, Excel, RichText, and others.
`javascript->reportsChart`	Includes components for working with all types of charts in the report.
`javascript->reportsMaps`	Contains components for working with regional and online maps.
`javascript->blocklyEditor`	Contains a visual editor based on Blockly for creating event scripts in the report. The event handler itself is built into the report engine.
`javascript->reportsImportXlsx`	Contains algorithms for working with Excel data sources.

Information

The report viewer and report designer components also have a `javascript` property, which allows you to manage script settings in the same way as described above. This enables control over which scripts are loaded and optimizes performance by customizing the functionality required for these components.

10.1.3 License Activation

After purchasing the product, you should activate license for used components. There are several ways of license key connection.

Activation using code string

To activate from a code string, simply copy the encrypted license text from your personal account on the website and register it using the static function `setPrimaryKey()`, which is located in the `StiLicense` class:

index.php

```
<?php
   use Stimulsoft\StiLicense;

   StiLicense::setPrimaryKey('Your activation code...');
?>
```

The full example code is available on GitHub.

Activation using a file

To activate using a license file, download the `license.key` file from your personal account on the website, copy it into your PHP project folder, and register it using the static function `setPrimaryFile()`, located in the `StiLicense` class:

index.php

```
<?php
   use Stimulsoft\StiLicense;

   StiLicense::setPrimaryFile('license.key');
```

```
?>
```

The full example code is available on GitHub.

Activation of the report generator only

In some cases, you may need to activate the report generator separately from other components. In this case, the `license` key can be registered using the `setKey()` or `setFile()` method of the license property of the report object:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->license->setKey('Your activation code...');
   $report->license->setFile('license.key');
?>
```

The full example code is available on GitHub.

Protection against license key theft

If the license is activated using a code string, it can be added conditionally. For example, you may want to add the license key only for registered users:

index.php

```
<?php
   use Stimulsoft\StiLicense;

   if (!empty($sessionID))
      StiLicense::setPrimaryFile('Your activation code...');
?>
```

Also, we recommend you change the location and name of the license key file, for example:

license.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->license->setFile('private/a15fc0ef64e6.key');
?>
```

Activation of a license in a single file

If the application components are used across multiple separate files, such as a report generator, viewer, and designer, it is more convenient to apply the license in a single file instead of activating each component separately. To achieve this, you need to create a dedicated file where the license will be applied, and in all other files containing components, simply include the prepared license file using the standard `require_once` statement.

license.php

```
<?php
   use Stimulsoft\StiLicense;

   StiLicense::setPrimaryKey('Your activation code...');
?>
```

license.php

```
<?php
   require_once 'license.php';

   use Stimulsoft\Report\StiReport;

   $report = new StiReport();

   ...
?>
```

10.1.4 Loading and Saving Reports

Information

Stimulsoft MRT and MDC files are a description of reports with XML or JSON markup. You can use MRT and MDC files, created in other Stimulsoft designers.

Loading a report

A report can be stored as a report template (MRT file) or as a built report (MDC document) intended for further viewing or exporting. To load a report using PHP code, you can use one of the methods listed below on the StiReport object. Each method accepts either the report file name or the report itself as a string:

Name	Description
`loadFile($filePath, $load = false)`	Loads a report template from an MRT file on the client-side, the path to which is specified in the function arguments. If the parameter $load is set to true, the report file will be loaded on the server side and passed to the client as a packed Base64 string.
`load($data, $fileName = 'Report')`	Loads a report template from an XML or JSON string and passes it to the client as a packed Base64 string. The $fileName parameter sets the file name that will be used for subsequent saving and exporting of the report.
`loadPacked($data, $fileName = 'Report')`	Loads and passes a report template to the client in the form of a packed Base64 string, specified in the $data parameter. The $fileName parameter sets the file name that will be used for subsequent saving and exporting of the report.
`loadDocumentFile($filePath, $load = false)`	Loads a built report from an MDC file on the client-side, the path to which is specified in the function

	parameters. If the $load parameter is set to true, the document file will be loaded on the server-side and passed to the client as a packed Base64 string.
`loadDocument($data, $fileName = 'Report')`	Loads a built report from an XML or JSON string and passes it to the client as a packed Base64 string. The $fileName parameter sets the file name that will be used for subsequent saving and exporting of the report.
`loadPackedDocument($data, $fileName = 'Report')`	Loads and passes a built report to the client in the form of a packed Base64 string, specified in the $data parameter. The $fileName parameter sets the file name that will be used for subsequent saving and exporting of the report. Loads and passes a built report to the client in the form of a packed Base64 string, specified in the $data parameter. The $fileName parameter sets the file name that will be used for subsequent saving and exporting of the report.

Example of loading a report from a file on the server-side from a private directory and passing it to the client as a packed string for subsequent building:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt', true);
   $report->render();
?>
```

Saving a report

In the report generation mode on the client-side JavaScript, the report generator on the PHP server-side doesn't have access to the report object. In this case, to save a report template or document, you need to use events and JavaScript functions. More detailed information on this can be found in the "Report Engine Events" section.

An example of saving a built report as a string for future use:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onAfterRender = 'afterRender';
   $report->loadFile('reports/SimpleList.mrt', true);
   $report->render();
?>

...

<script>
   function afterRender(args) {
      let reportJson = args.report.saveDocumentToJsonString();
...
   }
</script>
```

The full example code is available on GitHub.

In the report-building mode on the server-side, one of the methods listed below is used to save a report:

Name	Description
`saveDocument($filePath = null))`	Saves the built report as an MDC file at the path specified in the function arguments. If the `$filePath` parameter isn't specified, the method will return the report as a JSON

	string instead of saving the file.
`savePackedDocument($filePath = null)`	Saves the built report as a packed MDZ file at the path specified in the function arguments. If the `$filePath` parameter isn't specified, the method will return the report as a packed `Base64` string instead of saving the file.

An example of saving a built report as a file on the server-side:

index.php

```
<?php
   use Stimulsoft\Report\Enums\StiEngineType;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->engine = StiEngineType::ServerNodeJS;
   $report->loadFile('reports/SimpleList.mrt', true);
   $report->render();
   $report->saveDocument('reports/SimpleList.mdc');
?>
```

The full example code is available on GitHub.

Information

The Stimulsoft Reports.PHP report generator and the Stimulsoft Dashboards.PHP analytic panels are based on the JavaScript platform and support saving MRT and MDC files only in JSON format. XML format files are only supported in load mode, and will be automatically converted to JSON format upon saving.

Since analytic panels always require data, they can't be saved as MDC documents. Stimulsoft Dashboards.PHP supports saving panels only as templates using JavaScript events and functions.

When saving a document from the viewer menu, the file is also saved in JSON

format, and has the extension MDC for a standard document, MDZ for a packed document, and MDX for an encrypted document.

10.1.5 Rendering a Report

To build a loaded report, you should call the `render()` function on the report object. For example, you want to build a report before exporting it.

index.php

```
<?php
   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Export\Enums\StiExportFormat;

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $report->render();
   $report->exportDocument(StiExportFormat::Pdf);
   $report->printHtml();
?>
```

The full example code is available on GitHub.

To perform any actions with the report before building it using JavaScript, you can simply define the name of a JavaScript function for the `onBeforeRender` event. The event's arguments will include the action type and the report itself. Example of registering JSON data before report generation:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onBeforeRender = 'beforeRender';
   $report->loadFile('reports/SimpleList.mrt');
   $report->render();
?>

...

<script>
   function beforeRender(args) {
```

```
    let dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
    dataSet.readJsonFile("Demo.json");

    let report = args.report;
    report.regData(dataSet.dataSetName, "", dataSet);
  }
</script>
```

The full example code is available on GitHub.

To perform any actions after building the report using JavaScript, you can define the name of a JavaScript function for the `onAfterRender` event. The event's arguments will include the action type and the report itself. Example of displaying a message after report generation:

index.php

```
<?php
  use Stimulsoft\Report\StiReport;

  $report = new StiReport();
  $report->onAfterRender = 'afterRender';
  $report->loadFile('reports/SimpleList.mrt');
  $report->render();
?>

...

<script>
  function afterRender(args) {
    alert("The report rendering is completed.");
  }
</script>
```

The full example code is available on GitHub.

10.1.6 Rendering a Report on Server Side

To build a report on the server-side, the universal Node.js platform is used. This platform executes the necessary block of JavaScript code and returns the prepared result.

Deploying the Node.js Platform

Before using the report generator on the server-side, you need to install the Node.js

platform and configure it. This can be done separately by following the Node.js installation instructions for a specific operating system from the official platform website, or automatically using a special deployment function.

If the platform is already installed, it is enough to specify the path to the directory containing the executable files of the platform. If necessary, you can also specify the path to the working directory, where the required Node.js packages will be installed in the node_modules subdirectory:

index.php

```
<?php
   use Stimulsoft\StiNodeJs;

   $nodejs = new StiNodeJs();

   //$nodejs->binDirectory = "C:\\Program Files\\nodejs";
   //$nodejs->binDirectory = "/usr/bin/nodejs";

   //$nodejs->workingDirectory = "";
?>
```

The full example code is available on GitHub.

If the platform and packages aren't installed, special methods can be used to install them. These methods need to be called only once: the first method installs the Node.js platform itself, and the second method installs or updates all necessary packages to the latest version:

index.php

```
<?php
   use Stimulsoft\StiNodeJs;

   $nodejs = new StiNodeJs();

   $result = $nodejs->installNodeJS();
   if ($result)
      $result = $nodejs->updatePackages();

   $message = $result ? 'The installation was successful.' : $nodejs-
   >error;
?>
```

The full example code is available on GitHub.

The platform and packages will be installed in the specified working directory.

Building a report

To switch the report generator to server-side operation, you need to set the report's engine property to `StiEngineType::ServerNodeJS`. The rest of the events and methods for working with the report are exactly the same as when building a report on the client-side.

After calling the `render()` method on the report object, the built report can be saved using the `saveDocument()` or `savePackedDocument()` methods, as described in the Loading and Saving a Report section. Additionally, after building the report, it can be exported to one of many formats using the `exportDocument()` method, as described in the Exporting a Report from Code section.

Example of loading and building a report template, and subsequently saving the built report to a file on the server-side:

index.php

```
<?php
   use Stimulsoft\Report\Enums\StiEngineType;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->engine = StiEngineType::ServerNodeJS;
   $report->loadFile('reports/SimpleList.mrt', true);
   $report->render();
   $report->saveDocument('reports/SimpleList.mdc');
?>
```

The full example code is available on GitHub.

Any errors that occur when working with Node.js packages, as well as when building and exporting a report on the server-side, can be read in the `report->nodejs->error` and `report->nodejs->errorStack` properties. These properties will

contain the last error in the queue:

index.php

```
<?php
   use Stimulsoft\Report\Enums\StiEngineType;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->engine = StiEngineType::ServerNodeJS;
   $report->loadFile('reports/SimpleList.mrt', true);
   $report->render();

   if (!$report) {
      // The main text of the error as a string.
      $error = $report->nodejs->error;

      // The full error text as an array of strings.
      $errorStack = $report->nodejs->errorStack;
   }
?>
```

The full example code is available on GitHub.

10.1.7 PHP Events Handler

The report generator, as well as the report viewer and designer, can trigger client-side events, send them to the PHP server for further processing, and receive the prepared response. All actions are implemented in the event handler, so there is no need to use any additional functions to establish communication between the client and server or to transfer data. To work with a selected event, you simply need to add the function name to the handler, and the specified function will automatically be triggered when the selected event occurs. Events can be triggered both on the client-side on JavaScript and on the server-side on PHP. If needed, multiple functions of any type can be added to the same event.

Triggering JavaScript events on the client-side

To trigger a JavaScript event, you need to add the function name as a string to the handler. All necessary data will be passed in the event arguments. A list of available properties passed in the arguments for all events can be found in the Report Engine Events section. Here's an example of displaying a message with the number of pages in the generated document after the report is built:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onAfterRender = 'afterRender';
   $report->loadFile('reports/SimpleList.mrt');
   $report->render();
?>

...

<script>
   function afterRender(args) {
      let pagesCount = args.report.renderedPages.count;
      alert("The report is rendered, pages: " + pagesCount);
   }
</script>
```

The full example code is available on GitHub.

In this example, you can retrieve the JavaScript report object from the event arguments and read the number of pages built in the document.

Information

You can find more detailed information about the available functions and parameters of the JavaScript report generator in the documentation Stimulsoft Reports.JSand Stimulsoft Dashboards.JS products.

Triggering JavaScript events on the Node.js server-side

When using the Node.js platform for working with reports, there is no possibility to call a JavaScript function by name, as no HTML template is used. In this case, to trigger a JavaScript event, you need to add the function itself as a string or lines of code to the handler. The event arguments will be stored in a predefined variable args, which can be used in the event code. Here's an example of clearing the data dictionary in the template before building the report:

index.php

```
<?php
   use Stimulsoft\Report\Enums\StiEngineType;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->engine = StiEngineType::ServerNodeJS;
   $report->onBeforeRender = 'args.report.dictionary.clear();';
   $report->loadFile('reports/SimpleList.mrt', true);
   $report->render();
?>
```

Information

The same method for JavaScript events can be used when displaying the viewer or designer without an HTML template, where it isn't possible to predefine the necessary JavaScript function.

Triggering PHP events on the server-side

To trigger a PHP event, you need to add the function name as a variable or the function itself to the handler. All necessary data will be passed in the event arguments. A list of available properties passed in the event arguments can be found in the Report Engine Events section. Here's an example of adjusting the password in the connection string before making a data request:

index.php

```
<?php
   use Stimulsoft\Events\StiDataEventArgs;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onBeginProcessData = function (StiDataEventArgs $args) {
      args->connectionString = str_replace('Pwd=;', 'Pwd=******;', args-
      >connectionString);
   };

   $report->loadFile('reports/SimpleList.mrt', true);
   $report->render();
?>
```

The full example code is available on GitHub.

In this example, you can retrieve and modify all the database connection parameters from the event arguments.

In a PHP server-side event, there is an option to return a textual message about the successful completion or an error in event processing, which will be displayed in the viewer or designer after the event finishes. To display an error window, you need to return the result of the function `StiResult::getError('Error message')`. To display an informational message window, you can return the result of the function `StiResult::getSuccess('Info message')` or simply the string `'Info message'`.

index.php

```
<?php
   $report->onBeginProcessData = function (StiDataEventArgs $args) {
...
      return StiResult::getError('Error message');
      // return StiResult::getSuccess('Info message');
      // return 'Info message';
   };
?>
```

Information

If an error occurs in the event handler itself, such as a database connection error, file processing error, etc., an internal message will be displayed regardless of whether a message is defined within the event component.

Information

The dialog window with the message will only be shown when using the `StiViewer` or `StiDesigner` components. The report generator itself doesn't have visual forms, so the event processing message will be displayed in the browser console.

Triggering multiple identical events

It's possible to add an unlimited number of functions to the event handler. They will all be grouped by event type and called sequentially in the order they were added. Instead of assigning the function name, you need to use the special `append()` method, passing the function name or the function itself as a parameter.

Here's an example of modifying an SQL query on the client-side using JavaScript and modifying the connection string on the PHP server-side:

index.php

```
<?php
   use Stimulsoft\Events\StiDataEventArgs;
   use Stimulsoft\Report\StiReport;

   function beginProcessData(StiDataEventArgs $args) {
      $args->connectionString = str_replace('Pwd=;', 'Pwd=******;', $args-
      >connectionString);
   };

   $report = new StiReport();
   $report->onBeginProcessData->append('beginProcessData');
   $report->onBeginProcessData->append(beginProcessData);
   $report->loadFile('reports/SimpleList.mrt');
   $report->render();
?>

...

<script>
   function beginProcessData(args) {
      args.queryString = args.queryString.replace("TableName", "Products");
   }
</script>
```

Information

Some events can only be triggered on the client-side using JavaScript and don't have the option to trigger on the PHP server-side, or, they can only be triggered on the PHP server-side. When adding a function of an unsupported type to such an event, no error will occur; the added function simply will not be executed. All supported options are listed in the Report Engine Events section.

Encrypting data transferred to the PHP server

To prevent data theft by malicious actors, we recommend using the HTTPS protocol, which is sufficient in most cases. In addition to this, by default, all transmitted data is passed through a special encoding algorithm and sent to the server in encrypted form. This helps protect sensitive data, such as the login and password in the connection string, from curious users working with your application.

However, if encryption isn't necessary, or if you need to display the original request data for debugging purposes, it is possible to disable encryption. To do this, set the `$encryptData` property to `false` in the event handler, after which all data will be transmitted in JSON format.

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->handler->encryptData = false;
   $report->process();
?>
```

Passing GET parameter values to the PHP event handler

There's a feature that allows the automatic transfer of all GET request parameter values to the event handler, where their values can be accessed in all events. To enable this feature, simply set the `passQueryParameters` property to `true`. After this, all GET request parameters will be passed with each request to the event handler.

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->handler->passQueryParameters = true;
   $report->process();
?>
```

10.1.8 Connecting Data Files

Typically, data source connection parameters are stored within the report template. For working with file data sources such as XML, JSON, Excel, and CSV, no additional actions are required as all the necessary algorithms are included in the report generator script. However, if needed, you can use other methods to connect data by leveraging the report generator's JavaScript functions. This can be done by using the `onBeforeRender` event of the report object. Data can be directly loaded into a special `DataSet` object, which is used to store it. The `DataSet` contains functions for loading data from XML/XSD and JSON files. After loading the files, you need to call the `regData()` function of the report object to connect the data, passing the prepared `DataSet` object as an argument.

Example of loading data from an XML file using an XSD schema:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onBeforeRender = 'beforeRender';
   $report->render();
?>

function onBeforeRender(args) {
   let dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
   dataSet.readXmlSchemaFile("Demo.xsd");
   dataSet.readXmlFile("Demo.xml");

   let report = args.report;
   report.regData(dataSet.dataSetName, "", dataSet);
   report.dictionary.synchronize();
}
```

Information

Data scheme loading is not essential. If you want to use data scheme, you should add it before XML data loading.

Example of loading data from a JSON file:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onBeforeRender = 'beforeRender';
   $report->render();
?>

function onBeforeRender(args) {
   let dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
   dataSet.readJsonFile("Demo.json");

   let report = args.report;
   report.regData(dataSet.dataSetName, "", dataSet);
   report.dictionary.synchronize();
}
```

The full example code is available on GitHub.

In addition to the `readXmlFile()` and `readJsonFile()` functions, there are also `readXml()` and `readJson()` functions, which accept data in the form of a string or object.

Information

The function `report.dictionary.synchronize()` is necessary for synchronizing the connected data with the report template's data dictionary. When this function is called, the report dictionary will be created based on the structure of the data loaded into the `DataSet`. The synchronization function is not required if the dictionary is pre-created and its structure matches the connected data.

Data loading event

To view and modify file data connection parameters before loading, you need to define the `onBeginProcessData` event for the component, for example:

index.php

```
<?php
```

```
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onBeginProcessData = 'beginProcessData';
   $report->render();
?>

function onBeginProcessData(args) {
   if (args.connection == "MyJsonConnection")
      args.pathData = "Data/Demo.json";
}
```

The event arguments will include information about the connection to the file data source, specifically, the connection name and type in the report template, as well as the path to the data file. A detailed description of the available property values transmitted in the event arguments can be found in the Report Engine Events section.

You are allowed to change the path to the data file. In this case, after the event completes, the report generator will request the file from the new path specified in the arguments. For example, if you need to change the path to the JSON data file for the specified connection:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onBeginProcessData = 'beginProcessData';
   $report->render();
?>

function onBeginProcessData(args) {
   if (args.connection == "MyJsonConnection")
      args.pathData = "Data/Demo.json";
}
```

Information

The `onBeginProcessData` event will be invoked twice for an XML data source: first time to read an XSD scheme, second time to read an XML data file.

Data Processing Event

To view or modify the loaded data before connecting it and generating the report, you need to define the `onEndProcessData` event for the component.

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onEndProcessData = 'onEndProcessData';
   $report->renderHtml();
?>

function onEndProcessData(args) {
   let dataSet = args.dataSet;
}
```

In the event arguments, information about the file data source connection will be passed, including the connection name and type saved in the report template, as well as the prepared `DataSet` object containing the tables and rows of data obtained from the file source. A detailed description of the available property values passed in the event arguments is available in the Report Engine Event section.

Loading file data on the PHP server-side

Sometimes it's necessary to control file data loading on the server-side or, for example, create a data array using code. For this, instead of specifying the file path, you can specify the path to a PHP script or service that contains the logic for fetching the data. A simple PHP data loading script might look like this:

Example of a simple PHP data loading script:

json.php

```
<?php
   echo file_get_contents('Data/Demo.json');
```

```
?>
```

In this case, the data path in the report template should be set to the URL of this file, for example:

File Data Source

```
https://localhost/data/json.php
```

Using variables in file data

It is possible to use variables or expressions when specifying the path to the file data source. Variables or expressions are defined in curly braces. You can use multiple expressions anywhere in the data file path, for example:

File Data Source

```
https://localhost/data/{VariableJsonFileName}.json
https://localhost/data/json.php?id={VariableId}
https://localhost/{VariableCategory}/{VariableId}
```

This way, one data source can be transformed into REST syntax, eliminating the need to create multiple similar data sources for retrieving similar data. Combined with server-side PHP logic and report generator events, this can make the data source even more flexible.

Using Odata

You can also use data from OData stores to create reports. In this case, you must perform authentication using a username, password, or token. The authentication parameters are specified in the OData connection string, separated by a ";" delimiter.

index.php

```
// Authorization using a user account
var oDataDatabase = new
Stimulsoft.Report.Dictionary.StiODataDatabase("OData", "OData", "https://
services.odata.org/V4/Northwind/
```

```
Northwind.svc;AddressBearer=adress;UserName=UserName;Password=Password;Cli
ent_Id=Your Client ID", false, null);

// Authorization using a user token
var oDataDatabase = new
Stimulsoft.Report.Dictionary.StiODataDatabase("OData", "OData", "https://
services.odata.org/V4/Northwind/Northwind.svc;Token=Enter your token",
false, null);

report.dictionary.databases.add(oDataDatabase);
report.dictionary.synchronize();

// Query with data filter
var productsDataSource =
report.dictionary.dataSources.getByName("Products");
if (productsDataSource != null) productsDataSource.sqlCommand =
"Products?$filter=ProductID eq 2";
```

10.1.9 Connecting SQL Data Adapters

The report generator allows the use of data from various SQL sources for building reports. Since pure JavaScript doesn't have built-in methods for working with remote databases, this functionality is implemented through server-side PHP code. No additional actions are required to work with SQL data sources, as all data adapters are already connected and configured.

Connection creation event

If you need to control all possible parameters for connecting to a database, the `onDatabaseConnect` event is available. The event arguments will include all necessary parameters for connecting to the SQL data source, as well as the type and name of the database driver used. All connection parameters can be modified. Additionally, it's possible to pass an already established database connection in the event arguments. A detailed description of the available property values passed in the event arguments can be found in the Report Engine Events section.

Example of creating a connection to a MySQL database with a private SSL key:

index.php

```
<?php
  use Stimulsoft\Events\StiConnectionEventArgs;
  use Stimulsoft\Report\StiReport;

  $report = new StiReport();
  $report->onDatabaseConnect = function (StiConnectionEventArgs $args)
```

```
    {
        $args->link = mysqli_init();
        mysqli_ssl_set($args->link, null, null, "./private/cert.pem", null,
        null);
        $args->link = mysqli_real_connect(
            $args->link, $args->info->host, $args->info->userId, $args->info-
            >password,
            $args->info->database, $args->info->port, NULL, MYSQLI_CLIENT_SSL);
    };

    $report->render();
?>
```

Data loading event

If you need to process the parameters used for connecting to the database, the `onBeginProcessData` event is available. The event arguments will include all necessary parameters for connecting to the SQL data source, as well as the SQL query parameters. All connection parameters can be modified both on the client-side through JavaScript and on the server-side through PHP. A detailed description of the available property values passed in the event arguments is available in the Report Engine Events section.

Example of modifying an SQL query on the JavaScript client-side and the database connection string on the PHP server-side:

index.php

```
<?php
    use Stimulsoft\Events\StiDataEventArgs;
    use Stimulsoft\Report\StiReport;

    function beginProcessData(StiDataEventArgs $args) {
        if ($args->connection == 'MyConnectionName')
            $args->connectionString =
            'Server=localhost;Database=test;uid=root;password=******;';
    };

    $report = new StiReport();
    $report->onBeginProcessData->append(beginProcessData);
    $report->onBeginProcessData->append('beginProcessData');
    $report->render();
?>

...

<script>
    function beginProcessData(args) {
        if (args.dataSource == "MyDataSource")
```

```
        args.queryString = "SELECT * FROM ProductsTable";
    }
</script>
```

The full example is available on GitHub.

Thus, in the `onBeginProcessData` event, you can get the database type, connection name, and data source, as well as retrieve and, if necessary, modify the connection string and SQL query. If any property values are changed in the event arguments on the PHP server-side, the modified data will not be sent to the client. This allows the use of sensitive information such as login, password, table names, and prefixes.

Data processing event

The `onEndProcessData` event is provided for viewing or modifying the loaded data before registering it and generating the report. In the event arguments, connection parameters for the SQL data source are passed, as well as the result of the query execution, containing the column names, data types, and rows of data obtained from the SQL source. A detailed description of the available properties in the event arguments can be found in the Report Engine Events section.

Here's an example of the result of an executed SQL query, where the result already contains data prepared for the report:

index.php

```
<?php
    use Stimulsoft\Events\StiDataEventArgs;
    use Stimulsoft\Report\StiReport;

    function endProcessData(StiDataEventArgs $args) {
        $args->result->columns = ['id', 'username', 'phone'];
        $args->result->types = ['int', 'string', 'string'];
        $args->result->rows = [
            [1, 'Mario Pontes', '555-6874'],
            [2, 'Helen Bennett', '555-2376']
        ];
    };

    $report = new StiReport();
    $report->onEndProcessData->append(endProcessData);
    $report->onEndProcessData->append('endProcessData');
    $report->render();
```

```
?>

...

<script>
   function endProcessData(args) {
      args.result.columns = ["id", "username", "phone"];
      args.result.types = ["int", "string", "string"];
      args.result.rows = [
         [1, "Mario Pontes", "555-6874"],
         [2, "Helen Bennett", "555-2376"]
      ];
   }
</script>
```

The available properties of the SQL query result object are in the table:

Name	Description
count	Total number of columns in the SQL source table.
columns	Column names of the SQL source table.
types	Column types of the SQL source table, converted to recognized types for the report generator.
rows	Data rows from the SQL source table, represented as an array of arrays containing all rows from the table.

All data in the SQL query result can be modified both on the client-side JavaScript and on the PHP server-side. The number of columns and data types must match to avoid incorrect data interpretation by the report generator.

Using parameters in SQL queries

SQL queries can use parameters. For this, parameters need to be added to a special collection in the data source, with each parameter assigned a type and a default value. These parameters can then be used in the SQL query, for example:

SQL Data Source

```
SELECT * FROM @Parameter1 WHERE UserID = @Parameter2
```

All parameter values are stored in the data source as a collection. The collection is an array of objects containing the parameter name, its type, and value. Here is an example of the parameter array structure on the client-side JavaScript:

index.php

```
args.parameters = [
   {
      name: "ParameterString",
      type: 752,
      typeName: "Text",
      value: "Text value"
   },
   {
      name: "ParameterInt",
      type: 3,
      typeName: "Int32",
      value: 20
   }
];
```

To access query parameters, you can use the `onBeginProcessData` event, where the parameter collection is passed in the event arguments. You are allowed to modify the values of any parameters in the collection. Here's an example of changing the same parameter value on both the client-side JavaScript and PHP server-side:

index.php

```
<?php
   use Stimulsoft\Events\StiDataEventArgs;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onBeginProcessData->append('
      args.parameters["Parameter1"] = "TableName";
   ');

   $report->onBeginProcessData->append(function (StiDataEventArgs $args) {
      $args->parameters['Parameter1']->value = 'TableName';
```

```
});

$report->render();
?>
```

When modifying query parameter values, the type of the new value must match the type of the parameter being changed. Otherwise, the execution of the SQL query may return incorrect data or trigger an internal execution error.

Information

When changing parameter values on the PHP server-side, the modified values will not be passed to the client-side, so confidential data can be used for these values.

If the report uses multiple data sources, you should check parameters before assigning them. Otherwise, a PHP script execution error may occur if a parameter is missing from the current data source. For example, if the report contains two data sources, both having one common parameter and the second data source having an additional unique parameter:

index.php

```
$report->onBeginProcessData->append(function (StiDataEventArgs $args) {
   $args->parameters['Parameter1']->value = 'TableName';

   if ($args->dataSource == 'DataSource2')
      $args->parameters['Parameter2']->value = 10;
});
```

Using a report variable as an SQL parameter

It's possible to use a variable as an SQL parameter. To do this, simply enable the **Allow using as SQL parameter** property in the report variable editor. Once this is done, the variable can be used in any SQL query. The syntax will be the same as when using parameters from a data source.

Information

Such a variable will only be passed in the parameter collection if it is used in the query. Parameters from the data source collection are always passed, even if they aren't used in the query.

Escaping parameter values

All parameter values will be automatically escaped to prevent the possibility of SQL injections and to maintain query execution security. If escaping isn't required, and you are managing parameter value security yourself, automatic escaping can be disabled. To do this, simply set the `escapeQueryParameters` property to `false` in the event handler:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->handler->escapeQueryParameters = false;
?>
```

Once this property is set, the use of parameters becomes unsafe, and you must strictly control the values before executing SQL queries.

Information

Escaping is applied only to SQL query parameters and variables used as parameters. If a variable is used as an expression, i.e., inside curly braces e.g., `{VariableName}`, escaping will not be applied in any case. A detailed description of working with variables can be found in the Working with Report Variables section.

10.1.10 Work with Report Variables

The report generator allows you to use variables in expressions, queries, filters, and other report elements. You can preview and modify variable values from code

before building the report.

Access to values of variables from a code

The report generator provides an easy way to directly access variables in the report template through the data dictionary. To do this, you need to define the `onBeforeRender` event for the report object, in which the report object will be passed as an argument on the JavaScript client-side. A detailed description of the available properties passed in the event arguments can be found in the Report Engine Events section.

To access a report variable, you should use the `getByName()` JavaScript function from the variable collection in the report's data dictionary. To modify the value of a variable, simply assign a new value to the `value` property. There's no need to check the variable's type as type conversion will be handled automatically.

Example of modifying a string and an integer value of selected variables on the JavaScript client- side:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onBeforeRender = 'beforeRender';
   $report->loadFile('reports/Variables.mrt');
   $report->render();
?>

<script>
   function beforeRender(args) {
      let report = args.report;

      let variableString =
      report.dictionary.variables.getByName("VariableString");
      variableString.value = "Text value";

      let variableInt =
      report.dictionary.variables.getByName("VariableInt");
      variableInt.value = "20";
   }
</script>
```

Direct access to variables in the report template on the PHP server-side isn't supported.

Information

Variable values will be changed within the report template, and when the report is saved, the changes will be stored in the file. To modify variable values without altering the template, you can use the `onPrepareVariables` event, which is triggered when preparing the report's variable values before building the report.

The event of preparing values of variables

The report generator allows easy access to variables before the report is built. To do this, define the `onPrepareVariables` event for the report object. The event arguments will contain a collection of report variables, including their types and values. If a variable is initialized as an expression, the calculated value of the expression will be passed into the collection. A detailed description of the available properties passed in the event arguments can be found in the Report Engine Events section.

On the JavaScript client-side, the collection of variables is represented as an array of objects, each containing the variable's name, type, and value. It's permissible to change the values of variables, but the type of the new value must match the type of the variable being modified.

index.php

```
<script>
   function prepareVariables(args) {
      args.variables = [
         {
            name: "VariableString",
            type: "String",
            value: "Text value"
         },
         {
            name: "VariableInt",
            type: "Int32",
            value: 20
         }
     ]};
</script>
```

Example of modifying a variable on the JavaScript client-side:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onPrepareVariables = 'prepareVariables';
   $report->loadFile('reports/Variables.mrt');
   $report->render();
?>

...

<script>
   function prepareVariables(args) {
      let variables = args.variables;

      variables.find(item => item.name == "VariableString").value = "Text
      value";
      variables.find(item => item.name == "VariableInt").value = 20;
   }
</script>
```

On the PHP server-side, it's also possible to change the values of variables, ensuring that the new value matches the type of the variable being modified. Additionally, it is possible to create a new report variable if needed. The collection passed to the client will only contain variables whose values have been changed, as well as any newly created variables.

Example of modifying a variable on the PHP server-side:

index.php

```
<?php
   use Stimulsoft\Events\StiVariablesEventArgs;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->onPrepareVariables = function (StiVariablesEventArgs $args) {
      $args->variables['VariableString']->value = 'Text value';
      $args->variables['VariableInt']->value = 20;
   };
   $report->loadFile('reports/Variables.mrt');
   $report->render();
```

```
?>
```

The full example code is available on GitHub.

To modify the value of a simple variable, you just need to replace the value in the variable collection. The value must be of the same type as the original. Variables of the `DateTime` type are passed as string values in the format "YYYY-MM-dd HH-mm-ss".

index.php

```
$args->variables['VariableString']->value = 'Value from Server-Side';
$args->variables['VariableInt']->value = 123;
$args->variables['VariableDecimal']->value = 123.456;
$args->variables['VariableDateTime']->value = '2021-03-20 22:00:00';
```

To modify a variable of the `Range` type, you need to use the nested `value->from` and `value->to` values of the selected variable in the collection. The format for each of these two values is the same as for a simple variable:

index.php

```
$args->variables['VariableStringRange']->value->from = 'Aaa';
$args->variables['VariableStringRange']->value->to
```

For modifying a variable of the `List` type, you can access the list value by its index. It's also allowed to set the entire list at once using a prepared array:

index.php

```
$args->variables['VariableStringList']->value[0] = 'Test';
$args->variables['VariableStringList']->value = ['1', '2', '2'];
```

To create a new variable not defined in the report, you need to assign a prepared

associative array in the format `['value' => 'New Value']` to the variable collection, using the new variable name. After that, the variable can be used for report generation, though it will not be saved in the report template.

handler.php

```
$args->variables['NewVariable'] = ['value' => 'New Value'];
```

Information

If a variable is used in an SQL query as an expression, written in curly braces like `{VariableName}`, its value will not be automatically escaped. You must ensure the security of the values yourself, or use the variable as a query parameter, for example `@VariableName`. Detailed information on parameter handling can be found in the Connecting SQL Data Adapters section.

Report variables transferred in a URL query

The report generator has the ability to automatically assign values to variables passed in a URL request. To enable this, you need to set the `passQueryParametersToReport` property to true in the event handler:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->handler->passQueryParametersToReport = true;
   $report->process();
?>
```

All other actions will be handled automatically by the report generator. If the variable exists in the report, its value will be updated to the value from the URL request. If the variable doesn't exist in the report, it will be created for the purpose of report generation, though the new variable will not be saved in the report template. Variable names passed in the URL request are case-insensitive.

Information

All variable values will be assigned before the `onPrepareVariables` event is triggered, so within this event, you can further control and adjust the assigned values if necessary.

10.1.11 Connecting Custom Fonts

The report generator allows you to load custom fonts from a file or a directory. This is achieved using the static class `StiFontCollection`, which provides the necessary functions for working with fonts.

To connect a font file, use the `addFontFile` function. You need to specify the path to the font file and, optionally, the font name and style. If the font name or style is not specified, the parameters from the font file will be used. Here's an example of connecting a font file:

report.php

```
<?php
   use Stimulsoft\Enums\FontStyle;
   use Stimulsoft\StiFontCollection;

   StiFontCollection::addFontFile('Roboto-Black.ttf', 'Roboto',
   FontStyle::Bold);
?>
```

Additionally, you can load all fonts from a single directory. To do this, specify the directory containing the fonts using the `setFontsFolder` function. For example:

report.php

```
<?php
   use Stimulsoft\StiFontCollection;

   StiFontCollection::setFontsFolder('/fonts');
?>
```

10.1.12 Printing Report from Code

The report generator provides the ability to print a report from code. You can use the `print()` method of the report object for this purpose. Here's an example of invoking the print dialog for a pre-built report:

index.php

```
<?php
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $report->render();
   $report->print();
   $report->printHtml();
?>
```

The full example code is available on GitHub.

Information

Printing a report doesn't automatically trigger the report generation, so you need to call the `render()` method beforehand to build the report for a loaded report template. For completed documents (already built reports), this method is not necessary.

By default, all pages of the built report will be printed. There's also an option to specify a particular page or range of pages for printing. You can pass the desired value as a parameter to the `print()` function, which can be a specific page or a range of pages. Here's an example of printing specified pages of the report:

index.php

```
<?php
   $report->print(5);
   $report->print('1,3-8');
?>
```

10.1.13 Export Report from Code

The report generator allows exporting reports or dashboards to various formats. The table below lists all the available export formats for reports and dashboards:

Export format	Reports	Dashboards
Document (Snapshot)	+	+
Adobe PDF	+	+
XPS (XML Paper Specification)	+	-
Microsoft PowerPoint	+	-
HTML	+	+
HTML5	+	-
Text	+	-
Microsoft Word	+	-
Microsoft Excel	+	+
OpenDocument Writer	+	-
OpenDocument Calc	+	-
RTF (Rich Text Format)	+	-

Data format	Reports	Dashboards
CSV (Comma Separated Value)	+	+
JSON (JavaScript Object Notation)	+	+
XML (Extensible Markup Language)	+	+
DBF (dBase/FoxPro)	+	+
DIF	+	+
SYLK (Symbolic Link)	+	+

Image format	Reports	Dashboards

PNG (Portable Network Graphics)	+	+
JPEG (Joint Photographic Experts Group)	+	+
GIF (Graphics Interchange)	+	+
TIFF (Tagged Image File Format)	+	+
SVG (Scalable Vector Graphics)	+	+
SVGZ (Compressed SVG)	+	+
PCX (Picture Exchange)	+	+
BMP (Windows Bitmap)	+	+

To export a report, you need to use the `exportDocument()` method of the report object.

index.php

```
<?php
   use Stimulsoft\Export\Enums\StiExportFormat;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $report->render();
   $report->exportDocument(StiExportFormat::Pdf);
   $report->printHtml();
?>
```

Full example code is available on GitHub.

Information

Exporting a report doesn't automatically trigger the report generation, so for a loaded report template, you must first call the `render()` method to build the report. For completed documents (already built reports), this method isn't required.

As arguments for the `exportDocument()` method, you must specify the desired export format from the `StiExportFormat` enumeration, and optionally, the export settings. The available formats are listed in the table below:

Name	Description
`StiExportFormat::Document`	Saving the document (built report).
`StiExportFormat::Pdf`	Saving the document (built report).
`StiExportFormat::Xps`	Saving in XPS (XML Paper Specification) format.
`StiExportFormat::PowerPoint`	Saving in Microsoft PowerPoint format.
`StiExportFormat::Html`	Saving in HTML format.
`StiExportFormat::Html5`	Saving in HTML5 format using SVG markup elements.
`StiExportFormat::Text`	Saving in text format.
`StiExportFormat::Word`	Saving in Microsoft Word format.
`StiExportFormat::Excel`	Saving in Microsoft Excel format.
`StiExportFormat::Odt`	Saving in OpenDocument Writer format.
`StiExportFormat::Ods`	Saving in OpenDocument Calc format.
`StiExportFormat::Rtf`	Saving in RTF (Rich Text Format) format.
`StiExportFormat::Csv`	Saving in CSV (Comma Separated Values) data format.
`StiExportFormat::Json`	Saving in JSON (JavaScript Object Notation) data format.
`StiExportFormat::Xml`	Saving in XML (Extensible Markup Language) data format.

StiExportFormat::Dbf	Saving in DBF (dBase/FoxPro) data format.
StiExportFormat::Dif	Saving in DIF data format.
StiExportFormat::Sylk	Saving in SYLK (Symbolic Link) data format.
StiExportFormat::ImagePng	Saving in PNG (Portable Network Graphics) image format.
StiExportFormat::ImageJpeg	Saving in JPEG (Joint Photographic Experts Group) image format.
StiExportFormat::ImageGif	Saving in GIF (Graphics Interchange) image format.
StiExportFormat::ImageTiff	Saving in TIFF (Tagged Image File Format) image format.
StiExportFormat::ImageSvg	Saving in SVG (Scalable Vector Graphics) image format.
StiExportFormat::ImageSvgz	Saving in SVGZ (Compressed SVG) image format.
StiExportFormat::ImagePcx	Saving in PCX (Picture Exchange) image format.
StiExportFormat::ImageBmp	Saving in BMP (Windows Bitmap) image format.

After exporting the report, the resulting data stream will be sent to the browser for download as a file. The file name and MIME type will be determined automatically. There's also an option to display the exported report directly in the browser window (only for PDF, HTML, and images) by setting the `openAfterExport` argument to `true`.

index.php

```
<?php
   $report->exportDocument(StiExportFormat::Pdf, null, true);
?>
```

Exporting a report on the server-side

To switch the report generator to server-side mode, set the report's engine property to `StiEngineType::ServerNodeJS`. All other events and methods for working with the report remain the same as when exporting on the client-side. When exporting on the server-side, the `exportDocument()` method will return a byte stream of the exported report.

Example of exporting a report to text format on the server-side:

index.php

```
<?php
   use Stimulsoft\Export\Enums\StiExportFormat;
   use Stimulsoft\Report\Enums\StiEngineType;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->engine = StiEngineType::ServerNodeJS;
   $report->loadFile('reports/SimpleList.mrt');
   $report->render();
   $result = $report->exportDocument(StiExportFormat::Text);
?>
```

Full example code is available on GitHub.

It's possible to automatically save the exported report as a file on the server. To do this, specify the save path in the `filePath` argument of the export method. In this case, the `exportDocument()` method will return a boolean result indicating whether the export and file save were successful.

Example of exporting a report to Adobe PDF format on the server-side and saving the result to a specified directory:

index.php

```
<?php
   use Stimulsoft\Export\Enums\StiExportFormat;
   use Stimulsoft\Report\Enums\StiEngineType;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->engine = StiEngineType::ServerNodeJS;
```

```
    $report->loadFile('reports/SimpleList.mrt');
    $report->render();
    $exportedFilePath = 'reports/SimpleList.pdf';
    $result = $report->exportDocument(StiExportFormat::Pdf, null, false,
    $exportedFilePath);
?>
```

Export report settings

When exporting a report from code, you can set the necessary export settings. Each export format has its own settings class, as listed in the table below:

Report Export Format	Settings class
StiExportFormat::Document	No settings are provided.
StiExportFormat::Pdf	StiPdfExportSettings()
StiExportFormat::Xps	StiXpsExportSettings()
StiExportFormat::PowerPoint	StiPowerPointExportSettings()
StiExportFormat::Html StiExportFormat::Html5	StiHtmlExportSettings()
StiExportFormat::Text	StiTxtExportSettings()
StiExportFormat::Word	StiWordExportSettings()
StiExportFormat::Excel	StiExcelExportSettings()
StiExportFormat::Odt	StiOdtExportSettings()
StiExportFormat::Ods	StiOdsExportSettings()
StiExportFormat::Rtf	StiRtfExportSettings()
StiExportFormat::Csv StiExportFormat::Json StiExportFormat::Xml StiExportFormat::Dbf StiExportFormat::Dif StiExportFormat::Sylk	StiDataExportSettings()

StiExportFormat::ImagePng StiExportFormat::ImageJpeg StiExportFormat::ImageGif StiExportFormat::ImageTiff StiExportFormat::ImageSvg StiExportFormat::ImageSvgz StiExportFormat::ImagePcx StiExportFormat::ImageBmp	StiImageExportSettings()

Dashboard Export Format	Settings class
StiExportFormat::Pdf	StiPdfDashboardExportSettings()
StiExportFormat::Html	StiHtmlDashboardExportSettings()
StiExportFormat::Excel	StiExcelDashboardExportSettings()
StiExportFormat::Csv	StiDataDashboardExportSettings()
StiExportFormat::ImageSvg	StiImageDashboardExportSettings()

To apply the export settings, simply pass the prepared settings object as a parameter to the report's export method. All other actions will be handled automatically.

Example of exporting a report to Adobe PDF format on the server-side, specifying company details in the settings and disabling embedded fonts:

index.php

```
<?php
   use Stimulsoft\Export\Enums\StiExportFormat;
   use Stimulsoft\Export\StiPdfExportSettings;
   use Stimulsoft\Report\Enums\StiEngineType;
   use Stimulsoft\Report\StiReport;

   $report = new StiReport();
   $report->engine = StiEngineType::ServerNodeJS;
```

```
    $report->loadFile('reports/SimpleList.mrt');
    $report->render();

    $settings = new StiPdfExportSettings();
    $settings->creatorString = 'My Company Name';
    $settings->keywordsString = 'SimpleList PHP Report Export';
    $settings->embeddedFonts = false;

    $result = $report->exportDocument(StiExportFormat::Pdf, $settings);
?>
```

Full example code is available on GitHub.

Information

The class of the export settings object must match the export format. Otherwise, the format returned by the `getExportFormat()` method of the export settings object will take precedence.

10.1.14 Engine Events

The report generator supports events that provide the ability to perform necessary operations before certain actions—both on the client-side JavaScript, including the Node.js platform, and on the PHP server-side.

To trigger an event on the client-side JavaScript, you need to add the name of the JavaScript function defined in the current HTML template to the event. If there's no HTML template, or when generating or exporting a report on the server-side using the Node.js platform, instead of the function name, the event should be assigned the function itself as a string or lines of code. The event arguments will be passed in the function parameters or in a pre-defined args variable, which can be used in the event code.

To trigger an event on the PHP server-side, you need to add the PHP function itself, previously defined in the code, to the event. Assigning an anonymous PHP function to the event is also allowed. The event arguments will be passed in the function parameters.
You can add any number of functions, both client-side and server-side, to a single event. In this case, you need to use the special `append()` method instead of assigning a function, and pass the function as a parameter. The event handler will group the functions by type and execute them in the order they were added.

Here's an example of different ways to add functions of various types to an event:

index.php

```
<?php
   use Stimulsoft\Events\StiDataEventArgs;
   use Stimulsoft\Events\StiVariablesEventArgs;
   use Stimulsoft\Report\StiReport;

   function prepareVariables(StiVariablesEventArgs $args) {
      $variables = $args->variables;
   };

   $report = new StiReport();
   $report->onPrepareVariables->append(prepareVariables);
   $report->onPrepareVariables->append('prepareVariables');
   $report->onBeginProcessData = function(StiDataEventArgs $args) {
      $args->connectionString =
      'Server=localhost;Database=test;uid=root;password=******;';
   };

   $report->onBeforeRender = 'args.report.dictionary.clear();';
   $report->onAfterRender = 'afterRender';

   $report->loadFile('reports/Variables.mrt');
   $report->render();
?>

...

<script>
   function prepareVariables(args) {
      let variables = args.variables;
   }

   function afterRender(args) {
      alert("The report rendering is completed.");
   }
</script>
```

Depending on the event, it can be triggered on both the client-side JavaScript and the PHP server-side simultaneously, or only on the client-side JavaScript, or only on the PHP server-side. This is due to the architecture of the components—the report generator uses a JavaScript core for building and exporting reports, which operates on the client-side, while server-side PHP code is used for data handling. These two components do not have direct access to each other, so events work separately, and data transfer is handled by the event handler. The description of each event will specify which type can be used. A detailed description of the event handler and

usage examples can be found in the Event Handler section.

Infformation

All events that work on the client-side JavaScript also work on the Node.js server-side since the same JavaScript report core is used in that case.

The report generator supports the following events:

- `onDatabaseConnect`
- `onBeforeRender`
- `onAfterRender`
- `onBeginProcessData`
- `onEndProcessData`
- `onPrepareVariables`

onDatabaseConnect
[-] JavaScript [+] PHP

The event is triggered before connecting to the database after all parameters have been received. A detailed description and usage examples can be found in the SQL Data Adapter Connections section. The table below lists the properties passed in the event arguments on the PHP server-side:

Name	Description
`event`	The identifier of the current event for this specific event has the value `StiEventType::DatabaseConnect`
`sender`	The component that initiated this event can have the following types: • `StiViewer`

	• `StiDesigner`
`database`	The database type can take one of the values from the `StiDatabaseType` enumeration.
`driver`	The name of the PHP database driver being used.
`info`	The database connection parameters obtained from the connection string.
`link`	The database connection ID. By default, it's set to `null`. In this case, the connection will be created by the data adapter.

onBeforeRender
[+] JavaScript [-] PHP

The event is triggered before the report is generated. A detailed description and usage examples can be found in the Rendering Report section.

The table below lists the properties passed in the event arguments on the client-side JavaScript:

Name	Description
`event`	The identifier of the current event has the value `BeforeRender`.
`sender`	The component identifier that initiated this event can have the following values: • `"Report"`
`report`	The current report object.

onAfterRender
[+] JavaScript [-] PHP

The event is triggered after the report is generated. A detailed description and usage examples can be found in the Rendering Report section.

The table below lists the properties passed in the event arguments on the client-side JavaScript:

Name	Description
`event`	The identifier of the current event has the value `AfterRender`.
`sender`	The component identifier that initiated this event can have the following values: • `"Report"`
`report`	The current report object.

onBeginProcessData

The event is triggered before requesting the data necessary for building the report. Detailed descriptions and usage examples can be found in the Connecting Data Files and Connecting SQL Data Adapters sections.

The table below lists the properties passed in the event arguments on the client-side JavaScript:

Name	Description
`event`	The identifier of the current event has the value `"BeginProcessData"`.
`sender`	The component that initiated this event can have the following values: • `Report` • `Viewer` • `Designer`
`report`	Current report object

command	The identifier of the current command can have the following values: • TestConnection - a connection check is performed; • ExecuteQuery - data query is performed from the specified SQL source. • GetSchema - the XSD scheme is read from a file source. • GetData - data is read from a file source.
connection	The name of the current connection to a data source, specified in report template.
database	The name of the current database. It can take the following values: • "XML" • "JSON" • "Excel" • "CSV" • "MySQL" • "MS SQL" • "PostgreSQL" • "Firebird" • "Oracle" • "ODBC"
dataSource	The name of the current data source, specified in the report template. It is set only for SQL data sources.
connectionString	Connection string to the SQL data source.
queryString	The SQL query for getting data. It is used only with the ExecuteQuery

	command.
`pathData`	The path to the data source file, specified in the report template. It's set only for XML and JSON data sources.
`pathSchema`	The path to the data scheme file, specified in the report template. It is set only for XML data source.
`parameters`	The collection of parameters and their values, specified in an SQL data source.
`preventDefault`	This flag is an ability to stop further event processing by the report generator. The false value is set by default.

In the table below, you can find the list of event handler arguments on the PHP server-side.

Name	Description
`event`	The identifier of the current event for this specific event has the value `StiEventType::BeginProcessData`.
`sender`	The component that initiated this event can have the following types: • `StiReport` • `StiViewer` • `StiDesigner`
`command`	The identifier of the current command can have the following values: • `StiDataCommand::TestConnection` - a connection test is

	performed; • `StiDataCommand::RetrieveSchema` - a database schema request is executed for NoSQL data sources; • `StiDataCommand::ExecuteQuery` - a data query from the specified SQL source is executed; • `StiDataCommand::Execute` - a stored procedure is executed from the specified SQL source.
`connection`	The name of the current connection to a data source, specified in report template.
`database`	The name of the current database. It can take the following values: • `StiDatabaseType::MySQL` • `StiDatabaseType::MSSQL` • `StiDatabaseType::PostgreSQL` • `StiDatabaseType::Firebird` • `StiDatabaseType::Oracle` • `StiDatabaseType::MongoDB` • `StiDatabaseType::ODBC`
`dataSource`	The name of the current data source, specified in report template.
`connectionString`	Connection string to the SQL data source.
`queryString`	The SQL query for getting data. It is used only with the `StiDataCommand::ExecuteQuery` command.
`parameters`	A collection of parameters and their values specified in the SQL data source. The parameter values are

	always passed in their original (unescaped) form.

onEndProcessData

The event is triggered after data is loaded but before the report is generated. Detailed descriptions and usage examples can be found in the Connecting Data Files and Connecting SQL Data Adapters sections.

The table below lists the properties passed in the event arguments on the client-side JavaScript:

Name	Description
`event`	The identificator of the current event has the "`EndProcessData`" value.
`sender`	The identificator of the component, which initialized this event can take the following values: • `Report` • `Viewer` • `Designer`
`report`	Current report object.
`command`	The identificator of the current command can take the following values: • `ExecuteQuery` - data is obtained from the specified SQL source. • `GetData` - data is obtained from a file source.
`connection`	The name of the current connection to a data source, specified in report table.
`database`	The name of the current database. It can take the following values:

	• "XML" • "JSON" • "Excel" • "CSV" • "MySQL" • "MS SQL" • "PostgreSQL" • "Firebird" • "Oracle" • "ODBC"
dataSource	The name of the current data source, specified in report template. It is set only for SQL data sources.
dataSet	The prepared Stimulsoft.System.Data.DataSet object contains tables and data rows, received from a file source.
result	The collection of columns, their types and data rows, received from an SQL source.

In the table below, you can find the list of the event handler arguments on the PHP server-side.

Name	Description
event	The identifier of the current event for this specific event has the value StiEventType::EndProcessData.
sender	The identificator of the component, which initialized this event can take the following values: • StiReport

	• StiViewer • StiDesigner
command	The identifier of the current command can have the following values: • StiDataCommand::TestConnection - a connection test is performed; • StiDataCommand::RetrieveSchema - a database schema request is executed for NoSQL data sources; • StiDataCommand::ExecuteQuery - a data query from the specified SQL source is executed; • StiDataCommand::Execute - a stored procedure from the specified SQL source is executed.
connection	The name of the current connection to a data source, specified in report template.
database	The name of the current database. It can take the following values: • StiDatabaseType::MySQL • StiDatabaseType::MSSQL • StiDatabaseType::PostgreSQL • StiDatabaseType::Firebird • StiDatabaseType::Oracle • StiDatabaseType::MongoDB • StiDatabaseType::ODBC

`dataSource`	The name of the current data source, specified in report template. It is set only for SQL data sources.
`queryString`	The final SQL query with all parameters that was executed to retrieve data. Used only with the StiDataCommand::ExecuteQuery command.
`result`	The collection of columns, their types and data rows, received from an SQL source.

onPrepareVariables

The event is triggered before report generation after the report variables have been prepared. Detailed descriptions and usage examples can be found in the Working with Report Variables section.

The table below lists the properties passed in the event arguments on the client-side JavaScript:

Name	Description
`event`	The identificator of the current event has the "`PrepareVariables`" variable.
`sender`	The identificator of the component, which initialized this event can take the following values: • `Report` • `Viewer` • `Designer`
`report`	Current report object.
`variables`	The collection of report variables and their values.

`preventDefault`	This flag gives an ability to stop the further event handler by the report generator. The `false` value is set by default.

In the table below you can find the list of the event handler arguments on the PHP server-side:

Name	Description
`event`	The identifier of the current event for this specific event has the value `StiEventType::PrepareVariables`.
`sender`	The identificator of the component, which initialized this event can take the following values: • `StiReport` • `StiViewer` • `StiDesigner`
`variables`	The collection of report variables and their values.

10.2 HTML5 Viewer

The report viewer is a PHP component (`StiViewer`) designed for viewing, printing, and exporting reports and dashboards in a browser window on any computer, regardless of the operating system. The viewer supports various themes, an animated interface, bookmarks, interactive reports, in-page editing of report elements, full-screen mode, search functionality, and other essential features for report viewing.

The viewer can display a report template, a built report, or a dashboard. If a report template is used, the viewer will automatically build it using the JavaScript report generator. The operation of the generator is discussed in more detail in the Engine section.

The viewer's interface is built using HTML5, allowing it to run on almost any modern platform. The component utilizes AJAX technology to perform all actions (loading and building the report, connecting to data, page navigation and scaling, interactivity in reports, printing, exporting, etc.), eliminating the need to reload the entire page, thus improving performance and making it suitable for use in one-page applications. The JavaScript technology used for report building enables the use of virtually any server-side, even with low performance.

Information

Since both dashboards and reports use the same unified MRT template format, as well as methods for loading templates and working with data, the term "report" will be used in the documentation.

- Deployment
- Activation
- Showing Reports and Dashboards
- Localization
- Printing Report
- Report Export
- Viewing Modes
- Work with Report Variables
- Work with Bookmarks
- Dynamic Sorting, Collapsing, and Drill-Down
- Editing Report
- Sending Report By Email
- Calling Designer from Viewer
- Appearance
- Viewer Events
- Viewer Settings

10.2.1 Usage

To use the product, simply download the ZIP archive from the Downloads page on our website, unpack it, and copy the contents of the **/PHP** folder to your web server. This folder is a web project that contains all the necessary files and resources for the product to function, as well as examples of how to work with the viewer and designer.

To install the product into an existing project, just copy the `/vendor` folder from **/ PHP** to the root directory of your project, or use the Composer dependency manager by running the following console command:

console

```
composer require stimulsoft/reports-php
```

To work with dashboards, you will need to install the following package:

console

```
composer require stimulsoft/dashboards-php
```

In most cases, using only PHP code is sufficient for the product to work, enabling all the core features. For more detailed product customization and to leverage all available features, JavaScript code should be used.

To use the components in a web project, simply include the automatic script loader at the beginning of the PHP file. After that, all available PHP classes and functions for working with reports and dashboards can be utilized.

index.php

```
<?php
   require_once 'vendor/autoload.php';

...
?>
```

The `StiViewer` class is designed to display the viewer in a web project. Using this class, you can create a viewer object, configure settings, assign a report for viewing, handle requests, and manage viewer events.

Example of displaying a report in the viewer on an HTML page:

viewer.php

```
<?php
   require_once 'vendor/autoload.php';

   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->process();

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $viewer->report = $report;
?>

<html>
<head>
   <?php
      $viewer->javascript->renderHtml();
   ?>
</head>
<body>
   <?php
      $viewer->renderHtml();
   ?>
</body>
</html>
```

The full example code is available on GitHub.

In this example, the following steps are performed sequentially:

- A `StiViewer` object instance is created;
- The current request is processed;
- A `StiReport` object instance is created;
- The report template is loaded from the SimpleList.mrt file;
- The created report is assigned to the viewer;
- The necessary JavaScript and HTML code for the component is output in the HTML file template.

The `$viewer->process()` method processes the current request and, if successful, automatically returns the result to the client side. More details on this can be found in the Event Handler section.

The `$viewer->javascript->renderHtml()` method outputs the code required

to include the necessary component scripts. The `$viewer->renderHtml()` method outputs the JavaScript and HTML code of the component itself. There are several usage options, which are described in more detail in the Using the Report Engine section.

Example of a simplified viewer display without using an HTML page template:

viewer.php

```
<?php
   require_once 'vendor/autoload.php';

   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->process();

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $viewer->report = $report;

   $viewer->printHtml();
?>
```

The full example code is available on GitHub.

10.2.2 License Activation

After purchasing the product, you need to activate the license for the components you are using. There are several ways to connect the license key.

All options for component activation are described in the License Activation for the Report Engine section, and they have the same functions and parameters for invocation.

10.2.3 Showing Reports

To display a report in the viewer, simply create a `StiReport` object, load the report template into it, and assign the resulting object to the viewer. All other actions will be performed automatically, the viewer will build the report and display the first page:

viewer.php

```
<?php
   require_once 'vendor/autoload.php';

   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->process();

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $viewer->report = $report;

   $viewer->printHtml();
?>
```

The full example code is available on GitHub.

The viewer can automatically build and display both report templates and documents (built reports), so a separate call to the report building method $report->render() is not required. Detailed instructions on working with different report and document formats can be found in the Loading and Saving Reports section.

10.2.4 Localization

The viewer fully supports localization of its interface. To localize the interface to the desired language, simply set the required filename for the `localization` option in the viewer:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->localization = 'de.xml';
?>
```

The full example code is available on GitHub.

All available localization XML files are located in the resources of the installed product package. If necessary, a localization file can be loaded from any other

location by specifying the full path to the desired XML file in the `localization` option.

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->localization = '/resources/loc/de.xml';
?>
```

10.2.5 Printing Report

The viewer provides several options for printing a report, each with its own features, advantages, and disadvantages.

Print to PDF
Printing will be done by exporting the report to PDF format. The advantages include higher accuracy in the positioning and printing of report elements compared to other printing options. One disadvantage is the requirement of a browser plugin for viewing PDF files (modern browsers have a built-in PDF viewer and print functionality).

Print with preview
The report will be printed in a separate browser pop-up window in HTML format. The report can be previewed before printing or copied to another location as text or HTML code. Advantages include cross-browser compatibility and no need for installing special plugins. The downside is relatively lower accuracy in element positioning due to the limitations of HTML formatting.

Print without preview
The report will be sent directly to the printer without preview. After selecting this option, the system print dialog will appear. Since printing is done in HTML format in this mode, the print quality is similar to that of printing with preview.

Information

Printing is handled using the browser's built-in methods, so the print dialog may look different depending on the operating system and browser. Also, browsers do

not allow JavaScript to control print settings, so the required adjustments must be made in the print dialog itself.

Report print settings

When selecting the print option from the viewer's toolbar, a menu with print options appears. The component allows for enforcing a specific print mode by setting the `printDestination` property to one of the values from the `StiPrintDestination` enumeration:

Name	Description
`StiPrintDestination::Default`	The menu will show all available print options (this is the default property value).
`StiPrintDestination::Pdf`	The report will be printed in PDF format.
`StiPrintDestination::Direct`	The report will be printed directly in HTML format using the system print dialog.
`StiPrintDestination::WithPreview`	The report will be displayed in a pop-up window for preview and then printed in HTML format.

For example, if you want to set the print mode to PDF only:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Viewer\Enums\StiPrintDestination;

   $viewer = new StiViewer();
   $viewer->options->toolbar->printDestination = StiPrintDestination::Pdf;
?>
```

The viewer also allows disabling the print option entirely if it's not needed. To do this, set the `showPrintButton` property to `false`:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->toolbar->showPrintButton = false;
?>
```

Report print event

The `onPrintReport` event is triggered if any actions need to be performed before printing the report. The event arguments include the print type, page range, and the report itself. It's possible to modify the page range or report properties before printing.

Example of actions performed on the JavaScript client-side before printing a report:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->onPrintReport = 'printReport';
   $viewer->process();
?>

<script>
   function printReport(args) {
      if (args.printAction == 'PrintPdf'){
         args.pageRange.rangeType =
         Stimulsoft.Report.StiRangeType.CurrentPage;
         args.pageRange.currentPage = 1;
      }
   }
</script>
```

Example of actions performed on the PHP server-side before printing a report:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Events\StiPrintEventArgs;
   use Stimulsoft\Report\Enums\StiRangeType;
   use Stimulsoft\Viewer\Enums\StiPrintAction;

   $viewer = new StiViewer();
   $viewer->onPrintReport = function (StiPrintEventArgs $args) {
      if ($args->printAction == StiPrintAction::PrintPdf){
         $args->pageRange->rangeType = StiRangeType::CurrentPage;
         $args->pageRange->currentPage = 1;
      }
   };
   $viewer->process();
?>
```

Detailed descriptions of the available argument values can be found in the Viewer Events section.

Printing a report from code

There's also the option to print a report directly from code without using the viewer's functions. Detailed instructions for this functionality can be found in the Printing a Report from Code section.

10.2.6 Report Export

The viewer allows exporting displayed reports or dashboards into various formats. No additional viewer configuration is required for exporting. The table below lists all available export formats for reports and dashboards:

Export format	Reports	Dashboards
Document (Snapshot)	+	+
Adobe PDF	+	+
XPS (XML Paper Specification)	+	-
Microsoft PowerPoint	+	-
HTML	+	+

HTML5	+	-
Text	+	-
Microsoft Word	+	-
Microsoft Excel	+	+
OpenDocument Writer	+	-
OpenDocument Calc	+	-
RTF (Rich Text Format)	+	-

Data format	Reports	Dashboards
CSV (Comma Separated Value)	+	+
JSON (JavaScript Object Notation)	+	+
XML (Extensible Markup Language)	+	+
DBF (dBase/FoxPro)	+	+
DIF	+	+
SYLK (Symbolic Link)	+	+

Image format	Reports	Dashboards
PNG (Portable Network Graphics)	+	+
JPEG (Joint Photographic Experts Group)	+	+
GIF (Graphics Interchange)	+	+
TIFF (Tagged Image File Format)	+	+
SVG (Scalable Vector Graphics)	+	+
SVGZ (Compressed SVG)	+	+
PCX (Picture Exchange)	+	+

BMP (Windows Bitmap)	+	+

Begin export event

If any actions need to be performed before exporting a report, the `onBeginExportReport` event is used. This event is triggered after the export settings dialog is shown. The event arguments include the export type, export file name, the report itself, and the settings for the selected export format. You can modify the report properties, export settings, and file name.

Example of performing actions on the client JavaScript before exporting a report:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->onBeginExportReport = 'beginExportReport';
   $viewer->process();
?>

...

<script>
   function beginExportReport(args) {
      if (args.format == Stimulsoft.Report.StiExportFormat.Pdf) {
         args.settings.creatorString = 'My Company Name';
         args.settings.embeddedFonts = false;
      }
   }
</script>
```

Example of performing actions on the server PHP side before exporting a report:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Events\StiExportEventArgs;
   use Stimulsoft\Export\Enums\StiExportFormat;
   use Stimulsoft\Export\StiPdfExportSettings;

   $viewer = new StiViewer();
   $viewer->onBeginExportReport = function (StiReportEventArgs $args) {
```

```
    $args->fileName = "MyExportedFileName.$args->fileExtension";

    if ($args->format == StiExportFormat::Pdf) {
      /** @var StiPdfExportSettings $settings */
      $settings = $args->settings;
      $settings->creatorString = 'My Company Name';
      $settings->embeddedFonts = false;
    }
  };
  $viewer->process();
?>
```

The full example code is available on GitHub.

Detailed descriptions of the available event arguments can be found in the Viewer Events section. A detailed explanation of the available export settings is found in the Exporting a Report from Code section.

End export event

If any actions need to be performed after a report is exported but before it's saved, the `onEndExportReport` event is used. The event arguments include the export type, file name, and the byte data of the exported file. You can modify the file name and byte data of the exported file.

Example of performing actions on the client JavaScript side after exporting a report:

viewer.php

```
<?php
  use Stimulsoft\Viewer\StiViewer;

  $viewer = new StiViewer();
  $viewer->onEndExportReport = 'endExportReport';
  $viewer->process();
?>

...

<script>
  function endExportReport(args) {
    if (args.format == Stimulsoft.Report.StiExportFormat.Html) {
      let fileName = args.fileName;
      let htmlText = args.data;
    }
  }
```

```
</script>
```

Example of performing actions on the server PHP side after exporting a report:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Events\StiExportEventArgs;
   use Stimulsoft\Export\Enums\StiExportFormat;

   $viewer = new StiViewer();
   $viewer->onEndExportReport = function (StiReportEventArgs $args) {
      $fileName = $args->fileName;
      if ($args->format == StiExportFormat::Pdf) {
         $htmlText = base64_decode($args->data);
      }
   };
   $viewer->process();
?>
```

Information

Byte data is sent to the server in `Base64` encoding, so it must be decoded back to its original byte stream using a standard PHP function like `base64_decode()` before saving.

Detailed descriptions of the available event arguments can be found in the Viewer Events section.

Export settings

Sometimes it is necessary to disable unused report export formats and leave only the required ones. This simplifies the interface and makes the viewer easier to use. To disable unused export formats, simply set the corresponding viewer properties to `false`. For example:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->exports->showExportToDocument = false;
   $viewer->options->exports->showExportToWord = false;
   $viewer->options->exports->showExportToCsv = false;
?>
```

Additionally, you can completely disable the export dialog windows, and exporting will always be done with default settings. You can manage these settings in the export event. To disable the export dialogs, set the `showExportDialog` property to `false`:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->exports->showExportDialog = false;
?>
```

The full list of available options is in the Viewer Settings section.

Exporting a report from code

To export a report directly from code without using the viewer, a special method `exportDocument()` is available for the report object. Detailed instructions are provided in the Exporting a Report from Code section.

10.2.7 Viewing Modes

The viewer allows configuring various interface and report page display modes, as well as controlling the display on mobile devices.

Scrollbars

The viewer offers two report display modes: with and without scrollbars. By default, the mode without scrollbars is enabled. To enable the scrolling mode, simply set the `scrollbarsMode` property to `true`.

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->appearance->scrollbarsMode = true;
?>
```

In the first mode (without scrollbars), the viewer displays the page or report in full, automatically resizing the viewing area. If width and height are set, the viewer will crop any parts of the page that exceed these boundaries. In the second mode, scrollbars will appear when the page exceeds the viewer's dimensions, allowing users to scroll and view the entire report without cropping.

Information

When using the scrollbars mode, you need to specify the viewer's height. If no height is set, the default height will be 650 pixels.

Full-Screen mode

The viewer supports full-screen mode for viewing reports and dashboards. By default, the standard viewing mode is enabled with predefined dimensions. To enable full-screen mode, set the `fullScreenMode` property to `true`.

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->appearance->fullScreenMode = true;
?>
```

Alternatively, full-screen mode can be toggled using the button on the viewer's

control panel.

Report page display modes

The viewer supports three modes for displaying reports: page-by-page display, continuous scroll (report as a strip) and table of pages display. To control these modes, use the `viewMode` property, which accepts one of the corresponding values.

Name	Description
`StiWebViewMode::SinglePage`	Displays a single page selected from the toolbar.
`StiWebViewMode::Continuous`	Displays all report pages as a continuous strip.
`StiWebViewMode::MultiplePages`	Displays all report pages in a table format.

For example, to enable the continuous scroll mode:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Viewer\Enums\StiWebViewMode;

   $viewer = new StiViewer();
   $viewer->options->toolbar->viewMode = StiWebViewMode::Continuous;
?>
```

Mobile mode

The viewer is designed to support both desktop and mobile devices, including touchscreen functionality. To control the interface modes, use the `interfaceType` property, which accepts one of the following values:

Name	Description
`StiInterfaceType::Auto`	The viewer interface type will be automatically selected based on the

	device in use (default value).
`StiInterfaceType::Mouse`	Forced use of the standard interface for viewer control via mouse.
`StiInterfaceType::Touch`	Forced use of the `Touch` interface for control via a touchscreen monitor, where viewer interface elements are enlarged for easier control.
`StiInterfaceType::Mobile`	Forced use of the `Mobile` interface for control via a smartphone screen, where the viewer interface is simplified and adapted for mobile device control.

For example, to completely disable mobile mode:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Viewer\Enums\StiInterfaceType;

   $viewer = new StiViewer();
   $viewer->options->appearance->interfaceType = StiInterfaceType::Mouse;
?>
```

10.2.8 Work with Report Variables

The viewer includes a dedicated parameters panel for handling report variables. To add a parameter to this panel, you need to define a variable in the report that will be requested from the user. When viewing the report in the viewer, such a variable will automatically appear on the parameters panel. All types of report variables are supported, including standard variables, date and time, ranges, lists, and more.

Managing variables on the parameters panel

To perform actions before applying parameters, there's a special event called `onInteraction`, triggered during viewer interactions. The event arguments will include the type of action and the collection of variables and their values located on the parameters panel. The action type will have the string value `"Variables"`.

Here's an example of performing actions on the client JavaScript side before applying report parameters:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->onInteraction = 'interaction';
   $viewer->process();
?>

...

<script>
   function interaction(args) {
      if (args.action == "Variables") {
         let variables = args.variables;
      }
   }
</script>
```

The collection of variables is an object containing all the variables on the parameters panel and their values, for example:

viewer.php

```
var variables = {
   VariableString: "Text value",
   VariableInt: 20
}
```

It's allowed to change the values of the variables, but the new value's type must match the type of the variable being modified. A detailed description of the available argument values can be found in the Viewer Events section.

Configuring the parameters panel

If you don't need to work with variables in the viewer, you can completely disable this feature. To do so, set the `showParametersButton` property to `false`.

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->toolbar->showParametersButton = false;
?>
```

Information

In this configuration, the parameters panel will not be shown even if parameters are present in the displayed report.

Variable values during report generation

To control all report variables, the special event `onPrepareVariables` is triggered before the report is generated. A detailed description can be found in the Working with Report Variables section.

10.2.9 Work with Bookmarks

The viewer supports report bookmarks. When such a report is displayed, a bookmarks panel will appear on the left side of the page. By selecting a bookmark, the viewer will automatically navigate to the corresponding page, and the bookmarked report element will be highlighted.

Bookmarks settings

By default, the width of the bookmarks panel is set to 180 pixels, but the viewer allows you to change this value using the `bookmarksTreeWidth` property, specified in pixels.

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
```

```
   $viewer->options->appearance->bookmarksTreeWidth = 300;
?>
```

If bookmarks aren't required in the report, this feature can be disabled entirely by setting the `showBookmarksButton` property to `false`.

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->toolbar->showBookmarksButton = false;
?>
```

Information

In this case, report bookmarks will not be displayed, even if they are present in the report. This property doesn't affect printing or exporting the report.

When printing a report with bookmarks, the bookmarks tree will be hidden. If you need to print both the report and its bookmarks, set the `bookmarksPrint` property to `true`.

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->appearance->bookmarksPrint = true;
?>
```

10.2.10 Dynamic Sorting, Collapsing, and Drill-Down

In addition to variables that can be set on the parameters panel, the viewer supports other types of interactivity that enhance convenience and functionality when using the report generator. These include sorting, collapsing, and drill-down features.

Sorting

Dynamic sorting allows users to change the sorting direction in the generated report. To sort, click on a component where dynamic sorting has been enabled. Sorting can be done in the following directions: **Ascending** and **Descending**. Each time you click on the component, the direction alternates.

Multilevel sorting is also supported. To perform multilevel sorting, hold down the **Ctrl** key and click on the components you want to sort. To reset sorting, click on any sortable component without holding **Ctrl**.

Collapsing

A report with dynamic collapsing is interactive, where certain blocks can collapse or expand their content by clicking on the block's header. Collapsible/expandable elements are marked with a special icon, usually a **[-]** or **[+]** sign.

Drill-Down

When drill-down is enabled, a drill-down panel with tabs of detailed reports will appear below the main viewer panel. The report currently displayed will be highlighted. Users can close any drill-down pages that are no longer needed.

Dashboard sorting

In dashboard viewing mode, many elements allow sorting by data fields either in **Ascending** or **Descending** order.

Dashboard filtering

Filtering on a dashboard can be applied through special filtering elements as well as other data components. The filter will affect all interconnected elements on the dashboard.

Dashboard Drill-Down

As with reports, dashboards can also have detailed drill-down dashboards or reports. Additionally, for some dashboard elements, data can be drilled down at the element level.

Viewer interactivity event

No additional viewer settings are required for dynamic sorting, collapsing, or drill-down actions. However, if you want to execute actions before these interactions, the special event onInteraction is triggered when interactive actions occur in the viewer.

A detailed description of available argument values is provided in the Viewer Events section.

Name	Desccription
`InitVars`	The action occurs during the initialization of report variables requested from the user.
`Variables`	The action occurs when using variables on the parameters panel. A detailed description can be found in the Working with Report Variables section.
`Sorting`	The action occurs when using column sorting.
`DrillDown`	The action occurs when using column drill-down.
`Collapsing`	The action occurs when collapsing report blocks.
`DashboardFiltering`	The action occurs when using filters within a dashboard element.
`DashboardSorting`	The action occurs when sorting within a dashboard element.
`DashboardResetAllFilters`	The action occurs when resetting sorting and filters within a dashboard element to the values defined in the template.
`DashboardElementDrillDown`	The action occurs when using drill-down on a dashboard element.
`DashboardElementDrillUp`	The action occurs when using drill-down on a dashboard element.

The corresponding collections of parameters, `sortingParameters`, `collapsingParameters`, and `drillDownParameters`, are passed as arguments, containing data in a specific format necessary for the current interactive action. If

needed, the values of the parameter collections can be adjusted while maintaining the structure and order of the transmitted values. Here's an example of the transmitted parameter values for some interactive actions:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->onInteraction = 'interaction';
   $viewer->process();
?>

...

<script>
   function interaction(args) {
      switch (args.action) {
         case "Sorting":
            args.sortingParameters = {
               ComponentName: "Text10;false",
               DataBand: "DataBand1;DESC;CompanyName"
            };
         break;

         case "DrillDown":
            drillDownParameters = [{
               ComponentIndex: "1"
               DrillDownMode: null
               ElementIndex: "6"
               PageGuid: "b916d048d3f446dc97c356d4ff47f48f"
               PageIndex: "0"
               ReportFile: null
            }];
         break;

         case "Collapsing":
            args.collapsingParameters = {
               CollapsingStates: {
                  GroupHeaderBand1: {
                     keys: [1],
                     values: [false]
                  },
               ComponentName: "GroupHeaderBand1"
            };
         break;
      }
   }
</script>
```

A more detailed description of available argument values is provided in the Viewer

Events section.

10.2.11 Editing Report

The viewer allows editing of elements in the generated report, such as text fields and checkboxes. To enable editing, the necessary components must be marked as editable in the report template itself. No additional viewer settings are required. Once the report is displayed in the viewer, click the corresponding button on the viewer's toolbar to begin editing. After finishing the edits, click the same button again, and all changes will be applied to the report.

Information

Edited values will be applied during printing or exporting the report, but the original report will remain unchanged. Upon reloading the viewer, all values will revert to their original state.

10.2.12 Sending Report By Email

The viewer provides the ability to send a report by email. To enable this feature, you need to set the viewer's `showSendEmailButton` property to `true` and add an event handler for `onEmailReport`.

Pure JavaScript doesn't have email handling functions, so this feature uses PHP server-side functions. To send an email, in the PHP server event arguments, you must set parameters such as the login and password of the account sending the email, as well as the server settings — its address, port, and other details. This is done through the `$args->settings` property in the event arguments. This property is an object of the `StiEmailSettings` class, containing all the necessary email sending parameters.

Here is an example of sending a report by email with the minimum required parameters:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Events\StiEmailEventArgs;
   use Stimulsoft\Report\StiReport;
   use Stimulsoft\StiResult;
```

```
    $viewer = new StiViewer();
    $viewer->options->toolbar->showSendEmailButton = true;

    $viewer->onEmailReport = function (StiEmailEventArgs $args) {
        $args->settings->from = 'mail.sender@stimulsoft.com';
        $args->settings->host = 'smtp.stimulsoft.com';
        $args->settings->login = '********';
        $args->settings->password = '********';

        return StiResult::getSuccess('The Email has been sent
        successfully.');
    };

    $viewer->process();
?>
```

The full example code is available on GitHub.

Additionally, the event arguments allow you to retrieve and modify the email details (such as the subject, message, and report file name), get the report itself, and access or modify the report export settings if needed. A detailed description of available event arguments is in the Viewer Events section.

If necessary, a JavaScript event can also be triggered, where you can access data required for sending the email, the export type of the report, and the report itself, as well as export settings to modify them if needed.

Here's an example of modifying the email subject before sending:

viewer.php

```
<?php
    use Stimulsoft\Viewer\StiViewer;

    $viewer = new StiViewer();
    $viewer->options->toolbar->showSendEmailButton = true;
    $viewer->onEmailReport = 'emailReport';
    $viewer->process();
?>

...

<script>
    function emailReport(args) {
        args.settings.subject = "Invoice: " + args.settings.subject;
    }
</script>
```

When sending an email, the JavaScript event (if defined) will be triggered first, followed by the PHP server event. This allows you to validate and adjust values right before sending the email.

Example of checking and modifying the email subject on the PHP server-side before sending:

viewer.php

```
$viewer->onEmailReport = function (StiEmailEventArgs $args) {
   if (strlen($args->settings->subject ?? '') == 0)
      $args->settings->subject = "{$args->formatName} report {$args-
      >settings->attachmentName}";

   return StiResult::getSuccess('The Email has been sent successfully.');
};
```

Example of adding additional recipients to the email with the report:

viewer.php

```
$viewer->onEmailReport = function (StiEmailEventArgs $args) {
   $args->settings->cc[] = 'extra_recipient_one@stimulsoft.com';
   $args->settings->bcc[] = 'hidden_recipient_one@stimulsoft.com';
   $args->settings->bcc[] = 'hidden_recipient_two@stimulsoft.com John
   Smith';

   return StiResult::getSuccess('The Email has been sent successfully.');
};
```

Email sending settings

The viewer allows you to set default values for the email dialog. These are controlled by the properties `defaultEmailAddress`, `defaultEmailSubject`, and `defaultEmailMessage`. By default, these properties are empty.

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->toolbar->showSendEmailButton = true;
   $viewer->options->email->defaultEmailAddress =
   'recipient_address@stimulsoft.com';
   $viewer->options->email->defaultEmailSubject = 'New Invoice';
   $viewer->options->email->defaultEmailMessage = 'Please check the new
   invoice in the attachment';
?>
```

10.2.13 Calling Designer from Viewer

The viewer can invoke the report designer. A special button labeled **Design** on the viewer's toolbar is available for this purpose. By default, this button is disabled. To use this feature, you need to set the `showDesignButton` property to `true` and define the `onDesignReport` event:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();
   $viewer->options->toolbar->showDesignButton = true;
   $viewer->onDesignReport = 'designReport';
   $viewer->process();
?>

<script>
   function designReport(args) {
      window.open("designer.php?fileName=" + args.fileName);
   }
</script>
```

The full example code is available on GitHub.

A detailed description of available event arguments is in the Viewer Events section.

Information

The viewer itself doesn't launch the designer, it simply triggers the event and passes the report and file name as arguments. In the event, you can redirect to a

PHP page that hosts the report designer.

10.2.14 Appearance

The viewer offers the ability to change the visual themes of the control elements. To do this, simply set the `theme` property in the component's options:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Viewer\Enums\StiViewerTheme;

   $viewer = new StiViewer();
   $viewer->options->appearance->theme =
   StiViewerTheme::Office2022BlackGreen;
?>
```

The full example code is available on GitHub.

Currently, there are **8** available themes with different color accents, resulting in over **60** layout options. This flexibility allows you to customize the viewer's appearance to match virtually any web project design.

Additional settings
By default, the viewer has only a top toolbar, which contains all the report control elements. If needed, the toolbar can be split into an upper and a lower toolbar. The upper toolbar will contain the print and export menu, along with buttons for working with parameters and bookmarks. The lower toolbar will include page navigation elements and zoom control. To enable this mode, use the **displayMode** property, which can have the following values:

Name	Description
`StiToolbarDisplayMode::Simple`	Simple display mode, all control elements are located on a single toolbar (this is the default setting).
`StiToolbarDisplayMode::Separated`	Split display mode. The toolbar is divided into an upper section for

	interacting with the report and a lower section for interacting with the pages.

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Viewer\Enums\StiToolbarDisplayMode;

   $viewer = new StiViewer();
   $viewer->options->toolbar->displayMode = StiToolbarDisplayMode::Simple;
   $viewer->options->appearance->scrollbarsMode = true;
?>
```

Additionally, you can customize the design of the viewer's core elements. For example, you can change the font and color of the viewer's toolbar text, set the background color, specify the page border color, and more. Below is a list of available properties for customizing the viewer's appearance and their default values:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;

   $viewer = new StiViewer();

   $viewer->options->appearance->backgroundColor = 'white';
   $viewer->options->appearance->pageBorderColor = 'red';
   $viewer->options->appearance->showPageShadow = false;

   $viewer->options->toolbar->backgroundColor = 'aqua';
   $viewer->options->toolbar->borderColor = 'darkgreen';
   $viewer->options->toolbar->fontColor = 'white';
   $viewer->options->toolbar->fontFamily = 'Arial';
?>
```

For color values, you can use either standard HTML color constants or a color code in RGB format, such as `#ff2020`.

10.2.15 Viewer Events

The report viewer supports events that allow executing necessary operations before certain actions, both on the client JavaScript side and the server PHP side. Detailed descriptions of event handling can be found in the Report Engine Events section.

The viewer supports the following events:

- onDatabaseConnect
- onBeginProcessData
- onEndProcessData
- onPrepareVariables
- onOpenReport
- onOpenedReport
- onPrintReport
- onBeginExportReport
- onEndExportReport
- onInteraction
- onEmailReport
- onDesignReport

onDatabaseConnect
[-] JavaScript [+] PHP

This event is triggered before connecting to the database after receiving all the parameters. Detailed descriptions and usage examples can be found in the Connetting SQL Data Adapter section. A list of event arguments is available in the Report Engine Events section.

onBeginProcessData
[+] JavaScript [+] PHP

This event is triggered before requesting the data needed to generate the report. Detailed descriptions and usage examples can be found in the Connecting Data and Connetting SQL Data Adapter sections. A list of event arguments is available in the

Report Engine Events section.

onEndProcessData
[+] JavaScript [+] PHP

This event is triggered before requesting the data needed to generate the report. Detailed descriptions and usage examples can be found in the Connecting Data and Connetting SQL Data Adapter sections. A list of event arguments is available in the Report Engine Events section.

onPrepareVariables
[+] JavaScript [+] PHP

This event is triggered before generating the report, after preparing the report variables. Detailed descriptions and usage examples can be found in the Working with Report Variables section. A list of event arguments is available in the Report Engine Events section.

onOpenReport
[+] JavaScript [-] PHP

This event is triggered before opening a report after clicking the toolbar button. The table below contains a list of properties passed as event arguments on the JavaScript client-side:

Name	Description
`event`	Current event identifier, has the value `"OpenReport"`.
`sender`	The identifier of the component that triggered this event can take the following values: • `"Viewer"` • `"Designer"`
`report`	The current report object. The value passed in the event arguments will be `null`.

`preventDefault`	This flag allows you to stop further processing of the event by the viewer. The default value is `false`.

onOpenedReport
[+] JavaScript [+] PHP

This event is triggered after the report file is opened but before it is sent to the viewer. The table below contains a list of properties passed as event arguments on the JavaScript client-side.

Name	Description
`event`	Current event identifier, has the value "`OpenedReport`".
`sender`	The identifier of the component that triggered this event can take the following values: • "`Viewer`" • "`Designer`"
`report`	The current report object.
`preventDefault`	This flag allows you to stop further processing of the event by the viewer. The default value is `false`.

The table below contains a list of properties passed as event arguments on the JavaScript client-side and on the PHP server-side, arguments are of the `StiReportEventArgs` type.

Name	Description
`event`	Current event identifier for this event is `StiEventType::OpenedReport`.
`sender`	The component that triggered this event can have the following types:

	• StiViewer • StiDesigner
report	The current report object.

onPrintReport
[+] JavaScript [+] PHP

This event is triggered before printing the report. Detailed descriptions and usage examples can be found in the Report Printing section.

The table below contains a list of properties passed as event arguments on the JavaScript client-side.

Name	Description
event	Current event identifier, has the value "PrintReport".
sender	The identifier of the component that triggered this event can take the following values: • "Viewer" • "Designer"
report	The current report object.
printAction	Report print type can take the following values: • "PrintPdf" - print in PDF format; • "PrintWithoutPreview" - print in HTML format directly to the printer, displaying the system print dialog; • "PrintWithPreview" - print in HTML format directly to the printer, displaying the system print dialog;

pageRange	The object containing the page range settings for printing.
preventDefault	This flag allows you to stop further processing of the event by the viewer. The default value is false.

A list of event arguments is available for both JavaScript client-side and PHP server-side, StiPrintEventArgs type.

Name	Description
event	Current event identifier for this event is StiEventType::PrintReport.
sender	The component that triggered this event can have the following types: • StiViewer • StiDesigner
report	The current report object.
printAction	Report print type can take the following values: • StiPrintAction::PrintPdf - print in PDF format; • StiPrintAction::PrintWithoutPreview - print in HTML format directly to the printer, displaying the system print dialog; • StiPrintAction::PrintWithPreview - print in HTML format with a preview in a pop-up window.
fileName	The report file name for saving.
pageRange	The object containing the page range settings for printing.

onBeginExportReport
[+] JavaScript [+] PHP

This event is triggered before exporting the report, right after the export settings dialog. Detailed descriptions and usage examples can be found in the Report Export section.

The table below contains a list of properties passed as event arguments on the JavaScript client-side.

Name	Description
`event`	Current event identifier, has the value `"BeginExportReport"`.
`sender`	The identifier of the component that triggered this event can take the following values: • `"Viewer"` • `"Designer"`
`report`	The current report object.
`action`	The action that triggered the export event can take the following following values `Stimulsoft.Viewer.StiExportAction`: • `StiExportAction.ExportReport` • `StiExportAction.SendEmail`
`format`	The selected report export format. Can take the following enumeration values `Stimulsoft.Report.StiExportFormat`: • `StiExportFormat.Document`

	• StiExportFormat.Pdf • StiExportFormat.Xps • StiExportFormat.PowerPoint • StiExportFormat.Html • StiExportFormat.Html5 • StiExportFormat.Text • StiExportFormat.Word • StiExportFormat.Excel • StiExportFormat.Odt • StiExportFormat.Ods • StiExportFormat.Rtf • StiExportFormat.Csv • StiExportFormat.Json • StiExportFormat.Xml • StiExportFormat.Dbf • StiExportFormat.Sylk • StiExportFormat.ImagePng • StiExportFormat.ImageJpeg • StiExportFormat.ImageGif • StiExportFormat.ImageTiff • StiExportFormat.ImageSvg • StiExportFormat.ImageSvgz • StiExportFormat.ImagePcx

	• `StiExportFormat.ImageBmp`
`formatName`	The name of the selected export format matches the constant names in the export format enumeration.
`settings`	Settings for the selected export format. The type of the settings object and the list of available properties will depend on the selected export type.
`fileName`	The report file name for saving after the export is completed.
`openAfterExport`	This flag allows enabling automatic opening of the exported report in a new browser tab instead of saving it to a file. It works only for formats supported by the browser. The default value is `false`.
`preventDefault`	This flag allows stopping further processing of the event by the viewer. The default value is `false`.

A list of event arguments is available for both JavaScript client-side and PHP server-side `StiExportEventArgs` type:

Name	Description
`event`	Current event identifier, has the value `StiEventType::BeginExportReport`.
`sender`	The component that triggered this event can be of the following types: • `StiViewer`

	• StiDesigner
action	The action that triggered the export event can take the following values: • StiExportAction::ExportReport • StiExportAction::SendEmail
format	The selected export format can take the following values: • StiExportFormat::Document • StiExportFormat::Pdf • StiExportFormat::Xps • StiExportFormat::PowerPoint • StiExportFormat::Html • StiExportFormat::Html5 • StiExportFormat::Text • StiExportFormat::Word • StiExportFormat::Excel2007 • StiExportFormat::Odt • StiExportFormat::Ods • StiExportFormat::Rtf • StiExportFormat::Csv • StiExportFormat::Json • StiExportFormat::Xml • StiExportFormat::Dbf • StiExportFormat::Sylk • StiExportFormat::ImagePng

	• `StiExportFormat::ImageJpeg` • `StiExportFormat::ImageGif` • `StiExportFormat::ImageTiff` • `StiExportFormat::ImageSvg` • `StiExportFormat::ImageSvgz` • `StiExportFormat::ImagePcx` • `StiExportFormat::ImageBmp`
`formatName`	The name of the selected export format corresponds to the constant names in the export format enumeration.
`fileName`	The report file name for saving after the export is completed.
`settings`	Settings for the selected export format. The type of settings object and the list of available properties will depend on the selected export type.
`openAfterExport`	This flag allows enabling automatic opening of the exported report in a new browser tab instead of saving it to a file. It works only for formats supported by the browser. The default value is `false`.

onEndExportReport
[+] JavaScript [+] PHP

This event is triggered after exporting the report but before saving it as a file. Detailed descriptions and usage examples can be found in the Report Export section.

The table below contains a list of properties passed as event arguments on the JavaScript client-side.

Name	Description
`event`	Current event identifier, has the value `"EndExportReport"`.
`sender`	The identifier of the component that triggered this event can take the following values: • `"Viewer"` • `"Designer"`
`report`	The current report object.
`format`	The selected export format can take the following values • `StiExportFormat.Document` • `StiExportFormat.Pdf` • `StiExportFormat.Xps` • `StiExportFormat.PowerPoint` • `StiExportFormat.Html` • `StiExportFormat.Html5` • `StiExportFormat.Text` • `StiExportFormat.Word` • `StiExportFormat.Excel` • `StiExportFormat.Odt` • `StiExportFormat.Ods` • `StiExportFormat.Rtf` • `StiExportFormat.Csv` • `StiExportFormat.Json`

	• StiExportFormat.Xml • StiExportFormat.Dbf • StiExportFormat.Sylk • StiExportFormat.ImagePng • StiExportFormat.ImageJpeg • StiExportFormat.ImageGif • StiExportFormat.ImageTiff • StiExportFormat.ImageSvg • StiExportFormat.ImageSvgz • StiExportFormat.ImagePcx • ·StiExportFormat.ImageBmp
formatName	The name of the selected export format corresponds to the constant names in the export format enumeration.
data	Byte data of the exported report prepared for saving to a file.
fileName	The report file name for saving after the export is completed.
openAfterExport	This flag allows enabling automatic opening of the exported report in a new browser tab instead of saving it to a file. It works only for formats supported by the browser. The default value is false.
preventDefault	This flag allows stopping further processing of the event by the

	viewer. The default value is `false`.

A list of event arguments is available for both JavaScript client-side and PHP server-side `StiExportEventArgs` type:

Name	Description
	Current event identifier, has the value `StiEventType::EndExportReport`.
`sender`	The identifier of the component that triggered this event can take the following values: • `StiViewer` • `StiDesigner`
`format`	The selected export format can take the following values from the Stimulsoft.Report.StiExportFormat enumeration: • `StiExportFormat::Document` • `StiExportFormat::Pdf` • `StiExportFormat::Xps` • `StiExportFormat::PowerPoint` • `StiExportFormat::Html` • `StiExportFormat::Html5` • `StiExportFormat::Text` • `StiExportFormat::Word` • `StiExportFormat::Excel2007` • `StiExportFormat::Odt` • `StiExportFormat::Ods`

	• StiExportFormat::Rtf • StiExportFormat::Csv • StiExportFormat::Json • StiExportFormat::Xml • StiExportFormat::Dbf • StiExportFormat::Sylk • StiExportFormat::ImagePng • StiExportFormat::ImageJpeg • StiExportFormat::ImageGif • StiExportFormat::ImageTiff • StiExportFormat::ImageSvg • StiExportFormat::ImageSvgz • StiExportFormat::ImagePcx • StiExportFormat::ImageBmp
formatName	The name of the selected export format corresponds to the constant names in the format enumeration.
data	The byte data of the exported report, prepared for saving to a file, is in Base64 format.
fileName	The report file name for saving after the export is completed.
fileExtension	The file extension for the report corresponds to the selected export format.

`mimeType`	The MIME type for the selected export format.
`openAfterExport`	This flag allows enabling automatic opening of the exported report in a new browser tab instead of saving it to a file. It works only for formats supported by the browser. The default value is `false`.

onInteraction
[+] JavaScript [-] PHP

This event is triggered during any interactive action in the viewer (such as dynamic sorting, collapsing, drilling down, or applying parameters) before the values are processed by the report generator. Detailed descriptions and usage examples can be found in the Dynamic Sorting, Collapsing, and Drilling Down section.

The table below contains a list of properties passed as event arguments on the JavaScript client-side.

Name	Description
`event`	Current event identifier, has the value "`Interaction`".
`sender`	The identifier of the component that triggered this event can take the following values: • "`Viewer`" • "`Designer`"
`report`	The current report object.
`action`	The identifier of the current interactive action can take the following values: • "`InitVars`" - the action occurs during the initialization of report variables requested from

	the user; • "Variables" - the action occurs when applying values of variables requested from the user; • "Sorting" - the action occurs when using column sorting; • "DrillDown" - the action occurs during report drill-down; • "Collapsing" - the action occurs when collapsing report blocks; • "DashboardFiltering" - the action occurs when using filters within a dashboard element; • "DashboardSorting" - the action occurs when using sorting within a dashboard element; • "DashboardResetAllFilters" - the action occurs when resetting sorting and filters within a dashboard element to the values set in the template; • "DashboardElementDrillDown" - the action occurs when using drill-down within a dashboard element; • "DashboardElementDrillUp" - the action occurs when using drill-up within a dashboard element.
variables	The collection of report variables and their values set on the parameter panel.
sortingParameters	The collection of parameters necessary for dynamic sorting of the report.

`collapsingParameters`	The collection of parameters necessary for dynamically collapsing report elements.
`drillDownParameters`	The collection of parameters necessary for report drill-down.
`filteringParameters`	The collection of parameters necessary for sorting, filtering, and drill-down of dashboard elements.
`preventDefault`	This flag allows stopping further event processing. The default value is `false`.

onEmailReport
[+] JavaScript [+] PHP

The event is triggered after the report is exported, before it is sent by Email. Detailed descriptions and usage examples can be found in the Sending Report by Email section.

The table below contains a list of properties passed as event arguments on the JavaScript client-side.

Name	Description
`event`	Current event identifier, has the value "`EmailReport`".
`sender`	The identifier of the component that triggered this event can take the following values: • "`Viewer`" • "`Designer`"
`report`	The current report object.
`format`	The selected export format of the

	report can take the following values from the `Stimulsoft.Report.StiExportFormat` enumeration: • `StiExportFormat.Document` • `StiExportFormat.Pdf` • `StiExportFormat.Xps` • `StiExportFormat.PowerPoint` • `StiExportFormat.Html` • `StiExportFormat.Html5` • `StiExportFormat.Text` • `StiExportFormat.Word` • `StiExportFormat.Excel` • `StiExportFormat.Odt` • `StiExportFormat.Ods` • `StiExportFormat.Rtf` • `StiExportFormat.Csv` • `StiExportFormat.Json` • `StiExportFormat.Xml` • `StiExportFormat.Dbf` • `StiExportFormat.Sylk` • `StiExportFormat.ImagePng` • `StiExportFormat.ImageJpeg` • `StiExportFormat.ImageGif`

	• StiExportFormat.ImageTiff • StiExportFormat.ImageSvg • StiExportFormat.ImageSvgz • StiExportFormat.ImagePcx • ·StiExportFormat.ImageBmp
formatName	The name of the selected export format corresponds to the constant names in the export format enumeration.
data	Byte data of the exported report prepared for sending by email.
fileName	The report file name for sending by email.
settings	An object containing the parameters filled out in the viewer's email sending dialog. A description of all parameters is provided in a separate table below.

The table below lists the Email sending parameters on the JavaScript client-side:

Name	Description
email	The email address to which the exported report will be sent.
subject	The subject of the email.
message	The message text of the email.

A list of event arguments is available for both JavaScript client-side and PHP server-

side `StiEmailEventArgs` type:

Name	Description
	Current event identifier, has the value `StiEventType::EmailReport`.
`sender`	The identifier of the component that triggered this event can take the following values: • `StiViewer` • `StiDesigner`
`format`	The selected export format can take the following values: • `StiExportFormat::Document` • `StiExportFormat::Pdf` • `StiExportFormat::Xps` • `StiExportFormat::PowerPoint` • `StiExportFormat::Html` • `StiExportFormat::Html5` • `StiExportFormat::Text` • `StiExportFormat::Word` • `StiExportFormat::Excel2007` • `StiExportFormat::Odt` • `StiExportFormat::Ods` • `StiExportFormat::Rtf` • `StiExportFormat::Csv` • `StiExportFormat::Json` • `StiExportFormat::Xml` • `StiExportFormat::Dbf`

	• StiExportFormat::Sylk • StiExportFormat::ImagePng • StiExportFormat::ImageJpeg • StiExportFormat::ImageGif • StiExportFormat::ImageTiff • StiExportFormat::ImageSvg • StiExportFormat::ImageSvgz • StiExportFormat::ImagePcx • StiExportFormat::ImageBmp
formatName	The name of the selected export format corresponds to the constant names in the format enumeration.
data	The byte data of the exported report, prepared for sending by Email, is in Base64 format.
fileName	The report file name for sending by Email.
settings	An object containing the Email sending parameters on the server-side. A detailed description of all parameters is provided in a separate table below.

A list of event arguments is available for both JavaScript client-side and PHP server-side StiEmailSettings type:

Name	Description

from	The sender's Email address.
name	The sender's first and last name.
to	The recipient's Email address to which the exported report will be sent, provided from the viewer dialog.
subject	The subject of the Email, provided from the viewer dialog.
message	The mesage of the Email, provided from the viewer dialog.
attachmentName	The report file name in the attachment, by default, the report file name is used.
charset	The character encoding used for the Email body, "UTF-8" is used by default.
host	The SMTP server address. This is a required field.
port	The SMTP server port, 465 is used by default.
secure	The type of encryption for the connection with the mail server, either "ssl" (default) or "tls" encryption can be used.
login	The login for connecting to the mail server. This is a required field.
password	The password for connecting to the mail server. This is a required field.
cc	An array of CC (Carbon Copy) addresses for secondary Email recipients.
bcc	An array of BCC (Blind Carbon Copy) addresses for hidden Email recipients.

onDesignReport
[+] JavaScript [-] PHP

This event is triggered when clicking the Design button on the viewer's toolbar. Detailed descriptions and usage examples can be found in the Call the Designer from the Viewer section.

The table below contains a list of properties passed as event arguments on the JavaScript client-side.

Name	Description
`event`	Current event identifier, has the value `"DesignReport"`.
`sender`	The identifier of the component that triggered this event can take the following values: • `"Viewer"`
`report`	The current report object.
`fileName`	The current report file name.

10.2.16 Viewer Settings

The viewer is configured by modifying the values of properties located in the main `options` property container of the component. All properties are divided into groups for ease of use.

Example of setting some viewer properties:

viewer.php

```
<?php
   use Stimulsoft\Viewer\StiViewer;
   use Stimulsoft\Viewer\Enums\StiToolbarDisplayMode;
   use Stimulsoft\Viewer\Enums\StiViewerTheme;
   use Stimulsoft\Viewer\Enums\StiHtmlExportMode;

   $viewer = new StiViewer();
   $viewer->options->appearance->theme =
```

```
    StiViewerTheme::Office2022WhiteGreen;
    $viewer->options->appearance->reportDisplayMode =
    StiHtmlExportMode::FromReport;
    $viewer->options->width = '1000px';
    $viewer->options->height = '1000px';
    $viewer->options->toolbar->displayMode =
    StiToolbarDisplayMode::Separated;
    $viewer->options->toolbar->zoom = 50;
    $viewer->options->appearance->fullScreenMode = true;
    $viewer->options->appearance->scrollbarsMode = true;
    $viewer->options->exports->ShowExportToWord = false;
    $viewer->options->exports->showExportToCsv = false;
?>
```

The full example code is available on GitHub.

Main settings (without groups)

Name	Description
`width`	Sets the width of the component in “px” or “%”. The "100%" value is set by default.
`height`	It sets height of a component in "px" or "%". The "100%" value is set by default for standard mode and "650px" for the mode of display with scroll bars.
`localization`	Sets the selected localization of the component. By default, the English localization is set. It is built into the component.

Appearance

Name	Description
`theme`	Sets the theme of the viewer. The list

	of available themes can be found in the `StiViewerTheme` enumeration. The default value is `StiViewerTheme::Office2022WhiteBlue`.
`iconSet`	Allows setting the icon set: • `StiWebUIIconSet::Auto` (default) - automatically sets the icon set. For `Office2022` themes, the `Monoline` icon set is used, and for `Office2013` themes, the `Regular` icon set is used; • `StiWebUIIconSet::Monoline` - sets the Monoline style icon set; • `StiWebUIIconSet::Regular` - sets the Regular style icon set.
`backgroundColor`	Sets the background color of the viewer. By default. it is set to `'white'`.
`pageBorderColor`	Sets the border color of the viewer. By default, it is set to `'gray'`.
`rightToLeft`	Sets the **Right to Left** mode for viewer controls. By default, the property is set to `false`.
`fullScreenMode`	It sets the full screen mode of the viewer display. If the property is set in the `true` value, the values of **width** and **height** properties are ignored. The `false` value is set by dedault.
`scrollbarsMode`	Sets the preview mode with

	scrollbars. By default, the property is set to `false`.
`openLinksWindow`	Sets the target window to open links contained in the report. By default, it is set to `'_blank'` (new window). It can have one of the standard values `'_blank'`, `'_self'`, `'_top'`, as well as the name of the window or frame.
`openExportedReportWindow`	Sets the target window or frame for opening the exported report. The default is `'_blank'` (new browser tab). It can take one of the standard values: `'_blank'`, `'_self'`, `'_top'`, or the name of a window or frame.
`showTooltips`	Enables displaying tips for the viewer controls when the mouse hovers over. By default, the property is set to `true`.
`showTooltipsHelp`	Sets a value which indicates that show or hide the help link in tooltips. By default, the property is set to `true`.
`showDialogsHelp`	Sets a value which indicates that show or hide the help button in dialogs. By default, the property is set to `true`.
`pageAlignment`	Sets the position of the report page in the viewer window: • `StiContentAlignment::DefaultValue` - page alignment is determined from the template settings (value is set by default); • `StiContentAlignment::Left` – the page will be aligned left;

	• `StiContentAlignment::Center` – the page will be centered (set by default); • `StiContentAlignment::Right` – the page will be aligned right.
`showPageShadow`	Enables displaying shadow for report pages. By default, the property is set to `false`.
`bookmarksPrint`	Enables printing of report bookmarks (besides the report itself). By default, the property is set to `false`.
`bookmarksTreeWidth`	Sets the width of the bookmarks panel in pixels. By default, the width is 180 pixels.
`parametersPanelPosition`	It sets location of the panel parameters in the viewer: • `StiParametersPanelPosition::FromReport` - the location of the panel is determined from the template settings (value is set by default); • `StiParametersPanelPosition::Top` - the panel is located upper report page; • `StiParametersPanelPosition::Left` - the panel is located to the left from report page.
`parametersPanelMaxHeight`	It sets max height of the parameter panel in pixels. The 300 value is set by default.
`parametersPanelColumnsCount`	It sets the number of columns in the parameter panel. The 2 value is set by default.
`minParametersCountForMultiColumns`	Sets the minimum number of variables on the parameters panel for multi-column display mode. The

	default value is 5.
`parametersPanelDateFormat`	It sets date and time format for the variables, which are displayed in the parameter panel. The `String.empty` value is set by default.
`parametersPanelSortDataItems`	It sets or disables the sorting variable values mode. The option is set in the `true` value by default, i.e variable values are sorted.
`interfaceType`	It sets the type of the viewer interface. The following values can be used: • `StiInterfaceType::Auto` – the type of the viewer interface will be selected automatically depending on the device you use (value is set by default); • `StiInterfaceType::Mouse` – forced using of standard interface to control the viewer using a computer mouse; • `StiInterfaceType::Touch` – forced using the `Touch` interface to control the viewer using touch screen of a monitor. In this mode, the viewer interface elements have enlarged sizes for comfortable control; • `StiInterfaceType::Mobile` – forced using the `Mobile` interface to control the viewer using smartphone screen. In this mode, the viewer interface has a simplified appearance to control using a mobile device.

`allowMobileMode`	Enables or disables displaying a report or dashboard in the mobile mode. If the option is set to `false`, then the mobile view will not be used. If the option is set to `true`, the mobile view mode will be used when opening the viewer on mobile devices. By default, the option is set to `true`.
`chartRenderType`	It sets the type of chart drawing in a report: • `StiChartRenderType::AnimatedVector` – charts will be drawn in the vector mode with animation (value is set by default); • `StiChartRenderType::Vector` – charts will be drawn as a vector image without animation.
`reportDisplayMode`	It sets the export mode to display report pages. It can take one of the following values: • `StiHtmlExportMode::FromReport` - the export mode of the report elements is defined from report template settings - Div or Table (value is set by default); • `StiHtmlExportMode::Table` – report elements are exported using HTML tables; • `StiHtmlExportMode::Div` – report elements are exported using DIV markup.
`datePickerFirstDayOfWeek`	It gives an ability to set the first day of the week for the **Date Picker** tool: • `StiFirstDayOfWeek::Auto` - Monday or Sunday will be set as the first day of the week

	depending on browser culture (value is set by default); • `StiFirstDayOfWeek::Monday` - Monday will be set as the first day of the week; • `StiFirstDayOfWeek::Sunday` - Sunday will be set as the first day of the week.
`datePickerIncludeCurrentDayForRanges`	It gives an ability to include or not the current day into the range of the **Date Picker** element values. By default, the option is set in the `false` value i.e. the current day is not included into the range of the element values.
`allowTouchZoom`	It gives an ability change the viewer zoom by touching. By default, the option is set in the `true` value.
`combineReportPages`	It allows you to combine processed pages of report template into one template or present each page of the template as a separate tab in the viewer. By default, the option is set in the `false` value i.e. each page of report template will be presented as a separate tab in the viewer.

Toolbar

Name	Description
`visible`	It allows you to display or not to display the viewer toolbar. By default, the `true` value is set.
`displayMode`	It sets the display of the viewer toolbar. It can take one of the enumeration values below:

	• `StiToolbarDisplayMode::Simple` – simple display mode, all elements of control are located in one control panel (value is set by default); • `StiToolbarDisplayMode::Separated` – separated display mode, toolbar is divided into upper and bottom.
`backgroundColor`	It allows you to change the color of toolbar. The `'transparent'` value is set by default.
`borderColor`	It allows you to change toolbar border color. The `'transparent'` value is set by default.
`fontColor`	It gives an ability to change the font of all elements in the toolbar and in all menus of this panel. The `'transparent'` value is set by default.
`fontFamily`	It allows you to change the font for all elements in the toolbar and in all menus of this panel. By default, the `'Arial'` value is set.
`alignment`	It sets the alignment of elements in the control panel: • `StiContentAlignment::Default` – alignment of elements depends on the `RightToLeft` option (value is set by default); • `StiContentAlignment::Left` – all elements will be aligned to the left side of the toolbar; • `StiContentAlignment::Center` – all elements will be aligned to the center of the toolbar;

	• `StiContentAlignment::Right` – all elements will be aligned to the right side of the toolbar.
`showButtonCaptions`	It enables or disables the display of the viewer toolbar buttons text. By default, the property is set to `true`.
`showPrintButton`	It allows you to display or not to display the **Print** button. By default, the property is set to `true`.
`showOpenButton`	It enables the display of the **Open** button in the viewer toolbar when viewing reports or dashboards. By default, the property is set to `true`.
`showSaveButton`	It enables the display of the **Save** button in the toolbar when viewing reports or dashboards. By default, the property is set to `true`.
`showSendEmailButton`	It allows you to display or not to display the **Send Email** button in the toolbar. By default, the `false` value is set. Also, you should add the `onEmailReport` event handler.
`showFindButton`	It allows you to display or not to display the **Find** button in the toolbar. By default, the property is set to `true`.
`showSignatureButton`	
`showBookmarksButton`	It allows you to display or not to display the **Bookmarks** button in the toolbar. If this button is not displayed, the bookmark panel, the bookmark panel will not be displayed in a report. By default, the property is set to `true`.
`showParametersButton`	It allows you to display or not to

	display the **Parameters** button in the toolbar. If this button is not displayed, the parameter panel will not be displayed in a report. By default, the property is set to `true`.
`showResourcesButton`	It allows you to display or not to display the **Resources** button in the toolbar. If this button is not displayed, the resources panel will not be displayed in a report. By default, the property is set to `true`.
`showEditorButton`	It allows you to display or not to display the **Editor** button in the toolbar. If this button is not displayed, you won't be able to change edited data. By default, the property is set to `true`.
`showFullScreenButton`	It enables the display of the **Full Screen** button in the viewer toolbar when viewing reports or dashboards. By default, the property is set to `true`.
`showRefreshButton`	It allows you to display or not to display the **Refresh** button in the viewer toolbar when viewing dashboards. By default, the property is set to `true`.
`showFirstPageButton`	It allows you to display or not to display the **First Page** button in the toolbar. By default, the property is set to `true`.
`showPreviousPageButton`	It allows you to display or not to display the **Previous Page** in the toolbar.By default, the property is set to `true`.
`showCurrentPageControl`	It allows you to display or not to

	display an indicator of the current page in the toolbar.By default, the property is set to `true`.
`showNextPageButton`	It allows you to display or not to display the **Next Page** button in the toolbar. By default, the property is set to `true`.
`showLastPageButton`	It allows you to display or not to display the **Last Page** button in the toolbar. By default, the property is set to `true`.
`showZoomButton`	It allows you to display or not to display the **Zoom** selection button in the toolbar. By default, the property is set to `true`.
`showViewModeButton`	It allows you to display or not to display the report pages display modes button. By default, the property is set to `true`.
`showDesignButton`	It enables the display of the **Design** button in the viewer toolbar when viewing reports or dashboards. By default, the property is set to `false`.
`showAboutButton`	It allows you to display or not to display the **About** button. By default, the property is set to `true`.
`showPinToolbarButton`	It allows you to display or not to display the **Pin** button in the mobile mode of report viewing. By default, the property is set to `true`.
`printDestination`	It sets the report print mode. It can take one of the enumeration values below: • `StiPrintDestination::Default` – the menu with the selection print mode will be

	displayed (value is set by default); • `StiPrintDestination::Pdf` – print will be made in PDF format; • `StiPrintDestination::Direct` – print will be made in HTML format directly to the printer. System print dialog will be displayed; • `StiPrintDestination::PopupWindow` – print will be made in HTML format via the pop-up window of report preview.
`viewMode`	It sets the report pages display mode: • `StiWebViewMode::OnePage` – one page selected in the toolbar is displayed (value is set by default); • `StiWebViewMode::Continuous` – all report pages are displayed as a ribbon; • `StiWebViewMode::MultiplePages` – all report pages are displayed as a table.
`zoom`	It allows you to set the scale of report pages when loading the viewer. 100 percent by default. Max value is 500 percent. • `StiZoomMode::PageWidth` – report pages scale by page width; • `StiZoomMode::PageHeight` – report pages scale by page height.
`menuAnimation`	It allows you to enable or disable the animation of display and closing

	various menus in the viewer. By default, the property is set to `true`.
`showMenuMode`	It sets the mode opening of various menus in the viewer when hovering or clicking. • `StiShowMenuMode::Click` – click-to-open menu mode (value is set by default); • `StiShowMenuMode::Hover` – hover-to-open menu mode.
`autoHide`	It sets the mode of automatic collapsing the toolbar when viewing a report in the mobile mode. By default, the property is set to `true`.

Exports

Name	Description
`storeExportSettings`	It allows you to save export settings in cookies. The `true` value is set by default.
`showExportDialog`	It allows you to display or not to display the **Export Settings** menu. If the menu is hidden, the export will be made with values by default. The `true` value is set by default.
`showExportToDocument`	It allows you to display or not to display the **Document File** element in the **Save** menu. The `true` value is set by default.
`showExportToPdf`	It enables the display of the **Adobe PDF File** export menu item when viewing reports and the **Adobe PDF** when viewing dashboards. The property has the `true` value by

	default.
showExportToXps	It enables the display of the **XPS File** export menu item. The property has the true value by default.
showExportToPowerPoint	It enables the display of the **Microsoft PowerPoint** export menu item. The property has the true value by default.
showExportToHtml	It allows you to display or not to display the **HTML File** element in the export settings menu. The true value is set by default.
showExportToHtml5	It allows you to display or not to display the **HTML5 File** element in the export settings menu. The true value is set by default.
showExportToText	It enables the display of the the **Text File** export menu item. The property has the true value by default.
showExportToWord	It enables the display of the the **Microsoft Word** export menu item. The property has the true value by default.
showExportToOpenDocumentWriter	It enables the display of the **OpenDocument Writer File** export menu item. The property has the true value by default.
showExportToExce	It enables the display of the **Microsoft Word** export menu item. The property has the true value by default.
showExportToOpenDocumentCalc	It enables the display of the **OpenDocument Calc File** export menu item. The property has the true value by default.

showExportToRtf	It enables display of the **RTF** type in the settings of the **Data** export menu item. By default, the property is set to true.
showExportToCsv	It enables display of the **CSV** type in the settings of the **Data** export menu item. By default, the property is set to true.
showExportToJson	It enables display of the **JSON** type in the settings of the **Data** export menu item. By default, the property is set to true..
showExportToXml	It enables display of the **XML** type in the settings of the **Data** export menu item. By default, the property is set to true.
showExportToDbf	It enables display of the **DBF** type in the settings of the **Data** export menu item. By default, the property is set to true.
showExportToDif	It enables display of the **DIF** type in the settings of the **Data** export menu item. By default, the property is set to true.
showExportToSylk	It enables display of the **SYLK** type in the settings of the **Data** export menu item. By default, the property is set to true.
showExportToImagePng	It enables display of the **PNG** type in the settings of the **Image** export menu item. By default, the property is set to true.
showExportToImageJpeg	It enables display of the **JPEG** type in the settings of the **Image** export menu item. By default, the property is set to true.

showExportToImageSvg	It enables display of the **SVG** type in the settings of the **Image** export menu item. By default, the property is set to true.
showExportToImageSvgz	It enables display of the **SVGZ** type in the settings of the **Image** export menu item. By default, the property is set to true.
showExportToImagePcx	It enables display of the **PCX** type in the settings of the **Image** export menu item. By default, the property is set to true.
showExportToImageBmp	It enables display of the **BMP** type in the settings of the **Image** export menu item. By default, the property is set to true.
showExportToImageGif	It enables display of the **GIF** type in the settings of the **Image** export menu item. By default, the property is set to true.
showExportToImageTiff	It enables display of the **TIFF** type in the settings of the **Image** export menu item. By default, the property is set to true.
showExportDataOnly	It enables display of the **Export Data Only** type in the settings of the **Data** export menu item. By default, the property is set to true.

Email

Name	Description
showEmailDialog	It enables the display of the parameter dialog window of report sending by Email. If the dialog

	window is disabled, the sending by Email will be done with the settings specified by default `onEmailReport`. The `true` value is set by default.
`showExportDialog`	It enables the display of the parameter dialog window when sending an Email. If the property has the `false` value, the export will be done with specified by default settings. The `true` value is set by default.
`defaultEmailAddress`	It sets an Email recipient by default, i.e. the address which will receive a Email with an attached report. The value is not set by default.
`defaultEmailSubject`	It sets the theme (header) of an Email by default. The value is not set by default.
`defaultEmailMessage`	It sets a message (text) of an Email by default. The value is not set by default.

10.3 HTML5 Designer

The report designer is a PHP component, `StiDesigner`, designed for creating reports and dashboards in a browser window on any computer with any operating system installed. The designer features a modern interface, various themes, a wide selection of professionally prepared report and dashboard templates, report wizards, and numerous components for building reports and dashboards of nearly any complexity.

The designer includes a convenient preview mode within the built-in viewer, significantly speeding up the report development process. The functionality of the built-in viewer is almost identical to that of the standalone component, which is covered in more detail in the Report Viewer section.

The designer interface is built using HTML5, allowing it to run on virtually any

modern platform. The component uses AJAX technology to perform all actions (loading and building reports, connecting to data, working with components and their settings, previewing reports, printing, exporting, etc.), eliminating the need for full page reloads, enhancing performance, and enabling use in One-Page applications. The JavaScript technology employed for report creation allows the use of almost any low-performance server backend.

Information

Since a unified template format (MRT) is used for both indicator panels (dashboards) and reports, as well as common methods for loading templates and working with data, the term "report" will be used throughout the documentation.

- Deployment
- Activation
- Creating and Editing Reports
- Saving Reports
- Localization
- Preview
- Appearance
- Designer Events
- Designer Settings

10.3.1 Usage

To use the product, simply download the ZIP archive from the Downloads page of our website, extract it, and copy the contents of the /PHP folder to your web server. This folder represents a web project that includes all the necessary files and resources for the product to work, as well as examples for working with the viewer and designer.

To integrate the product into an existing project, simply copy the `/vendor` folder from the **/PHP** directory into the root directory of your project, or use the Composer dependency manager by running the following console command:

console

```
composer require stimulsoft/reports-php
```

To work with dashboards, the following package needs to be included:

console

```
composer require stimulsoft/dashboards-php
```

When using the product, in most cases, PHP code alone is sufficient to enable the primary features. For more advanced configuration and to utilize all the capabilities of the product, **JavaScript** code is required.

To use the components in a web project, you only need to connect the automatic script loader at the beginning of your PHP file. After that, you can utilize all the available PHP classes and functions to work with reports and dashboards.

designer.php

```
<?php
   require_once 'vendor/autoload.php';

...
?>
```

To use the designer in a web project, the `StiDesigner` class is provided. Using this class, you can create a designer object, set the necessary configurations, assign a report for editing, handle requests, and manage designer events.

Example of editing a report in the designer on an HTML page:

designer.php

```
<?php
   require_once 'vendor/autoload.php';

   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Designer\StiDesigner;
```

```
   $designer = new StiDesigner();
   $designer->process();

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $designer->report = $report;
?>

<html>
<head>
   <?php
      $designer->javascript->renderHtml();
   ?>
</head>
<body>
   <?php
      $designer->renderHtml();
   ?>
</body>
</html>
```

The full example code is available on GitHub.

In this example, the following steps are performed sequentially:

- An instance of the StiDesigner object is created;
- The current request is processed;
- An instance of the StiReport object is created;
- A report template is loaded from the SimpleList.mrt file;
- The created report is assigned to the designer;
- The necessary JavaScript and HTML code for the component is rendered in the HTML file template.

The `$designer->process()` method processes the current request and, if successful, automatically returns the result to the client-side. More details on this can be found in the Event Handler section.

The `$designer->javascript->renderHtml()` method renders the code required to connect the component's scripts. The `$designer->renderHtml()` method renders the JavaScript and HTML code for the component itself. Several usage options are available, which are described in detail in the Using the Report Engine section.

Example of simplified designer rendering without using an HTML page template:

designer.php

```
<?php
   require_once 'vendor/autoload.php';

   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Designer\StiDesigner;

   $designer = new StiDesigner();
   $designer->process();

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $designer->report = $report;

   $designer->printHtml();
?>
```

The full example code is available on GitHub.

10.3.2 License Activation

After purchasing the product, you need to activate the license for the components you are using. There are several ways to connect the license key.

All options for component activation are described in the License Activation for the Report Engine section, and they have the same functions and parameters for invocation.

10.3.3 Creating and Editing Reports

No actions are required to launch the designer without a report. After the component loads, the designer's main menu will be displayed. If you need to start the designer with a new (empty) report, you can create a new `StiReport` object and assign it to the designer.

To edit a report in the designer, you simply need to create an `StiReport` object, load a report template into it, and assign the resulting object to the designer. All other actions will be performed automatically, and the designer will display the first page of the template.

designer.php

```php
<?php
   require_once 'vendor/autoload.php';

   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Designer\StiDesigner;

   $designer = new StiDesigner();
   $designer->process();

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $designer->report = $report;

   $designer->printHtml();
?>
```

The full example code is available on GitHub.

The designer can work with standard, packed, and encrypted templates. A detailed description of working with various report formats can be found in the Loading and Saving Reports section.

Report creation event

A new report can be created using the designer's main menu. To perform any necessary actions with the new report, the `onCreateReport` event is available. This event will be triggered when a new empty report is created from the main menu or when a report is created using a wizard. The event arguments will contain the report object created in the designer. If necessary, you can modify it, connect data, or load a pre-prepared report template. After the event is completed, this report will be automatically loaded into the designer for editing.

Example of connecting data and synchronizing the data dictionary when creating a new report:

designer.php

```php
<?php
   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Designer\StiDesigner;

   $designer = new StiDesigner();
   $designer->onCreateReport = 'createReport';
   $designer->process();
```

```
    $report = new StiReport();
    $report->loadFile('reports/SimpleList.mrt');
    $designer->report = $report;
?>

...

<script>
    function createReport(args) {
        let dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
        dataSet.readJsonFile("Data/Demo.json");

        args.report.regData(dataSet.dataSetName, "", dataSet);
        args.report.dictionary.synchronize();
    }
</script>
```

The full example code is available on GitHub.

If needed, the process of creating a new report can be controlled on the PHP server-side. Example of modifying new report parameters on the PHP server-side:

designer.php

```
<?php
    use Stimulsoft\Designer\StiDesigner;
    use Stimulsoft\Events\StiReportEventArgs;

    $designer = new StiDesigner();
    $designer->onCreateReport = function (StiReportEventArgs $args) {
        $args->report->ReportAlias = 'New Report Alias';
    };

    $designer->process();
    $designer->printHtml();
?>
```

Example of loading a pre-prepared template with pre-configured data connections:

designer.php

```
<?php
    use Stimulsoft\Designer\StiDesigner;
    use Stimulsoft\Events\StiReportEventArgs;
```

```
   $designer = new StiDesigner();
   $designer->onCreateReport = function (StiReportEventArgs $args) {
      $reportJson = file_get_contents('reports/NewTemplateWithData.mrt');
      $args->setReportJson($reportJson);
   };

   $designer->process();
   $designer->printHtml();
?>
```

A detailed description of available argument values can be found in the Designer Events section.

10.3.4 Saving Reports

The designer provides two options for saving a report, which are available in the main menu and on the designer's main panel: **Save** and **Save As**. Each of these saving options has its own modes and settings.

Saving a report on the client-side via JavaScript

When the **Save** button is pressed, the report template file is saved using browser mechanisms, and no settings are required for this. If you need to save the report using your own methods, the `onSaveReport` event is available. The event arguments will include the report file name and the report itself. The report can be saved, for example, as a JSON string and sent to the server using your own methods.

Example of saving a report as a JSON string on the client-side via JavaScript for further use in the application:

designer.php

```
<?php
   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Designer\StiDesigner;

   $designer = new StiDesigner();
   $designer->onSaveReport = 'saveReport';
   $designer->process();

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $designer->report = $report;
```

```
?>

...

<script>
   function saveReport(args) {
      let fileName = args.fileName;
      let report = args.report;
      let reportJson = args.report.saveToJsonString();
      ...
   }
</script>
```

If necessary, after saving the report, you can display a dialog with an error message or text notification. For this, the special function `StiError.showError()` is available. You can decide whether to display the error message.

designer.php

```
<script>
   function saveReport(args) {
      let report = args.report;

      // Error message
      Stimulsoft.System.StiError.showError("An error occurred while saving
      the report.");

      // Info message
      Stimulsoft.System.StiError.showError("The report was saved
      successfully.", true, true);
   }
</script>
```

When the **Save As** button is pressed, a dialog box will be displayed requesting the report file name. After that, the report template file will be saved using browser mechanisms. If you need to save the report using your own methods, the `onSaveAsReport` event is available. The event arguments will include the report file name and the report itself.

The functionality of this event is no different from the `onSaveReport` event, except that after the event is completed, the report template will be automatically saved on the computer using the browser. To prevent this action, simply set the `preventDefault` property to `true` in the event arguments, and the automatic

saving will not be performed.

designer.php

```
<script>
   function saveReport(args) {
      args.preventDefault = true;
   }
</script>
```

If necessary, you can access the original report name or the name from the save dialog as follows:

designer.php

```
<script>
   function saveReport(args) {
      // Report name from the designer save dialog
      let reportName1 = args.fileName;

      // Original report name from properties
      let reportName2 = args.report.reportName;
   }
</script>
```

A detailed description of available argument values can be found in the Designer Events section.

Saving a report on the PHP server-side

To save a report on the PHP server-side, you simply need to define the `onSaveReport` event. The event arguments will include the report file name and the report itself as an object. You can use standard PHP functions to save the report.

Example of saving an editable report as a file in a specified directory:

designer.php

```
<?php
   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Designer\StiDesigner;
```

```
  use Stimulsoft\Events\StiReportEventArgs;

  $designer = new StiDesigner();
  $designer->onSaveReport = function (StiReportEventArgs $args) {
    $reportJson = $args->getReportJson();
    file_put_contents('reports/' . $reportFileName, $reportJson);
  };

  $designer->process();

  $report = new StiReport();
  $report->loadFile('reports/SimpleList.mrt');
  $designer->report = $report;

  $designer->printHtml();
?>
```

The full example code is available on GitHub.

A detailed description of available argument values can be found in the Designer Events section.

Information

Similarly, the `onSaveAsReport` event works on the PHP server-side, and all event arguments have the same names and values.

10.3.5 Localization

The designer fully supports localization of its interface. To localize the interface in the desired language, you only need to set the required file name for the `localization` option in the designer:

designer.php

```
<?php
  use Stimulsoft\Designer\StiDesigner;

  $designer = new StiDesigner();
  $designer->options->localization = 'de.xml';
  $designer->printHtml();
?>
```

The full example code is available on GitHub.

All available localization XML files can be found in the resources of the installed product package. If necessary, the localization file can be loaded from any other location by specifying the full path to the required XML file for the `localization` option:

designer.php

```
<?php
   use Stimulsoft\Designer\StiDesigner;

   $designer = new StiDesigner();
   $designer->options->localization = '/resources/loc/de.xml';
   $designer->printHtml();
?>
```

The designer also provides the ability to choose the necessary interface localization via a special menu in the toolbar. By default, this menu includes the English (built-in) localization as well as the one set via the `localization` property. To add additional localizations to the menu, a special method `addLocalization()` is available in the designer options. You can specify the localization file or the full path to this file as a parameter.

Example of adding additional localizations located in the component resources:

designer.php

```
<?php
   use Stimulsoft\Designer\StiDesigner;

   $designer = new StiDesigner();
   $designer->options->localization = 'de.xml';
   $designer->options->addLocalization('fr.xml');
   $designer->options->addLocalization('es.xml');
   $designer->options->addLocalization('pt.xml');
   $designer->printHtml();
?>
```

The full example code is available on GitHub.

10.3.6 Preview

The designer provides a preview mode for the editable report. To do this, simply switch to the corresponding tab in the designer window. The report template will be generated and displayed in the built-in viewer.

Preview event

Before previewing the report, there's an option to perform necessary actions, such as connecting data for the report. For this, there is a special `onPreviewReport` event, which will be triggered before the report is previewed. The event arguments will contain the report intended for preview.

Example of connecting data on the client-side using JavaScript for report preview:

designer.php

```
<?php
   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Designer\StiDesigner;

   $designer = new StiDesigner();
   $designer->onPreviewReport = 'previewReport';
   $designer->process();

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $designer->report = $report;
?>

...

<script>
   function previewReport(args) {
      let dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
      dataSet.readJsonFile("Data/Demo.json");

      args.report.regData(dataSet.dataSetName, "", dataSet);
   }
</script>
```

Example of modifying report properties on the PHP server-side before report preview:

designer.php

```
<?php
   use Stimulsoft\Report\StiReport;
   use Stimulsoft\Designer\StiDesigner;

   $designer = new StiDesigner();
   $designer->onPreviewReport = function (StiReportEventArgs $args) {
      $args->report->ReportDescription = 'This is a report description from
      the PHP server-side.';
   };

   $designer->process();

   $report = new StiReport();
   $report->loadFile('reports/SimpleList.mrt');
   $designer->report = $report;

   $designer->printHtml();
?>
```

A detailed description of the available event arguments can be found in the Designer Events section.

Additional options

The report preview window in the designer is a fully interactive viewer that allows printing and exporting the report and supports working with report parameters. All interactive actions, such as dynamic sorting, drill-down, and collapsing, are supported. No additional settings are required to use these features in the report designer.

10.3.7 Appearance

The designer allows for changing the themes of the visual elements. To do this, simply set the `theme` property in the component options:

designer.php

```
<?php
   use Stimulsoft\Designer\StiDesigner;
   use Stimulsoft\Designer\Enums\StiDesignerTheme;

   $designer = new StiDesigner();
   $designer->options->appearance->theme =
   StiDesignerTheme::Office2022BlackGreen;
?>
```

Currently, there are **2 different themes** with their own color accents. As a result, more than **50 design options** are available. This allows customizing the appearance of the designer to match almost any web project design.

10.3.8 Adding Custom Functions

When integrating the report designer into a custom application, you can add your own JavaScript functions to the data dictionary of the report designer. Once added, these functions can be used when creating reports and dashboards. Below is an example of adding a function to calculate a sum total:

designer.php

```
<?php
   use Stimulsoft\Designer\StiDesigner;
   use Stimulsoft\Enums\Types;
   use Stimulsoft\Report\StiFunctions;

   StiFunctions::addFunction(
      "MyCategory", "MySum", "MySum", "MySum", "", Types::string,
      "Return Description", [Types::int], ["value"], ["Descriptions"],
      "myFunc");


   $designer = new StiDesigner();
   $designer->printHtml();
?>

...

<script>
   function myFunc (value) {
      if (!Stimulsoft.Data.Extensions.ListExt.isList(value))
         return Stimulsoft.Base.Helpers.StiValueHelper.tryToNumber(value);

      return Stimulsoft.Data.Functions.Funcs
         .skipNulls(Stimulsoft.Data.Extensions.ListExt.toList(value))
         .tryCastToNumber()
         .sum();
   };
</script>
```

If you are using the designer without an HTML template, you can pass the function code instead of its name. The other parameters of the function will remain the same:

designer.php

```
<?php
   use Stimulsoft\Designer\StiDesigner;
   use Stimulsoft\Enums\Types;
   use Stimulsoft\Report\StiFunctions;

   StiFunctions::addFunction(
      "MyCategory", "MySum", "MySum", "MySum", "", Types::string,
      "Return Description", [Types::int], ["value"], ["Descriptions"],
      "
         function myFunc (value) {
            if (!Stimulsoft.Data.Extensions.ListExt.isList(value))
               return
               Stimulsoft.Base.Helpers.StiValueHelper.tryToNumber(value);

            return Stimulsoft.Data.Functions.Funcs
               .skipNulls(Stimulsoft.Data.Extensions.ListExt.toList(value))
               .tryCastToNumber()
               .sum();
         };
      ");

   $designer = new StiDesigner();
   $designer->printHtml();
?>
```

10.3.9 Designer Events

The report viewer supports events that allow executing necessary operations before certain actions, both on the client JavaScript side and the server PHP side. Detailed descriptions of event handling can be found in the Report Engine Events section.

The viewer supports the following events:

- onDatabaseConnect
- onBeginProcessData
- onEndProcessData
- onPrepareVariables
- onCreateReport
- onOpenReport
- onOpenedReport
- onSaveReport
- onSaveAsReport

- onPreviewReport
- onCloseReport
- onExit

onDatabaseConnect
[-] JavaScript [+] PHP

This event is triggered before connecting to the database after receiving all the parameters. Detailed descriptions and usage examples can be found in the Connetting SQL Data Adapter section. A list of event arguments is available in the Report Engine Events section.

onBeginProcessData
[+] JavaScript [+] PHP

This event is triggered before requesting the data needed to generate the report. Detailed descriptions and usage examples can be found in the Connecting Data and Connetting SQL Data Adapter sections. A list of event arguments is available in the Report Engine Events section.

onEndProcessData
[+] JavaScript [+] PHP

This event is triggered before requesting the data needed to generate the report. Detailed descriptions and usage examples can be found in the Connecting Data and Connetting SQL Data Adapter sections. A list of event arguments is available in the Report Engine Events section.

onPrepareVariables
[+] JavaScript [+] PHP

This event is triggered before generating the report, after preparing the report variables. Detailed descriptions and usage examples can be found in the Working with Report Variables section. A list of event arguments is available in the Report Engine Events section.

onCreateReport
[+] JavaScript [+] PHP

The event is triggered after a new report is created in the designer. Detailed descriptions and usage examples can be found in the Creating and Editing a Report section.

The table below lists the properties passed in the event arguments on the client-side using JavaScript:

Name	Description
`event`	The identifier of the current event, which has the value `"CreateReport"`.
`sender`	The identifier of the component that triggered this event, which can have the following values: • `"Designer"`
`report`	The current report object.
`isWizardUsed`	A flag indicating whether the new report is created using a wizard (value `true`) or as a blank report (value `false`).

The table below lists the properties passed in the event arguments on the server-side using PHP. The arguments are of type `StiReportEventArgs`:

Name	Description
`event`	The identifier of the current event, which for this event has the value `StiEventType::CreateReport`.
`sender`	The component that triggered this event, which can have the following types:

	• StiDesigner
report	The current report object.
isWizardUsed	A flag indicating whether the new report is created using a wizard (value true) or as a blank report (value false).

onOpenReport
[+] JavaScript [-] PHP

This event is triggered after opening a report after clicking the toolbar button.

The table below contains a list of properties passed as event arguments on the JavaScript client-side:

Name	Description
event	Current event identifier, has the value "OpenReport".
sender	The identifier of the component that triggered this event can have the following values: • "Designer"
preventDefault	This flag allows you to stop further processing of the event by the viewer. The default value is true.

onOpenedReport
[+] JavaScript [+] PHP

The event is triggered after the report is opened from the designer menu and before it is passed to the designer itself.

The table below lists the properties passed in the event arguments on the JavaScript client-side:

Name	Description
event	Current event identifier, has the value "OpenedReport".
sender	The identifier of the component that triggered this event can have the following values: • "Designer"
report	The current report object.
preventDefault	This flag allows you to stop further processing of the event by the viewer. The default value is true.

The table below lists the properties passed in the event arguments on the PHP server-side, the arguments are of type StiReportEventArgs:

Name	Description
event	The identifier of the current event, which for this event has the value StiEventType::OpenedReport
sender	The component that triggered this event, which can have the following types: • StiDesigner
report	The current report object.

onSaveReport
[+] JavaScript [+] PHP

The event is triggered when saving a report in the designer. Detailed descriptions and usage examples can be found in the Saving a Report section.

The table below lists the properties passed in the event arguments on the client-side using JavaScript:

Name	Description
`event`	Current event identifier, has the value "`SaveReport`".
`sender`	The identifier of the component that triggered this event can have the following values • "`Designer`"
`report`	The current report object.
`fileName`	The name of the report file for saving.
`autoSave`	A flag indicating whether the report is being saved automatically (value `true`) or manually by pressing the save button (value `false`).
`preventDefault`	A flag that provides the ability to stop further processing of the event by the designer. The default value is `true`.

The table below lists the properties passed in the event arguments on the server-side using PHP. The arguments are of type `StiReportEventArgs`:

Name	Description
`event`	The identifier of the current event, which for this event has the value `StiEventType::SaveReport`.
`sender`	The component that triggered this event, which can have the following types: • `StiDesigner`

report	The current report object.
fileName	The name of the report file for saving.
autoSave	A flag indicating whether the report is being saved automatically (value true) or manually by pressing the save button (value false).

onSaveAsReport
[+] JavaScript [+] PHP

The event is triggered when saving a report in the designer. Detailed descriptions and usage examples can be found in the Saving a Report section.

The table below lists the properties passed in the event arguments on the client-side using JavaScript:

Name	Description
event	Current event identifier, has the value "SaveAsReport".
sender	The identifier of the component that triggered this event can have the following values • "Designer"
report	The current report object.
fileName	The name of the report file for saving.
preventDefault	A flag that provides the ability to stop further processing of the event by the designer. The default value is false.

The table below lists the properties passed in the event arguments on the server-

side using PHP. The arguments are of type `StiReportEventArgs`:

Name	Description
`event`	The identifier of the current event, which for this event has the value `StiEventType::SaveAsReport`.
`sender`	The component that triggered this event, which can have the following types: • `StiDesigner`
`report`	The current report object.
`fileName`	The name of the report file for saving.

onPreviewReport
[+] JavaScript [+] PHP

The event is triggered when switching to the report preview tab. Detailed descriptions and usage examples can be found in the Preview section.

The table below lists the properties passed in the event arguments on the client-side using JavaScript:

Name	Description
`event`	Current event identifier, has the value `"PreviewReport"`.
`sender`	The identifier of the component that triggered this event can have the following values • `"Designer"`
`report`	The current report object.
`viewer`	The current object of the `StiViewer` component embedded in the designer.

`preventDefault`	A flag that provides the ability to stop further processing of the event by the designer. The default value is `true`.

The table below lists the properties passed in the event arguments on the server-side using PHP. The arguments are of type `StiReportEventArgs`:

Name	Description
`event`	The identifier of the current event, which for this event has the value `StiEventType::PreviewReport`.
`sender`	The component that triggered this event, which can have the following types: • `StiDesigner`
`report`	The current report object.

onCloseReport
[+] JavaScript [+] PHP

The event is triggered after the report is closed from the designer menu and before the report has been unassigned from the report designer.

The table below lists the properties passed in the event arguments on the JavaScript client-side:

Name	Description
`event`	Current event identifier, has the value `"CloseReport"`.
`sender`	The identifier of the component that triggered this event can have the following values: • `"Designer"`

report	The current report object.
preventDefault	This flag allows you to stop further processing of the event by the viewer. The default value is true.

The table below lists the properties passed in the event arguments on the PHP server-side, the arguments are of type StiReportEventArgs:

Name	Description
event	The identifier of the current event, which for this event has the value StiEventType::CloseReport
sender	The component that triggered this event, which can have the following types: • StiDesigner
report	The current report object.

onExit
[+] JavaScript [-] PHP

The event is triggered when the **Exit** button is clicked in the designer's main menu.

The table below lists the properties passed in the event arguments on the client-side using JavaScript:

Name	Description
event	Current event identifier, has the value "Exit".
sender	The identifier of the component that triggered this event can have the following values

	• "Designer"

10.3.10 Designer Settings

The designer is set using the properties found in the `Stimulsoft.Designer.StiDesignerOptions` class. All properties are divided into groups for comfortable using. All designer classes and enums you may find in the `\Stimulsoft\Designer` namespace. To set the designer you should create the class of options, set required properties and transfer an object of options to the designer constructor as the first argument.

designer.php

```
<?php
   use Stimulsoft\Designer\StiDesigner;
   use Stimulsoft\Designer\Enums\StiDesignerTheme;

   $designer = new StiDesigner();
   $designer->options->appearance->theme =
   StiDesignerTheme::Office2022WhiteGreen;
   $designer->options->toolbar->showFileMenuExit = false;
   $designer->options->toolbar->showFileMenuOptions = false;
   $designer->options->bands->showChildBand = false;
   $designer->options->components->showPanel = false;
   $designer->options->appearance->showReportTree = false;
   $designer->options->appearance->showTooltips = false;
?>
```

Main settings (without groups)

Name	Description
`Width`	It sets component width in "px or "%". The 100 % value is set by default.
`Height`	It sets component height in "px" or "%". The "800px" value is set by default.
`localization`	Sets the selected localization of the component. By default, the English localization is set. It is built into the

	component.

Appearance

Name	Description
`theme`	Sets the theme of the designer. The list of available themes can be found in the `StiDesignerTheme` enumeration. The default value is `Office2022WhiteBlue`.
`iconSet`	Allows setting the icon set: • `StiWebUIIconSet::Auto` (default) – automatically selects the icon set. For `Office2022` themes, the `Monoline` icon set is used, and for `Office2013` themes, the `Regular` icon set is used; • `StiWebUIIconSet::Monoline` - sets the `Monoline` icon set; • `StiWebUIIconSet::Regular` - sets the `Regular` icon set.
`defaultUnit`	It sets the units of measure of sizes for a report and all its components: • `StiReportUnitType::Centimeters` (value is set by default); • `StiReportUnitType::HundredthsOfInch`; • `StiReportUnitType::Inches`; • `StiReportUnitType::Millimeters`.

zoom	Sets the zoom for displaying report pages. The default setting is 100 percent. Values between 10% and 200% are allowed. It can take one of the following values of the StiZoomMode enumeration: • StiZoomMode::PageWidth – when the designer runs, the zoom, necessary to display the report by the page width, will be set; • StiZoomMode::PageHeight – when the designer runs, the zoom, required to display the page height of the report, will be set.
interfaceType	It sets the type of the designer interface. The following values can be used: • StiInterfaceType::Auto – the type of the designer interface will be selected automatically depending on the device you use (the value by default); • StiInterfaceType::Mouse – the forced using Mouse interface for controlling the designer using a computer mouse; • StiInterfaceType::Touch – the forced using of the Touch interface to control the designer using touch screen of a monitor. In this mode, the designer interface elements have enlarged sizes for comfortable control.

showAnimation	It allows you to enable or disable the animation of display and closing various menus in the designer. The true value is set by default.
showSaveDialog	It enables the display of the input dialog of report name when its saving. Report name will be transferred in the designer parameters. The true value is set by default.
showTooltips	It enables the display of prompts for the designer controls when hovering. The true value is set by default.
showTooltipsHelp	It enables the display of the link to the online documentation in prompts for the designer controls. The true value is set by default.
showDialogsHelp	It allows you to display or not to display the reference invoke button in various menus. The true value is set by default.
fullScreenMode	It sets the full screen mode of the designer display. If the property is set in the true value, the values of the width and height properties are ignored. The false value is set by default.
maximizeAfterCreating	It allows you to set max size of the report designer. The false value is set by default.
showLocalization	It allows you to display or not to display the localization control in the report designer. The true value is set by default.
allowChangeWindowTitle	It allows you to use the header of browser window to display file name

	of edited report. The `true` value is set by default.
`showPropertiesGrid`	It enables the display of the property panel in the report designer. The `true` value is set by default.
`showReportTree`	It enables the display of report components tree. The `true` value is set by default.
`propertiesGridPosition`	It allows you to define the position of the property panel to the left or to the right: • `StiPropertiesGridPosition::Left` – the property panel will be displayed on the left (value is set by default); • `StiPropertiesGridPosition::Right` – the property panel will be displayed on the right.
`showSystemFonts`	It allows you to display or not to display system fonts in the list of fonts. The property has the `true` value by default, i.e. system fonts are displayed in the list of fonts.
`datePickerFirstDayOfWeek`	It allows you to set the first day of the week for the **Date picker** element: • `StiFirstDayOfWeek::Auto` - Monday or Sunday will be set as the first day of the week depending on browser culture (value is set by default); • `StiFirstDayOfWeek::Monday` - Monday will be set as the first day of the week; • `StiFirstDayOfWeek::Sunday` - Sunday will be set as the first day of the week.

formatForDateControls	This feature allows you to customize the format for date controls. By default, the current option doesn't have a specified value, and the date format is determined based on the browser's locale.
undoMaxLevel	It sets max depth of report changes cancel when its editing. It exerts influence over memory usage. 6 changes are set by default.
wizardTypeRunningAfterLoad	It allows you to call the master of report creation after the report designer is run. It can take one of the specified below enumeration values: • StiWizardType::None - the report designer will be run without the invoke of report creation master (value is set by default); • StiWizardType::StandardReport - the report designer will be run with the invoke of standard report creation master; • StiWizardType::MasterDetailReport - the report designer will be run with the invoke of master-detail report creation master; • StiWizardType::LabelReport - the report designer will be run with the invoke of the report creation master with labels; • StiWizardType::InvoicesReport - the report designer will be run with the invoke of the invoice creation master; • StiWizardType::OrdersRep

	ort - the report designer will be run with the invoke of the order creation master; • StiWizardType::Quotation Report - the report designer will be run with the invoke of the quota creation master.
allowWordWrapTextEditors	It enables or disables line break in editors of text in the designer. The property has the true value by default.
allowLoadingCustomFontsToClientSide	Enables or disables the transfer of custom fonts to the client-side. The default value is false.
enableShortCutKeys	Enables or disables the handling of keyboard shortcuts. By default, this property is set to true.
defaultRibbonType	Allows setting the default style for the designer's toolbar. It can take one of the following values from the enumeration: • StiDesignerRibbonType::Classic - the classic style will be set (default value); • StiDesignerRibbonType::SingleLine – a compact single-line style will be set.

Toolbar

Name	Description
visible	It enables the display of toolbar in the report designer. The property has the true value by default.
showPreviewButton	It enables or disables the display of

	the **Preview** button in the designer toolbar. The property has the `true` value by default.
`showSaveButton`	It enables the display of the **Save** button in the designer toolbar. The property has the `true` value by default.
`showAboutButton`	It enables the display of the **About** button in the designer toolbar. The property has the `false` value by default.
`showFileMenu`	It enables the display of the main menu of the report designer. The property has the `true` value by default.
`showFileMenuNew`	It enables the display of the **New** main menu item. The property has the `true` value by default.
`showFileMenuOpen`	It enables the display of the **Open** main menu item. The property has the `true` value by default.
`showFileMenuSave`	It enables the display of the **Save** main menu item. The property has the `true` value by default.
`showFileMenuSaveAs`	It enables the display of the **Save as** main menu item. The property has the `true` value by default.
`showFileMenuClose`	It enables the display of the **Close** main menu item. The property has the `true` value by default.
`showFileMenuExit`	It enables the display of the **Exit** main menu item. The property has the `false` value by default.
`showFileMenuReportSetup`	It enables the display of the **Report Setup** main menu item. The property

	has the `true` value by default.
`showFileMenuOptions`	It enables the display of the **Options** main menu item. The property has the `true` value by default.
`showFileMenuInfo`	It enables the display of the **Info** main menu item. The property has the `true` value by default.
`showFileMenuAbout`	It enables the display of the **About** main menu item. The property has the `true` value by default.
`showFileMenuNewReport`	It enables or disables the display of the **New Page** main menu item. The property has the `true` value by default.
`showFileMenuNewDashboard`	It enables or disables the display of the **New Dashboard** main menu item. The property has the `true` value by default.
`showSetupToolboxButton`	It enables or disables the display of the report components side panel settings invoke button. The property has the `true` value by default.
`showNewPageButton`	It enables or disables the display of the **New Page** button in the toolbar. The property has the `true` value by default.
`showNewDashboardButton`	It enables or disables the display of the **New Dashboard** button in the toolbar. The property has the `true` value by default.

Bands

Name	Description

showReportTitleBand	It enables the display of the **Report Title** band in the designer components insert menu. The property has the `true` value by default.
showReportSummaryBand	It enables the display of the **Report Summary** band in the designer components insert menu. The property has the `true` value by default.
showPageHeaderBand	It enables the display of the **Page Header** band in the designer components insert menu. The property has the `true` value by default.
showPageFooterBand	It enables the display of the **Page Footer** band in the designer components insert menu. The property has the `true` value by default.
showGroupHeaderBand	It enables the display of the **Group Header** band in the designer components insert menu. The property has the `true` value by default.
showGroupFooterBand	It enables the display of the **Group Footer** band in the designer components insert menu. The property has the `true` value by default.
showHeaderBand	It enables the display of the **Header** band in the designer components insert menu. The property has the `true` value by default.
showFooterBand	It enables the display of the **Footer** band in the designer components

	insert menu. The property has the `true` value by default.
`showColumnHeaderBand`	It enables the display of the **Column Header** band in the designer components insert menu. The property has the `true` value by default.
`showColumnFooterBand`	It enables the display of the **Column Footer** band in the designer components insert menu. The property has the `true` value by default.
`showDataBand`	It enables the display of the **Data** band in the designer components insert menu. The property has the `true` value by default.
`showHierarchicalBand`	It enables the display of the **Hierarchical** band in the designer components insert menu. The property has the `true` value by default.
`showChildBand`	It enables the display of the **Child** band in the designer components insert menu. The property has the `true` value by default.
`showEmptyBand`	It enables the display of the **Empty** band in the designer components insert menu. The property has the `true` value by default.
`showOverlayBand`	It enables the display of the **Overlay** band in the designer components insert menu. The property has the `true` value by default.
`showTable`	It enables the display of the **Table** component in the designer components insert menu. The

	property has the `true` value by default.
`showTableOfContents`	It enables the display of the **Table of Contents** band in the designer components insert menu. The property has the `true` value by default.

Cross-Bands

Name	Description
`showCrossTab`	It enables the display of the **Cross-Tab** component in the designer components insert menu. The property has the `true` value by default.
`showCrossGroupHeaderBand`	It enables the display of the **Cross-Group Header** band in the designer components insert menu. The property has the `true` value by default.
`showCrossGroupFooterBand`	It enables the display of the **Cross-Group Footer** band in the designer components insert menu. The property has the `true` value by default.
`showCrossHeaderBand`	It enables the display of the **Cross-Header** band in the designer components insert menu. The property has the `true` value by default.
`showCrossFooterBand`	It enables the display of the **Cross-Footer** band in the designer components insert menu. The property has the `true` value by

	default.
showCrossDataBand	It enables the display of the **Cross-Data** band in the designer components insert menu. The property has the true value by default.

Components

Name	Description
showText	It enables the display of the **Text** component in the designer components insert menu. The property has the true value by default.
showTextInCells	It enables the display of the **Text in Cells** component in the designer components insert menu. The property has the true value by default.
showRichText	It enables the display of the **Rich Text** component in the designer components insert menu. The property has the true value by default.
showImage	It enables the display of the **Image** component in the designer components insert menu. The property has the true value by default.
showBarCode	It enables the display of the **Bar Code** component in the designer components insert menu. The property has the true value by default.

showShape	It enables the display of the **Shape** component in the designer components insert menu. The property has the `true` value by default.
showPanel	It enables the display of the **Panel** component in the designer components insert menu. The property has the `true` value by default.
showClone	It enables the display of the **Clone** component in the designer components insert menu. The property has the `true` value by default.
showCheckBox	It enables the display of the **Check Box** component in the designer components insert menu. The property has the `true` value by default.
showSubReport	It enables the display of the **Text** component in the designer components insert menu. The property has the `true` value by default.
showZipCode	It enables the display of the **Sub-Report** component in the designer components insert menu. The property has the `true` value by default.
showChart	It enables the display of the **Chart** component in the designer components insert menu. The property has the `true` value by default.
showGauge	It enables the display of the **Gauge**

	component in the designer components insert menu. The property has the `true` value by default.
`showSparkline`	It enables the display of the **Sparkline** component in the designer components insert menu. The property has the `true` value by default.
`showMathFormula`	It enables the display of the **Math Formula** component in the designer components insert menu. The property has the `true` value by default.
`showMap`	It enables the display of the **Map** component in the designer components insert menu. The property has the `true` value by default.
`showElectronicSignature`	It enables or disables the display of the **Electronic Signature** component in the toolbox or Insert tab in the designer. The property has the `true` value by default.
`showPdfDigitalSignature`	It enables or disables the display of the **PDF Digital Signature** component in the toolbox or Insert tab in the designer. The property has the `true` value by default.
`showHorizontalLinePrimitive`	It enables or disables the display of the **Horizontal Line** component in the toolbox or Insert tab in the designer. The property has the `true` value by default.
`showVerticalLinePrimitive`	It enables or disables the display of the **Vertical Line** component in the

	toolbox or Insert tab in the designer. The property has the `true` value by default.
`showRectanglePrimitive`	It enables or disables the display of the **Rectangle** component in the toolbox or Insert tab in the designer. The property has the `true` value by default.
`showRoundedRectanglePrimitive`	It enables or disables the display of the **Rounded Rectangle** component in the toolbox or Insert tab in the designer. The property has the `true` value by default.

dashboardElements

Name	Description
`showTableElement`	It enables the display of the **Table** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showCardsElement`	It enables the display of the **Cards** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showChartElement`	It enables the display of the **Chart** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showGaugeElement`	It enables the display of the **Gauge** dashboard element in the toolbox or the **Insert** tab in the designer. The

	property has the `true` value by default.
`showPivotTableElement`	It enables the display of the **Pivot** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showIndicatorElement`	It enables the display of the **Indicator** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showProgressElement`	It enables the display of the **Progress** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showRegionMapElement`	It enables the display of the **Region Map** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showOnlineMapElement`	It enables the display of the **Online Map** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showImageElement`	It enables the display of the **Image** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showWebContentElement`	It enables the display of the dashboard element **Web Content** in the toolbox or the **Insert** tab in the designer. The property has the `true`

	value by default.
`showTextElement`	It enables the display of the **Text** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showPanelElement`	It enables the display of the **Panel** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showShapeElement`	It enables the display of the **Shape** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showButtonElement`	It enables the display of the **Button** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showListBoxElement`	It enables the display of the **List Box** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showComboBoxElement`	It enables the display of the **Combo Box** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.
`showTreeViewElement`	It enables the display of the **Tree View** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the `true` value by default.

showTreeViewBoxElement	It enables the display of the **Tree View Box** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the true value by default.
showDatePickerElement	It enables the display of the **Date Picker** dashboard element in the toolbox or the **Insert** tab in the designer. The property has the true value by default.

Dictionary

Name	Description
showAdaptersInNewConnectionForm	It enables the display of the **Object** category in the new connection creation window. The property has the true value by default.
showDictionary	It enables the display of the report dictionary. The property has the true value by default.
newReportDictionary	Allows you to use aliases in the data dictionary. It can take one of the specified below enumeration values: • StiNewReportDictionary::Auto - defines the mode of using aliases from a saved value in cookies (default value); • StiNewReportDictionary::DictionaryNew - sets the mode to create a new data dictionary when creating a new report; • StiNewReportDictionary::DictionaryMerge - sets the mode to join the existing data

	dictionary with a new one when creating a new report in the designer
`useAliases`	It allows you to create a new data dictionary or join the existing one when creating a new report in the designer. It can take one of the specified below enumeration values: • `StiUseAliases::Auto` - defines the mode to create or join the data dictionary from a saved value in cookies (default value); • `StiUseAliases::True` - sets the mode of using aliases in the data dictionary • `StiUseAliases::False` - disables the mode of using aliases in the data dictionary.
`showDictionaryContextMenuProperties`	Sets a visibility of the **Properties** item in the dictionary context menu. By default, the property is set to `true`.
`showDictionaryActions`	Sets a visibility of the **Actions** menu on the dictionary toolbar. By default, the property is set to **true**.
`dataSourcesPermissions`	It sets available actions on report data sources. It can take one or several values from the `StiDesignerPermissions` enumeration.
`dataConnectionsPermissions`	It sets available actions on connections to report data. It can take one or several values from the `StiDesignerPermissions` enumeration.

`dataColumnsPermissions`	It sets available actions on report data columns. It can take one or several values from the `StiDesignerPermissions` enumeration.
`dataRelationsPermissions`	It sets available actions on report data connections. It can take one or several values from the `StiDesignerPermissions` enumeration.
`businessObjectsPermissions`	It sets available actions on report business objects. It can take one or several values from the `StiDesignerPermissions` enumeration.
`variablesPermissions`	It sets available actions on report variables. It can take one or several values from the `StiDesignerPermissions` enumeration.
`resourcesPermissions`	It sets available actions on sources in report dictionary. It can take one or several values from the `StiDesignerPermissions` enumeration.
`dataTransformationsPermissions`	Sets the available actions on data transformation. It can take one or more values from the **StiDesignerPermissions** enumeration.

In the table below you can see all available values for the `StiDesignerPermissions` enumeration. They can be set for report dictionary elements.

Name	Description

`StiDesignerPermissions::None`	Disables any action on the item of the data dictionary.
`StiDesignerPermissions::Create`	It allows you to create definite element of the dictionary.
`StiDesignerPermissions::Delete`	It allows you to delete a definite element of the dictionary.
`StiDesignerPermissions::Modify`	It allows you to edit a definite element of the dictionary.
`StiDesignerPermissions::View`	It allows you to view a definite element of the dictionary.
`StiDesignerPermissions::ModifyView`	It allows you to edit and view a definite element of the dictionary.
`StiDesignerPermissions::All`	It allows you to make any actions on the dictionary element.

You can configure the built-in `StiViewer` component used to preview the report. To get access to all of its settings, you should use the `viewerOptions` property, which is an object of the viewer options. All its properties are described in the Viewer settings section. For example, you want to change the report display mode and disable unnecessary export formats:

designer.php

```
<?php
   use Stimulsoft\Designer\StiDesigner;
   use Stimulsoft\Viewer\Enums\StiHtmlExportMode;

   $designer = new StiDesigner();
   $designer->options->viewerOptions->appearance->reportDisplayMode =
   StiHtmlExportMode::FromReport;
   $designer->options->viewerOptions->exports->ShowExportToWord = false;
   $designer->options->viewerOptions->exports->showExportToCsv = false;
?>
```

11 Reports and Dashboards for WinForms

YouTube

Watch the video tutorials for working with the components of the Stimulsoft Reports.NET product. Subscribe to the Stimulsoft channel and be the first to watch new video lessons. Questions and suggestions is recommended be left in the comments to the video.

Samples

See on our website examples for working with the Reports.NET components. All examples are separate projects, grouped into one solution for Visual Studio. Also, you can view and download specific examples on GitHub.

11.1 Activation

YouTube

Watch videos which show how to activate the WinForms components. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this done in various ways. Below is an example of activating the **WinForms** component.

Form1.cs

```
...
public partial class Form1 : Form
{
   public Form1()
   {
      //Activation with using license code
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";

      //Activation with using license file
```

```
      Stimulsoft.Base.StiLicense.LoadFromFile("license.key");

      //Activation from byte array
      Stimulsoft.Base.StiLicense.LoadFromBytes(bytes);

      //Activation from stream
      Stimulsoft.Base.StiLicense.LoadFromStream(stream);

      //Activation from assembly
      Stimulsoft.Base.StiLicense.LoadFromEntryAssembly(assembly,
      "stimulsoft-license.key");

      InitializeComponent();
   }
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

11.2 WinForms Viewer

The **StiViewerControl** component is used to view reports in the WinForms. The component can show a report, zoom, save rendered reports to various formats, print reports, send them to a recipient via Email.

- How to Show Report
- Dot-Matrix Viewer for WinForms

11.2.1 How to Show Report

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `Report.Render()` method if you want to perform specific actions with the rendered report before it is displayed in the viewer. Likewise, when using the compilation mode, you need to call the `Report.Compile()` method only if you have actions to perform with the compiled report before it is built and shown in the viewer.

Just call only one method to show a report:

C#

```
...
StiReport report = new StiReport();
report.Load("report.mrt");
report.Show();
...
```

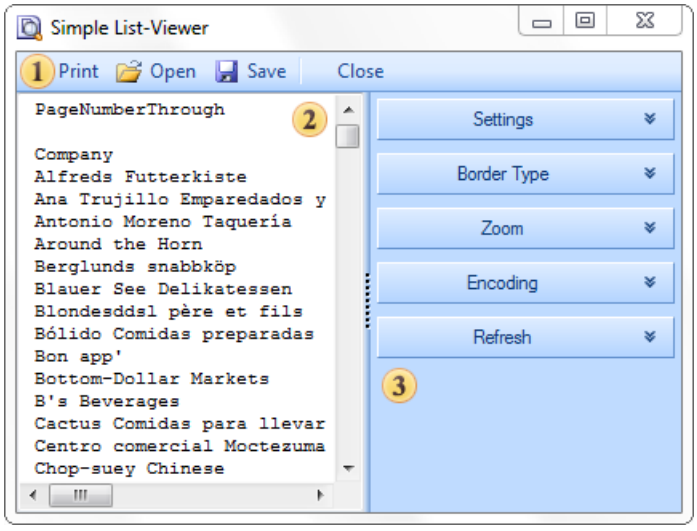
VB.NET

```
...
Dim Report As StiReport = New StiReport()
Report.Load("report.mrt")
Report.Show()
...
```

If the report was not rendered before showing, the **Show** method will render a report using the **Render** method.

11.2.2 Dot-Matrix Mode of WinForms Viewer

The **Dot-Matrix** viewer is designed to preview the report before printing it on dot matrix printer. The Dot matrix printer is used to print only the text and characters of pseudographics. Accordingly the viewer displays only the text and borders of objects as pseudographics characters. The picture below shows the Dot-matrix viewer dialog box:

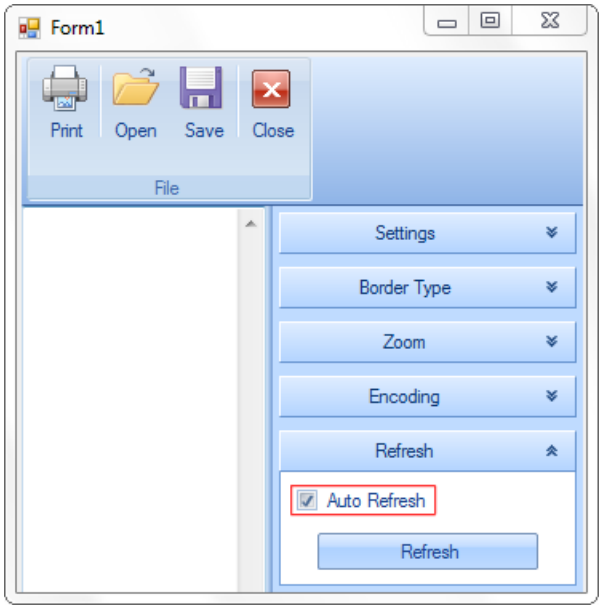


1. The Dot-matrix viewer toolbar.
2. The panel displays the text of a report

3. The options bar of a report.

Setting Dot-Matrix Viewer in WinForms

The **Dot-Matrix** viewer can be configured from code using static properties. Depending on the value of the static properties in the Dot-matrix viewer, these or that parameters will be specified. For example, the **AutoRefresh** property. The picture below shows the **Dot-Matrix** viewer dialog box:

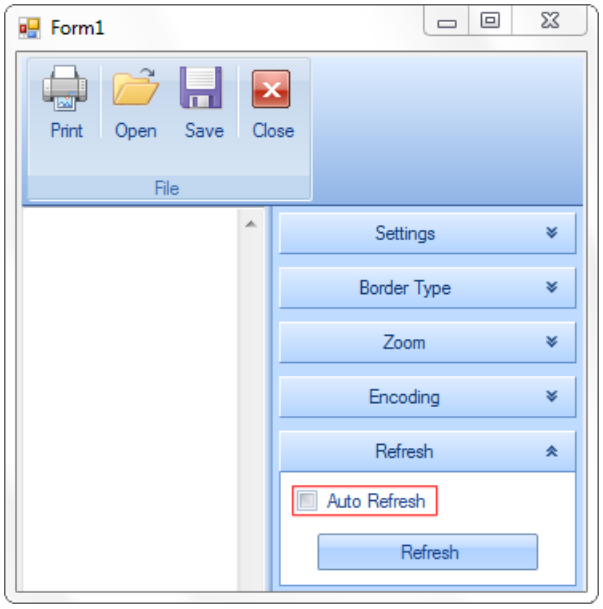


As can be seen on the picture above, the **Auto Refresh** property is enabled. This means that the **AutoRefresh** static property of the **Dot-Matrix** viewer is set to true. If the AutoRefresh static property is set to false, then the AutoRefresh property in the Dot-Matrix viewer is disabled. Add the following code into the project code:

C#

```
...
StiOptions.Viewer.DotMatrix.AutoRefresh = false;
...
```

Thus, the AutoRefresh property will be disabled. The picture below shows the Dot-Matrix viewer dialog box with disabled auto refresh function:



Most parameters can be set using the static properties.

Dot-Matrix and Escape Codes

For inserting the escape sequence to text the commands that may look like <#command> should be used as seen in the code sample below:

```
Normal text <#b> Bold text <#/b><#i> Italic text <#/i> Again normal text
```

Also commands of selecting bold, italic or underlined text are automatically inserted depending on the style of the text box font. When printing to matrix printer and exporting to text format these commands are changed on appropriate escape sequences.
The **StiEscapeCodesCollection** is used for this process. It is inherited from the Hashtable class. This is a collection of "key-value" pairs where the key is the command and value is the escape-sequence. For different types of printers different collections with different set of command can be defined. Collections are stored in the **StiOptions.Export.Txt.EscapeCodesCollectionList** static variable. By default, the following collections will be created: "None", "EpsonFX", "Oki ML92/93". The "None" collection is empty and used to output the text without escape codes.

Command/Collection	EpsonFX	Oki ML92/93
b	ESC E	ESC T
/b	ESC F	ESC I
i	ESC 4	
/i	ESC 5	
u	ESC -1	ESC H
/u	ESC -0	ESC D
sup	ESC S0	ESC J
/sup	ESC T	ESC K
sub	ESC S1	ESC L
/sub	ESC T	ESC M
condensed	0x0F	0x1d
/condensed	0x12	0x1e
elite	ESC M	0x1c
pica	ESC P	0x1e
doublewidth	ESC W1	0x1f
/doublewidth	ESC W0	0x1e

It is possible to add new collections of change the existing ones. The selection of the required collection is done by the name. If the collection with the name is not found then the "None" collection is used. The collection name can be selected from the DotMatrixViewer settings and passed as an option to the exporting and printing methods.

11.3 Add custom functions

Information

When working in the report designer and using undo/redo commands, the operation stack is saved as JSON. However, if it is necessary to save the operation stack as XML, you should set the option `StiOptions.Designer.UndoRedoMode = StiUndoRedoMode.Xml`.

You can add a custom function to the Dictionary in the report designer when you integrate it into your application. After adding the custom function, you can use this in creating reports and dashboards. Below is the example of adding a function for calculating the sum total.

Designer.cs

```
...
StiFunctions.AddFunction("MyCategory", "MySum",
"description", typeof(MyClass),
typeof(decimal), "Calculates a sum of the specified set of values.",
new[] { typeof(object) },
new[] { "values" },
new[] { "A set of values" }).UseFullPath = false;
...
public static decimal MySum(object value)
{
   if (!ListExt.IsList(value))
   return Stimulsoft.Base.Helpers.StiValueHelper.TryToDecimal(value);
   return
   Stimulsoft.Data.Functions.Funcs.SkipNulls(ListExt.ToList(value)).TryCast
   ToDecimal().Sum();
}
...
```

12 Reports and Dashboards for Python

Stimulsoft Reports.PYTHON and Stimulsoft Dashboards.PYTHON comprise a combination of a JavaScript client and a Python server, along with all the necessary functionality for their interaction. The report is built and exported on the client side using JavaScript, while the Python server side contains everything needed to work with report files and communicate with various SQL data sources. Communication between the client and the server is carried out through AJAX requests that transmit and receive JSON data in a specific format. To enhance the usability of the product, special events and functions have been introduced both on the JavaScript client side and on the Python server side.

12.1 Report Engine

The reporting tool Stimulsoft Reports.PYTHON allows you to download, build and export a report in various formats without deploying a viewer and designer. This allows you to avoid loading a large number of scripts on the client side and increases the speed and ease of operation.

The report is built and exported on the client side using JavaScript; the Python server side contains everything needed to work with report files and communicate with various SQL data sources. Communication between the client and the server is carried out through AJAX requests that transmit and receive JSON data in a specific format. For ease of use of the product, special events and functions have been developed both on the JavaScript client side and on the Python server side.

Dashboards

The dashboard tool Stimulsoft Dashboards.PYTHON allows loading and analyze data, exporting and printing analytical panels to various formats without deployment of the viewer and designer.

All data analysis, except for certain SQL operations, is performed on the client side using JavaScript; the Python server-side contains everything needed to work with dashboard files and communicate with various SQL data sources. The client communicates with the server in exactly the same way as in the reporting tool; the same events and functions are used.

12.1.1 Deployment

Examples
The complete code sample can be found on GitHub.

To use the reporting tool, just install `stimulsoft-reports` or `stimulsoft-`

dashboards package using the package installer by running the following command:

console

```
python -m pip install stimulsoft-reports
```

console

```
python -m pip install stimulsoft-dashboards
```

The latest available version of the reporting tool will be installed in the current workspace, after which you can use classes and functions for working with reports.

Information

For code examples, the **Flask** framework is used as one of the most popular and easy-to-understand frameworks. You can use any web framework because all classes and functions for working with the reporting tool are completely independent.

The `StiReport` class is designed to work with a reporting tool in a web project. Using this class, you can create a report, load a report from a file or a line, build a report, or export a report. For example, if you need to load a report from a file, build it, and export it to HTML:

app.py

```
from flask import Flask, render_template, url_for, request
from stimulsoft_reports.report import StiReport
from stimulsoft_reports.report.enums import StiExportFormat

app = Flask(__name__)

@app.route('/report', methods = ['GET', 'POST'])
def report():
    report = StiReport()
```

```
    if report.processRequest(request):
        return report.getFrameworkResponse()

    report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
    report.render()
    report.exportDocument(StiExportFormat.HTML)

    js = report.javascript.getHtml()
    html = report.getHtml()
    return render_template('report.html', reportJavaScript = js,
reportHtml = html)
```

index.html

```
<!DOCTYPE html>
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Render and Export a Report</title>

   {{ reportJavaScript|safe }}
</head>

<body>
   {{ reportHtml|safe }}
</body>

</html>
```

In this example, an instance of the `StiReport` object is created, a report template is loaded from the SimpleList.mrt file located in the static files directory, and the command to build and export the report is called. Next, the JavaScript and HTML code of the components necessary for the operation of the reporting tool are generated. The `report.javascript.getHtml()` function generates HTML code for connecting the necessary resources, while the `report.getHtml()` function generates HTML code for working with the reporting tool. The generated HTML code is passed as parameters to the HTML template `report.html`, where it is displayed in the required places.

The special function `report.processRequest(request)` processes the current request. If it returns `True`, the request is intended for the reporting tool, and you need to return the result of its execution instead of the page template. This is described in more detail in a separate section of the documentation.

Information

Our products Stimulsoft Reports.PYTHON and Stimulsoft Dashboards.PYTHON do not have a native Python report engine. Report generation and export are performed on the client side using JavaScript code. Therefore, when using Python code to work with components, you should call the getHtml() function, which will return all the necessary JavaScript code to add to the web page.

Framework support

Currently, there is built-in support for the three main Python frameworks - **Flask**, **Django**, and **Tornado**. Universal functions for working in any other frameworks are also available. For code examples, the **Flask** framework was used, as it is one of the most popular and easy-to-understand codes; product deployment for it was discussed above. Below are examples of the same code for other frameworks. They are all very similar and differ only in the features used for a specific framework.

Django

app.py

```
from django.shortcuts import render
from django.templatetags.static import static
from stimulsoft_reports.report import StiReport
from stimulsoft_reports.report.enums import StiExportFormat

def report(request):
    report = StiReport()
    if report.processRequest(request):
        return report.getFrameworkResponse()

    report.loadFile(static('reports/SimpleList.mrt'))
    report.render()
    report.exportDocument(StiExportFormat.HTML)

    js = report.javascript.getHtml()
    html = report.getHtml()
    return render(request, 'report.html', {'reportJavaScript': js,
'reportHtml': html})
```

report.html

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Render and Export a Report</title>

    {{ reportJavaScript|safe }}
</head>

<body>
    {{ reportHtml|safe }}
</body>

</html>
```

Tornado

app.py

```
import asyncio, os
from tornado.web import Application, RequestHandler, url
from stimulsoft_reports.report import StiReport
from stimulsoft_reports.report.enums import StiExportFormat

class ReportHandler(RequestHandler):
    def get(self):
        report = StiReport()
        if report.processRequest(request):
            return report.getFrameworkResponse(self)

        report.loadFile(self.static_url('reports/SimpleList.mrt'))
        report.render()
        report.exportDocument(StiExportFormat.HTML)

        js = report.javascript.getHtml()
        html = report.getHtml()
        self.render('report.html', reportJavaScript = js, reportHtml =
html)

    def post(self):
        handler = StiHandler()
        if handler.processRequest(self.request):
            return handler.getFrameworkResponse(self)
```

report.html

```
<!DOCTYPE html>
```

```
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Render and Export a Report</title>

    {% raw reportJavaScript %}
</head>

<body>
    {% raw reportHtml %}
</body>

</html>
```

Universal function

app.py

```
from stimulsoft_reports.report import StiReport
from stimulsoft_reports.report.enums import StiExportFormat

def report():
    report = StiReport()
    query = 'query string'
    body = 'post data'
    if report.processRequest(None, query, body):
        response = report.getResponse()
        data = response.data
        contentType = response.contentType
        mimetype = response.mimetype

    report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
    report.render()
    report.exportDocument(StiExportFormat.HTML)

    js = report.javascript.getHtml()
    html = report.getHtml()
```

Information

In most cases, to work with the product, you should use only Python code, which provides interaction with all the main features. To configure the product more precisely and utilize all the capabilities of the JS reporting software, you must use JavaScript code. The option to deploy a product using only JavaScript code is described in the Reports and Dashboards for JS section. In this case, the use of

Python code is required only to connect data adapters.

Various deployment and optimization options are discussed in the section Optimizing scripts loading.

12.1.2 Optimizing Scripts Loading

Due to the impressive functionality of the product, the scripts are quite large. When loading a web application for the first time, or when browser caching is disabled, loading may take some time, especially on a low-speed internet connection. We offer two options for solving this problem: using packaged scripts or employing partial functionality and loading only what is required.

Packed scripts

Packed scripts have the same structure as regular scripts but end with `*.pack.js` in the file name. Such scripts contain a block of packed data in the form of a JavaScript variable and a compact unpacker. When loading all scripts, the unpacker automatically unpacks all downloaded data and launches the prepared script for execution. Unpacking takes some time, but under certain circumstances - such as with a slow internet connection - this time is much less than the loading speed of regular scripts.

To use packaged scripts, all you should do is set the `javascript.usePacked` property of the report object to `True`, for example:

app.py

```
report = StiReport()
report.javascript.usePacked = True
```

Partial loading of scripts

When deploying the reporting tool, by default only one file with scripts, `stimulsoft.reports.js`, is loaded. It contains all the functionality for building and exporting reports. If you only need some of the capabilities to generate reports, you can download only the required parts of the generator that contain a specific set

of capabilities. For example, if your reports do not use maps, you don't have to download them. This will speed up the loading of the web project and reduce memory consumption by the browser.

Information

This feature is implemented only for the report engine; the viewer and designer cannot be divided into parts; their scripts will be loaded entirely in one block.

To use partial loading of scripts, just set the necessary options for the `javascript` property of the report object:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()

report.javascript.reportsSet = False
report.javascript.reportsChart = True
report.javascript.reportsExport = True
report.javascript.reportsImportXlsx = True
report.javascript.reportsMaps = True
```

Each `javascript` property option controls the loading of a script containing specific functionality. This table presents the entire set of scripts that can be loaded separately:

Name	Description
`javascript.reportsSet`	Contains a complete set of scripts for working with the reporting tool. It should be set to False when using partial script loading.
`javascript.reportsExport`	Contains algorithms for exporting the generated report to various formats - PDF, HTML, Excel, RichText,

	and others.
`javascript.reportsChart`	Contains components for working with all types of charts in a report.
`javascript.reportsMaps`	Contains components for working with regional and online maps.
`javascript.blocklyEditor`	Contains the Blockly visual editor for creating event scripts in the report. The event handler itself is built into the reporting engine.
`javascript.reportsImportXlsx`	Contains algorithms for working with Excel data sources.

Information

The report viewer and report designer components also have a `javascript` property, with which you can control the configuration of scripts the way described above.

12.1.3 License Activation

After purchasing the product, you need to activate the license for the components you use. There are several ways to activate the license.

Activation using code

To activate using a string, just copy the encrypted license text from your account, and register it using the special `setKey()` function of the `license` property of the report object:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.license.setKey('Your activation code...')
```

Activation using a license.key file

To activate using a license file, just download the `license.key` file from your account, and copy it to the Web project folder, for example, to the `static` folder with static files. Then, just register it using the special `setFile()` function of the `license` property of the report object:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.license.setFile(url_for('static', filename='license.key'))
```

Information

The report viewer and report designer components also have the `license` property using which you can manage the license key in the way described above.

Protecting the license key

If you activate a license using a string, you can add its code under a certain condition. For example, the `sessionID` variable stores information about the current client session; you need to add a license key only for authorized users:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()

if sessionID != None:
    report.license.setKey('Your activation code...')
```

It would also work if you change the location and name of the license key file, for example:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.license.setFile(url_for('private', filename='a15fc0ef64e6.key'))
```

12.1.4 Loading and Saving Reports

Information

The Stimulsoft MRT format file is a description of reports with XML or JSON markup. You can use MRT files created in other Stimulsoft report designers.

To load a report using Python code, you can use one of the functions listed below on the `StiReport` object. Each of the functions takes as input either the name of the report file or the report itself as a string.

Name	Description
`loadFile(filePath: str, load: bool = False)`	Loads a report from an MRT file on the client side, with the file path specified in the function arguments. If the `load` parameter is set to `True`, the report file will be loaded on the server side and then transferred to the client side as a `Base64` encoded string.
`load(data: str, fileName: str = 'Report')`	Loads a report from an XML or JSON string and sends it to the client side as a `Base64`-encoded string. The `fileName` parameter specifies the file name to be used for saving and exporting the report in the future.
`loadPacked(data: str, fileName: str = 'Report')`	Transmits a report to the client side as a `Base64`-encoded string specified in the `data` parameter. The `fileName` parameter specifies the file name that will be used for subsequent saving and exporting of

	the report.

For example, you may need to load a report from a file on the server side located in a private directory and transfer it to the client side:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.loadFile(url_for('private', filename='reports/SimpleList.mrt'),
True)
report.render()
```

Since the reporting tool for Python is based on a JavaScript platform and renders and exports reports on the client side, you should use events and JavaScript functions to save a report or document template. You can find more details in the section Reporting tool events. An example of saving a generated report as a string can be implemented as follows:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.onAfterRender += 'afterRender'
report.loadFile(url_for('private', filename='reports/SimpleList.mrt'),
True)
report.render()
```

report.html

```
<script>
    function afterRender(args) {
        let json = args.report.saveDocumentToJsonString();
    }
</script>
```

Information

Stimulsoft Reports.PYTHON and Stimulsoft Dashboards.PYTHON support saving

MRT files only in JSON format. MRT files in XML format are only supported in download mode and will be automatically converted to JSON format upon saving.

Loading and saving documents (rendered reports)

The rendered report can be saved as a document in JSON format for later viewing or exporting. The document contains the rendered report pages but lacks the source data and does not maintain a connection to it.

To load a document using Python code, use one of the functions listed below on the `StiReport` object. Each function takes either the name of the document file or the document itself as a string as input.

Name	Description
`loadDocumentFile(filePath: str, load: bool = False)`	Loads a document from an MDC file on the client side, with the path specified in the function parameters. If the `load` parameter is set to `True`, the document file will be loaded on the server side and transferred to the client side as a `Base64`-encoded string.
`loadDocument(data: str, fileName: str = 'Report')`	Loads a document from an XML or JSON string and transfers it to the client side as a `Base64`-encoded string. The `fileName` parameter specifies the file name to be used for subsequent saving and exporting of the report.
`loadPackedDocument(data: str, fileName: str = 'Report')`	Transfers the document to the client side as a `Base64`-encoded string specified in the `data` parameter. The `fileName` parameter specifies the file name to be used for subsequent saving and exporting of the report.

For example, you need to load a document from a file and then export it.

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.loadDocumentFile(url_for('static', filename='reports/
SimpleList.mdc'))
report.exportDocument(StiExportFormat.HTML)
```

Information

Because dashboards always require data, they cannot be saved as documents. Stimulsoft Dashboards.PYTHON supports saving dashboards only as templates.

Information

When saving a document from the viewer menu, the file is saved in JSON format and uses the extension MDC for a standard document, MDZ for a packed document, and MDX for an encrypted document.

12.1.5 Report Rendering

To build a loaded report, you need to call the `render()` function on the `StiReport` report object. For example, you need to build a report before exporting it:

app.py

```
from stimulsoft_reports.report import StiReport
from stimulsoft_reports.report.enums import StiExportFormat

report = StiReport()
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
report.exportDocument(StiExportFormat.PDF)
```

If you need to perform any actions after building a report using JavaScript code, simply utilize the `onAfterRender` event on the report object. More details about this are provided in the Reporting tool engine. For instance, after building a report, you may need to display the following message:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.onAfterRender += 'afterRender'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

render.html

```
<script>
    function afterRender(args) {
        alert("The report rendering is completed.");
    }
</script>
```

If you need to perform any actions with the report before building it using JavaScript code, just use the `onBeforeRender` event on the report object. For example, before rendering a report, you need to register JSON data:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.onBeforeRender += 'beforeRender'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

render.html

```
<script>
    function onBeforeRender(args) {
```

```
        var dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");
        dataSet.readJsonFile("Demo.json");

        var report = args.report;
        report.regData(dataSet.dataSetName, "", dataSet);
    }
</script>
```

12.1.6 Event Handler

The reporting tool, along with the viewer and report designer, can trigger events on the client side, transfer them to the Python server for further processing, and receive a response prepared on the server. All actions are implemented in the event handler, eliminating the need for any additional functions to communicate between the client and server and transfer data. To handle a specific event, simply add the function name to the handler, and the specified function will be automatically called when the event occurs. Events can be called on both the JavaScript client side and the Python server side. If needed, you can add multiple functions of any type to the same event.

Calling a JavaScript event on the client side

To trigger a JavaScript event on the client side, add the function name as a string to the handler. All necessary data will be passed through the event arguments, which can be viewed in the section Reports Engine Events.

For example, after generating a report, you may want to display a message indicating the number of pages in the resulting document:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.onAfterRender += 'afterRender'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

report.html

```
<script>
    function afterRender(args) {
        let pagesCount = args.report.renderedPages.count;
        alert("The report is rendered, pages: " + pagesCount);
```

```
    }
</script>
```

In this example, you can obtain a JavaScript report object from the event arguments and read the number of pages in the document that was generated.

Information

More information about the available functions and parameters of the JavaScript reporting tool can be found in the documentation for the Stimulsoft Reports.JS and Stimulsoft Dashboards.JS products.

Calling a Python event on the server side

To call a Python event, add the function name as a variable to the handler. All necessary data will be passed in the event arguments; the list of available arguments can be viewed in the Report Engine Events.

For example, before requesting data, you may need to adjust the password in the connection string:

app.py

```
from stimulsoft_reports.events import StiDataEventArgs

def beginProcessData(args: StiDataEventArgs):
    args.connectionString = args.connectionString.replace('Pwd=;',
'Pwd=******;')

report = StiReport()
report.onBeginProcessData += beginProcessData
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

In this example, you can retrieve and modify all the database connection parameters from the event arguments.

The event on the Python server side allows you to return a text message indicating successful completion or an error during event processing; this message will be displayed in the viewer or designer after the event concludes. To display an error message window, you must return the result of the `StiResult.getError('Error message')` function. To display a window with an information message, you can return the result of the `StiResult.getSuccess('Info message')` function or simply the string `'Info message'`.

app.py

```
from stimulsoft_reports import StiResult
from stimulsoft_reports.designer import StiDesigner
from stimulsoft_reports.events import StiReportEventArgs
from stimulsoft_reports.report import StiReport

def saveReport(args: StiReportEventArgs):
    #StiResult.getError('An error occurred while saving.')
    #StiResult.getSuccess('The report was successfully saved.')
    return 'The report was successfully saved.'

designer = StiDesigner()
designer.onSaveReport += saveReport
```

Information

When an error occurs in the event handler, such as a failure to connect to the database or an error processing a file, an internal message will be displayed regardless of whether a message is defined in the component event.

Information

The message dialog box will only appear when using the StiViewer or StiDesigner components. The reporting tool does not include visual forms, so event processing messages will be displayed in the browser console.

Calling several identical events

You can add an unlimited number of functions to the event handler; they will all be

grouped by event type and called sequentially in the order they were added. For example, you might need to modify the SQL query on the JavaScript client side and adjust the connection string on the Python server side:

app.py

```
from stimulsoft_reports.events import StiDataEventArgs

def beginProcessData(args: StiDataEventArgs):
    args.connectionString = args.connectionString.replace('Pwd=;',
'Pwd=;')

report = StiReport()
report.onBeginProcessData += beginProcessData
report.onBeginProcessData += 'beginProcessData'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

viewer.html

```
<script>
    function beginProcessData(args) {
        args.queryString = args.queryString.replace('TableName',
'Products')
    }
</script>
```

Information

Some events can only be triggered on the JavaScript client side and cannot be triggered on the Python server side. When adding a function to such an event, no error will occur, but the function will not be called. All supported options are listed in the section Report Engine Events.

12.1.7 Connecting Data Files

Typically, the connection parameters for data sources are stored in the report template itself. For working with file-based data sources such as XML, JSON, Excel, or CSV, don't required additional actions, as all algorithms are contained within the report generator script. If necessary, other methods of data connection can be used with the report generator's JavaScript functions. To do this, the `onBeforeRender` event of the `StiReport` object can be used. Data can be loaded directly into a

special `DataSet` object, which is used for storing them. It contains functions for loading data from XML/XSD and JSON file formats. After loading the files, the `regData()` function of the report's JavaScript object must be called to connect the data, with the prepared `DataSet` object specified in the function's arguments.

Example of loading data from a file:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.onBeforeRender += 'beforeRender'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

report.html

```
<script>
    function beforeRender(args) {
        var dataSet = new Stimulsoft.System.Data.DataSet("SimpleDataSet");

        dataSet.readXmlSchemaFile("Demo.xsd");
        dataSet.readXmlFile("Demo.xml");

        //dataSet.readJsonFile("Demo.json");

        var report = args.report;
        report.regData(dataSet.dataSetName, "", dataSet);
        report.dictionary.synchronize();
    }
</script>
```

Information

Loading the data schema isn't necessarily. If you want to use a data schema, you should add it before loading the XML data.

In addition to the mentioned functions `readXmlFile()` and `readJsonFile()`, there are also functions `readXml()` and `readJson()`, which accept data in the form of a string or an object.

Information

The `report.dictionary.synchronize()` function is required to synchronize the connected data with the report template's data dictionary. In other words, when this function is called, the report dictionary will be created based on the structure of the data loaded into the `DataSet`. The synchronization function is not needed if the dictionary is created in advance and its structure matches the connected data.

Data loading event

To view and modify the connection parameters of file data before loading, you need to define the `onBeginProcessData` event for the report object, or the viewer or designer component:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.onBeginProcessData += 'beginProcessData'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

report.html

```
<script>
    function beginProcessData(args) {
        let pathData = args.pathData;
    }
</script>
```

In the event arguments will contain information about the connection to the file data source, including the connection name and type in the report template, as well as the path to the data file. A detailed description of the available argument values can be found in the Report Engine Events section.

It is allowed to change the path to the data file. In this case, after the event

completes, the report generator will request the file from the new path specified in the arguments. For example, if you need to change the path to the JSON data file for the specified connection:

report.html

```
<script>
    function beginProcessData(args) {
        if (args.connection == "MyJsonConnection")
            args.pathData = "Data/Demo.json";
    }
</script>
```

Information

For an XML data source, the `onBeginProcessData` event will be triggered twice: first for reading the XSD schema, and second for reading the actual XML data file.

Data Processing Event

To view and modify the connection parameters of file data before loading, you need to define the `onEndProcessData` event for the report object:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.onEndProcessData += 'endProcessData'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

report.html

```
<script>
    function endProcessData(args) {
        let dataSet = args.dataSet;
    }
</script>
```

In the event arguments will include information about the connection to the file data source such as the connection name and type saved in the report template and the prepared `DataSet` object containing tables and rows of data retrieved from the file source. A detailed description of the available argument values can be found in the Report Engine Events section.

Information

The report viewer and report designer components also have properties for the events mentioned above, which can be used to manage data loading in the manner described.

Using variables in the data file

It is possible to use variables in the form of expressions (as well as using expressions) when specifying the path to the file data source. A variable or expression is defined within curly braces. Multiple expressions can be used anywhere in the file path, for example:

File Data Source

```
https://localhost/data/{VariableJsonFileName}.json
https://localhost/data/json?id={VariableId}
https://localhost/{VariableCategory}/{VariableId}
```

So, a single data source can be transformed into REST syntax, eliminating the need to create multiple similar data sources to obtain uniform data. Combined with server-side Python logic and report generator events, this makes the data source even more flexible.

Using OData

You can also use data retrieved from OData repositories for creating reports. In this case, authorization is required using a username, password, or token. The authorization parameters are specified in the connection string to the OData

repository, using the ";" separator.

report.html

```
<script>
    function beginProcessData(args) {
        let report = args.report;

        // Authorization using a user account
        let oDataDatabase = new
Stimulsoft.Report.Dictionary.StiODataDatabase("OData", "OData", "https://
services.odata.org/V4/Northwind/
Northwind.svc;AddressBearer=adress;UserName=UserName;Password=Password;Cli
ent_Id=Your Client ID", false, null);

        // Authorization using a user token
        let oDataDatabase = new
Stimulsoft.Report.Dictionary.StiODataDatabase("OData", "OData", "https://
services.odata.org/V4/Northwind/Northwind.svc;Token=Enter your token",
false, null);

        report.dictionary.databases.add(oDataDatabase);
        report.dictionary.synchronize();

        // Query with data filter
        var productsDataSource =
report.dictionary.dataSources.getByName("Products");
        if (productsDataSource != null) productsDataSource.sqlCommand =
"Products?filter=ProductID eq 2";
    }
</script>
```

12.1.8 Connecting SQL Data Adapters

To generate reports, the report generator allows using data from various SQL sources. Since pure JavaScript doesn't have built-in methods for working with remote databases, this functionality is implemented using server-side Python code.

Modifying connection parameters and data retrieval

Working with SQL data sources does not require any additional actions; all data adapters are already connected and configured. If it is necessary to process parameters used for connecting to the data, the `onBeginProcessData` event of the report object is provided. This event can be triggered on either the client- or server-side. The event arguments will contain all necessary parameters for connecting to the SQL data source, as well as the SQL query parameters. Detailed descriptions of the available argument values can be found in the Report Engine Events section.

All connection parameters to the data source can be modified both on the client- and server-sides. For example, you may need to change the SQL query on the client-side and the connection string on the server-side:

app.py

```
from stimulsoft_reports.events import StiDataEventArgs
from stimulsoft_reports.report import StiReport

def beginProcessData(args: StiDataEventArgs):
    if args.connection == 'MyConnectionName':
        args.connectionString =
'Server=localhost;Database=test;uid=root;password=******;'

report = StiReport()
report.onBeginProcessData += beginProcessData
report.onBeginProcessData += 'beginProcessData'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

report.html

```
<script>
    function beginProcessData(args) {
        if (args.dataSource == "MyDataSource")
            args.queryString = "SELECT * FROM ProductsTable";
    }
</script>
```

For each SQL data source, you can specify SQL query parameters, which are stored in the data source itself as a collection. This collection is also passed in the event arguments. It is an array of objects containing the parameter name, type, and value, for example:

app.py

```
from stimulsoft_reports.events import StiDataEventArgs

def beginProcessData(args: StiDataEventArgs):
    args.parameters = [
        {
            name: 'ParameterString',
            type: 752,
            typeName: 'Text',
            value: 'Text value'
```

```
        },
        {
            name: 'ParameterInt',
            type: 3,
            typeName: 'Int32',
            value: 20
        }
    ]
```

It is allowed to modify the values of query parameters, however, the type of the new value must match the type of the parameter being modified, for example:

app.py

```
from stimulsoft_reports.events import StiDataEventArgs

def beginProcessData(args: StiDataEventArgs):
    if args.parameters != None:
        args.parameters['Parameter1'].value = 'TableName'
```

Information

All SQL query parameter types will correspond to the available types of the database tables being connected to. More details about SQL query parameters are discussed in this section in the chapter Using Parameters in SQL Queries.

Thus, in the `onBeginProcessData` event, you can determine the database type, connection name, and data source name, as well as view and, if necessary, adjust the connection string and SQL query for data retrieval, and set the query parameter values. When modifying argument values on the server-side, the modified values will not be passed to the client-side, allowing the use of confidential data such as login and password in the connection string, table names, prefixes, etc.

Processing loaded data

To view or adjust the loaded data before connecting and generating the report, the `onEndProcessData` event of the report object is provided. The event arguments

will include all necessary connection parameters to the SQL data source, as well as the query result, containing column names, column types, and data rows retrieved from the SQL source. A detailed description of the available argument values can be found in the Report Engine Events section.

The data object of the executed SQL query has the following structure:

app.py

```
from stimulsoft_reports.events import StiDataEventArgs
from stimulsoft_reports.report import StiReport

def endProcessData(args: StiDataEventArgs):
    args.result = {
        columns: ['id', 'username', 'phone'],
        types: ['int', 'string', 'string'],
        rows: [
            [1, 'Mario Pontes', '555-6874'],
            [2, 'Helen Bennett', '555-2376']
        ]
    }

report = StiReport()
report.onEndProcessData += endProcessData
report.onEndProcessData += 'endProcessData'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

report.html

```
<script>
    function endProcessData(args) {
        args.result = {
            columns: ["id", "username", "phone"],
            types: ["int", "string", "string"],
            rows: [
                [1, "Mario Pontes", "555-6874"],
                [2, "Helen Bennett", "555-2376"]
            ]
        };
    }
</script>
```

The available properties of the data object are listed in the table:

Name	Description
columns	Column names of the SQL data source table.
types	Column types of the SQL data source table, converted to known types for the report generator.
rows	Data rows from the SQL data source, represented as an array of arrays of all table rows.

All data from the SQL query result can be adjusted, and in this case, after the event is completed, the report will be generated using the modified data.

Using parameters in an SQL query

If necessary, you can use parameters in an SQL query. To do this, add parameters to a special collection in the data source and set the required type and default value for each parameter. After that, the parameters can be used in the SQL query as follows:

SQL Data Source

```
SELECT * FROM @Parameter1 WHERE UserID = @Parameter2
```

All parameter values will be sent to the server-side as a separate collection and can be reviewed and modified before executing the SQL query. To access the parameter values, use the `args.parameters` collection in the `onBeginProcessData` event, for example:

app.py

```
from stimulsoft_reports.events import StiDataEventArgs

def beginProcessData(args: StiDataEventArgs):
    if args.dataSource == 'DataSourceWithParams':
        args.parameters['Parameter1'].value = 'TableName'
        args.parameters['Parameter2'].value = 10
```

Information

New parameter values in this collection will not be sent to the client-side, so it is permissible on the Python server-side to assign confidential data as values.

Using report variables as SQL parameters

It is possible to use a variable as an SQL parameter. To do this, set the property **Allow using as SQL parameter** in the report variable editor, after which it can be used in any SQL query. The syntax will be exactly the same as when using parameters in the data source.

Information

Such a variable will be included in the parameters collection only if it is used in the query. Parameters from the data source collection are always passed, even if they are not used in the query.

Escaping parameter values

All parameter values will be automatically escaped to prevent SQL injections and ensure query execution security. If escaping is not required and you control the security of parameter values yourself, automatic escaping can be disabled. To do this, set the `escapeQueryParameters` property to `False` in the event handler:

app.py

```
from stimulsoft_reports.report import StiReport

@app.route('/report', methods = ['GET', 'POST'])
def report():
    report = StiReport()
    report.handler.escapeQueryParameters = False
    if report.processRequest(request):
        return report.getFrameworkResponse()
```

After setting the specified property, using parameters becomes unsafe, and you must strictly control the values before executing SQL queries.

Information

Escaping applies only to SQL query parameters and variables used as parameters. If a variable is used as an expression, i.e., within curly braces, such as `{VariableName}`, escaping will not be applied in any case. A detailed description of how variables work can be found in the Working with Report Variables section.

12.1.9 Working with Report Variables

The report generator allows using variables in expressions, queries, filters, and other report elements. It also supports the preview and modification of variable values from code before the report is generated.

Report template variable values

The report generator provides easy access to variables within a report template through the data dictionary. To achieve this, define the `onBeforeRender` event for the report object, where a JavaScript report object will be passed in the event arguments on the client-side. A detailed description of available argument values can be found in the Report Engine Events section.

To access a report variable, use the JavaScript function `getByName()` on the collection of variables within the report's data dictionary. To change a variable's value, simply assign a new value to its `value` property; there is no need to check the variable's type, as type conversion will be handled automatically. For example, to change the string and integer values of specific variables:

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.onBeforeRender += 'beforeRender'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

report.html

```
<script>
    function beforeRender(args) {
        let report = args.report;

        let variableString =
report.dictionary.variables.getByName("VariableString");
        variableString.value = "Text value";

        let variableInt =
report.dictionary.variables.getByName("VariableInt");
        variableInt.value = "20";
    }
</script>
```

Information

The variable values will be changed in the report template, and if the template is subsequently saved, these changes will be saved in the file. To modify variable values without changes the template, you can use the `onPrepareVariables` event, which is triggered when preparing variable values before generating the report.

Direct access to variables in the report template on the Python server-side isn't provided.

Variable values when generating a report

The report generator allows easy access to variables before the report is built. To achieve this, define the `onPrepareVariables` event for the report object. The event arguments will contain a collection of report variables with their types and values. If a variable is initialized as an expression, the evaluated value of the expression will be included in the collection. A detailed description of available argument values can be found in the Report Engine Events section.

In the client-side JavaScript event, the collection of variables is represented as an array of objects containing the variable name, type, and value. It is permissible to change variable values, but the new value's type must match the type of the variable being modified.

app.py

```
from stimulsoft_reports.report import StiReport

report = StiReport()
report.onPrepareVariables += 'prepareVariables'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

report.html

```
<script>
    function prepareVariables(args) {
        args.variables = [
            {
                name: "VariableString",
                type: "String",
                value: "Text value"
            },
            {
                name: "VariableInt",
                type: "Int32",
                value: 20
            }
        ];
</script>
```

In the Python server-side event, the collection of variables is represented as a dictionary of `StiVariable` objects, where each object contains the variable's name, type, and value. The dictionary key corresponds to the name of the report variable.

app.py

```
from stimulsoft_reports.report import StiReport
from stimulsoft_reports.events import StiVariablesEventArgs

def prepareVariables(args: StiVariablesEventArgs):
    variableName = args.variables['Variable1'].name
    variableType = args.variables['Variable1'].typeName
    variableValue = args.variables['Variable1'].value

report = StiReport()
report.onPrepareVariables += prepareVariables
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

It is allowed to change variable values, but the new value's type must match the type of the variable being modified. To change the value, simply assign the new value to the variable's `value` property, for example:

app.py

```
from stimulsoft_reports.events import StiVariablesEventArgs

def prepareVariables(args: StiVariablesEventArgs):
    args.variables['VariableString'].value = 'Value from Server-Side'
    args.variables['VariableInt'].value = 123
    args.variables['VariableDecimal'].value = 123.456
```

Variables of type **DateTime** are passed as `datetime` objects. To change its value, simply assign a new `datetime` object, for example:

app.py

```
from datetime import datetime
from stimulsoft_reports.events import StiVariablesEventArgs

def prepareVariables(args: StiVariablesEventArgs):
    args.variables['VariableDateTime'].value = datetime.today()
```

Variables of type Range are passed as a dictionary with two keys: `from` and `to`. To access and modify the values of a **Range** variable, use these keys. The format of each of the two values is the same as for a simple variable. For example, to change the values of a variable of type `StringRange`:

app.py

```
from stimulsoft_reports.events import StiVariablesEventArgs

def prepareVariables(args: StiVariablesEventArgs):
    args.variables['VariableStringRange'].value['from'] = 'Aaa'
    args.variables['VariableStringRange'].value['to'] = 'Zzz'
```

To change the value of a **List** variable, you need to access the list value by its index. You can also set the values of the entire list at once by using a prepared array, for example

app.py

```
from stimulsoft_reports.events import StiVariablesEventArgs

def prepareVariables(args: StiVariablesEventArgs):
    args.variables['VariableStringList'].value[0] = 'Test'
    args.variables['VariableIntList'].value = [1, 22, 333]
```

It is also permissible to create a new report variable if needed. To create a new variable not defined in the report, you must add a new `StiVariable` object to the variables dictionary using the new variable name. After that, the variable can be used for generating the report, meaning the variable will not be saved in the report template.

app.py

```
from stimulsoft_reports.report import StiVariable
from stimulsoft_reports.report.enums import StiVariableType

def prepareVariables(args: StiVariablesEventArgs):
    args.variables['NewVarInt'] = StiVariable('NewVarInt',
StiVariableType.INT, 10)
```

After processing the `onPrepareVariables` event, a new collection will be sent to the client, containing only those variables whose values have been changed, as well as any new variables.

Information

If a variable is used in an SQL query as an expression, i.e., enclosed in braces, such as `{VariableName}`, its value will not be automatically escaped. You need to ensure the security of these values yourself or use the variable as a query parameter, such as **@VariableName**. A detailed description of parameter you can

found in the Connecting SQL Data Adapters section.

Report variables passed in URL query

The report generator can automatically assign values to variables passed in the URL query. To enable this, set the `passQueryParametersToReport` property to `True` in the event handler:

app.py

```
@app.route('/report', methods = ['GET', 'POST'])
def report():
    report = StiReport()
    report.handler.passQueryParametersToReport = True
    if report.processRequest(request):
        return report.getFrameworkResponse()
```

All other actions the report generator will handle automatically. If a variable is present in the report, its value will be updated with the value from the URL query. If the variable does not exist in the report, it will be created for the report generation, meaning that the new variable will not be saved in the report template. Variable names passed in the URL query are case-insensitive.

Information

All variable values will be assigned before the `onPrepareVariables` event is triggered, allowing you to further control and adjust the assigned values if necessary during this event.

12.1.10 Printing a Report From Code

The reporting tool allows you to print a report from code. To achieve this, you can utilize the special `print()` method on the report object.

app.py

```
report = StiReport()
```

```
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.print()
```

By default, all pages of the generated report will be printed. It is possible to specify a page or range of pages to print. To achieve this, simply pass the required value as a parameter to the `print()` function. For example:

app.py

```
report = StiReport()
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))

report.print(5)
report.print('1,3-8')
```

12.1.11 Exporting a Report From Code

The reporting tool allows you to export the generated report or dashboard to various formats. Below is a list of all available export formats for reports and dashboards:

Name	Reports	Dashboards
Document (Snapshot)	+	+
Adobe PDF	+	+
Microsoft XPS	+	-
Microsoft PowerPoint (.pptx)	+	-
HTML	+	+
HTML5	+	-
Text	+	-
Microsoft Word (.docx)	+	-
Microsoft Excel (.xlsx)	+	+
OpenDocument Writer (.odt)	+	-
OpenDocument Calc (.ods)	+	-

Comma Separated Value (.csv)	+	+
Scalable Vector Graphics (.svg)	+	+

To export a report, you should utilize the special `exportDocument()` function on the report object.

app.py

```
from stimulsoft_reports.report import StiReport
from stimulsoft_reports.report.enums import StiExportFormat

report = StiReport()
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
report.exportDocument(StiExportFormat.PDF)
```

Information

Exporting a report does not automatically trigger its construction. Therefore, the loaded report template must first call the `render()` function, which initiates the pre-building of the report. For generated reports, calling the specified function is not necessary.

As arguments to the `exportDocument()` function, you should specify the required export format from the `StiExportFormat` enumeration. The available format options are as follows:

Name	Description
`StiExportFormat.DOCUMENT`	Saves the rendered document.
`StiExportFormat.PDF`	Saves to Adobe PDF.
`StiExportFormat.XPS`	Saves to XPS (XML Paper Specification).

StiExportFormat.POWERPOINT	Saves to Microsoft PowerPoint.
StiExportFormat.HTML	Saves to HTML.
StiExportFormat.HTML5	Saves to HTML5, using SVG markup elements.
StiExportFormat.TEXT	Saves to text.
StiExportFormat.WORD	Saves to Microsoft Word.
StiExportFormat.EXCEL	Saves to Microsoft Excel.
StiExportFormat.ODT	Saves to OpenDocument Text.
StiExportFormat.ODS	Saves to OpenDocument Spreadsheet.
StiExportFormat.CSV	Saves to CSV (Comma-Separated Values).
StiExportFormat.IMAGE_SVG	Saves to SVG.

After the report is exported, the resulting data stream will be transferred to the web browser for downloading as a file. The file name and MIME data type will be detected automatically.

12.1.12 Report Engine Events

The report generator supports events that provide the ability to perform necessary operations before specific actions—both on the JavaScript client-side and the Python server-side. To call an event on the client-side, you need to add the name of the JavaScript function to the event as a string. To call an event on the server-side, you need to add the Python function itself to the event. Any number of functions, both client and server, can be added to a single event.

Example of adding multiple functions of different types to an event:

app.py

```
def prepareVariables(args: StiVariablesEventArgs):
    variables = args.variables

report = StiReport()
report.onPrepareVariables += prepareVariables
```

```
report.onPrepareVariables += 'prepareVariables'
report.onAfterRender += 'afterRender'
report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
report.render()
```

report.html

```
<script>
    function prepareVariables(args) {
        let variables = args.variables;
    }

    function afterRender(args) {
        alert("The report rendering is completed.");
    }
</script>
```

Depending on the event, the handler may operate on both the JavaScript client-side and the Python server-side simultaneously, or only on the JavaScript client-side. This is because the JavaScript core of the generator, which operates on the client-side, is used for building and exporting the report, while the server-side Python code is used for working with data, to which the JavaScript side doesn't have direct access. Each event description will specify which handler can be used.

Some event arguments accept values from enumerations that are located in specific namespaces. All enumerations used in the report engine events are listed in the code block below:

app.py

```
from stimulsoft_reports.enums import StiDataCommand, StiDatabaseType,
StiEventType
```

The report generator supports the following events:

- onBeforeRender
- onAfterRender
- onPrepareVariables

- onBeginProcessData
- onEndProcessData

onBeforeRender
[v] JavaScript [x] Python

The event is triggered before the report is generated. The list of properties passed in the event arguments on the JavaScript client-side:

Name	Description
`event`	Identifier of the current event, with the value `"BeforeRender"`.
`sender`	Identifier of the component that initiated this event, which can have the following values: • `"Report"` • `"Viewer"` • `"Designer"`
`report`	The current report object.

onAfterRender
[v] JavaScript [x] Python

The event is triggered after the report is generated. The list of properties passed in the event arguments on the JavaScript client-side:

Name	Description
`event`	Identifier of the current event, with the value `"AfterRender"`.

`sender`	Identifier of the component that initiated this event, which can have the following values: • `"Report"` • `"Viewer"` • `"Designer"`
`report`	The current report object.

onPrepareVariables
[v] JavaScript [v] Python

The event is triggered before the report is generated, after the report variables are prepared. The table below lists the event handler arguments on the JavaScript client-side:

Name	Description
`event`	Identifier of the current event, with the value `"PrepareVariables"`.
`sender`	Identifier of the component that initiated this event, which can have the following values: • `"Report"` • `"Viewer"` • `"Designer"`
`report`	The current report object.
`variables`	Collection of report variables and their values.
`preventDefault`	This flag allows stopping further event processing by the report generator. The default value is

	false.

The list of properties passed in the event arguments on the Python server-side. The arguments are of type `StiVariablesEventArgs`:

Name	Description
event	Identifier of the current event, with the value `StiEventType.PREPARE_VARIABLES` for this event.
sender	Identifier of the component that initiated this event, which can have the following values: • `StiHandler` • `StiReport` • `StiViewer` • `StiDesigner`
variables	Collection of report variables and their values.

onBeginProcessData
[v] JavaScript [v] Python

The event is triggered before querying the data needed to generate the report. For file data sources, only the JavaScript event handler is supported. Below is the list of properties passed in the event arguments on the JavaScript client-side:

Name	Description

`event`	Identifier of the current event, with the value `"BeginProcessData"`.
`sender`	Identifier of the component that initiated this event, which can have the following values: • `"Report"` • `"Viewer"` • `"Designer"`
`report`	The current report object.
`command`	Identifier of the current command, which can have the following values: • `"TestConnection"` - checks the connection; • `"ExecuteQuery"` - queries data from the specified SQL source; • `"GetSchema"` - reads the XSD schema from the file source; • `"GetData"` - reads data from the file source.
`connection`	Name of the current data source connection specified in the report template.
`database`	Name of the current database, which can have the following values: • `"XML"` • `"JSON"` • `"Excel"` • `"CSV"` • `"MySQL"` • `"MS SQL"` • `"PostgreSQL"`

	• "Firebird" • "Oracle" • "MongoDB" • "ODBC"
dataSource	Name of the current data source specified in the report template. Set only for SQL data sources.
connectionString	Connection string to the SQL data source.
queryString	SQL query for data retrieval. Used only with the ExecuteQuery command.
pathData	Path to the data source file specified in the report template. Set only for XML and JSON data sources.
pathSchema	Path to the data schema file specified in the report template. Set only for the XML data source.
parameters	Collection of parameters and their values specified in the SQL data source.
preventDefault	This flag allows stopping further event processing by the report generator. The default value is false.

The list of properties passed in event arguments on the Python server-side.

Arguments are of type `StiDataEventArgs`:

Name	Description
`event`	Identifier of the current event, with the value `StiEventType.BEGIN_PROCESS_DATA` for this event.
`sender`	Identifier of the component that initiated this event, which can have the following values: • `StiHandler` • `StiReport` • `StiViewer` • `StiDesigner`
`command`	Identifier of the current command, which can have the following values: • `StiDataCommand.TEST_CONNECTION` - checks the connection; • `StiDataCommand.RETRIEVE_SCHEMA` - queries the database schema for NoSQL data sources; • `StiDataCommand.EXECUTE_QUERY` - queries data from the specified SQL or NoSQL source; • `StiDataCommand.EXECUTE` - executes a stored procedure from the specified SQL source.
`connection`	Name of the current data source connection specified in the report template.

database	Name of the current database, which can have the following values: • StiDatabaseType.MYSQL • StiDatabaseType.MSSQL • StiDatabaseType.POSTGRESQL • StiDatabaseType.FIREBIRD • StiDatabaseType.ORACLE • StiDatabaseType.MONGODB • StiDatabaseType.ODBC
dataSource	Name of the current data source specified in the report template.
connectionString	Connection string to the SQL data source.
queryString	SQL query for data retrieval. Used only with the StiDataCommand.EXECUTE_QUERY command.
parameters	Collection of parameters and their values specified in the SQL data source. Values are always passed in their original (unescaped) form.

onEndProcessData
[v] JavaScript [v] Python

This event is triggered after data is loaded and before the report is generated. For file data sources, only the JavaScript event handler is supported. Below is the list of properties passed in the event arguments on the JavaScript client-side:

Name	Description
`event`	Identifier of the current event, with the value `"EndProcessData"`.
`sender`	Identifier of the component that initiated this event, which can have the following values: • `"Report"` • `"Viewer"` • `"Designer"`
`report`	The current report object.
`command`	Identifier of the current command, which can have the following values: • `"ExecuteQuery"` - data retrieved from the specified SQL source. • `"GetData"` - data retrieved from the file source.
`database`	Name of the current database, which can have the following values:: • `"XML"` • `"JSON"` • `"Excel"` • `"CSV"` • `"MySQL"` • `"MS SQL"` • `"PostgreSQL"` • `"Firebird"` • `"Oracle"` • `"MongoDB"` • `"ODBC"`
`connection`	Name of the current data source

	connection specified in the report template.
`dataSource`	Name of the current data source specified in the report template. Set only for SQL data sources.
`dataSet`	The prepared `Stimulsoft.System.Data.DataSet`, object containing tables and data rows from the file source.
`result`	A collection of columns and their types, along with data rows retrieved from the SQL source..

The list of properties passed in the event arguments on the Python server-side. The arguments are of type `StiDataEventArgs`:

Name	Description
`event`	Identifier of the current event, with the value `StiEventType.END_PROCESS_DATA` for this event.
`sender`	Identifier of the component that initiated this event, which can have the following values: • `StiHandler`

	• StiReport • StiViewer • StiDesigner
command	Identifier of the current command, which can have the following values: • StiDataCommand.TEST_CONNECTION - checks the connection; • StiDataCommand.RETRIEVE_SCHEMA - requests the database schema for NoSQL data sources; • StiDataCommand.EXECUTE_QUERY - queries data from the specified SQL or NoSQL source; • StiDataCommand.EXECUTE - executes a stored procedure from the specified SQL source.
connection	Name of the current data source connection specified in the report template.
database	Name of the current database, which can have the following values: • StiDatabaseType.MYSQL • StiDatabaseType.MSSQL • StiDatabaseType.POSTGRESQL • StiDatabaseType.FIREBIRD • StiDatabaseType.ORACLE • StiDatabaseType.MONGODB • StiDatabaseType.ODBC
dataSource	Name of the current data source specified in the report template. Set only for SQL data sources.

`queryString`	The final SQL query with all parameters used for data retrieval. Used only with the StiDataCommand.EXECUTE_QUERY command.
`result`	A collection of columns and their types, along with data rows retrieved from the SQL or NoSQL source.

12.2 HTML5 Viewer

The report viewer is a Python component called `StiViewer`, designed for viewing, printing, and exporting reports in a browser window on any computer with any operating system. The viewer supports various themes, an animated interface, bookmarks, interactive reports, editing report elements on the page, full-screen mode, search, and other features essential for report viewing.

The viewer can display both the report template and an already generated report. If a report template is used, the viewer will automatically build it using the JavaScript report generator. Full support for working with interactive analytical dashboards is implemented. The functioning of the report generator is discussed in detail in the Report Engine section.

The viewer's interface is built using HTML5, making it usable on virtually any modern platform. The component uses AJAX technology to perform all actions (loading and building reports, connecting to data, paging and zooming, interactivity in reports, printing, exporting, etc.), eliminating the need to reload the entire page and increasing performance, as well as enabling the component's use in single-page applications. The JavaScript technology used for building reports allows for use with virtually any low-performance server-side.

Information

Such as dashboards and reports use the same unified MRT template format, as well as the same methods for loading templates and working with data, the term "report" will be used throughout the documentation.

12.2.1 Deployment

Examples

The complete code sample can be found on GitHub.

To use the viewer, simply install `stimulsoft-reports` or `stimulsoft-dashboards` package using the package manager by running the following command:

console

```
python -m pip install stimulsoft-reports
```

console

```
python -m pip install stimulsoft-dashboards
```

The latest available version of the product will be installed in the current workspace, after which you can use the classes and functions for working with reports and dashboards.

Information

The examples in the documentation use the **Flask** framework, as it's one of the most popular and easy-to-understand frameworks. You can use any web framework, as all classes and functions for working with the components are universal.

To add a viewer to a web project, the `StiViewer` class is intended. Using this class, you can create a viewer object, set the necessary settings, handle a request and return its result, and get the prepared JavaScript and HTML code for the component. Here is an example of displaying the viewer on an HTML page:

app.py

```
from flask import Flask, render_template, url_for, request
from stimulsoft_reports.viewer import StiViewer

app = Flask(__name__)

@app.route('/viewer', methods = ['GET', 'POST'])
def viewer():
    viewer = StiViewer()
    viewer.options.appearance.fullScreenMode = True

    if viewer.processRequest(request):
        return viewer.getFrameworkResponse()

    js = viewer.javascript.getHtml()
    html = viewer.getHtml()
    return render_template('viewer.html', viewerJavaScript = js,
viewerHtml = html)
```

viewer.html

```
<!DOCTYPE html>
<html>

<head>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Showing a Report in the Viewer</title>

    {{ viewerJavaScript|safe }}
</head>

<body>
    {{ viewerHtml|safe }}
</body>

</html>
```

In this example, an instance of the `StiViewer` object is created, and the necessary settings are configured.

The special component function `processRequest(request)` processes the current request. If the function returns `True`, it means the request was successfully processed, and the result of its execution should be returned. More details about this are provided in the Event Handler section.

Next, it's necessary to generate the prepared JavaScript and HTML code required for the viewer to work. The `viewer.javascript.getHtml()` function generates the HTML code to connect the necessary scripts and resources, and the `viewer.getHtml()` function generates the HTML code for the component itself. The generated code is passed as parameters to the HTML template viewer.html and displayed in the designated locations.

Information

Our products Stimulsoft Reports.PYTHON and Stimulsoft Dashboards.PYTHON don't have a native report generator core in Python. Report generation and export are performed on the client-side using JavaScript code. When using Python code to work with the components, you need to call the `getHtml()` function, which will return all the necessary JavaScript and HTML code for the report generator and components to work.

A simplified viewer deployment is available without using an HTML page template. For example, the same example can be implemented using only Python code:

app.py

```
from flask import Flask, url_for, request
from stimulsoft_reports.viewer import StiViewer

app = Flask(__name__)

@app.route('/viewer', methods = ['GET', 'POST'])
def viewer():
    viewer = StiViewer()
    viewer.options.appearance.fullScreenMode = True

    if viewer.processRequest(request):
        return viewer.getFrameworkResponse()

    # Here is the code for working with the report

    return viewer.getFrameworkResponse()
```

In this case, after calling the event handler and performing actions with the report, the final result of the request is returned immediately. That is, the viewer will return a fully prepared HTML page with all the necessary scripts and code. Other methods of operation are also available, which are described in more detail in the Event Handler section.

Information

In most cases, it is sufficient to use only Python code to work with the product, which allows interaction with all the main features of the components. For detailed product customization and using all the features of the JS generator, you need to use JavaScript code. The method of deploying the product using only JavaScript code is described in the section Reports and Dashboards for JS, in this case, Python

code is only required to connect data adapters.

To generate a response, the `getFrameworkResponse()` function is used, which returns a ready object suitable for the currently used web framework. The current framework is determined at the time of request processing. Our components support working with popular frameworks like **Django**, **Flask**, and **Tornado**. To process the request and generate a response in web projects that don't use these frameworks, you need to pass the current GET and POST data to the request processing function, and to generate a response, use the `getResponse()` function, which will return all the necessary data. For example, you can implement the viewer event handling in the following way:

app.py

```
from flask import Flask, make_response, render_template, url_for, request
from stimulsoft_reports.viewer import StiViewer

app = Flask(__name__)

@app.route('/viewer', methods = ['GET', 'POST'])
def viewer():
    viewer = StiViewer()
    viewer.options.appearance.fullScreenMode = True

    query = request.args.to_dict()
    body = request.get_data(False)
    if viewer.processRequest(None, query, body):
        viewerResponse = viewer.getResponse()
        response = make_response(viewerResponse.data)
        response.mimetype = viewerResponse.mimetype
        response.headers.add('Access-Control-Allow-Origin',
request.origin)
        return response

    js = viewer.javascript.getHtml()
    html = viewer.getHtml()
    return render_template('viewer.html', viewerJavaScript = js,
viewerHtml = html)
```

12.2.2 Activation

After purchasing the product, you need to activate the license for the components you use. There are several ways to connect a license key.

All options for activating components are described in the License activation section

and have the same functions and call parameters.

12.2.3 Showing Reports

Notice

When assigning a report viewer to a component, the report is automatically generated. Calling the `report.render()` method is necessary only if you need to perform some actions with the generated report before displaying it in the viewer.

To display a report in the viewer, simply create a `StiReport` object, load a report template into it, and assign the resulting object to the viewer. All other actions will be performed automatically; the viewer will build a report and display the first page.

app.py

```
from flask import Flask, url_for, request
from stimulsoft_reports.report import StiReport
from stimulsoft_reports.viewer import StiViewer

app = Flask(__name__)

@app.route('/viewer', methods = ['GET', 'POST'])
def viewer():
    viewer = StiViewer()
    viewer.options.appearance.fullScreenMode = True

    if viewer.processRequest(request):
        return viewer.getFrameworkResponse()

    report = StiReport()
    report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
    viewer.report = report

    return viewer.getFrameworkResponse()
```

The viewer carries out the rendering of the report and is able to display both report templates and documents (generated reports). A detailed description of working with various report and document formats is found in the section Loading and Savin a Report.

12.2.4 Localization

The viewer supports full localization of the UI. To localize the interface into the required language, just set the required file name for the `localization` option in the viewer:

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.options.localization = 'de.xml'
```

The localization XML files can be found in the resources of the installed product package. If necessary, the localization file can be loaded from any other location; for this you need to specify the full path to the desired XML file for the `localization` option:

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.options.localization = '/resources/loc/de.xml'
```

If the file is readable from the Python application, the localization will be loaded into the viewer. Otherwise, the built-in English localization of the interface will be used.

12.2.5 Printing a Report

The viewer provides several options for printing a report, each with its own features, advantages, and disadvantages.

Print to PDF

Printing is done by exporting the report to PDF format. The advantages include higher accuracy in the placement and printing of report elements compared to other printing options. A disadvantage is the requirement of a PDF viewer plugin installed

in the browser (modern browsers typically have built-in tools for viewing and printing PDF files).

Print with preview
The report is printed in a separate browser popup window in HTML format. The report can be previewed before sending it to the printer or copied as text or HTML code. Advantages include cross-browser compatibility for printing and no need for special plugins. The disadvantage is the relatively low accuracy in the placement of report elements due to the specifics of HTML formatting. The report is printed in a separate browser popup window in HTML format. The report can be previewed before sending it to the printer or copied as text or HTML code. Advantages include cross-browser compatibility for printing and no need for special plugins. The disadvantage is the relatively low accuracy in the placement of report elements due to the specifics of HTML formatting.

Print without preview
The report is printed directly to the printer without a preview. After selecting this menu option, the system print dialog is displayed. Since the report is printed in HTML format in this mode, the print quality is similar to that of printing with a preview.

Information

Report printing is carried out using the built-in methods of the browser in use, so the appearance of the print dialog window may vary across different operating systems and browsers. Additionally, the browser doesn't allow control over print settings through JavaScript code, so the necessary settings must be configured directly in the print dialog window.

Report print settings
When selecting report printing from the viewer's toolbar, a menu with print options is displayed. The component can forcibly set the required print mode. To do this, simply set the `printDestination` property to one of the values listed below from the `StiPrintDestination` enumeration:

Name	Description

`StiPrintDestination.DEFAULT`	When printing is selected, a menu with available print options will be displayed (this is the default property value).
`StiPrintDestination.PDF`	Print to PDF format.
`StiPrintDestination.DIRECT`	Print to HTML format directly to the printer, displaying the system print dialog.
`StiPrintDestination.WITH_PREVIEW`	Print to HTML format with a preview in a popup window. Print to HTML format with a preview in a popup window.

For example, to set the print mode to PDF format only:

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.viewer.enums import StiPrintDestination

viewer = StiViewer()
viewer.options.toolbar.printDestination = StiPrintDestination.PDF
```

The viewer provides the option to completely disable report printing if it isn't needed. To do this, set the `showPrintButton` property to `False`.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.options.toolbar.showPrintButton = False
```

Report print event

If any actions need to be performed before printing a report, the `onPrintReport`

event is available. The event arguments will include the type of report printing and the report itself that is being sent to print.

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.events import StiReportEventArgs

def printReport(args: StiReportEventArgs):
    printAction = args.printAction
    report = args.report

viewer = StiViewer()
viewer.onPrintReport += printReport
viewer.onPrintReport += 'printReport'
```

viewer.html

```
<script>
    function printReport(args) {
        let printAction = args.printAction;
        let report = args.report;
    }
</script>
```

A detailed description of the available argument values can be found in the Viewer Events section.

Printing a report from code

It is possible to print a report directly from code without using the viewer functions. A detailed description of this functionality can be found in the report generator section Printing a Report from Code.

12.2.6 Report Export

The viewer allows exporting the displayed report or dashboard into various formats. No additional viewer settings are required for export functionality. The table below lists all available export formats for reports and dashboards:

Name	Reports	Dashboards
Document (Snapshot)	+	+

Adobe PDF	+	+
Microsoft XPS	+	-
Microsoft PowerPoint (.pptx)	+	-
HTML	+	+
HTML5	+	-
Text	+	-
Microsoft Word (.docx)	+	-
Microsoft Excel (.xlsx)	+	+
OpenDocument Writer (.odt)	+	-
OpenDocument Calc (.ods)	+	-
Comma Separated Value (.csv)	+	+
Scalable Vector Graphics (.svg)	+	+

Begin export event

If you need to perform any actions before exporting a report, the `onBeginExportReport` event is provided. This event is triggered after the export settings dialog is displayed. The event arguments will include the report export type, the report itself, and all the selected export settings. Modifications to the report, its parameters, export settings, and file name are allowed.

app.py

```
from stimulsoft_reports.report.enums import StiExportFormat
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.events import StiExportEventArgs

def beginExportReport(args: StiExportEventArgs):
    if args.format == StiExportFormat.PDF:
        args.settings['imageQuality'] = 0.90
        args.settings['imageResolution'] = 200
```

```
viewer = StiViewer()
viewer.onBeginExportReport += beginExportReport
viewer.onBeginExportReport += 'beginExportReport'
```

viewer.html

```
<script>
    function beginExportReport(args) {
        if (args.format == Stimulsoft.Report.StiExportFormat.Pdf) {
            args.settings.imageQuality = 0.90;
            args.settings.imageResolution = 200;
        }
    }
</script>
```

A detailed description of the available argument values can be found in the Viewer Events section.

End export event

If you need to perform any actions after exporting a report but before saving it, the `onEndExportReport` event is provided. The event arguments will include the report export type, as well as the name and byte data of the exported file. Modifications to the file name and byte data of the exported file are allowed.

app.py

```
from stimulsoft_reports.report.enums import StiExportFormat
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.events import StiExportEventArgs

def endExportReport(args: StiExportEventArgs):
    if args.format == StiExportFormat.HTML:
        htmlText = args.data

viewer = StiViewer()
viewer.onEndExportReport += endExportReport
viewer.onEndExportReport += 'endExportReport'
```

viewer.html

```
<script>
    function endExportReport(args) {
        if (args.format == Stimulsoft.Report.StiExportFormat.Html) {
            htmlText = args.data
        }
    }
</script>
```

A detailed description of the available argument values can be found in the Viewer Events section.

Export settings

Sometimes it's necessary to disable unused report export formats, leaving only the required ones. This helps streamline the interface and make the viewer easier to use. To disable unused export formats, simply set the corresponding viewer properties to `False`, for example:

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.options.exports.showExportToDocument = False
viewer.options.exports.showExportToWord2007 = False
viewer.options.exports.showExportToCsv = False
```

If necessary, you can also completely hide the export dialog windows, and exporting will always be done with default settings. In this case, you can manage the settings in the export event. To disable the dialog windows, simply set the `showExportDialog` property to `False`:

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.options.exports.showExportDialog = False
```

A full list of available options can be found in the Viewer Settings section.

Exporting a report from code

It is also possible to export a report using code. This can be done with the `exportDocument()` method of the report object. A detailed description can be found in the Exporting a Report from Code section.

12.2.7 Display Modes

The viewer allows configuring various interface and report page display modes, as well as managing the display on mobile devices.

Scrollbars

The viewer offers two report display modes: with scrollbars and without them. By default, the mode without scrollbars is enabled. To enable the mode with scrollbars, simply set the `scrollbarsMode` property to `True`.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.appearance.scrollbarsMode = True
```

In the first mode (without scrollbars), the viewer displays the page or report in its entirety, automatically stretching the viewing area. If width and height dimensions are specified, the viewer will crop any content that extends beyond the page boundaries. In the second mode, unlike the first, no cropping occurs when the page extends beyond the viewer's dimensions. Instead, scrollbars appear, allowing the user to scroll through the page or report.

Information

Using the report viewing mode with scrollbars, the height of the viewer must be set, otherwise, a default height of 650 pixels will be applied.

Fullscreen mode

The viewer includes a fullscreen mode for displaying reports and dashboards. By default, the standard viewing mode is enabled, with the viewer sized according to the settings. To enable fullscreen mode, simply set the `fullScreenMode` property to `True`.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.appearance.fullScreenMode = True
```

To enable or disable fullscreen mode, you can also use the corresponding button on the viewer's control panel.

Report page display

The viewer offers three report display modes:

- Single-page display;
- Continuous display of the entire report as a scrollable ribbon;
- Table display of report pages.

The property `viewMode` is used to control these modes, and it accepts one of the following values:

Name	Description
`StiWebViewMode.SINGLE_PAGE`	Displays one page selected from the toolbar.
`StiWebViewMode.CONTINUOUS`	Displays all report pages as a continuous ribbon.
`StiWebViewMode.MULTIPLE_PAGES`	Displays all report pages in a table format.

For example, to set the mode to display all pages as a continuous ribbon:

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.viewer.enums import StiWebViewMode

viewer = StiViewer()
viewer.toolbar.viewMode = StiWebViewMode.CONTINUOUS
```

Mobile mode

The viewer supports both desktop and touchscreen mobile devices. To control the interface modes, use the `interfaceType` property, which accepts one of the following values:

Name	Description
`StiInterfaceType.AUTO`	The viewer's interface type will be automatically selected based on the device being used (default value).
`StiInterfaceType.MOUSE`	Forces the use of the standard interface for controlling the viewer with a mouse.
`StiInterfaceType.TOUCH`	Forces the use of the `Touch` interface for controlling the viewer with a touchscreen. In this mode, the viewer's interface elements are larger for easier interaction.
`StiInterfaceType.MOBILE`	Forces the use of the `Mobile` interface for controlling the viewer via a smartphone screen. In this mode, the viewer interface is simplified and adapted for mobile device control.

For example, to completely disable the mobile display mode:

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.viewer.enums import StiInterfaceType

viewer = StiViewer()
viewer.appearance.interfaceType = StiInterfaceType.MOUSE
```

12.2.8 Working with Report Variables

The viewer includes support for a special parameters panel for working with report variables. To add a parameter to the panel, a variable that requires user input must be defined in the report. When viewing the report in the viewer, this variable will automatically be added to the parameters panel. All types of report variables are supported (standard variables, date and time, range, lists, etc.).

Values of variables on the parameters panel

A special `onInteraction` event is provided for performing actions before applying the parameters, which will be triggered by any interactive actions in the viewer. The event arguments will include the action type, the report object, and a collection of variables and their values on the parameters panel. In this case, the action type will be `Variables`. A detailed description of the available argument values can be found in the Viewer Events section.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.onInteraction += 'interaction'
```

viewer.html

```
<script>
    function interaction(args) {
        if (args.action == "Variables") {
            let variables = args.variables;
        }
    }
</script>
```

The variable collection is an object that contains all the variables from the parameters panel. It's allowed to change the values of the variables, but the type of

the new value must match the type of the variable being modified.

viewer.html

```
<script>
    let variables = {
        VariableString: "Text value",
        VariableInt: 20
    }
</script>
```

Direct access to the variable values on the parameters panel from the Python server side isn't provided, the event works only on the JavaScript client-side.

Parameters panel settings

If working with variables in the viewer is not required, this feature can be completely disabled. To do so, set the `showParametersButton` property to `False`.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.toolbar.showParametersButton = False
```

Inforrmation

With this viewer configuration, the parameters panel will not be displayed, even if parameters are present in the report being viewed.

Variable values when building a report

If it is necessary to control all report variables, a special `onPrepareVariables` event is available, which will be triggered before the report is generated.

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.events import StiReportEventArgs

def prepareVariables(args: StiVariablesEventArgs):
    variables = args.variables

viewer = StiViewer()
viewer.onPrepareVariables += onPrepareVariables
viewer.onPrepareVariables += 'onPrepareVariables'
```

viewer.html

```
<script>
    function prepareVariables(args) {
        let variables = args.variables;
    }
</script>
```

A detailed description of this event can be found in the Working with Report Variables section of the report generator documentation.

12.2.9 Working with Bookmarks

The viewer supports report bookmarks. When such a report is displayed, a panel with bookmarks will appear to the left of the page. When you select a report bookmark, the viewer will automatically navigate to the desired page, and the report element associated with the bookmark will be highlighted.

Bookmark settings

By default, the width of the bookmarks bar is 180 pixels; the viewer allows you to change this value. The `bookmarksTreeWidth` property is used for this, the value of which is specified in pixels.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.appearance.bookmarksTreeWidth = 300
```

If you do not need to work with report tabs, you can completely disable this functionality. The `showBookmarksButton` property is used for this. It should be set to `False`.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.toolbar.showBookmarksButton = False
```

Information

In this case, report bookmarks will not be shown, even if they are present in the displayed report. This property does not affect printing and exporting a report with bookmarks.

When printing a report with bookmarks, the bookmarks tree will be hidden. If, in addition to the report itself, you also need to print bookmarks, then you need to set the `bookmarksPrint` property to `True`.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.appearance.bookmarksPrint = True
```

12.2.10 Dynamic collapsing, sorting, and detailing

In addition to variables whose values can be set on the parameter panel, the viewer supports other types of interactivity that enhance convenience and functionality when using the report generator. These include sorting, collapsing, and detailing.

Sorting

Dynamic sorting allows you to change the sorting direction in a built report. To do this, click on the component that has been set up for dynamic sorting. Dynamic sorting is performed in the following directions: **Ascending** and **Descending**. Each

click on the component toggles the sorting direction to the opposite.

Multi-level sorting is also supported in the report. To apply it, hold down the **Ctrl** key and click on the sortable components of the report in sequence. To reset the sorting, click on any sortable component without holding the **Ctrl** key.

Collapsing

A report with dynamic collapsing is an interactive report where specific blocks can collapse or expand their content when clicking on the block's header. Report elements that can be collapsed or expanded are marked with a special icon showing **[-]** or **[+]**.

Detailing

When drilling down into data, a drill-down panel with tabs for detailed reports will be displayed below the main viewer panel. The currently displayed report will be highlighted. There is an option to close the detailed pages that are not needed at the moment.

Dashboard Sorting

When viewing a dashboard, many elements allow sorting by the element's data fields in **Ascending** and **Descending** order.

Dashboard Filtering

Filtering on the dashboard can be done both using special filtering elements and with other elements. The filter will be applied to all related elements on the dashboard.

Dashboard Drill-Down

In dashboards, as in reports, it is possible to open a drill-down dashboard or report. Additionally, for some dashboard elements, data drill-down at the element level is available.

Viewer interactivity event

No additional viewer settings are required for any interactive actions in the report or dashboard. However, it’s possible to perform certain operations before an interactive action occurs. To do this, you can define the `onInteraction` event, which will be triggered at the moment an interactive action is performed, just before it is applied to the report. The event arguments will include the action type, the report object, and all parameters used for the current interactive action. Detailed descriptions of

the available argument values can be found in the Viewer Events section.

Each type of viewer interactivity has a specific action type listed in the following table:

Name	Description
`InitVars`	The action occurs during the initialization of report variables requested from the user.
`Variables`	The action occurs when using variables on the parameters panel. Detailed information can be found in the section Working with Report Variables.
`Sorting`	The action occurs when using column sorting.
`DrillDown`	The action occurs when using column detailing.
`Collapsing`	The action occurs when collapsing report blocks.
`DashboardFiltering`	The action occurs when using filters within a dashboard element.
`DashboardSorting`	The action occurs when using sorting within a dashboard element.
`DashboardResetAllFilters`	The action occurs when resetting sorting and filters within a dashboard element to the values set in the template.
`DashboardElementDrillDown`	The action occurs when using detailing in a dashboard element.
`DashboardElementDrillUp`	The action occurs when using detailing in a dashboard element.

The action type is passed in the event arguments:

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.onInteraction += 'interaction'
```

viewer.html

```
<script>
    function interaction(args) {
        switch (args.action) {
            case "Sorting":
                break;

            case "DrillDown":
                break;

            case "Collapsing":
                break;
        }
    }
</script>
```

The arguments include corresponding parameter collections: `sortingParameters`, `collapsingParameters`, `drillDownParameters`, and `filteringParameters`, which contain data in a specific format required for the current interactive action. If needed, the values of the parameter collections can be adjusted while maintaining the structure and order of the passed values. Below is an example of parameter values passed for some interactive actions:

viewer.html

```
<script>
    let sortingParameters = {
        ComponentName: "Text10;false",
        DataBand: "DataBand1;DESC;CompanyName"
    };

    let collapsingParameters = {
        CollapsingStates: {
        GroupHeaderBand1: {
            keys: [1],
            values: [false]
```

```
        },
        ComponentName: "GroupHeaderBand1"
    };

    let drillDownParameters = [
        {
            ComponentIndex: "1"
            DrillDownMode: null
            ElementIndex: "6"
            PageGuid: "b916d048d3f446dc97c356d4ff47f48f"
            PageIndex: "0"
            ReportFile: null
        }
    ];
</script>
```

Direct access to the parameters of viewer interactive actions on the Python server side isn't provided, the event only works on the JavaScript client-side.

12.2.11 Editing Rendered Reports

You may edit elements of the generated report, such as text fields and check boxes in the viewer. To enable editing, you should mark the required components as editable in the report template. No additional viewer settings are required. After displaying the report in the viewer, to begin editing, you need to click on the corresponding button on the viewer panel. Once editing is completed, you must click the specified button again, and all changes made will be applied to the report.

Information

The edited values will be applied when printing or exporting the report, while the original report will remain untouched. After reloading the viewer, all values will return to their original state.

12.2.12 Sending a report by Email

The viewer provides the ability to send a report by Email. To enable this feature, simply set the viewer's `showSendEmailButton` property to `True` and add an `onEmailReport` event handler on the Python server side, where all Email sending parameters should be specified. If necessary, a JavaScript event call can be added, in which arguments allow you to retrieve the necessary data for sending the email, get the report export type, obtain the report itself, and access the report export settings, which can be modified if needed. A detailed description of the available argument

values can be found in the Viewer's Events section.

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.events import StiEmailEventArgs

def emailReport(args: StiEmailEventArgs):
    settings = args.settions

viewer = StiViewer()
viewer.onEmailReport += emailReport
viewer.onEmailReport += 'emailReport'
```

viewer.html

```
<script>
    function emailReport(args) {
        let settings = args.settings;
    }
</script>
```

When sending a report by Email, a menu for selecting the attachment format is displayed, which corresponds to the report export format selection menu. After choosing the format, a dialog will appear for entering the sending parameters, such as the recipient's Email, subject, and message body. After confirming the sending, the `onEmailReport` event will be triggered, where you can check and correct the data entered in this form, for example:

viewer.html

```
<script>
    function emailReport(args) {
        args.settings.subject = "Invoice: " + args.settings.subject;
    }
</script>
```

Pure JavaScript doesn't have built-in functions for working with Email, functions for this purpose are implemented on the Python server-side. To send an Email, the Python server needs to be configured with data such as the login and password of

the account used for sending, as well as server settings like its address, port, and other parameters. This is done through the settings property, which is part of the event arguments and represents an object of the `StiEmailSettings` class that contains all the necessary properties. Additionally, the arguments allow you to access and modify the email's sending data (subject, body, and report file name). A detailed description of the available settings can be found in the Viewer Events section.

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.events import StiEmailEventArgs

def emailReport(args: StiEmailEventArgs):
    args.settings.fromAddr = 'mail.sender@stimulsoft.com'
    args.settings.host = 'smtp.stimulsoft.com'
    args.settings.port = 456
    args.settings.login = '********'
    args.settings.password = '********'
    return 'The Email has been sent successfully.'

viewer = StiViewer()
viewer.onEmailReport += emailReport
```

Example of checking and modifying the email subject before sending:

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.events import StiEmailEventArgs

def emailReport(args: StiEmailEventArgs):
    if len(args.settings.subject) == 0:
        args.settings.subject = args.formatName + ' report ' +
args.settings.attachmentName

viewer = StiViewer()
viewer.onEmailReport += emailReport
```

Example of adding addresses of additional recipients for the email with the report:

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.events import StiEmailEventArgs

def emailReport(args: StiEmailEventArgs):
    args.settings.cc.append('extra_recipient_one@stimulsoft.com')
    args.settings.bcc.append('hidden_recipient_one@stimulsoft.com')
    args.settings.bcc.append('hidden_recipient_two@stimulsoft.com John
Smith')

viewer = StiViewer()
viewer.onEmailReport += emailReport
```

Email sending settings

The viewer allows you to set default values for the email sending form. The properties `defaultEmailAddress`, `defaultEmailSubject`, and `defaultEmailMessage` are intended for this. By default, these properties are empty.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.options.email.defaultEmailAddress =
'recipient_address@stimulsoft.com'
viewer.options.email.defaultEmailSubject = 'New Invoice'
viewer.options.email.defaultEmailMessage = 'Please check the new invoice
in the attachment'
```

12.2.13 Calling Designer from Viewer

The viewer has the ability to call the report designer (use the **Design** button on the viewer toolbar). By default, this button is disabled. Set the `showDesignButton` property to `True`, and also define the `onDesignReport` event.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.onDesignReport += 'designReport'
```

viewer.html

```
<script>
    function designReport(args) {
        window.location.href = "/designer?report=" + args.fileName;
    }
</script>
```

A detailed description of the available argument values you may read in the Viewer Events section.

Information

The viewer itself does not initiate the designer; it solely triggers the specified event and passes the file name and the report being viewed as arguments. Within the event, you can redirect to the web page hosting the report designer and pass the required parameters.

12.2.14 Appearance

The viewer has the option to change the theme of the visual control elements. To do this, simply set the `theme` property in the component options:

app.py

```
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.viewer.enums import StiViewerTheme

viewer = StiViewer()
viewer.options.appearance.theme = StiViewerTheme.OFFICE_2022_DARKGRAY_BLUE
```

Currently, there are 8 themes available with various color accents, resulting in over 60 design options. This allows you to customize the appearance of the viewer to match almost any web project design.

Additional Settings

By default, the viewer has only a top toolbar with all report control elements. If needed, the toolbar can be split into a top and a bottom panel. The top panel will contain the print and export menu, as well as buttons for working with parameters

and bookmarks. The bottom toolbar will include page navigation controls and zoom management menus. The `displayMode` property is used to enable this mode, and it can have the following values:

Name	Description
`StiToolbarDisplayMode.SIMPLE`	Simple display mode, all control elements are on one control panel (default value).
`StiToolbarDisplayMode.SEPARATED`	Separated display mode, the control panel is divided into a top section - for interacting with the report, and a bottom section - for interacting with the report's pages and zoom controls.

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()
viewer.options.toolbar.displayMode = StiToolbarDisplayMode.SIMPLE
viewer.options.appearance.scrollbarsMode = True
```

Additionally, there is the option to configure the appearance of the main elements of the viewer. For example, you can change the font and color of the toolbar labels, set the background of the viewer, specify the page border color, and more. Below is a list of available properties for customizing the viewer's appearance and their default values:

app.py

```
from stimulsoft_reports.viewer import StiViewer

viewer = StiViewer()

viewer.options.appearance.backgroundColor = 'white'
viewer.options.appearance.pageBorderColor = 'gray'
viewer.options.appearance.showPageShadow = False

viewer.options.toolbar.backgroundColor = 'transparent';
viewer.options.toolbar.borderColor = 'transparent';
```

```
viewer.options.toolbar.fontColor = 'transparent';
viewer.options.toolbar.fontFamily = 'Arial';
```

For color values, you can specify either one of the standard HTML color constants or a color code in RGB format, such as '`#ff2020`'.

12.2.15 Viewer Events

The report viewer supports events that provide the ability to perform necessary operations before specific actions—both on the JavaScript client side and the Python server side. A detailed description of how events work can be found in the Report Engine Events section.

Some event arguments take values from enumerations that are located in specific namespaces. All the enumerations used in designer events are listed in the code block below:

app.py

```
from stimulsoft_reports.enums import StiEventType
from stimulsoft_reports.report.enums import StiExportFormat
from stimulsoft_reports.viewer.enums import StiPrintAction,
StiExportAction
```

viewer.html

```
<script>
    StiExportAction = Stimulsoft.Viewer.StiExportAction;
    StiExportFormat = Stimulsoft.Report.StiExportFormat;
</script>
```

The viewer supports the following events:

- onPrepareVariables
- onBeginProcessData
- onEndProcessData
- onOpenReport

- onOpenedReport
- onPrintReport
- onBeginExportReport
- onEndExportReport
- onInteraction
- onEmailReport
- onDesignReport

onPrepareVariables
[v] JavaScript [v] Python

The event is triggered before the report is built, after the report variables are prepared. A list of event arguments can be found in the Report Engine Events section. Detailed descriptions and usage examples can be found in the Working with Report Variables section.

onBeginProcessData
[v] JavaScript [v] Python

The event is triggered before requesting the data needed to build the report. A list of event arguments can be found in the Report Engine Events section. Detailed descriptions and usage examples can be found in the Connecting Data Files and Connecting SQL Data Adapters sections.

onEndProcessData
[v] JavaScript [v] Python

The event is triggered after the data is loaded, before the report is built. A list of event arguments can be found in the "Report Engine Events" section. Detailed descriptions and usage examples can be found in the Connecting Data Files and Connecting SQL Data Adapters sections.

onOpenReport

[v] JavaScript [x] Python

The event is triggered before opening the report after the toolbar button is clicked.

List of properties passed in the event arguments on the JavaScript client-side:

Name	Description
`event`	The identifier of the current event has the value `"OpenReport"`.
`sender`	The identifier of the component that initiated this event can take the following value: • `"Viewer"` • `"Designer"`
`report`	The current report object will be passed as `null` in the arguments of this event.
`preventDefault`	This flag allows you to stop further event handling by the viewer. By default, it is set to `false`.

onOpenedReport
[v] JavaScript [v] Python

The event is triggered after opening a report from the designer menu but before it is loaded into the viewer.

List of properties passed in the event arguments on the JavaScript client-side:

Name	Description
`event`	The identifier of the current event,

	with the value `"OpenedReport"`.
`sender`	The identifier of the component that initiated the event, which can have the following value: • `"Viewer"` • `"Designer"`
`report`	The current report object.
`preventDefault`	This flag allows you to stop further event processing by the designer. By default, it is set to `false`.

List of properties passed in the event arguments on the Python server side. The arguments are of type `StiReportEventArgs`:

Name	Description
`event`	The identifier of the current event, for this event the value is `StiEventType.OPENED_REPORT`
`sender`	The identifier of the component that initiated the event, which can take the following value: • `StiViewer` • `StiDesigner`
`report`	The current report object in the form of a JSON string or object.

onPrintReport

[v] JavaScript [v] Python

The event is triggered before printing the report. The table below shows the list of event handler arguments on the client-side in JavaScript:

Name	Description
`event`	The identifier of the current event, with the value `"PrintReport"`.
`sender`	The identifier of the component that triggered this event, possible values: • `"Viewer"` • `"Designer"`
`report`	The current report object.
`printAction`	The type of report printing. Possible values: • `PrintPdf` - prints in PDF format; • `PrintWithoutPreview` - prints in HTML format directly to the printer, displays the system print dialog; • `PrintWithPreview` - prints in HTML format with a preview in a popup.
`preventDefault`	This flag allows you to stop further event handling by the viewer. By default, it is set to `false`.

The list of properties passed as event arguments on the server side in Python has the type `StiReportEventArgs`:

Name	Description

`event`	The identifier of the current event, has the `StiEventType.PRINT_REPORT`.
`sender`	The identifier of the component that triggered this event, possible values: • `StiViewer` • `StiDesigner`
`report`	The current report object.
`printAction`	The type of report printing. Possible values: • `StiPrintAction.PRINT_PDF` - prints in PDF format; • `StiPrintAction.PRINT_WITHOUT_PREVIEW` - prints in HTML format directly to the printer, displays the system print dialog; • `StiPrintAction.PRINT_WITH_PREVIEW` - prints in HTML format with a preview in a popup.

For detailed descriptions and usage examples, refer to the Report Printing section.

onBeginExportReport
[v] JavaScript [v] Python

The event is triggered before exporting the report, after the export settings dialog. The table below shows the list of event handler arguments on the client-side in JavaScript:

Name	Description
`event`	The identifier of the current event, has the value `"BeginExportReport"`.
`sender`	The identifier of the component that triggered this event, possible values: • `"Viewer"` • `"Designer"`
`report`	The current report object.
`action`	The action that triggered the export event, possible values: • `StiExportAction.ExportReport` • `StiExportAction.SendEmail`
`format`	The selected report export format, possible values: • `StiExportFormat.Document` • `StiExportFormat.Pdf` • `StiExportFormat.Xps` • `StiExportFormat.Ppt2007` • `StiExportFormat.Html` • `StiExportFormat.Html5` • `StiExportFormat.Text` • `StiExportFormat.Word2007` • `StiExportFormat.Excel2007` • `StiExportFormat.Odt` • `StiExportFormat.Ods` • `StiExportFormat.Csv` • `StiExportFormat.ImageSvg`

`formatName`	The name of the selected export format, corresponds to the format enumeration constants.
`settings`	The settings of the selected export format. The available properties list will depend on the chosen export type.
`fileName`	The report file name to save after export completion.
`openAfterExport`	A flag indicating whether the report will be exported in a new browser tab (`` `true` ``), or if the file save dialog will be prompted after the export (`` `false` ``).
`preventDefault`	This flag allows you to stop further event handling by the viewer. By default, it is set to `false`.

The list of properties passed as event arguments on the server side in Python has the type `StiExportEventArgs`:

Name	Description
`event`	The identifier of the current event, has the value `StiEventType.BEGIN_EXPORT_REPORT`.
`sender`	The identifier of the component that triggered this event, possible values: • `StiViewer` • `StiDesigner`
`report`	The current report object.
`action`	The action that triggered the export

	event, possible values: • StiExportAction.EXPORT_REPORT • StiExportAction.SEND_EMAIL
format	The selected report export format, possible values: • StiExportFormat.DOCUMENT • StiExportFormat.PDF • StiExportFormat.XPS • StiExportFormat.PPT2007 • StiExportFormat.HTML • StiExportFormat.HTML5 • StiExportFormat.TEXT • StiExportFormat.WORD2007 • StiExportFormat.EXCEL2007 • StiExportFormat.ODT • StiExportFormat.ODS • StiExportFormat.CSV • StiExportFormat.IMAGE_SVG
formatName	The name of the selected export format, corresponds to the format enumeration constants.
settings	The settings of the selected export format. The available properties list will depend on the chosen export type.
fileName	The report file name to save after export completion.

`openAfterExport`	A flag indicating whether the report will be exported in a new browser tab (`true`), or if the file save dialog will be prompted after the export (`false`).

onEndExportReport
[v] JavaScript [v] Python

The event is triggered after the report has been exported, but before it is saved as a file. The table below shows the list of event handler arguments on the client-side in JavaScript:

Name	Description
`event`	The identifier of the current event, has the value `"EndExportReport"`.
`sender`	The identifier of the component that triggered this event, possible values: • `"Viewer"` • `"Designer"`
`report`	The current report object.
`format`	The selected report export format, possible values: • `StiExportFormat.Document` • `StiExportFormat.Pdf` • `StiExportFormat.Xps` • `StiExportFormat.Ppt2007` • `StiExportFormat.Html` • `StiExportFormat.Html5` • `StiExportFormat.Text`

	• `StiExportFormat.Word2007` • `StiExportFormat.Excel2007` • `StiExportFormat.Odt` • `StiExportFormat.Ods` • `StiExportFormat.Csv` • `StiExportFormat.ImageSvg`
`formatName`	The name of the selected export format, corresponds to the format enumeration constants.
`data`	The byte data of the exported report, prepared for saving to a file.
`fileName`	The report file name to save after export completion.
`openAfterExport`	A flag indicating whether the report will be exported in a new browser tab (`` `true` ``), or if the file save dialog will be prompted after the export (`` `false` ``).
`preventDefault`	This flag allows you to stop further event handling by the viewer. By default, it is set to `false`.

The list of properties passed as event arguments on the server side in Python has the type `StiExportEventArgs`:

Name	Description
`event`	The identifier of the current event,

	has the value `StiEventType.END_EXPORT_REPORT`.
`sender`	The identifier of the component that triggered this event, possible values: • `StiViewer` • `StiDesigner`
`format`	The selected report export format, possible values: • `StiExportFormat.DOCUMENT` • `StiExportFormat.PDF` • `StiExportFormat.XPS` • `StiExportFormat.PPT2007` • `StiExportFormat.HTML` • `StiExportFormat.HTML5` • `StiExportFormat.TEXT` • `StiExportFormat.WORD2007` • `StiExportFormat.EXCEL2007` • `StiExportFormat.ODT` • `StiExportFormat.ODS` • `StiExportFormat.CSV` • `StiExportFormat.IMAGE_SVG`
`formatName`	The name of the selected export format, corresponds to the format enumeration constants.
`data`	The byte data of the exported report, prepared for saving to a file.

`fileName`	The report file name to save after export completion.
`fileExtension`	The file extension for the report to save after export completion, corresponds to the selected format type.
`mimeType`	The MIME type for the selected export format.
`openAfterExport`	A flag indicating whether the report will be exported in a new browser tab (`true`), or if the file save dialog will be prompted after the export (`false`).

onInteraction

[v] JavaScript [x] Python

The event is triggered at the moment of an interactive action in the viewer (dynamic sorting, collapsing, drill-down, applying parameters) before the report generator processes the values. The table below shows the list of event handler arguments on the client-side in JavaScript:

Name	Description
`event`	The identifier of the current event, has the value `"Interaction"`.
`sender`	The identifier of the component that triggered this event, possible values: • `"Viewer"` • `"Designer"`
`report`	The current report object.
`action`	The identifier of the current interactive action, possible values: • `"InitVars"` - triggered when initializing report variables requested from the user;

	• "Variables" - triggered when applying the values of variables requested from the user; • "Sorting" - triggered when sorting columns; • "DrillDown" - triggered during report drill-down; • "Collapsing" - triggered when collapsing report sections; • "DashboardFiltering" - triggered when using filters within a dashboard element; • "DashboardSorting" - triggered when sorting within a dashboard element; • "DashboardResetAllFilters" - triggered when resetting all sorting and filters in a dashboard element to the values defined in the template; • "DashboardElementDrillDown" - triggered during drill-down of a dashboard element; • "DashboardElementDrillUp" - triggered during drill-up of a dashboard element;
variables	A collection of report variables and their values, set on the parameters panel..
sortingParameters	A collection of parameters required for dynamic report sorting.
collapsingParameters	A collection of parameters required for dynamic collapsing of report elements.
drillDownParameters	A collection of parameters required for report drill-down.
filteringParameters	A collection of parameters required

	for sorting, filtering, and drill-down in dashboard elements.
`preventDefault`	This flag allows stopping further event handling. By default, it is set to `false`.

onEmailReport
[v] JavaScript [v] Python

The event is triggered after the report is exported and before it is sent via Email. The table below lists the event handler arguments on the client side in JavaScript:

Name	Description
`event`	The identifier of the current event, has the value `"EmailReport"`.
`sender`	The identifier of the component that triggered this event, possible values: • `"Viewer"` • `"Designer"`
`report`	The current report object.
`format`	The selected report export format, possible values: • `StiExportFormat.Document` • `StiExportFormat.Pdf` • `StiExportFormat.Xps` • `StiExportFormat.Ppt2007` • `StiExportFormat.Html` • `StiExportFormat.Html5` • `StiExportFormat.Text`

	• StiExportFormat.Word2007 • StiExportFormat.Excel2007 • StiExportFormat.Odt • StiExportFormat.Ods • StiExportFormat.Csv • StiExportFormat.ImageSvg
formatName	The name of the selected report export format, corresponding to the format enum constants.
data	The byte data of the exported report, prepared for sending via Email.
fileName	The name of the report file for sending by Email.
settings	An object containing the parameters filled in the viewer's Email sending dialog.

The table below shows the list of Email sending parameters on the client-side in JavaScript.

Name	Description
settings.email	The Email address to which the exported report will be sent.
settings.subject	The subject of the email.
settings.message	The text of the email.

List of properties passed in the event arguments on the Python server-side. The arguments are of type `StiExportEventArgs`:

Name	Description
`event`	The identifier of the current event, for this event it is `StiEventType.EMAIL_REPORT`.
`sender`	The identifier of the component that triggered this event, possible values: • `StiViewer` • `StiDesigner`
`report`	The current report object in JSON format.
`format`	The selected report export format, possible values: • `StiExportFormat.DOCUMENT` • `StiExportFormat.PDF` • `StiExportFormat.XPS` • `StiExportFormat.PPT2007` • `StiExportFormat.HTML` • `StiExportFormat.HTML5` • `StiExportFormat.TEXT` • `StiExportFormat.WORD2007` • `StiExportFormat.EXCEL2007` • `StiExportFormat.ODT` • `StiExportFormat.ODS` • `StiExportFormat.CSV`

	• `StiExportFormat.IMAGE_SVG`
`formatName`	The name of the selected report export format, matching the enum constants.
`data`	The byte data of the exported report, prepared for sending via Email.
`fileName`	The name of the report file for sending by Email.
`settings`	An object containing the Email sending parameters on the server-side.

List of Email sending parameters on the Python server-side. All settings are in the `StiEmailSettings` class:

Name	Description
`fromAddr`	The email address of the sender.
`name`	The full name of the sender.
`toAddr`	The email address to which the exported report will be sent (from the viewer's dialog).
`subject`	The email subject (from the viewer's dialog).
`message`	The email text (from the viewer's dialog).
`attachmentName`	The name of the report file in the attachment (default: report file name).
`charset`	The character encoding used for the email body (default: "UTF-8").
`host`	The SMTP server address (required).

`port`	The SMTP server port (default: 465).
`login`	The login for the email server (required).
`password`	The password for the email server (required).
`secure`	The type of encryption for the email connection (`ssl` or tls, default: `ssl`).
`cc`	An array of CC (Carbon Copy) addresses for secondary recipients.
`bcc`	An array of BCC (Blind Carbon Copy) addresses for hidden recipients.

onDesignReport
[v] JavaScript [x] Python

The event is triggered when the Design button on the viewer panel is clicked. The table below lists the event handler arguments on the client side in JavaScript:

Name	Description
`event`	The identifier of the current event, with the value `"DesignReport"`.
`sender`	The identifier of the component that triggered the event, possible value: • `"Viewer"`
`report`	The current report object.
`fileName`	The name of the report file to be passed and loaded into the designer.

12.2.16 Viewer Settings

Viewer settings are configured by modifying the property values located in the main properties container named `options` of the component. All properties are divided into groups for ease of use. All enumerations used in the viewer settings are found

in the namespace `stimulsoft_reports.viewer.enums`.

The example of changing some viewer settings:

app.py

```
from flask import Flask, request
from stimulsoft_reports.viewer import StiViewer
from stimulsoft_reports.viewer.enums import StiHtmlExportMode,
StiToolbarDisplayMode, StiViewerTheme

@app.route('/viewer', methods = ['GET', 'POST'])
def viewer():
    viewer = StiViewer()
    viewer.options.localization = 'de.xml'
    viewer.options.appearance.theme =
StiViewerTheme.OFFICE_2022_DARKGRAY_BLUE
    viewer.options.appearance.fullScreenMode = True
    viewer.options.appearance.scrollbarsMode = True
    viewer.options.appearance.bookmarksTreeWidth = 200
    viewer.options.toolbar.displayMode = StiToolbarDisplayMode.SEPARATED
    viewer.options.exports.showExportToWord2007 = False
    viewer.options.exports.showExportToCsv = False

    if viewer.processRequest(request):
        return viewer.getFrameworkResponse()

    # Here is the code for working with the report

    return viewer.getFrameworkResponse()
```

Main (without group)

Name	Description
`width`	Sets the width of the component in "px" or "%". By default, the value is set to "100%".
`height`	Sets the height of the component in "px" or "%". By default, the value is set to "100%" for standard mode and "650px" for scroll mode.
`localization`	Sets the selected localization of the component. By default, the English

	localization is embedded in the component.

Appearance

Name	Description
`theme`	Sets the theme of the viewer. The list of available themes is in the StiViewerTheme enumeration. By default, the value is set to `StiViewerTheme.OFFICE_2022_WHITE_BLUE`.
`iconSet`	Allows setting the icon set: • `StiWebUIIconSet.AUTO` (default value) – automatically sets the icon set. For Office2022 themes, a Monoline icon set is used, and for Office2013 themes, a Regular icon set is used; • `StiWebUIIconSet.MONOLINE` – sets the Monoline icon set; • `StiWebUIIconSet.REGULAR` – sets the Regular icon set.
`backgroundColor`	Sets the background color of the viewer. By default, the value is `'white'`.
`pageBorderColor`	Sets the page border color in the report. By default, the value is `'gray'`.
`rightToLeft`	Sets the **Right to Left** mode for the viewer's controls. By default, the value is set to `False`.
`fullScreenMode`	Sets the full-screen mode for the viewer. If this property is set to `True`, the **width** and **height** properties are

	ignored. By default, the value is set to `False`.
`scrollbarsMode`	Sets the mode to display the report with scrollbars. By default, the value is set to `False`.
`openLinksWindow`	Sets the target window or frame for opening hyperlinks from the report. By default, the value is set to `'blank'` (new browser tab). It can take standard values `'blank'`, `'self'`, `'top'`, or a window or frame name.
`showTooltips`	Enables or disables the display of tooltips when hovering over the viewer's tools. By default, the value is set to `True`.
`showTooltipsHelp`	Allows displaying or hiding a link to the documentation in the tooltips when hovering over the viewer's tools. By default, the value is set to `True`.
`showDialogsHelp`	Allows displaying or hiding the help button in various menus. By default, the value is set to `True`.
`pageAlignment`	Sets the alignment of report pages in the viewer: • `StiContentAlignmen.DEFAULT` – the page alignment is determined by the template settings (default value); • `StiContentAlignment.LEFT` – pages are aligned to the left; • `StiContentAlignment.CENTER` – pages are centered; • `StiContentAlignment.RIGHT` – pages are aligned to the right.

showPageShadow	Enables or disables the display of page shadows in the report. By default, the value is set to False.
bookmarksPrint	Enables the printing of bookmarks in the report. By default, the value is set to False.
bookmarksTreeWidth	Sets the width of the bookmarks panel in pixels. By default, the value is set to 180.
parametersPanelPosition	Sets the position of the parameters panel in the viewer: • StiParametersPanelPosition.FROM_REPORT – the panel position is determined by the template settings (default value); • StiParametersPanelPosition.TOP - the panel is located at the top above the report page; • StiParametersPanelPosition.LEFT - the panel is located to the left of the report page.
parametersPanelMaxHeight	Sets the maximum height of the parameters panel in pixels. By default, the value is set to 300.
parametersPanelColumnsCount	Sets the number of columns on the parameters panel. By default, the value is set to 2.
parametersPanelDateFormat	Sets the date and time format for the variables displayed on the parameters panel. By default, no value is set.
parametersPanelSortDataItems	Enables or disables the sorting mode for the variable values. By default, the option is set to False, meaning the variable values are displayed in

	their original order.
interfaceType	Sets the viewer's interface type. The following values can be used: • `StiInterfaceType.AUTO` – the viewer's interface type will be automatically selected based on the device being used (default value); • `StiInterfaceType.MOUSE` – forces the use of the standard interface for controlling the viewer with a mouse; • `StiInterfaceType.TOUCH` – forces the use of the Touch interface for controlling the viewer with a touchscreen monitor; in this mode, the viewer's interface elements are larger for ease of use; • `StiInterfaceType.MOBILE` – forces the use of the Mobile interface for controlling the viewer with a smartphone screen; in this mode, the viewer's interface has a simplified appearance and is adapted for mobile device control.
allowMobileMode	Enables or disables the possibility of displaying the report or dashboard in mobile mode. If the option is set to `False`, the mobile view mode will not be used under any circumstances. If the option is set to `True`, the mobile view mode will be used when the viewer is launched on mobile devices. By default, the option is set to `True`.

chartRenderType	Sets the rendering type for charts in the report: • StiChartRenderType.ANIMATED_VECTOR – charts will be rendered in vector mode with animation (default value); • StiChartRenderType.VECTOR – charts will be rendered as vector images without animation.
reportDisplayMode	Sets the export mode for displaying report pages. It can take one of the following values: • StiHtmlExportMode.FROM_REPORT – the export mode of report elements is determined by the template settings, either Div or Table (default value); • StiHtmlExportMode.TABLE – report elements are exported using HTML tables; • StiHtmlExportMode.DIV – report elements are exported using DIV markup.
datePickerFirstDayOfWeek	Allows setting the first day of the week for the **Date Picker** tool. • StiFirstDayOfWeek.AUTO - Monday or Sunday will be set as the first day of the week based on the browser culture. • StiFirstDayOfWeek.MONDAY- Monday will be set as the first day of the week. • StiFirstDayOfWeek.SUNDAY - Sunday will be set as the first day of the week.
datePickerIncludeCurrentDayForRanges	Allows including or excluding the current day in the range of the **Date**

	Picker element. By default, the option is set to `False`, meaning the current day is not included in the range.
`allowTouchZoom`	Enables or disables the ability to zoom the viewer by touch. By default, the option is set to `True`.
`combineReportPages`	Allows combining the processed report template pages into one template or presenting each template page as a separate tab in the viewer. By default, the option is set to `False`, meaning each template page will be presented as a separate tab in the viewer.

Toolbar

Name	Description
`visible`	Allows the viewer's toolbar to be shown or hidden. By default, this is set to `True`.
`displayMode`	Sets the display mode of the viewer's toolbar. It can take one of the following values from the `displayMode` enumeration: • `StiToolbarDisplayMode.SIMPLE` – simple display mode, all controls are located on a single toolbar (default value); • `StiToolbarDisplayMode.SEPARATED` – split display mode, the toolbar is divided into upper and lower parts.
`backgroundColor`	Allows changing the toolbar's background color. By default, this is

	set to `'transparent'`.
`borderColor`	Allows changing the toolbar's border color. By default, this is set to `'transparent'`.
`fontColor`	Allows changing the font color for all elements on the toolbar and in all menus of this toolbar. By default, this is set to `'transparent'`.
`fontFamily`	Allows changing the font for all elements on the toolbar and in all menus of this toolbar. By default, this is set to `'Arial'`.
`alignment`	Sets the alignment of elements on the toolbar: • `StiContentAlignment.DEFAULT` – alignment depends on the **RightToLeft** option (default value); • `StiContentAlignment.LEFT` – all elements will be aligned to the left of the toolbar; • `StiContentAlignment.CENTER` – all elements will be aligned to the center of the toolbar; • `StiContentAlignment.RIGHT` – all elements will be aligned to the right of the toolbar.
`showButtonCaptions`	Enables or disables the display of button captions on the viewer's toolbar. By default, this is set to `True`.
`showPrintButton`	Allows showing or hiding the **Print** button on the toolbar. By default, this is set to `True`.
`showOpenButton`	Enables the display of the **Open** button on the viewer's toolbar when

	viewing reports or dashboards. By default, this is set to `True`.
`showSaveButton`	Enables the display of the **Save** button on the viewer's toolbar when viewing reports or dashboards. By default, this is set to `True`.
`showSendEmailButton`	Allows showing or hiding the **Send Email** button on the toolbar. By default, this is set to `False`. Additionally, the event handler for onEmailReport must be added.
`showFindButton`	Allows showing or hiding the **Find** button on the toolbar. By default, this is set to `True`.
`showBookmarksButton`	Allows showing or hiding the **Bookmarks** button on the toolbar. If this button is not displayed, the bookmarks panel in the report will not be displayed either. By default, this is set to `True`.
`showParametersButton`	Allows showing or hiding the **Parameters** button on the toolbar. If this button is not displayed, the parameters panel in the report will not be displayed either. By default, this is set to `True`.
`showResourcesButton`	Allows showing or hiding the **Resources** button on the toolbar. If this button is not displayed, the resources panel in the report will not be displayed either. By default, this is set to `True`.

showEditorButton	Allows showing or hiding the **Editor** button on the toolbar. If this button is not displayed, editable elements cannot be modified. By default, this is set to True.
showFullScreenButton	Enables the display of the **Full Screen** button on the viewer's toolbar when viewing reports or dashboards. By default, this is set to True.
showRefreshButton	Allows showing or hiding the **Refresh** button on the viewer's toolbar when viewing dashboards. By default, this is set to True.
showFirstPageButton	Allows showing or hiding the **First Page** button on the toolbar. By default, this is set to True.
showPreviousPageButton	Allows showing or hiding the **Previous Page** button on the toolbar. By default, this is set to True.
showCurrentPageControl	Allows showing or hiding the **Current Page** indicator on the toolbar. By default, this is set to True.
showNextPageButton	Allows showing or hiding the **Next Page** button on the toolbar. By default, this is set to True.
showLastPageButton	Allows showing or hiding the **Last Page** button on the toolbar. By default, this is set to True.
showZoomButton	Allows showing or hiding the **Zoom** selection button on the toolbar. By default, this is set to True.
showViewModeButton	Allows showing or hiding the page view **Mode** button on the toolbar. By

	default, this is set to `True`.
`showDesignButton`	Enables the display of the **Design** button on the toolbar when viewing reports or dashboards. By default, this is set to `False`.
`showAboutButton`	Allows showing or hiding the **About** button on the toolbar. By default, this is set to `True`.
`showPinToolbarButton`	Allows showing or hiding the **Pin** button in mobile report view mode. By default, this is set to `True`.
`printDestination`	Sets the report print mode. It can take one of the following values from the enumeration: • `StiPrintDestination.DEFAULT` – the menu with print mode options will be displayed (default); • `StiPrintDestination.PDF` – printing will be done in PDF format; • `StiPrintDestination.DIRECT` – printing will be done in HTML format directly to the printer, displaying the system print dialog; • `StiPrintDestination.WITH_PREVIEW` – printing will be done in HTML format through a pop-up report preview window.
`viewMode`	Sets the report page display mode: • `StiWebViewMode.SINGLE_PAGE` – a single page selected from the toolbar is displayed (default); • `StiWebViewMode.CONTINUOUS` – all report pages are

	displayed in a continuous scroll; • `StiWebViewMode.MULTIPLE_PAGES` – all report pages are displayed in a grid.
`zoom`	Allows setting the report page zoom level when loading the viewer. By default, this is set to 100%. The maximum value is 500%. Additionally, one of the following zoom values can be set: • `StiZoomMode.PAGE_WIDTH` – zoom pages to fit page width; • `StiZoomMode.PAGE_HEIGHT` – zoom pages to fit page height.
`menuAnimation`	Enables or disables animation for displaying and closing various menus in the viewer. By default, this is set to `True`.
`showMenuMode`	Sets the mode for revealing various menus in the viewer. It can take one of the following values: • `StiShowMenuMode.CLICK` – menus are revealed by clicking (default); • `StiShowMenuMode.HOVER` – menus are revealed by hovering the cursor.
`autoHide`	Sets the auto-hide mode for the toolbar when viewing reports in mobile mode. By default, this is set to `False`.

Exports

Name	Description

storeExportSettings	Allows saving **Export Settings** in cookies. By default, the value is set to True.
showExportDialog	Allows displaying or hiding the **Export Diallog**. If the menu is hidden, the export will be executed with default values. By default, the value is set to True.
showExportToDocument	Allows displaying or hiding the **Document File** option in the **Save** menu. By default, the value is set to True.
showExportToPdf	Enables the display of the **Adobe PDF File** export option in the report viewer, and the **Adobe PDF** option in the dashboard viewer. By default, the property is set to True.
showExportToXps	Enables the display of the **XPS File** export option. By default, the property is set to True.
showExportToPowerPoint	Enables the display of the **Microsoft PowerPoint 2007/2010 File** export option. By default, the property is set to True.
showExportToHtml	Allows displaying or hiding the **HTML File** option in the **Export Settings** menu. By default, the value is set to True.
showExportToHtml5	Allows displaying or hiding the **HTML5 File** option in the **Export Settings** menu. By default, the value is set to True.
showExportToText	Enables the display of the **Text File** export option. By default, the property is set to True.

showExportToWord2007	Allows displaying or hiding the **Microsoft Word 2007/2010 File** option in the **Save** menu. By default, the value is set to True.
showExportToOpenDocumentWriter	Enables the display of the **OpenDocument Writer File** export option. By default, the property is set to True.
showExportToExcel2007	Enables the display of the **Microsoft Excel 2007/2010 File** export option in the report viewer, and the **Microsoft Excel** option in the dashboard viewer. By default, the property is set to True.
showExportToOpenDocumentCalc	Enables the display of the **OpenDocument Calc File** export option. By default, the property is set to True.
showExportToCsv	Enables the display of the **CSV File** export option. By default, the property is set to True.
showExportToJson	Enables the display of the Image export option, with the ability to export the report to the **JSON File**. By default, the property is set to False.
showExportToImageSvg	Enables the display of the Image export option, with the ability to export the report to the **SVG file**. By default, the property is set to True.

Send by Email

Name	Description
showEmailDialog	Enables the display of the dialog box

	for sending the report by Email. If the dialog box is disabled, sending by Email will be done with default settings `onEmailReport`. By default, the value is set to `True`.
`showExportDialog`	Enables the display of the export settings dialog box when sending an Email. If the property is set to False, the export will be executed with default settings. By default, the value is set to `True`.
`defaultEmailAddress`	Sets the default recipient Email address, i.e., the address to which the email with the attached report will be sent. By default, the value isn't set.
`defaultEmailSubject`	Sets the default subject (title) of the email. By default, the value isn't set.
`defaultEmailMessage`	Sets the default message (text) of the email. By default, the value isn't set.

12.3 HTML5 Designer

The report designer is a Python component called `StiDesigner`, designed for creating reports and dashboards in a browser window on any computer with any operating system installed. The designer features a modern interface and various themes, offering a wide range of professionally prepared report and dashboard templates, report creation wizards, and numerous components to build reports and dashboards of virtually any complexity.

The designer features a convenient preview mode in the built-in viewer, which significantly speeds up report development. The built-in viewer functions the same as the standalone component, with more details covered in the Report Viewer section.

The designer's interface is built using HTML5, allowing it to be used on virtually any

modern platform. The component uses AJAX technology to perform all actions (loading and building reports, connecting to data, working with components and their settings, previewing reports, printing, exporting, etc.), eliminating the need to reload the entire page and increasing performance, making it suitable for use in single-page applications. The JavaScript technology used for report building allows for the use of almost any low-performance server-side technology.

Information

Since dashboards and reports use the same unified MRT template format, as well as the same methods for loading templates and working with data, the term "report" will be used throughout the documentation.

- Deployment
- Activation
- Creating and Editing Reports
- Saving a Report
- Localization
- Preview
- Designer Themes
- Designer Events
- Designer Settings

12.3.1 Deployment

Examples

The complete code sample can be found on GitHub.

To use the designer, simply install the `stimulsoft-reports` or `stimulsoft-dashboards` package using the package manager by executing the following command:

console

```
python -m pip install stimulsoft-reports
```

console

```
python -m pip install stimulsoft-dashboards
```

The latest available version of the product will be installed in the current workspace, after which you can use the classes and functions for working with reports and dashboards.

Information

The code examples in the documentation use the **Flask** framework, as it is one of the most popular and easy-to-understand frameworks. You can use any web framework; all classes and functions for working with components are universal.

To add the designer to a web project, the `StiDesigner` class is used. This class allows you to create a designer object, set the necessary settings, handle requests and return the result, and obtain the prepared JavaScript and HTML code of the component. Here is an example of displaying the designer on an HTML page:

app.py

```
from flask import Flask, render_template, url_for, request
from stimulsoft_reports.designer import StiDesigner

app = Flask(__name__)

@app.route('/designer', methods = ['GET', 'POST'])
def designer():
    designer = StiDesigner()
    designer.options.appearance.fullScreenMode = True

    if designer.processRequest(request):
        return designer.getFrameworkResponse()

    js = designer.javascript.getHtml()
    html = designer.getHtml()
```

```
    return render_template('designer.html', designerJavaScript = js,
designerHtml = html)
```

designer.html

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Editing a Report Template in the Designer</title>

    {{ designerJavaScript|safe }}
</head>

<body>
    {{ designerHtml|safe }}
</body>

</html>
```

In this example, an instance of the `StiDesigner` object is created, and the necessary settings are applied.

The special component function `processRequest(request)` processes the current request. If the function returns `True`, the request was successfully processed, and you need to return the result of its execution. More details about this can be found in the Event Handler section.

Next, you need to generate the prepared JavaScript and HTML code required for the designer to work. The `designer.javascript.getHtml()` function generates the HTML code to include the necessary scripts and resources, and the `designer.getHtml()` function generates the HTML code of the component itself. The generated code is passed as parameters to the HTML template designer.html and displayed in the specified places.

Information

Our products, Stimulsoft Reports.PYTHON and Stimulsoft Dashboards.PYTHON

don't have a native report generator core in Python; report building and exporting are performed on the client side using JavaScript code. When using Python code to work with components, you need to call the `getHtml()` function, which will return all the necessary JavaScript and HTML code for the report generator and components.

There is a simplified deployment of the designer without using an HTML page template. For example, the same example can be implemented using only Python code:

app.py

```
from flask import Flask, url_for, request
from stimulsoft_reports.designer import StiDesigner

app = Flask(__name__)

@app.route('/designer', methods = ['GET', 'POST'])
def designer():
    designer = StiDesigner()
    designer.options.appearance.fullScreenMode = True

    if designer.processRequest(request):
        return designer.getFrameworkResponse()

    # Here is the code for working with the report

    return designer.getFrameworkResponse()
```

In this case, after calling the event handler and performing actions with the report, the final result of the request is immediately returned. That is, the designer will return a fully prepared HTML page with all the necessary scripts and code.

Information

In most cases, using the product requires only Python code, which interacts with all the main capabilities of the components. For detailed product configuration and access to all the features of the JS generator, JavaScript code is needed. The deployment option using only JavaScript code is described in the Reports and Dashboards for JS section, in this case, Python code is only required for connecting SQL data adapters.

To form the response, the `getFrameworkResponse()` function is used, which returns a ready-made object suitable for the currently used web framework. The current framework is determined at the time of request processing. Our components support popular frameworks like **Django**, **Flask**, and **Tornado**. For handling requests and forming responses in web projects that do not use these frameworks, you need to pass the current GET and POST data to the request processing function and use the `getResponse()` function to generate the response, which will return all the necessary data. For example, handling designer events can be implemented in the following way:

app.py

```
from flask import Flask, make_response, render_template, url_for, request
from stimulsoft_reports.designer import StiDesigner

app = Flask(__name__)

@app.route('/designer', methods = ['GET', 'POST'])
def designer():
    designer = StiDesigner()
    designer.options.appearance.fullScreenMode = True

    query = request.args.to_dict()
    body = request.get_data(False)
    if designer.processRequest(None, query, body):
        designerResponse = designer.getResponse()
        response = make_response(designerResponse.data)
        response.mimetype = designerResponse.mimetype
        response.headers.add('Access-Control-Allow-Origin',
request.origin)
        return response

    js = designer.javascript.getHtml()
```

```
    html = designer.getHtml()
    return render_template('designer.html', designerJavaScript = js,
designerHtml = html)
```

12.3.2 Activation

After purchasing the product, you need to activate the license for the components you use. There are several ways to connect a license key.

All options for activating components are described in the License activation section and have the same functions and call parameters.

12.3.3 Creating and Editing Reports

No action is needed to start the designer without a report. Once the component is loaded, the main designer menu will appear. If you wish to initiate the designer with a new (empty) report, you can create a new `StiReport` report object and allocate it to the designer.

For editing a report within the designer, just create a `StiReport` object, load a report template into it, and assign the resulting object to the designer. All other actions will occur automatically; the designer will showcase the first page of the template.

app.py

```
from flask import Flask, url_for, request
from stimulsoft_reports.report import StiReport
from stimulsoft_reports.designer import StiDesigner

app = Flask(__name__)

@app.route('/designer', methods = ['GET', 'POST'])
def designer():
    designer = StiDesinger()
    designer.options.appearance.fullScreenMode = True

    if designer.processRequest(request):
        return designer.getFrameworkResponse()

    report = StiReport()
    report.loadFile(url_for('static', filename='reports/SimpleList.mrt'))
    designer.report = report

    return designer.getFrameworkResponse()
```

The designer is equipped to handle standard, packaged, and encrypted templates. Detailed instructions on working with various report formats are provided in the Loading and Saving Reports section.

Report creation event

A new report can be generated using the main menu of the designer. To preload data for a new report or execute any other necessary actions with a new report, the `onCreateReport` event is utilized. This event will be activated when a new empty report is created from the main menu or when a report is generated using the wizard.

Within an event on the Python server side, modifications to the report or its parameters are permitted.

app.py

```
from stimulsoft_reports.designer import StiDesigner
from stimulsoft_reports.events import StiReportEventArgs

def createReport(args: StiReportEventArgs):
    args.report['ReportDescription'] = 'This is a report description from
the Python server-side.'

designer = StiDesigner()
designer.onCreateReport += createReport
```

All functionalities of the reporting tool are accessible in the client-side JavaScript event. For instance, you can link data for a new report and synchronize it with a dictionary.

app.py

```
from stimulsoft_reports.designer import StiDesigner

designer = StiDesigner()
designer.onCreateReport += 'createReport'
```

designer.html

```
<script>
    function createReport(args) {
        let dataSet = new Stimulsoft.System.Data.DataSet("Demo");
        dataSet.readJsonFile("/static/data/Demo.json");

        let report = args.report;
        report.regData(dataSet.dataSetName, "", dataSet);
        report.dictionary.synchronize();
    }
</script>
```

A detailed description of the available argument values is in the Designer Events section.

12.3.4 Saving a Report

The designer provides two options for saving a report, available in the main menu and on the main toolbar: **Save** and **Save As**. Each of these saving options has its own modes and settings.

Saving a report on the client-side of JavaScript

When the **Save** button is clicked, the report template file is saved using the browser's functionality, and no additional settings are required. If you need to save the report using custom methods, the `onSaveReport` event is provided. The event arguments will include the report file name and the report itself. For example, the report can be saved as a JSON string and sent to the server using custom methods.

app.py

```
from stimulsoft_reports.designer import StiDesigner

designer = StiDesigner()
designer.onSaveReport += 'saveReport'
```

designer.html

```
<script>
    function saveReport(args) {
        let fileName = args.fileName;
        let report = args.report;
```

```
        let jsonReport = report.saveToJsonString();
    }
</script>
```

If the event is defined, after its completion, the designer continues to work without any additional actions or messages. If necessary, after saving the report, a dialog box with an error or text message can be displayed. For this purpose, there is a special function `StiError.showError()`. You determine whether displaying an error message is necessary.

designer.html

```
<script>
    function saveReport(args) {
        let fileName = args.fileName;
        let report = args.report;

        // Error message
        Stimulsoft.System.StiError.showError("An error occurred while
saving the report.");

        // Info message
        Stimulsoft.System.StiError.showError("The report was saved
successfully.", true, true);
    }
</script>
```

When the **Save As** button is clicked, a dialog box will appear requesting the report file name. After that, the report template file will be saved using the browser's built-in functionality. If you need to save the report using your own methods, the `onSaveAsReport` event is provided. The event arguments will include the report file name and the report itself.

The behavior of this event is the same as the `onSaveReport` event, except that after the event is completed, the report template will be automatically saved on the computer using the browser's functionality. To prevent this action, you can set the `preventDefault` property to `true` in the event arguments, which will stop the automatic saving.

app.py

```
from stimulsoft_reports.designer import StiDesigner

designer = StiDesigner()
designer.onSaveAsReport += 'saveAsReport'
```

designer.html

```
<script>
    function saveAsReport(args) {
        args.preventDefault = true;
    }
</script>
```

If necessary, you can access the original report name or the name from the save dialog as follows:

designer.html

```
<script>
    function saveAsReport(args) {
        // Report name from the designer save dialog
        var reportName1 = args.fileName;

        // Original report name from properties
        var reportName2 = args.report.reportName;
    }
</script>
```

A detailed description of available argument values can be found in the Designer Events section.

Saving the report on the Python server-side

To save the report on the Python server-side, simply define the `onSaveReport` event. The event arguments will include the report file name and the report itself as an object. You can use standard functions to save the report, for example, saving the edited report as a file in a specified directory:

app.py

```
import json
import os
from stimulsoft_reports import StiResult
from stimulsoft_reports.designer import StiDesigner
from stimulsoft_reports.events import StiReportEventArgs
from stimulsoft_reports.report import StiReport

def saveReport(args: StiReportEventArgs):
    filePath = os.path.normpath(os.getcwd() + '\\static\\reports\\' +
args.fileName)
    try:
        with open(filePath, mode='w', encoding='utf-8') as file:
            jsonReport = json.dumps(args.report, indent = 4)
            file.write(jsonReport)
            file.close()
    except Exception as e:
        return StiResult.getError(str(e))

    return f'The report was successfully saved to a {args.fileName} file.'

designer = StiDesigner()
designer.onSaveReport += saveReport
```

Similarly, the `onSaveAsReport` event works on the Python server-side, with all event arguments having the same names and values.

A detailed description of the available argument values can be found in the Designer Events section.

12.3.5 Localization

The designer supports full localization of its interface. To localize the interface into the required language, just set the required file name for the `localization` option in the designer:

app.py

```
from stimulsoft_reports.designer import StiDesigner

designer = StiDesigner()
designer.options.localization = 'de.xml'
```

All available localization XML files are located in the resources of the installed product package. If necessary, the localization file can be loaded from any other

location; for this you need to specify the full path to the desired XML file for the `localization` option:

app.py

```
from stimulsoft_reports.designer import StiDesigner

designer = StiDesigner()
designer.options.localization = '/resources/loc/de.xml'
```

If the file is readable from the Python application, the localization will load into the designer. Otherwise, the built-in English localization of the interface will be used.

The designer provides the option to select the necessary interface localization using a special menu on the toolbar. By default, English (built-in) localization is included in this menu, as well as any specified using the `localization` property. To add additional localizations to the menu, there's a special designer option, `localization`, which is a collection of localizations. The localization file or the full path to this file is specified as values.

app.py

```
from stimulsoft_reports.designer import StiDesigner

designer = StiDesigner()
designer.options.localization = 'de.xml'
designer.options.localizations.append('fr.xml')
designer.options.localizations.append('pl.xml')
designer.options.localizations.append('/resources/loc/it.xml')
```

12.3.6 Preview

The designer has a preview mode for the edited report. To access it, simply navigate to the appropriate tab in the designer window. The report template will be rendered and displayed in the built-in viewer.

Preview events

Before previewing the report, you have the opportunity to carry out any necessary actions. For this purpose, there is a special event called `onPreviewReport`, which

will be triggered before the report is viewed. The event arguments will include the report intended for preview.

Within an event on the Python server side, modifications to the report or its parameters are permitted.

app.py

```
from stimulsoft_reports.designer import StiDesigner
from stimulsoft_reports.events import StiReportEventArgs

def previewReport(args: StiReportEventArgs):
    args.report['ReportDescription'] = 'This is a report description from
the Python server-side.'

designer = StiDesigner()
designer.onPreviewReport += previewReport
```

All functionalities of the reporting tool are accessible in the client-side JavaScript event. For example, you can connect data for a report.

app.py

```
from stimulsoft_reports.designer import StiDesigner

designer = StiDesigner()
designer.onPreviewReport += 'previewReport'
```

designer.html

```
<script>
    function previewReport(args) {
        let dataSet = new Stimulsoft.System.Data.DataSet("Demo");
        dataSet.readJsonFile("/static/data/Demo.json");

        let report = args.report;
        report.regData(dataSet.dataSetName, "", dataSet);
    }
</script>
```

You can find a list of available built-in viewer events and descriptions of use cases in

the Viewer Events section.

Additional features

The report preview window in the designer is a fully interactive viewer that can print and export the report, and supports working with report parameters. All available interactive actions are supported, such as dynamic sorting, drill-down, and collapsing. You do not need any additional settings in the report designer to use these features.

12.3.7 Designer Themes

The designer has the ability to change the design themes of visual controls. To do this, set the theme property in the component options.

app.py

```
from stimulsoft_reports.designer import StiDesigner
from stimulsoft_reports.designer.enums import StiDesignerTheme

designer = StiDesigner()
designer.options.appearance.theme =
StiDesignerTheme.OFFICE_2022_DARKGRAY_BLUE
```

Currently, there are **two themes** available, each with its own color accents. As a result, more than 50 design options are available. This allows you to customize the appearance of the designer to suit almost any web project design.

12.3.8 Designer Events

The report designer supports events that provide the ability to perform necessary operations before specific actions-both on the JavaScript client side and the Python server side. A detailed description of how events work can be found in the Report Engine Events section.

Some event arguments take values from enumerations that are located in specific namespaces. All the enumerations used in designer events are listed in the code block below:

app.py

```
from stimulsoft_reports.enums import StiEventType
```

The designer supports the following events:

- onPrepareVariables
- onBeginProcessData
- onEndProcessData
- onCreateReport
- onOpenReport
- onOpenedReport
- onSaveReport
- onSaveAsReport
- onPreviewReport
- onCloseReport
- onExit

onPrepareVariables
[v] JavaScript [v] Python

The event is triggered before the report is built, after the report variables are prepared. A list of event arguments can be found in the Report Engine Events section. Detailed descriptions and usage examples can be found in the Working with Report Variables section.

onBeginProcessData
[v] JavaScript [v] Python

The event is triggered before requesting the data needed to build the report. A list of event arguments can be found in the Report Engine Events section. Detailed descriptions and usage examples can be found in the Connecting Data Files and Connecting SQL Data Adapters sections.

onEndProcessData
[v] JavaScript [v] Python

The event is triggered after the data is loaded, before the report is built. A list of event arguments can be found in the Report Engine Events section. Detailed descriptions and usage examples can be found in the Connecting Data Files and Connecting SQL Data Adapters sections.

onCreateReport
[v] JavaScript [v] Python

The event is triggered after a new report is created in the designer. The table below presents a list of event handler arguments on the JavaScript client-side:

Name	Description
`event`	The identifier of the current event has the value `"CreateReport"`.
`sender`	The identifier of the component that initiated this event can take the following value: • `"Designer"`
`report`	The current report object.
`isWizardUsed`	This flag indicates whether the new report is being created using a

	wizard (`true`) or as a blank report (`false`).
`preventDefault`	This flag provides the ability to stop further event processing. By default, it is set to `false`.

List of properties passed in the event arguments on the Python server-side. The arguments are of type `StiReportEventArgs`:

Name	Description
`event`	The identifier of the current event, for this event, it has the value `StiEventType.CREATE_REPORT`.
`sender`	The identifier of the component that initiated the event, which can have the following value: • `StiDesigner`
`report`	The current report object.
`isWizardUsed`	The flag indicates whether the new report is being created using a wizard (`true`), or as a blank report (`false`).

onOpenReport
[v] JavaScript [x] Python

List of properties passed in the event arguments on the JavaScript client-side:

Name	Desccription
`event`	The identifier of the current event, with the value "`OpenReport`".
`sender`	The identifier of the component that

	initiated the event, which can have the following value: • "Designer"
`preventDefault`	This flag allows you to stop further event processing by the designer. By default, it is set to `true`.

onOpenedReport
[v] JavaScript [v] Python

The event is triggered after opening a report from the designer menu but before it is loaded into the designer.

List of properties passed in the event arguments on the JavaScript client-side:

Name	Description
`event`	The identifier of the current event, with the value "`OpenReport`".
`sender`	The identifier of the component that initiated the event, which can have the following value: • "`Designer`"
`report`	The current report object.
`preventDefault`	This flag allows you to stop further event processing by the designer. By default, it is set to `true`.

List of properties passed in the event arguments on the Python server-side. Arguments have the type `StiReportEventArgs`:

Name	Description
`event`	The identifier of the current event, for this event the value is

	`StiEventType.OPENED_REPORT`.
`sender`	The identifier of the component that initiated the event, which can take the following value: • `StiDesigner`
`report`	The current report object.

onSaveReport
[v] JavaScript [v] Python

The event is triggered when saving a report in the designer. Below is the list of arguments passed to the event handler on the JavaScript client side:

Name	Description
`event`	The identifier of the current event, with the value "`SaveReport`".
`sender`	The identifier of the component that initiated the event, which can take the following value: • "`Designer`"
`report`	The current report object.
`fileName`	The report file name to save.
`autoSave`	This flag indicates whether the report is being saved automatically (`true`), or by clicking the save button (`false`).
`preventDefault`	This flag allows stopping further event processing by the designer. By default, it is set to `false`.

List of properties passed in the event arguments on the Python server-side. Arguments have the type `StiReportEventArgs`:

Name	Description
event	The identifier of the current event, for this event the value is StiEventType.SAVE_REPORT.
sender	The identifier of the component that initiated the event, which can take the following value: • StiDesigner
report	The current report, represented as an object. The current report, represented as an object.
fileName	The report file name to save.
autoSave	This flag indicates whether the report is being saved automatically (true), or manually (false).

onSaveAsReport
[v] JavaScript [v] Python

The event is triggered when saving a report in the designer with a pre-entered file name. The list of properties passed in the event arguments on the client-side JavaScript:

Name	Description
event	The identifier of the current event, has the value "SaveAsReport".
sender	The identifier of the component that initiated this event, can have the following values: • "Designer"
report	The current report object.
fileName	The report file name to save.
preventDefault	This flag allows stopping further

	event processing by the designer. By default, it is set to `false`.

List of properties passed in the event arguments on the Python server side. The arguments are of type `StiReportEventArgs`:

Name	Description
`event`	The identifier of the current event, for this event, it has the value `StiEventType.SAVE_AS_REPORT`.
`sender`	The identifier of the component that initiated this event, can have the following values: • `StiDesigner`
`report`	The current report object.
`fileName`	The name of the report file to save.

onPreviewReport
[v] JavaScript [v] Python

The event is triggered when switching to the report preview tab.

List of properties passed in the event arguments on the client-side JavaScript:

Name	Description
`event`	The identifier of the current event, has the value `"PreviewReport"`.
`sender`	The identifier of the component that initiated this event, can have the following values: • `"Designer"`
`report`	The current report object.

`viewer`	The current embedded viewer component in the designer.
`preventDefault`	This flag allows stopping further event processing by the designer. By default, it is set to `true`.

List of properties passed in the event arguments on the Python server-side. The arguments are of type `StiReportEventArgs`:

Name	Descrription
`event`	The identifier of the current event, for this event, it has the value `StiEventType.PREVIEW_REPORT`.
`sender`	The identifier of the component that initiated this event, can have the following values: • `StiDesigner`
`report`	The current report object.

onCloseReport
[v] JavaScript [v] Python

The event is triggered after the report is closed from the designer menu and before the report has been unassigned from the report designer.

List of properties passed in the event arguments on the JavaScript client-side:

Name	Description
`event`	The identifier of the current event, with the value `"CloseReport"`.
`sender`	The identifier of the component that initiated the event, which can have the following value: • `"Designer"`

`report`	The current report object.
`preventDefault`	This flag allows you to stop further event processing by the designer. By default, it is set to `true`.

List of properties passed in the event arguments on the Python server-side. Arguments have the type `StiReportEventArgs`:

Name	Description
`event`	The identifier of the current event, for this event the value is `StiEventType.CLOSE_REPORT`.
`sender`	The identifier of the component that initiated the event, which can take the following value: • `StiDesigner`
`report`	The current report object.

onExit
[v] JavaScript [x] Python

The event is triggered when the **Exit** button is clicked in the main menu of the designer.

List of properties passed in the event arguments on the client-side JavaScript:

Name	Description
`event`	The identifier of the current event, has the value "`Exit`".
`sender`	The identifier of the component that initiated this event, can have the following values: • "`Designer`"

12.3.9 Designer Settings

The designer is configured by modifying the property values located in the main properties container called `options` of the component. All properties are divided into groups for ease of use. All enumerations used in the viewer settings are located in the namespace `stimulsoft_reports.designer.enums`.

An example of modifying some designer settings:

app.py

```
from flask import Flask, request
from stimulsoft_reports.designer import StiDesigner
from stimulsoft_reports.designer.enums import StiReportUnitType,
StiDesignerTheme

@app.route('/designer', methods = ['GET', 'POST'])
def designer():
    designer = StiDesigner()
    designer.options.appearance.theme =
StiDesignerTheme.OFFICE_2022_DARKGRAY_BLUE
    designer.options.appearance.defaultUnit =
StiReportUnitType.CENTIMETERS
    designer.options.appearance.showReportTree = False
    designer.options.appearance.showTooltips = False
    designer.options.bands.showChildBand = False
    designer.options.components.showPanel = False
    designer.options.toolbar.showFileMenuExit = False
    designer.options.toolbar.showFileMenuOptions = False

    if designer.processRequest(request):
        return designer.getFrameworkResponse()

    # Here is the code for working with the report

    return designer.getFrameworkResponse()
```

Main (without group)

Name	Description
`width`	Sets the width of the component in "px" or "%". By default, the value is set to "100%".
`height`	Sets the height of the component in "px" or "%". By default, the value is

	set to "100%" for standard mode and "650px" for scroll mode.
`localization`	Sets the selected localization of the component. By default, the English localization is embedded in the component.

Appearance

Name	Description
`theme`	Sets the theme of the designer. The list of available themes is in the `StiDesignerTheme`. By default, the value is set to `StiDesignerTheme.OFFICE_2022_WHITE_BLUE`.
`iconSet`	Allows setting the icon set: • `StiWebUIIconSet.AUTO` (default value) -automatically sets the icon set. For `Office2022` themes, a `Monoline` icon set is used, and for `Office2013` themes, a `Regular` icon set is used. • `StiWebUIIconSet.MONOLINE` - sets the `Monoline` icon set; • `StiWebUIIconSet.REGULAR` - sets the `Regular` icon set.
`defaultUnit`	Sets the background color of the viewer. By default, the value is `'white'`. • `StiReportUnitType.CENTIMETERS` (default value); • `StiReportUnitType.HUNDREDTHS_OF_INCH`; • `StiReportUnitType.INCHES`;

	• `StiReportUnitType.MILLIMETERS`.
`zoom`	Sets the scale for displaying report pages. The default scale is set to 100 percent. Allowed values range from 10 to 200 percent. It can take one of the following values from the `StiZoomMode` enumeration: • `StiZoomMode.PAGE_WIDTH` – scales the report pages to the width of the page; • `StiZoomMode.PAGE_HEIGHT` – scales the report pages to the height of the page.
`interfaceType`	Sets the interface type of the designer . The following values can be used: • `StiInterfaceType.AUTO` – the designer interface type will be automatically selected based on the device being used (default value); • `StiInterfaceType.MOUSE` – forces the use of the standard interface for controlling the viewer with a mouse; • `StiInterfaceType.TOUCH` – forces the use of the Touch interface for controlling the viewer with a touchscreen monitor; in this mode, the viewer's interface elements are larger for ease of use;
`showAnimation`	Enables or disables animation for displaying and closing various menus in the viewer. By default, this is set to `True`.

showSaveDialog	Enables the display of the report name input dialog when saving a report. The report name will be passed in the designer parameters. By default, the property is set to True.
showTooltips	Enables the display of tooltips when hovering over the viewer's tools. By default, the value is set to True.
showTooltipsHelp	Allows displaying a link to the documentation in the tooltips when hovering over the viewer's tools. By default, the value is set to True.
showDialogsHelp	Allows displaying or hiding the help button in various menus. By default, the value is set to True.
fullScreenMode	Sets the full-screen mode for the designer. If this property is set to True, the width and height properties are ignored. By default, the value is set to False.
maximizeAfterCreating	Allows setting the maximum size of the report designer. By default, the property is set to False.
showLocalization	Provides the option to show or hide the localization control in the report designer. By default, the property is set to True.
allowChangeWindowTitle	Allows the use of the browser window title to display the name of the edited report file. By default, the property is set to True.
showPropertiesGrid	Enables the display of the properties panel in the report designer. By default, the property is set to True.

`showReportTree`	Enables the display of the report components tree. By default, the property is set to `True`.
`propertiesGridPosition`	Provides the option to define the position of the properties panel. The following values can be used: • `StiPropertiesGridPosition.LEFT` – the properties panel will be displayed on the left (default value); • `StiPropertiesGridPosition.RIGHT` – the properties panel will be displayed on the right.
`showSystemFonts`	Provides the option to show or hide system fonts in the font list. By default, the property is set to True, meaning system fonts are displayed in the font list.
`datePickerFirstDayOfWeek`	Allows setting the first day of the week for the **Date Picker** tool. • `StiFirstDayOfWeek.AUTO` - Monday or Sunday will be set as the first day of the week based on the browser culture; • `StiFirstDayOfWeek.MONDAY` - Monday will be set as the first day of the week; • `StiFirstDayOfWeek.SUNDAY` - Sunday will be set as the first day of the week.Sunday will be set as the first day of the week.
`formatForDateControls`	This feature allows you to customize the format for date controls. By default, the current option doesn't have a specified value, and the date format is determined based on the browser's locale.

undoMaxLevel	Sets the maximum undo depth for changes to the report while editing. This affects memory consumption. By default, it is set to 6 changes.
wizardTypeRunningAfterLoad	Provides the option to call the report creation wizard after starting the report designer. It can take one of the following values from the enumeration: • StiWizardType.NONE - the report designer will start without calling the report creation wizard (default value); • StiWizardType.STANDARD_REPORT - the report designer will start with the standard report creation wizard; • StiWizardType.MASTER_DETAIL_REPORT - the report designer will start with the master-detail report creation wizard; • StiWizardType.LABEL_REPORT - the report designer will start with the label report creation wizard; • StiWizardType.INVOICES_REPORT - the report designer will start with the invoice creation wizard; • StiWizardType.ORDERS_REPORT - the report designer will start with the order creation wizard; • StiWizardType.QUOTATION_REPORT - the report designer will start with the quotation creation wizard.

allowWordWrapTextEditors	Enables or disables line wrapping in text editors in the designer. By default, the property is set to True.

Toolbar

Name	Description
visible	Enables the display of the toolbar in the report designer. By default, the property is set to True.
showPreviewButton	Enables or disables the display of the **Preview** button on the designer toolbar. By default, the property is set to True.
showSaveButton	Enables the display of the **Save** button on the designer's toolbar. By default, this is set to False.
showAboutButton	Enables the display of the **About** button on the designer toolbar. By default, the property is set to False.
showFileMenu	Enables the display of the main menu in the report designer. By default, the property is set to True.
showFileMenuNew	Enables the display of the **New** item in the main menu. By default, the property is set to True.
showFileMenuOpen	Enables the display of the **Open** item in the main menu. By default, the property is set to True.
showFileMenuSave	Enables the display of the Save item in the main menu. By default, the property is set to True.
showFileMenuSaveAs	Enables the display of the **Save As** item in the main menu. By default,

	the property is set to `True`.
`showFileMenuClose`	Enables the display of the **Close** item in the main menu. By default, the property is set to `True`.
`showFileMenuExit`	Enables the display of the **Exit** item in the main menu. By default, the property is set to `False`.
`showFileMenuReportSetup`	Enables the display of the **Report Setup** item in the main menu. By default, the property is set to `True`.
`showFileMenuOptions`	Enables the display of the **Options** item in the main menu. By default, the property is set to `True`.
`showFileMenuInfo`	Enables the display of the **Info** item in the main menu. By default, the property is set to `True`.
`showFileMenuAbout`	Enables the display of the **About** item in the main menu. By default, the property is set to `True`.
`showFileMenuNewReport`	Enables or disables the display of the **New Page** item in the main menu. By default, the property is set to `True`.
`showFileMenuNewDashboard`	Enables or disables the display of the **New Dashboard** item in the main menu. By default, the property is set to `True`.
`showSetupToolboxButton`	Enables or disables the display of the settings button for the report component sidebar. By default, the property is set to `True`.
`showNewPageButton`	Enables or disables the display of the **New Page** button on the toolbar. By default, the property is set to `True`.

showNewDashboardButton	Enables or disables the display of the **New Dashboard** button on the toolbar. By default, the property is set to `True`.

Bands

Name	Description
showReportTitleBand	Enables the display of the **Report Title** section in the **Sections** menu. By default, the property is set to `True`.
showReportSummaryBand	Enables the display of the **Report Summary** section in the **Sections** menu. By default, the property is set to `True`.
showPageHeaderBand	Enables the display of the **Page Header** section in the **Sections** menu. By default, the property is set to `True`.
showPageFooterBand	Enables the display of the **Page Footer** section in the **Sections** menu. By default, the property is set to `True`.
showGroupHeaderBand	Enables the display of the **Group Header** section in the **Sections** menu. By default, the property is set to `True`.
showGroupFooterBand	Enables the display of the **Group Footer** section in the **Sections** menu. By default, the property is set to `True`.
showHeaderBand	Enables the display of the **Header** section in the **Sections** menu. By default, the property is set to `True`.

showFooterBand	Enables the display of the **Footer** section in the **Sections** menu. By default, the property is set to True.
showColumnHeaderBand	Enables the display of the **Column Header** section in the **Sections** menu. By default, the property is set to True.
showColumnFooterBand	Enables the display of the **Column Footer** section in the **Sections** menu. By default, the property is set to True.
showDataBand	Enables the display of the **Data** section in the **Sections** menu. By default, the property is set to True.
showHierarchicalBand	Enables the display of the **Hierarchical** section in the **Sections** menu. By default, the property is set to True.
showChildBand	Enables the display of the **Child** section in the **Sections** menu. By default, the property is set to True.
showEmptyBand	Enables the display of the **Empty** section in the **Sections** menu. By default, the property is set to True.
showOverlayBand	Enables the display of the **Overlay** section in the **Sections** menu. By default, the property is set to True.
showTable	Enables the display of the **Table** component in the **Sections** menu. By default, the property is set to True.
showTableOfContents	Enables or disables the display of the **Table of Contents** component in the **Sections** menu. By default, the property is set to True.

Cross-Bands

Name	Description
showCrossTab	Enables the display of the **Cross-Tab** component in the Cross menu. By default, the property is set to True.
showCrossGroupHeaderBand	Enables the display of the **Cross Group Header** section in the Cross menu. By default, the property is set to True.
showCrossGroupFooterBand	Enables the display of the **Cross Group Footer** section in the Cross menu. By default, the property is set to True.
showCrossHeaderBand	Enables the display of the **Cross Header** section in the Cross menu. By default, the property is set to True.
showCrossFooterBand	Enables the display of the **Cross Footer** section in the Cross menu. By default, the property is set to True.
showCrossDataBand	Enables the display of the **Cross Data** section in the Cross menu. By default, the property is set to True.

Dashboard Elements

Name	Description
showTableElement	Enables the display of the **Table** indicator panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to True.
showCardsElement	Enables the display of the **Cards**

	indicator panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showChartElement`	Enables the display of the **Chart** indicator panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showGaugeElement`	Enables the display of the **Gauge** indicator panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showPivotTableElement`	Enables the display of the **Pivot Table** indicator panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showIndicatorElement`	Enables the display of the **Indicator** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showProgressElement`	Enables the display of the **Progress** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showRegionMapElement`	Enables the display of the **Region Map** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showOnlineMapElement`	Enables the display of the **Online Map** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showImageElement`	Enables the display of the **Image** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.

`showTextElement`	Enables the display of the **Text** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showPanelElement`	Enables the display of the **Panel** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showShapeElement`	Enables the display of the **Shape** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showButtonElement`	Enables the display of the **Button** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showListBoxElement`	Enables the display of the **ListBox** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showComboBoxElement`	Enables the display of the **ComboBox** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showTreeViewElement`	Enables the display of the **Tree View** panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showTreeViewBoxElement`	Enables the display of the **Tree ViewBox** indicator panel item in the toolbar or the **Insert** tab in the designer. By default, the property is set to `True`.
`showDatePickerElement`	nables the display of the **Date Picker** indicator panel item in the toolbar or the **Insert** tab in the designer. By

	default, the property is set to `True`.

Components

Name	Description
`showText`	Enables the display of the **Text** component in the **Components** menu. By default, the property is set to `True`.
`showTextInCells`	Enables the display of the **Text** in Cells component in the **Components** menu. By default, the property is set to `True`.
`showRichText`	Enables the display of the **Rich** Text component in the **Components** menu. By default, the property is set to `False`.
`showImage`	Enables the display of the **Image** component in the **Components** menu. By default, the property is set to `True`.
`showBarCode`	Enables the display of the **Bar Code** component in the toolbar or the Insert tab in the designer. By default, the property is set to `True`.
`showShape`	Enables the display of the **Shape** component in the toolbar or the Insert tab in the designer. By default, the property is set to `True`.
`showPanel`	Enables the display of the **Panel** component in the **Components** menu. By default, the property is set to `True`.
`showClone`	Enables the display of the **Clone**

	component in the **Components** menu. By default, the property is set to `False`.
`showCheckBox`	Enables the display of the **Check Box** component in the **Components** menu. By default, the property is set to `True`.
`showSubReport`	Enables the display of the **Sub Report** component in the **Components** menu. By default, the property is set to `True`.
`showZipCode`	Enables the display of the **Zip Code** component in the **Components** menu. By default, the property is set to `False`.
`showChart`	Enables the display of the **Chart** component in the toolbar or the Insert tab in the designer. This applies to all chart types. By default, the property is set to `True`.
`showGauge`	Enables or disables the display of the **Gauge** component in the toolbar or the Insert tab in the designer. By default, the property is set to `True`.
`showSparkline`	Enables the display of the **Sparkline** component in the **Components** menu. By default, the property is set to `True`.
`showMathFormula`	Enables or disables the display of the **Math Formula** component in the **Components** menu. By default, the property is set to `False`.
`showMap`	Enables or disables the display of the **Map** component in the toolbar or the Insert tab in the designer. By default, the property is set to `True`.

Dictionary

Name	Description
showAdaptersInNewConnectionForm	Enables the display of the **Object** category in the new connection window. By default, the property is set to True.
showDictionary	Enables the display of the report's data dictionary. By default, the property is set to True.
newReportDictionary	Allows the creation of a new data dictionary or merging it with an existing one when creating a new report in the designer. It can take one of the following values: • StiNewReportDictionary.AUTO - determines the mode of creating or merging the data dictionary from the saved value in cookies (default value); • StiNewReportDictionary.DICTIONARY_NEW - sets the mode to create a new data dictionary when creating a new report; • StiNewReportDictionary.DICTIONARY_MERGE - sets the mode to merge an existing data dictionary with the new one when creating a new report in the designer.
useAliases	Allows the use of aliases in the data dictionary. It can take one of the specified values: • StiUseAliases.AUTO -

	determines the mode for using aliases from the saved value in `cookies` (default value); • `StiUseAliases.True` - enables the use of aliases in the data dictionary; • `StiUseAliases.False` - disables the use of aliases in the data dictionary.
`showDictionaryContextMenuProperties`	Sets the visibility of the **Properties** item in the context menu of the data dictionary. By default, the property is set to `True`.
`showDictionaryActions`	Sets the display of the **Actions** menu on the data dictionary toolbar. By default, the property is set to `True`.
`dataSourcesPermissions`	Sets the available actions for data sources in the report. It can take one or more values from the `StiDesignerPermissions` enumeration.
`dataConnectionsPermissions`	Sets the available actions for data connections in the report. It can take one or more values from the `StiDesignerPermissions` enumeration.
`dataColumnsPermissions`	Sets the available actions for data columns in the report. It can take one or more values from the `StiDesignerPermissions` enumeration.
`dataRelationsPermissions`	Sets the available actions for data relationships in the report. It can take one or more values from the `StiDesignerPermissions` enumeration.
`businessObjectsPermissions`	Sets the available actions for

	business objects in the report. It can take one or more values from the `StiDesignerPermissions` enumeration.
`variablesPermissions`	Sets the available actions for report variables. It can take one or more values from the `StiDesignerPermissions` enumeration.
`resourcesPermissions`	Sets the available actions for resources in the report's data dictionary. It can take one or more values from the `StiDesignerPermissions` enumeration.
`dataTransformationsPermissions`	Sets the available actions for data transformations. It can take one or more values from the `StiDesignerPermissions` enumeration.

The table below lists all available values for the `StiDesignerPermissions` enumeration, which can be set for report dictionary elements.

Value	Description
`StiDesignerPermissions.NONE`	Prohibits any action on a data dictionary element.
`StiDesignerPermissions.CREATE`	Allows creating a specific data dictionary element.
`StiDesignerPermissions.DELETE`	Allows deleting a specific data dictionary element.
`StiDesignerPermissions.MODIFY`	Allows editing a specific data dictionary element.
`StiDesignerPermissions.VIEW`	Allows viewing a specific data dictionary element.

`StiDesignerPermissions.MODIFY_VIEW`	Allows editing and viewing a specific data dictionary element.
`StiDesignerPermissions.ALL`	Allows any actions on a data dictionary element (default setting for all properties using this enumeration).

The built-in `StiViewer` component is designed for report preview customization. To access all its settings, use the `viewerOptions` property, which represents a viewer options object. All its properties are described in the Viewer Settings section. For example, you may need to change the report display mode and disable unnecessary export formats:

app.py

```
from flask import Flask, request
from stimulsoft_reports.designer import StiDesigner
from stimulsoft_reports.designer.enums import StiReportUnitType,
StiDesignerTheme

@app.route('/designer', methods = ['GET', 'POST'])
def designer():
    designer = StiDesigner()
    designer.options.viewerOptions.exports.showExportToWord2007 = False
    designer.options.viewerOptions.exports.showExportToCsv = False

    if designer.processRequest(request):
        return designer.getFrameworkResponse()

    # Here is the code for working with the report

    return designer.getFrameworkResponse()
```

13 Reports.Java

YouTube

Watch the video tutorials for working with the components of the Stimulsoft Reports.JAVA product. Subscribe to the Stimulsoft channel and be the first to watch new video lessons. Questions and suggestions is recommended be left in the comments to the video.

Samples

See on our website examples for working with the Reports.JAVA components. All examples are separate projects, grouped into one solution for Visual Studio. Also, you can view and download specific examples on GitHub.

Tools for creating and editing reports:	Tools for viewing and converting reports:
› HTML5 designer	› HTML5 Viewer

13.1 Activation

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this by specifying a license key or by downloading a file with the license key. Below is an example of activating the **StiWebViewer** component.

index.jsp

```
...
<body>
   <%
      //Activation with using license code
      com.stimulsoft.base.licenses.StiLicense.setKey(
      "Your activation code...";

      //Activation with using license file
      com.stimulsoft.base.licenses.StiLicense.loadFromFile(request.getSessi
      on().getServletContext().getRealPath("/license/license.key"));
   %>
   <stiwebviewer:webviewer report="${report}" options="${options}" />
</body>
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

13.2 Java Viewer

The **StiViewerFx** component is delivered as a part of Stimulsoft Reports.JAVA. This component is used to display reports in Java applications. To run the viewer you will need to place the **StiViewerFx** component on the **Frame** and set necessary properties to it. For running the report viewer it requires the Java 1.5 platform or higher.

- Showing Reports,
- Custom Functions.

13.2.1 Showing Reports

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `Report.Render()` method if you want to perform specific actions with the rendered report before it is displayed in the viewer. Likewise, when using the compilation mode, you need to call the `Report.Compile()` method only if you have actions to perform with the compiled report before it is built and shown in the viewer.

Add the **StiViewerFx** component on the required component (for example on JFx):

Jfx

```
...
StiViewerFx viewerfx = new StiViewerFx(parentFrame);
fx.add(viewerfx);
...
```

where the **JFrameparentFrame** is a main Frame of the application. For better showing **StiViewerFx**, the parent component must have **BoxLayoutManager**. For loading and showing the report use the following method:

Jfx

```
...
viewerfx.getStiViewModel().loadDocumentFile(documentFile, showProgress);
```

```
...
```

where the argument **documentFile** is a file of **mdc** documents, and the boolean value **showProgress** provides the ability to define whether to show the loading progress of the document (if it is set to true then the process is displayed, if false - it is not displayed.) It is also possible to display a report in the form of a dialog box, you can use the method:

Jfx

```
...
StiViewerFx viewerfx = new StiViewerFx(parentFrame);
JDialogviewerPopup = viewerfx.createPopup(parentFrame, modal);
viewerPopup.setVisible(true);
...
```

Where the argument **parentFrame** is a frame from which the dialog is displayed, and the Boolean value **modal** - dialog modality. The method returns **JDialog**, which subsequently can make the necessary manipulations, such as resizing, changing dialog parameters, visibility, etc.

13.2.2 Custom Functions

In addition to the standard (built-in) functions, there is an ability to define your own (custom) functions. To do this, we have the list customFunctions implemented in the class **StiReport**. Before rendering a report all required functions must be added in it. Classes of user-defined functions must implement the interface **com.stimulsoft.report.StiCustomFunction**. The description of the interface **StiCustomFunction**:

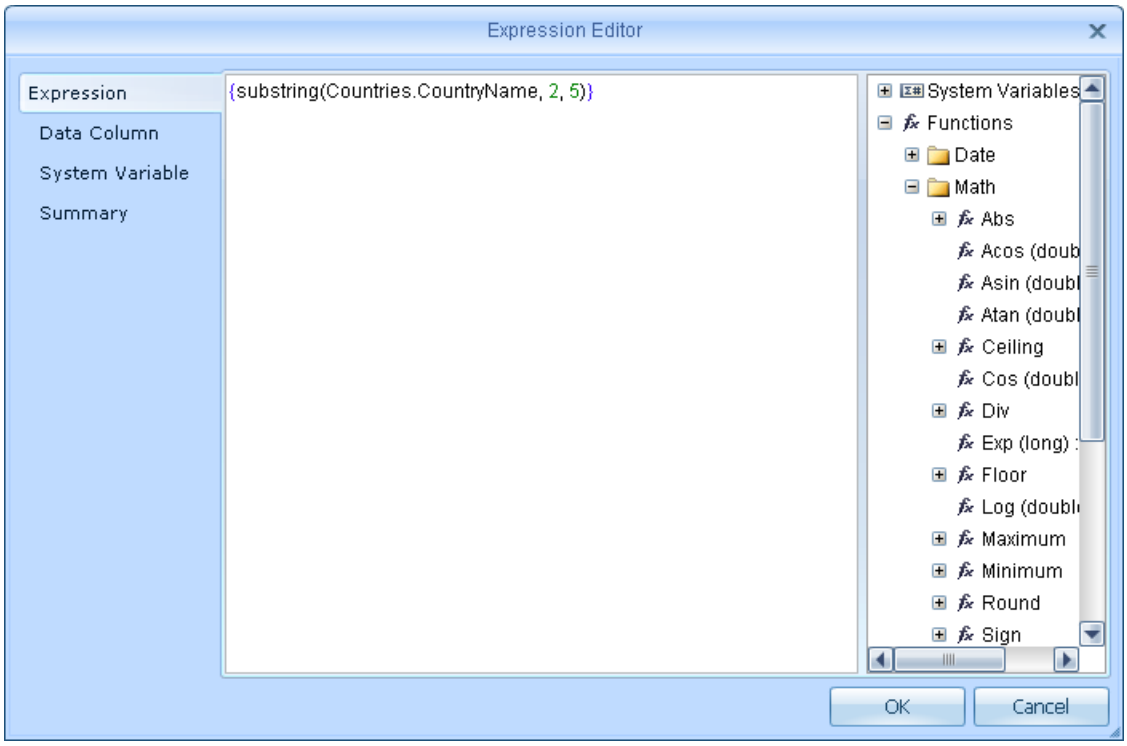
› **public String getFunctionName()** – the function class should return the name of the custom function. Register is taken into account. Do not use the names of existing built-in functions, methods, variables, reserved words as true/false/null, etc.
› **public List<Class> getParametersList()** – the function class should return a list of classes of variables used in the custom function.
› **public Object invoke(List<Object> args)** – there must be a realization of a custom function.

An example of using on the base of **Samples\webfx**. Suppose you need to implement a custom **substring** function. In the class **my.actions.MyRenderReportAction** write the following:

webfx

```
...
public StiReport render(StiReport report) throws IOException, StiException
{
   report.getCustomFunctions().add(new StiCustomFunction() {
      public Object invoke(List<Object> args) {
         return ((String) args.get(0)).substring((Integer)args.get(1),
         (Integer) args.get(2));
      }

      public List<Class> getParametersList() {
         return new ArrayList<Class>(Arrays.asList(String.class,
         Integer.class, Integer.class));
      }

      public String getFunctionName() {
         return "substring";
      }
   });
   return super.render(report);
}
...
```

Now you can use a custom substring function in a report:



13.3 Java HTML5 Viewer

The Java Angular Viewer will be described in this section.

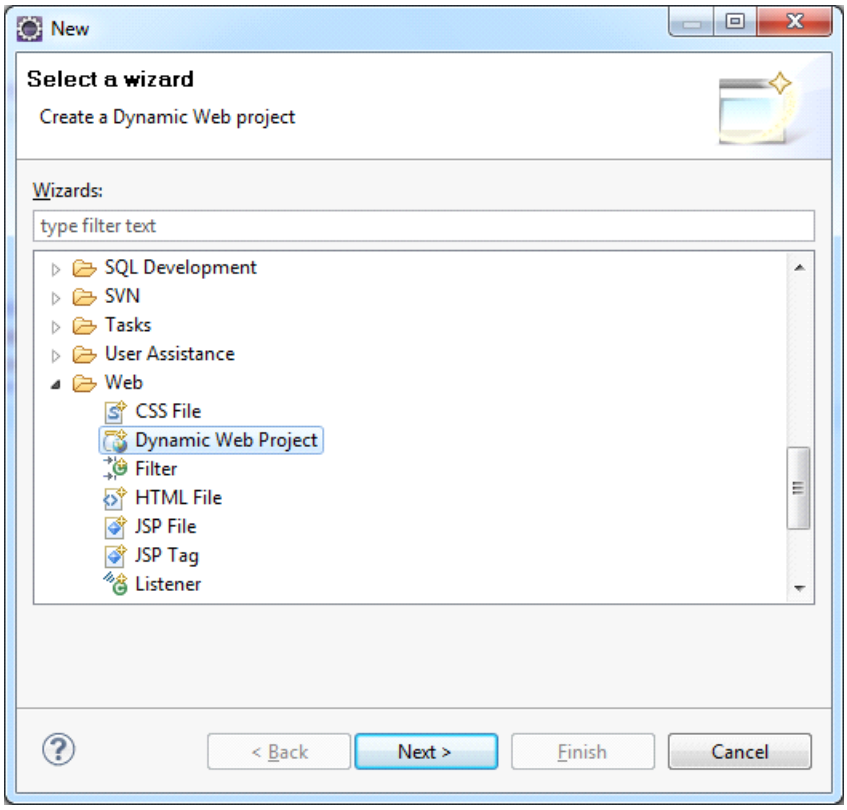
13.3.1 Installation

Installation:

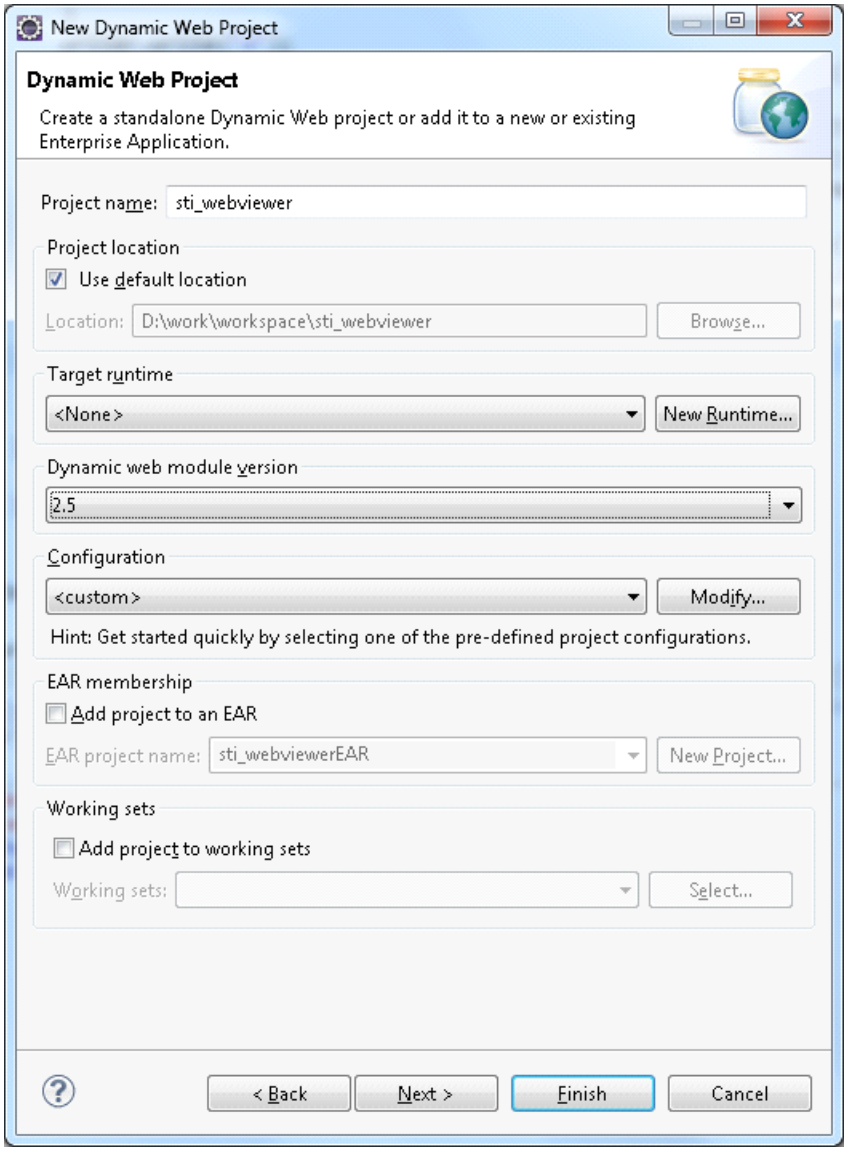
- Download and install **Java™ SE** version 1.5 or higher (for the version 1.5 jaxb-impl and jaxb-api libraries are required).
- Download and install **EclipsePlatform**.
- Download an archive with **jar** files on stimulsoft

13.3.2 Creating Project

Launch the **Eclipse IDE**, choose **File> New> Project**. In the project wizards open the Web type and in the drop-down list select **Dynamic Web Project** wizard and click Next (Figure 1). Select dynamic Web project:



In the window opened fill in the Project name (e.g. sti_webviewer, as shown on Figure2). Then configure the web server, on which the application will run. Create a new dynamic Web project:



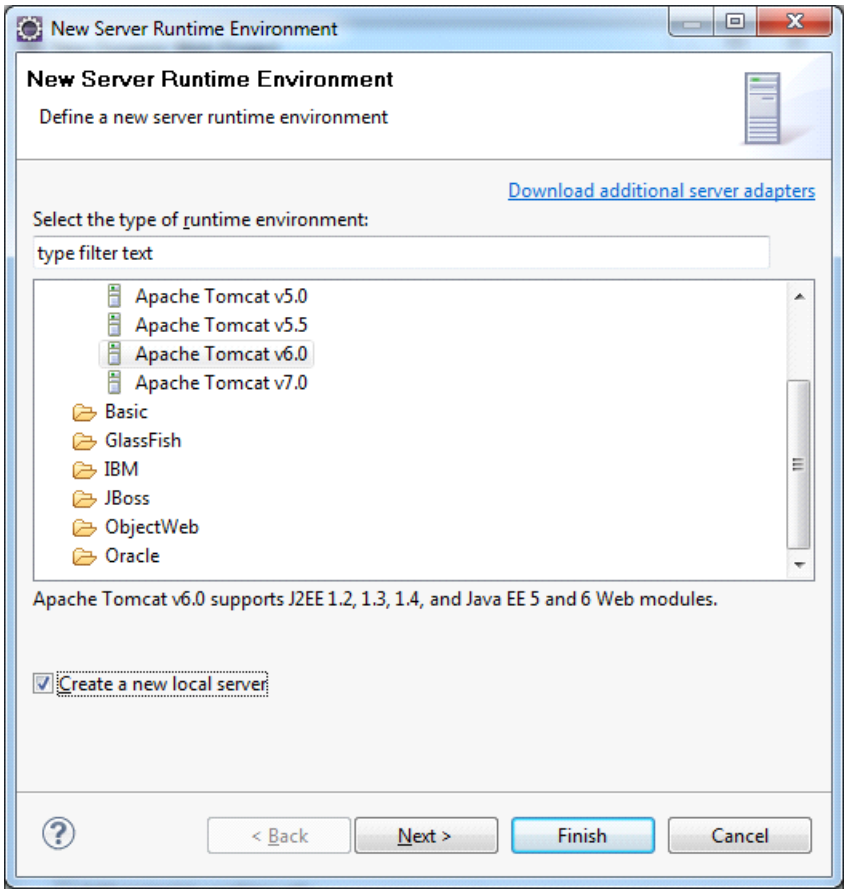
Target a runtime

Under Target Runtime, you see <None>, as shown in Figure 1, because you haven't created a runtime yet for Apache Tomcat. Click New Runtime to open the New Target Runtime Wizard. Select Apache Tomcat of the required version from the list of available, check the Create a new local server as shown in Figure 3, then click Next.

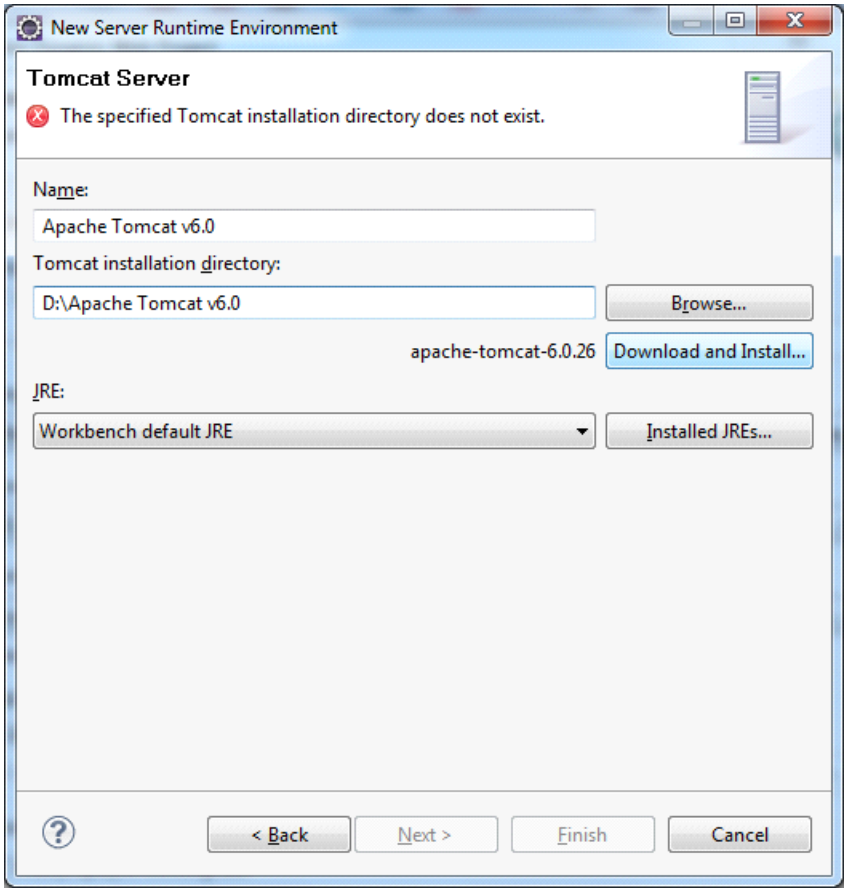
Target a runtime

Under Target Runtime, you see <None>, as shown in Figure 1, because you haven't created a runtime yet for Apache Tomcat. Click New Runtime to open the New Target Runtime Wizard. Select Apache Tomcat of the correct version from a

list. Check Create a new local server as shown on Figure 3, then click Next.Create a new server runtime:

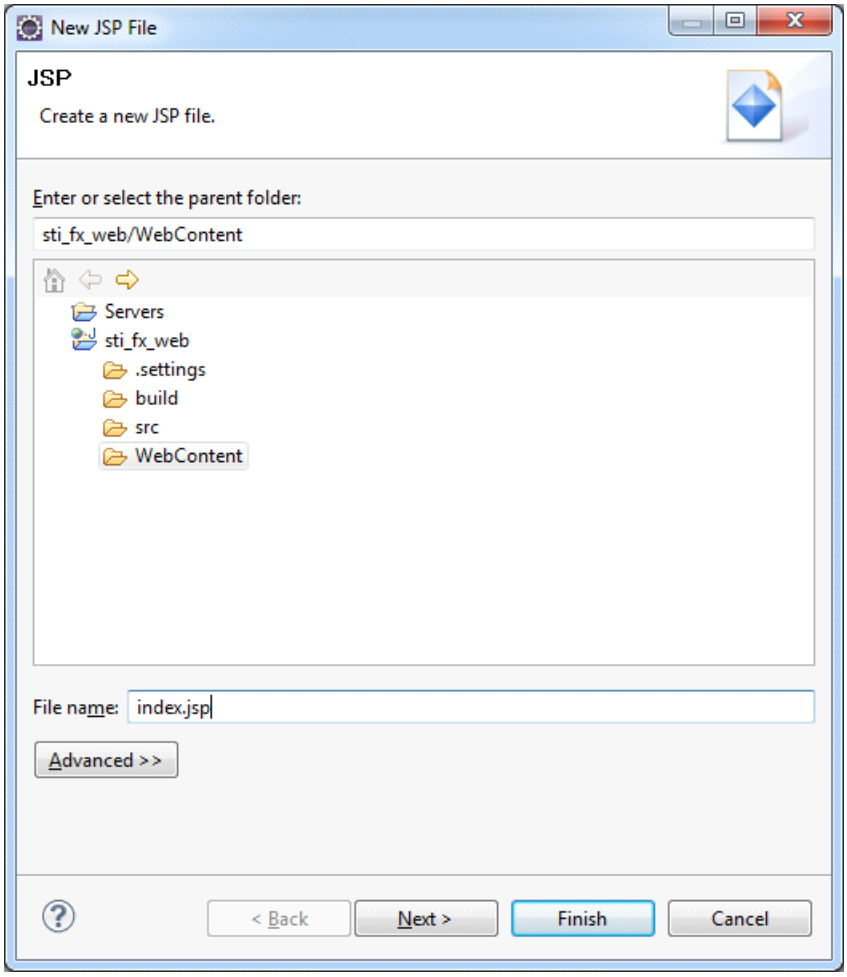


Then define the Tomcat installation directory, in which Apache Tomcat is installed, or in which one needs to install it, as shown on Figure 4. If it is not installed, then click Download and Install. After all fields are specified, click Finish. Define the server location:



13.3.3 Creating a Sample Page

In order to verify the project and the Tomcat server, create a simple JSP and deploy it on Tomcat. To do this, one can create a new JSP, by choosing **File> New> Other**, or one can use the context menu, right-click the project name in the **Project Explorer** and select **New> JSP** file. In the next window (see Figure 5) define the directory **WebContent**, and in the **File** name write **index.jsp**. Click **Finish** to create pages using the default template:



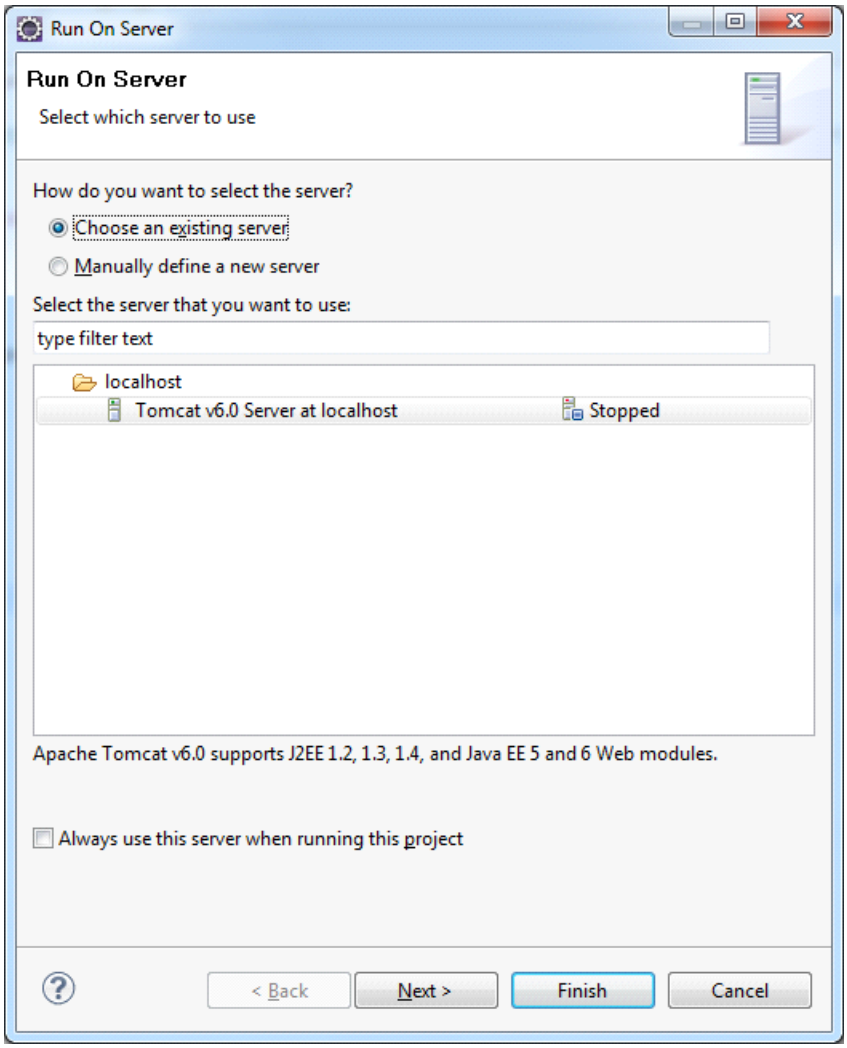
Now open the **index.jsp** and edit it so that it displays the current date. The code page is specified in the Listing 1.

index.jsp

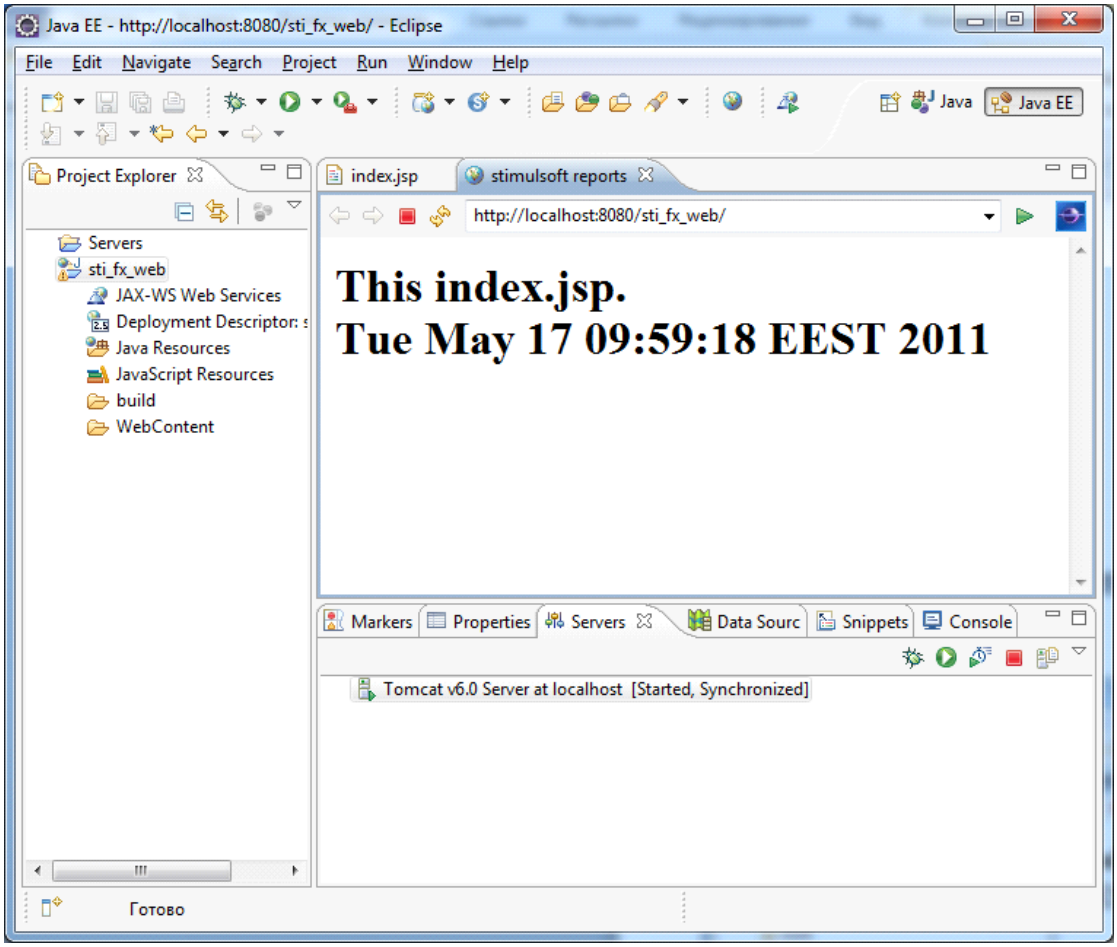
```
...
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>stimulsoft reports</title>
</head>
<body>
	<%java.util.Date date = new java.util.Date();%>
	<h1>
		This index.jsp.<br>
		<%=date.toString()%>
```

```
      </h1>
</body>
</html>
...
```

Now deploy it on the server. For this one need to use the context menu, right-click the project name, select **Run> Run as> Run** on server. Define a previously created server and click **Finish**:

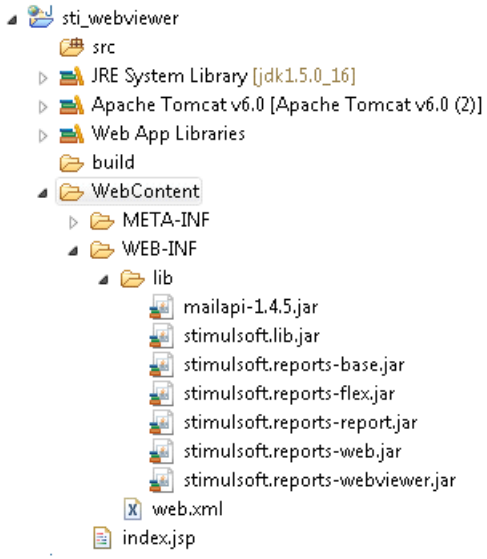


As a result, you receive the following (see Figure 7). This page will be available from any browser at **http://localhost:8080/{ProjectName}** (where the **{ProjectName}** name of the created project, in our case **sti_webviewer**):



13.3.4 Create a Sample Page With Report HTML5 Viewer

Create a simple page with a report webviewer. To do this, put the following libraries into the **WebContent\WEB-INF\lib** directory: stimulsoft.lib.jar, stimulsoft.reports-base.jar, stimulsoft.reports-report.jar, stimulsoft.reports-flex.jar, stimulsoft.reports-web.jar, stimulsoft.reports-webviewer.jar . As a result, one can see the following (Figure 8):



Next, open the web.xml for editing, it should look like in Listing 2:

web.xml

```
...
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/
xml/ns/javaee/webapp_2_5.xsd"
	xsi:schemaLocation="http://java.sun.com/xml/ns/javaee"
	id="WebApp_ID" version="2.5">
	<display-name>sti_webviewer</display-name>
	<welcome-file-list>
		<welcome-file>index.jsp</welcome-file>
	</welcome-file-list>
	<!-- configuration, this parameter indicates the main application
directory -->
	<servlet>
		<servlet-name>StimulsoftResource</servlet-name>
		<servlet-class>com.stimulsoft.web.servlet.StiWebResourceServlet</
servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>StimulsoftResource</servlet-name>
		<url-pattern>/stimulsoft_web_resource</url-pattern>
	</servlet-mapping>
	<servlet>
		<servlet-name>StimulsoftAction</servlet-name>
		<servlet-
class>com.stimulsoft.webviewer.servlet.StiWebViewerActionServlet</servlet-class>
	</servlet>
	<servlet-mapping>
```

```
            <servlet-name>StimulsoftAction</servlet-name>
            <url-pattern>/stimulsoft_webviewer_action</url-pattern>
      </servlet-mapping>
</web-app>
...
```

Leave unchanged the remaining **web.xml** blocks, which defines the servlets required for working. Then, edit the **index.jsp** (Listing 4).

index.jsp

```
...
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<%@page import="com.stimulsoft.report.StiReport"%>
<%@ page language="java" contentType="text/html; charset=utf-8"
      pageEncoding="UTF-8"%>
<%@ taglib uri="http://stimulsoft.com/webviewer" prefix="stiwebviewer"%>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Stimulsoft Reports for Java</title>
<stiwebviewer:resources />
<style type="text/css">
.t1 td {
      padding-right: 30px
}
</style>
</head>
<body>
      <%
          pageContext.setAttribute("report", new StiReport());
      %>
      <h1 align="center">My first report!</h1>
      <stiwebviewer:webviewer  report="${report}" />
</body>
</html>
...
```

It will display empty webviewer (because of empty StiReport object). Add taglib directives in the JSP. They will work with custom tags on the page.

Custom Stimulsoft tag

```
...
<%@ taglib uri="http://stimulsoft.com/webviewer" prefix="stiwebviewer"%>
...
```

Add a tag `<stiwebviewer:resources />`, tag used to load necessary resources (css & js) for webviewer, it haven't any attributes, it must be placed inside HTML <head> tag.

13.3.5 Description of Webviewer Tag

index.jsp

```
...
<stiwebviewer:webviewer  report="${report}" />
...
```

This tag contains next attributes:

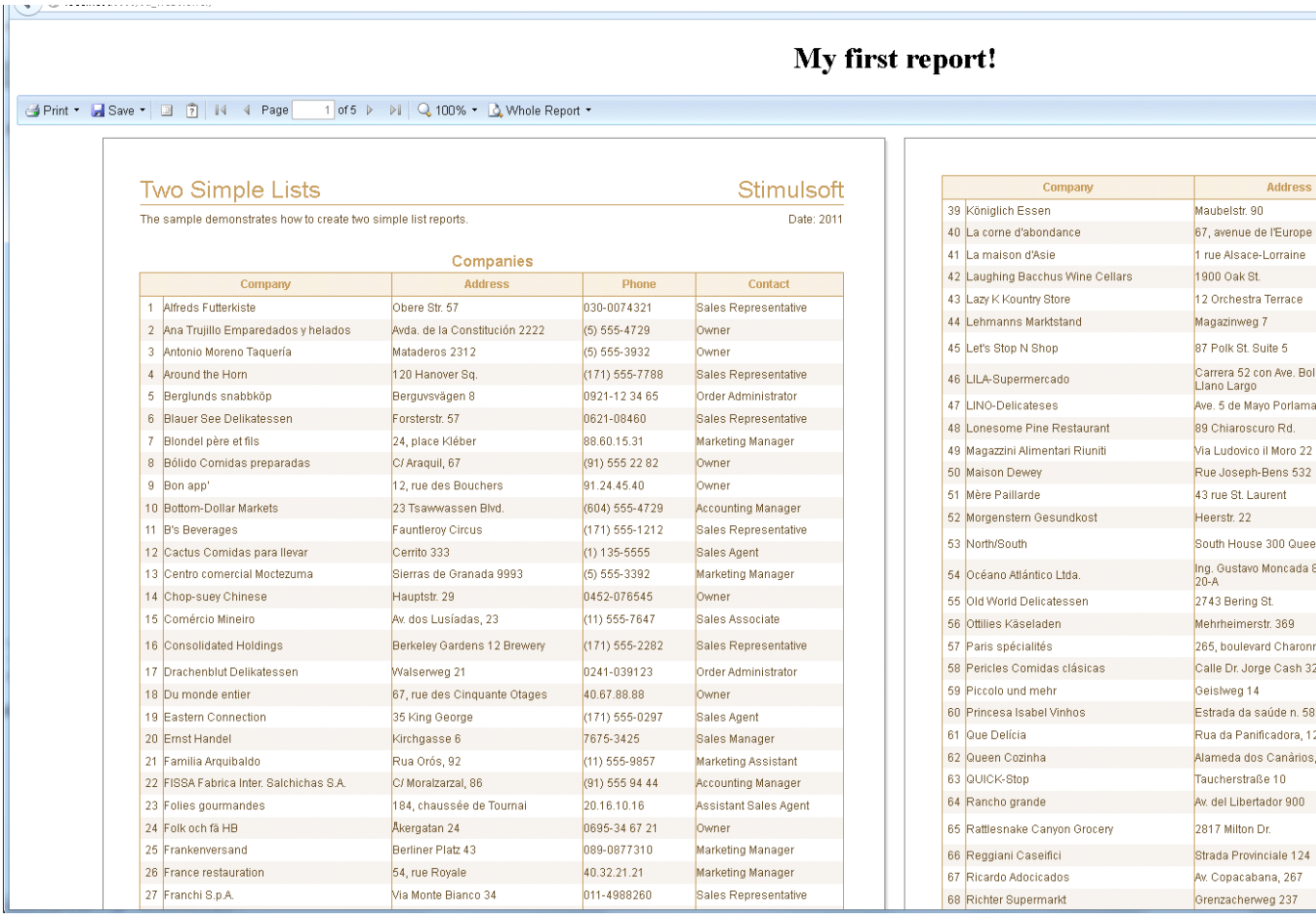
- **report** [required] – StiReport object to display in webviewer;
- **options** [optional] – StiWebViewerOptions object to customize webviewer. If not present – default options are used;
- **viewerID** [optional] – String value identifier of webviewer HTML element. If more than one webviewer present in HTML page each webviewer must have different identifier.

Example of usage webviewer tag (display generated (mdc) report from d:\repots \TwoSimpleLists.mdc with custom parameters)

index.jsp

```
...
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<%@page import="com.stimulsoft.webviewer.enums.StiWebViewerTheme"%>
<%@page import="com.stimulsoft.webviewer.enums.StiPagesViewMode"%>
<%@page import="com.stimulsoft.webviewer.StiWebViewerOptions"%>
<%@page import="java.io.File"%>
<%@page import="com.stimulsoft.report.StiSerializeManager"%>
<%@page import="com.stimulsoft.report.StiReport"%>
<%@ page language="java" contentType="text/html; charset=utf-8"
      pageEncoding="UTF-8"%>
<%@ taglib uri="http://stimulsoft.com/webviewer" prefix="stiwebviewer"%>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Stimulsoft Reports for Java</title>
<stiwebviewer:resources />
</head>
<body>
      <%
          StiReport report = StiSerializeManager.deserializeDocument(new
```

```
File("d:/reports/TwoSimpleLists.mdc")).getReport();
            StiWebViewerOptions options = new StiWebViewerOptions();
            options.setTheme(StiWebViewerTheme.Office2007Blue);
            options.setPagesViewMode(StiPagesViewMode.WholeReport);

            pageContext.setAttribute("report", report);
            pageContext.setAttribute("options", options);
        %>
        <h1 align="center">My first report!</h1>
        <stiwebviewer:webviewer  report="${report}" options="${options}"/>
</body>
</html>
...
```



13.3.6 Options

Webviewer have described below options:

- String viewerID - the viewerID
- StiWebViewerTheme theme - The current visual theme which is used for drawing visual elements of the webviewer.

- String width - The width of webviewer, must ends width % or px, default is "100%".
- String height - The height of webviewer, must ends width % or px, default is "100%".
- StiColor backColor - The background color, default is White.
- int countColumnsParameters - A count columns in parameters Panel, default is 2;

- String localization - A path to the localization file for the web viewer.
- boolean rightToLeft - A value which controls of output objects in the right to left mode, default is **false**.
- boolean scrollbarsMode - A value which indicates that the web viewer will show the report with, default is **false**.
- boolean menuAnimation - A value which indicates that menu animation is enabled, default is **true**.
- StiShowMenuMode menuShowMode - The mode that shows menu of the viewer, default id StiShowMenuMode.Click.
- StiPrintDestination menuPrintDestination - The default mode of the report print destination, default is StiPrintDestination.Default.

- StiPagesViewMode pagesViewMode - The mode of showing a report in the web viewer - one page or the whole report, default is StiPagesViewMode.OnePage.
- int menuZoom - The report showing zoom. The default value is **100**.
- StiContentAlignment pageAlignment - The alignment of the web viewer page, default is StiContentAlignment.Center.
- boolean pageShowShadow - A value which indicates that the shadow of the page will be displayed in the webviewer, default is **true**.
- StiColor pageBorderColor - A color of the page border, default is **Gray**.
- boolean bookmarksVisible - A visibility of the Toolbar of the web viewer, default is **true**;

- boolean bookmarksPrint - A value which allows printing report bookmarks, default is **false**;
- int bookmarksTreeWidth - A width of the bookmarks tree in the web viewer, default is 180.
- boolean toolbarVisible - A value which indicates that report bookmarks will be shown in the web viewer, default is **true**;
- StiColor toolbarBackgroundColor - A color of the toolbar background.
- StiColor toolbarFontColor - A color of the toolbar texts.
- String toolbarFontFamily - A value which indicates which font family will be

used for drawing texts in the webviewer, default is Arial;

- StiContentAlignment toolbarAlignment - The alignment of the web viewer toolbar, default is StiContentAlignment.Default;
- boolean toolbarButtonCaptions - A value which allows displaying or hiding toolbar buttons captions, default is **false**;
- boolean toolbarMenuCaptions - A value which allows displaying or hiding toolbar menu captions, default is **true**;
- boolean showCurrentPageControl - A visibility of the current page control in the toolbar of the web viewer, default is **true**;
- boolean showButtonPrint - A visibility of the Print button in the toolbar of the web viewer, default is **true**;
- boolean showButtonSave - A visibility of the Save button in the toolbar of the web viewer, default is **true**;

- boolean showButtonBookmarks - A visibility of the Parameters button in the toolbar of the web viewer, default is **true**;
- boolean showButtonParameters - A visibility of the Parameters button in the toolbar of the web viewer, default is **true**;
- boolean showButtonFirstPage - A visibility of the First Page button in the toolbar of the web viewer, default is **true**;
- boolean showButtonPreviousPage - A visibility of the Prev Page button in the toolbar of the web viewer, default is **true**;
- boolean showButtonNextPage - A visibility of the Next Page button in the toolbar of the web viewer, default is **true**;
- boolean showButtonLastPage - A visibility of the Last Page button in the toolbar of the web viewer, default is **true**;

- boolean showButtonZoom - A visibility of the Zoom control in the toolbar of the webviewer, default is **true**;
- boolean showButtonViewMode - visibility of the View Mode button in the toolbar of the web viewer, default is **true**;
- boolean showExportDialog - A value which allows to display the export dialog, or to export with the default settings, default is **true**;
- boolean showExportToDocument - A value which indicates that the user can save the report from the web viewer to the report document file, default is **true**;
- boolean showExportToPdf - A value which indicates that the user can save the report from the web viewer to the PDF format, default is **true**;
- boolean showExportToXps - A value which indicates that the user can save the report from the web viewer to the XPS format, default is **true**;

- boolean showExportToHtml - A value which indicates that the user can save the report from the web viewer to the HTML format, default is **true**;
- boolean showExportToText - A value which indicates that the user can save the report from the web viewer to the TEXT format, default is **true**;
- boolean showExportToRtf - A value which indicates that the user can save the report from the web viewer to the RTF format, default is **true**;
- boolean showExportToWord2007 - A value which indicates that the user can save the report from the web viewer to the Word 2007-2010 format, default is **true**;
- boolean showExportToExcel - A value which indicates that the user can save the report from the web viewer to the Excel BIFF format, default is **true**;
- boolean showExportToExcelXml - A value which indicates that the user can save the report from the web viewer to the ExcelXML format, default is **true**;

- boolean showExportToExcel2007 - A value which indicates that the user can save the report from the web viewer to the Excel 2007-2010 format, default is **true**;
- boolean showExportToCsv - A value which indicates that the user can save the report from the web viewer to the CSV format, default is **true**;
- boolean showExportToXml - A value which indicates that the user can save the report from the web viewer to the XML format, default is **true**;
- boolean showExportToSylk - A value which indicates that the user can save the report from the web viewer to the Sylk format, default is **true**;
- boolean showExportToImageBmp - A value which indicates that the user can save the report from the web viewer to the BMP image format, default is **true**;
- boolean showExportToImageJpeg - A value which indicates that the user can save the report from the web viewer to the JPEG image format, default is **true**;

- boolean showExportToImagePcx - A value which indicates that the user can save the report from the web viewer to the PCX image format, default is **true**;
- boolean showExportToImagePng - A value which indicates that the user can save the report from the web viewer to the PNG image format, default is **true**;
- boolean showExportToImageSvg - A value which indicates that the user can save the report from the web viewer to the SVG image format, default is **true**;
- boolean showExportToImageSvgz - A value which indicates that the user can save the report from the web viewer to the SVGZ image format, default is **true**;
- int refreshTimeout - A value wich indicates timeout in minutes for execute dummy request to avoid end session, default is 0 (disabled);

13.3.7 Template JDBC Connections

Template

```
...
jdbc.driver={myDriver};
jdbc.url={myConnectionUrl};
jdbc.username={myUserName };
jdbc.password={ myUserPassword };
...
```

An example for a SQLServer

```
...
jdbc.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver;
jdbc.url= jdbc:sqlserver://[serverName[\instanceName][:portNumber]]
[;property=value[;property=value]];
jdbc.username={myUserName };
jdbc.password={ myUserPassword };
...
```

http://msdn.microsoft.com/en-us/library/ms378428

An example for a Oracle

```
...
jdbc.driver=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@[HOST][:PORT]:SID;
jdbc.username={myUserName };
jdbc.password={ myUserPassword };
...
```

http://www.orafaq.com/wiki/JDBC

An example for a postgresql

```
...
jdbc.driver= org.postgresql.Driver
jdbc.url= jdbc:postgresql://[host]:[port]/[database]
jdbc.username={myUserName };
jdbc.password={ myUserPassword };
...
```

http://jdbc.postgresql.org/documentation/80/connect.html

13.4 HTML5 Designer

The Java designer will be described in this section.

- Installation and Description HTML5 Designer,
- Template JDBC Coonections

13.4.1 Installation and Description HTML5 Designer

Create a sample page with report webdesigner

Create a simple page with a report webdesigner. To do this, put the following libraries into the **WebContent\WEB-INF\lib** directory: stimulsoft.lib.jar, stimulsoft.reports-base.jar, stimulsoft.reports-report.jar, stimulsoft.reports-flex.jar, stimulsoft.reports-web.jar, stimulsoft.reports-webdesigner.jar .

Next, edit **web.xml** it should look like in Listing 1:

web.xml

```
...
<?xml version="1.0" encoding="UTF-8" ?>
  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://
    java.sun.com/xml/ns/javaee/webapp_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee"
    id="WebApp_ID" version="2.5">
  <display-name>sti_webdesigner</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
    <!-- configuration, this parameter indicates the main application
    directory -->
  <servlet>
    <servlet-name>StimulsoftResource</servlet-name>
    <servlet-class>com.stimulsoft.web.servlet.StiWebResourceServlet</
    servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>StimulsoftResource</servlet-name>
    <url-pattern>/stimulsoft_web_resource/*</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>StimulsoftDesignerAction</servlet-name>
    <servlet-
    class
    >com.stimulsoft.webdesigner.servlet.StiWebDesignerActionServlet</
    servlet-class>
  </servlet>
  <servlet-mapping>
```

```
    <servlet-name>StimulsoftDesignerAction</servlet-name>
    <url-pattern>/stimulsoft_webdesigner_action</url-pattern>
  </servlet-mapping>
</web-app>
...
```

Leave unchanged the remaining web.xml blocks, which defines the servlets required for working. Then, edit the index.jsp (Listing 2).

index.jsp

```
...
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<%@page import="java.io.FileOutputStream"%>
<%@page  import="java.io.FileInputStream"%>
<%@page  import="com.stimulsoft.report.utils.data.StiDataColumnsUtil"%>
<%@page
import="com.stimulsoft.report.dictionary.StiDataColumnsCollection"%>
<%@page  import="com.stimulsoft.report.dictionary.StiDataColumn"%>
<%@page  import="com.stimulsoft.report.utils.data.StiSqlField"%>
<%@page
import="com.stimulsoft.report.dictionary.dataSources.StiDataTableSource"%>
<%@page  import="com.stimulsoft.report.utils.data.StiXmlTable"%>
<%@page
import="com.stimulsoft.report.utils.data.StiXmlTableFildsRequest"%>
<%@page  import="com.stimulsoft.webdesigner.StiWebDesigerHandler"%>
<%@page  import="com.stimulsoft.webdesigner.StiWebDesignerOptions"%>
<%@page
  import="com.stimulsoft.report.dictionary.databases.StiXmlDatabase"%>
<%@page  import="java.io.File"%>
<%@page  import="com.stimulsoft.report.StiSerializeManager"%>
<%@page  import="com.stimulsoft.report.StiReport"%>
<%@page  language="java" contentType="text/html; charset=utf-8"
  pageEncoding="UTF-8"%>
<%@taglib uri="http://stimulsoft.com/webdesigner" prefix="stiwebdesigner"%
>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Stimulsoft Webdesigner for Java</title>
  <style type="text/css">
  </style>
</head>
<body>
  <%
    final String reportPath =
    request.getSession().getServletContext().getRealPath("/reports/
    Master-Detail.mrt");
    final String xmlPath =
    request.getSession().getServletContext().getRealPath("/data/
    Demo.xml");
    final String xsdPath =
    request.getSession().getServletContext().getRealPath("/data/
    Demo.xsd");
```

```
final String savePath =
request.getSession().getServletContext().getRealPath("/save/");

StiWebDesignerOptions options = new StiWebDesignerOptions();

StiWebDesigerHandler handler = new StiWebDesigerHandler(){
  public StiReport getEditedReport(HttpServletRequest request){
    try{
      StiReport report = StiSerializeManager.deserializeReport(new
      File(reportPath));
      report.getDictionary().getDatabases().add(new
      StiXmlDatabase("Demo", xsdPath, xmlPath));
      return report;
    } catch (Exception e){
        e.printStackTrace();
    }
    return null;
  }

  public void onOpenReportTemplate(StiReport report,
  HttpServletRequest request){
    report.getDictionary().getDatabases().add(new
    StiXmlDatabase("Demo", xsdPath, xmlPath));
  }

  public void onNewReportTemplate(StiReport report,
  HttpServletRequest request){
    report.getDictionary().getDatabases().add(new
    StiXmlDatabase("Demo", xsdPath, xmlPath));
    try{
      // In new report if you want to use wizard, you must populate
      report with datasources
      StiXmlTableFildsRequest tables =
      StiDataColumnsUtil.parceXSDSchema(new
      FileInputStream(xsdPath));
      for (StiXmlTable table : tables.getTables()){
        StiDataTableSource tableSource = new
        StiDataTableSource("Demo." + table.getName(),
        table.getName(), table.getName());
        tableSource.setColumns(new StiDataColumnsCollection());
        for (StiSqlField field : table.getColumns()){
          StiDataColumn column = new StiDataColumn(field.getName(),
          field.getName(), field.getSystemType());
          tableSource.getColumns().add(column);
        }
        tableSource.setDictionary(report.getDictionary());
        report.getDictionary().getDataSources().add(tableSource);
      }
    } catch (Exception e){
      e.printStackTrace();
    }
  }

  public void onSaveReportTemplate(StiReport report, StiRequestParams
  requestParams, HttpServletRequest request) {
    try {
      FileOutputStream fos = new FileOutputStream(savePath +
```

```
            requestParams.designer.fileName);
            if (requestParams.designer.password != null) {
               StiSerializeManager.serializeReport(report, fos,
               requestParams.designer.password);
            } else {
               StiSerializeManager.serializeReport(report, fos, true);
            }
            fos.close();
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
   };
   pageContext.setAttribute("handler", handler);
   pageContext.setAttribute("options", options);
%>

<stiwebdesigner:webdesigner
   handler="${handler}" options="${options}" />
</body>
</html>
...
```

Add taglib directives in the JSP. They will work with custom tags on the page.

Listing 3. Custom Stimulsoft tag

index.jsp

```
...
<%@ taglib uri="http://stimulsoft.com/webdesigner"
prefix="stiwebdesigner"%>
...
```

Add a tag `<stiwebdesigner:resources />`, tag used to load necessary resources (css & js) for webdesigner, it haven't any attributes, it must be placed inside HTML <head> tag.

Description of webdesigner tag

web.xml

```
...
<stiwebdesigner:webdesigner handler="${handler}" report="${report}" />
...
```

This tag contains next attributes:

- **handler** [required] – com.stimulsoft.webdesigner.StiWebDesigerHandler object to handle webdesigner;
- **options** [optional] – StiWebdesignerOptions object to customize webdesigner. If not present – default options are used;
- **designerID** [optional] – String value identifier of webdesigner HTML element. If more than one webdesigner present in HTML page each webdesigner must have different identifier.

Description of StiWebDesigerHandler

To handle designer events, class that implement **StiWebDesigerHandler** must be created and setup in stiwebdesigner tag. Occured on opening {@link StiReport}.

- **public StiReport getEditedReport(HttpServletRequest request)**;
  Occurred on loading webdesinger. Here must present implementation of loading report and population it with Database\Data sources (if required).

- **public void onOpenReportTemplate(StiReport report, HttpServletRequest request)**;
  Occurred on opening StiReport. Method intended for populate report with Database\Data sources (if required).

- **public void onNewReportTemplate(StiReport report, HttpServletRequest request);**
  Occurred on new StiReport. Method intended for populate report with Database \Data sources (if required).
  In new report if you want to use wizard, you must populate report with datasources.

- **public void onSaveReportTemplate(StiReport report, String reportName, HttpServletRequest request);**
  Occurred on save StiReport. Method must implement saving StiReport.

13.4.2 Template JDBC Coonections

***.mrt**

```
...
jdbc.driver={myDriver};
jdbc.url={myConnectionUrl};
jdbc.username={myUserName };
jdbc.password={ myUserPassword };
...
```

An example for a **SQLServer**:

***.mrt**

```
...
jdbc.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver;
jdbc.url= jdbc:sqlserver://[serverName[\instanceName][:portNumber]]
[;property=value[;property=value]];
jdbc.username={myUserName };
jdbc.password={ myUserPassword };
...
```

http://msdn.microsoft.com/en-us/library/ms378428

An example for a **Oracle**:

***.mrt**

```
...
jdbc.driver=oracle.jdbc.driver.OracleDriver;
jdbc.url=jdbc:oracle:thin:@[HOST][:PORT]:SID;
jdbc.username={myUserName };
jdbc.password={ myUserPassword };
...
```

http://www.orafaq.com/wiki/JDBC

An example for a **POSTGreSQL**:

***.mrt**

```
...
jdbc.driver= org.postgresql.Driver
jdbc.url= jdbc:postgresql://[host]:[port]/[database]
jdbc.username={myUserName };
jdbc.password={ myUserPassword };
...
```

http://jdbc.postgresql.org/documentation/80/connect.html

14 Reports.WPF

YouTube

Watch the video tutorials for working with the components of the Stimulsoft Reports.WPF product. Subscribe to the Stimulsoft channel and be the first to watch new video lessons. Questions and suggestions is recommended be left in the comments to the video.

Samples

See on our website examples for working with the Reports.WPF components. All examples are separate projects, grouped into one solution for Visual Studio. Also, you can view and download specific examples on GitHub.

14.1 Activation

YouTube

Watch videos which show how to activate the WPF components. Subscribe to the Stimulsoft channel to find out about the new video lessons uploaded. Leave your questions and suggestions in the comments to the video.

After purchasing a Stimulsoft product, you need to activate the license for the components you are using. You can do this done in various ways. Below is an example of activating the **WPF** component.

MainWindows.xaml..cs

```
...
public partial class MainWindow : Window
{
   public MainWindow()
   {
      //Activation with using license code
      Stimulsoft.Base.StiLicense.Key = "Your activation code...";

      //Activation with using license file
      Stimulsoft.Base.StiLicense.LoadFromFile("license.key");

      //Activation from byte array
      Stimulsoft.Base.StiLicense.LoadFromBytes(bytes);

      //Activation from stream
      Stimulsoft.Base.StiLicense.LoadFromStream(stream);

      //Activation from assembly
      Stimulsoft.Base.StiLicense.LoadFromEntryAssembly(assembly,
      "stimulsoft-license.key");

      InitializeComponent();
   }
}
...
```

You can get a license key or download a file with a license key in the user's account. To log in to your account, please use the username and password specified when purchasing the product.

14.2 Wpf Viewer

The **StiWpfViewerControl** component is used to view reports in **Reports.WPF**. The component can show a report, zoom, save rendered reports to various formats, print reports, send them to a recipient via Email.

- How to Show Report,
- Dot-Matrix Viewer for Wpf

14.2.1 How to Show Report

Notice

When a report is assigned to a viewer component, it is automatically generated. You only need to call the `Report.Render()` method if you want to perform specific actions with the rendered report before it is displayed in the viewer.

Likewise, when using the compilation mode, you need to call the `Report.Compile()` method only if you have actions to perform with the compiled report before it is built and shown in the viewer.

Just call one method to show a report:

C#

```
...
StiReport report = new StiReport();
report.Load("report.mrt");
report.ShowWithWpf();
...
```

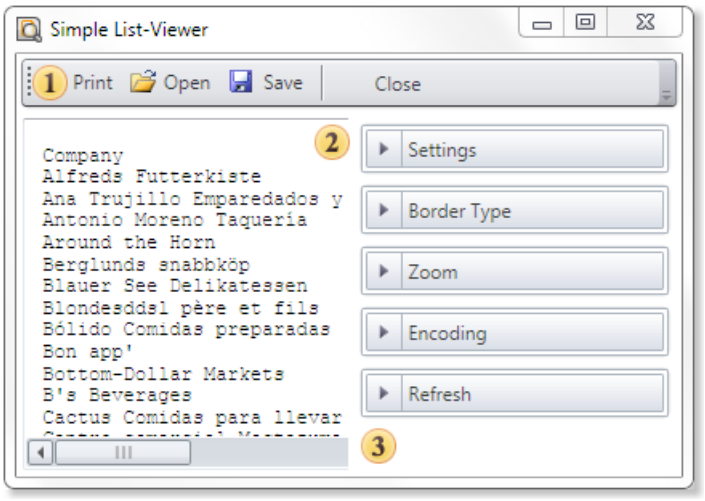
VB.NET

```
...
Dim Report As StiReport = New StiReport()
Report.Load("report.mrt")
Report.ShowWithWpf()
...
```

If the report was not rendered before showing, the **ShowWithWpf** method will render a report using the **RenderWithWpf** method.

14.2.2 Dot-Matrix Mode of Wpf Viewer

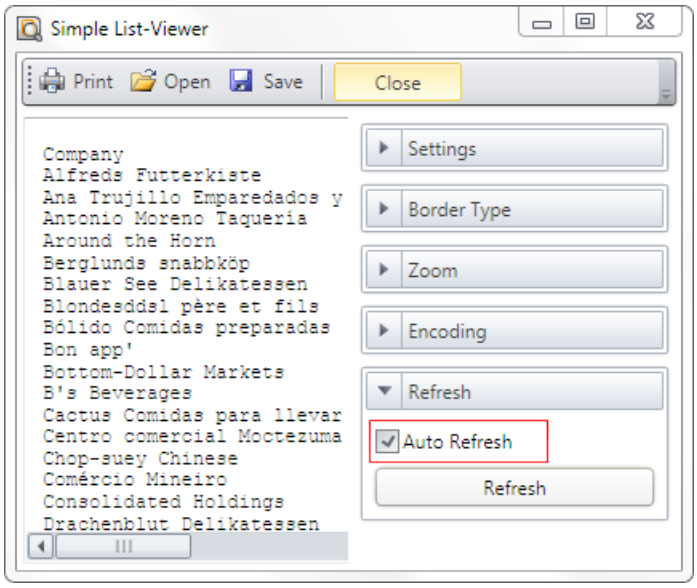
The **Dot-matrix** viewer is designed to preview the report before printing it on dot matrix printer. The Dot matrix printer is used to print only the text and characters of pseudographics. Accordingly the viewer displays only the text and borders of objects as pseudographics characters. The picture below shows the Dot-matrix viewer dialog box:



1. The **Dot-Matrix** viewer toolbar.
2. The panel displays the text of a report
3. The options bar of a report.

Dot-Matrix Viewer Settings for WPF

The Dot-Matrix viewer can be configured from code using static properties. Depending on the value of the static properties in the Dot-matrix viewer, these or that parameters will be specified. For example, the AutoRefresh property. The picture below shows the Dot-matrix viewer dialog box:

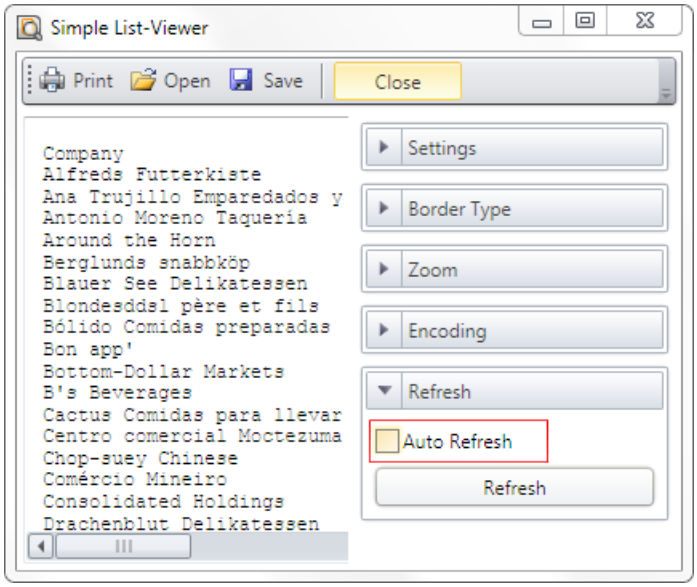


As can be seen on the picture above, the AutoRefresh property is enabled. This means that the AutoRefresh static property of the Dot-matrix viewer is set to true. If the AutoRefresh static property is set to false, then the AutoRefresh property in the Dot-matrix viewer is disabled. Add the following code into the project code:

C#

```
...
StiOptions.Viewer.DotMatrix.AutoRefresh = false;
...
```

Thus, the **AutoRefresh** property will be disabled. The picture below shows the **Dot-matrix** viewer dialog box with disabled auto refresh function:



Most parameters can be set using the static properties.

DotMatrix and Escape Codes

For inserting the escape sequence to text the commands that may look like <#command> should be used as seen in the code sample below:

```
Normal text <#b> Bold text <#/b><#i> Italic text <#/i> Again normal text
```

Also commands of selecting bold, italic or underlined text are automatically inserted depending on the style of the text box font. When printing to matrix printer and exporting to text format these commands are changed on appropriate escape sequences.
The **StiEscapeCodesCollection** is used for this process. It is inherited from the Hashtable class. This is a collection of "key-value" pairs where the key is the command and value is the escape-sequence. For different types of printers different collections with different set of command can be defined. Collections are stored in the **StiOptions.Export.Txt.EscapeCodesCollectionList** static variable. By default, the following collections will be created: "None", "EpsonFX", "Oki ML92/93". The "None" collection is empty and used to output the text without escape codes.

Command/Collection	EpsonFX	Oki ML92/93
b	ESC E	ESC T
/b	ESC F	ESC I
i	ESC 4	
/i	ESC 5	
u	ESC -1	ESC H
/u	ESC -0	ESC D
sup	ESC S0	ESC J
/sup	ESC T	ESC K
sub	ESC S1	ESC L
/sub	ESC T	ESC M
condensed	0x0F	0x1d
/condensed	0x12	0x1e
elite	ESC M	0x1c
pica	ESC P	0x1e
doublewidth	ESC W1	0x1f
/doublewidth	ESC W0	0x1e

It is possible to add new collections of change the existing ones. The selection of the required collection is done by the name. If the collection with the name is not found

then the "None" collection is used. The collection name can be selected from the DotMatrixViewer settings and passed as an option to the exporting and printing methods.

15 Engine

In this chapter you can read questions related to the capabilities of the report engine.

- Data
- Exports
- Report Inheritance
- Scripts
- Right to Left

15.1 Data

YouTube

Watch the video tutorials for working with the components of the Stimulsoft product. Subscribe to the Stimulsoft channel and be the first to watch new video lessons. Questions and suggestions is recommended be left in the comments to the video.

- Business Objects in Net, Web
- Working with OData Using Business Objects

15.1.1 Business Objects in Net, Web

Business Object is a data type, which is a set of objects related to each other, using what it is possible to present data in various structures: tables, lists, arrays, etc. These data can be passed to a reporting tool based on them the report can be rendered. Business Objects are created, registered and passed to the report generator from code.

Filling the Business Objects manually in .NET

This example creates a report with a business object. First we need to create the structure of the business object. Below is a sample code to create a business object class:

C#

```
...
public class MyObject
{
   public class Category
   {
      public int number;
      public int Number
      {
         get
         {
            return number;
         }
      }

      public string name;
      public string Name
      {
         get
         {
            return name;
         }
      }

      public string description;
      public string Description
      {
         get
         {
            return description;
         }
      }
   }

   public Category[] list = null;
   public Category[] List
   {
      get
      {
         return list;
      }
   }
}
...
```

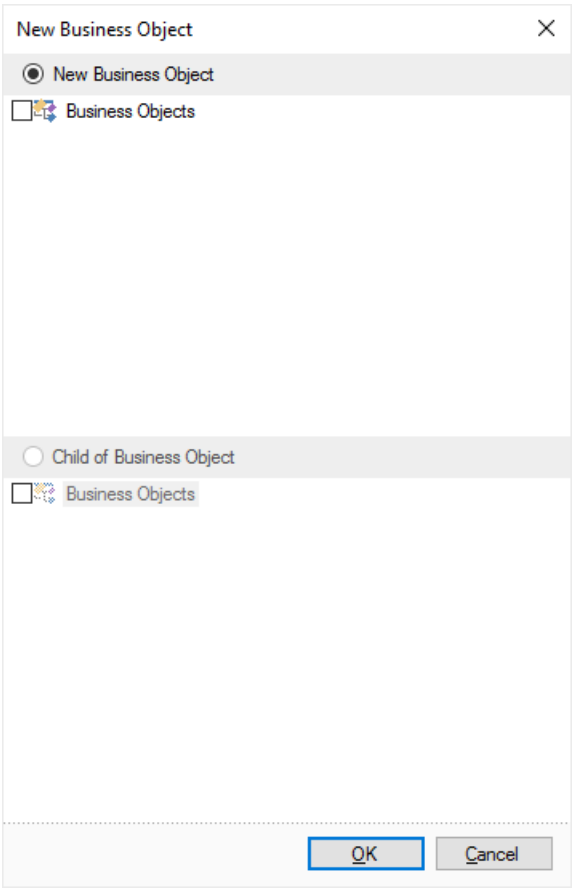
Now, you should populate the business object. Below is an example of code to populate a Business Object is with data:

C#

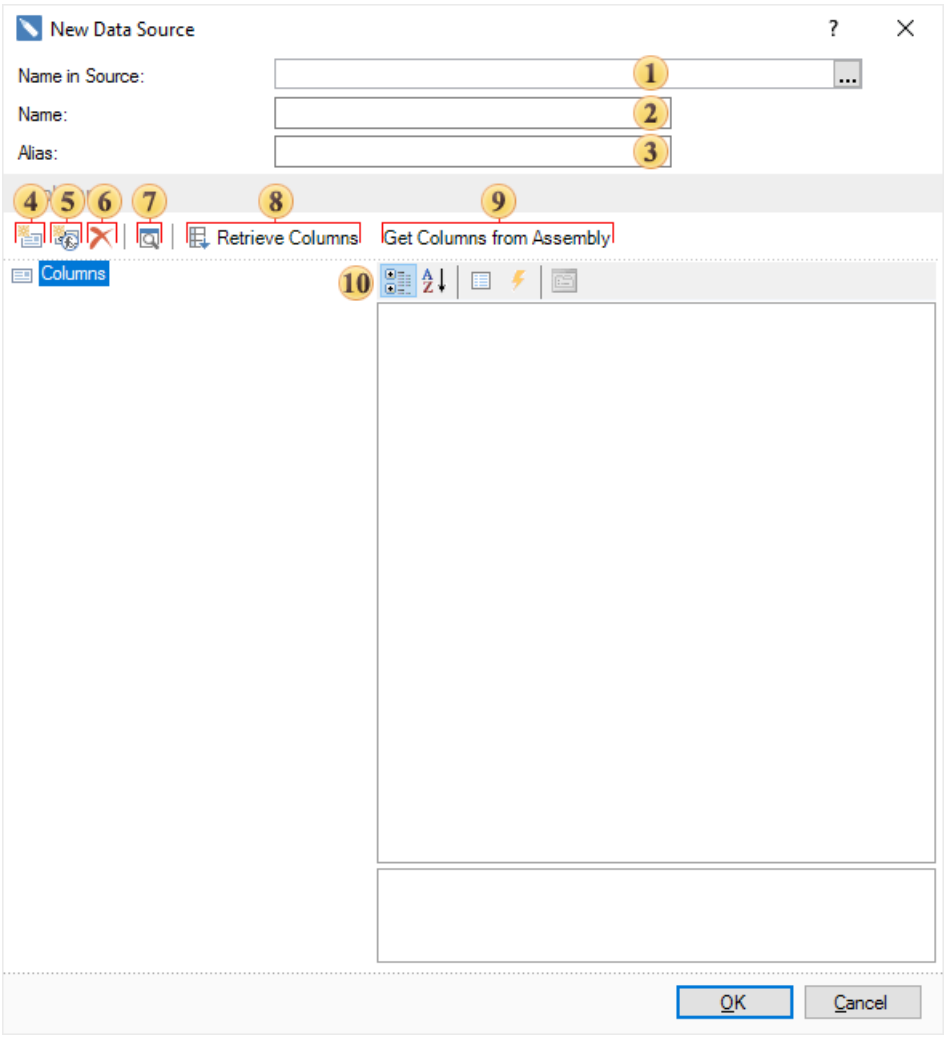
```
...
MyObject obj = new MyObject();
obj.list = new MyObject.Category[2];

MyObject.Category c1 = new MyObject.Category();
c1.number = 1;
c1.name = "Cat1";
c1.description = "desc for n1";

MyObject.Category c2 = new MyObject.Category();
c2.number = 2;
c2.name = "Cat2";
c2.description = "desc for n2";

obj.list[0] = c1;
obj.list[1] = c2;

StiReport mainreport = new StiReport();
mainreport.RegBusinessObject("MyObject", obj);
mainreport.Design();
...
```

Using Business Object in Report

After that, the business object is created, filled with data, registered and passed to the reporting tool. In order to create a report in the designer using business objects, you should create a data description in the report dictionary. To do this, select MyObject (created Business Object) in the report dictionary in and choose New Business Object... from the context menu or the menu New Item. After selecting this command, the window will open a New Business Object, in which you should specify the Child Business Object and select lists of data. The picture below shows the dialog New Business Object.



After you click Ok, you will be shown the second dialog box form of the New Business Object, where you can change the detail business object. The picture below shows the second dialog box form of the New Business Object.



1 The field Category displays the category name. When you create a business object the field is not editable and is purely informative. Also, it may be empty, as in this case.

2 The field Name is used to specify the name of the business object. This field is always available for editing, and, in this case, the name List is used.

3 The field Alias specifies an alias of the business object. This field is always available for editing, and, in this case, the name List is used.

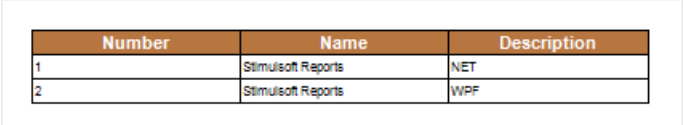
4 The button New Column. Pressing it a new data column will be created in the business object. It should be noted that the data column created this way is a virtual data column and it does not contain actual data.

5 The button New Calculated Column is used to insert a new calculated column into the business object.

6 The button Delete is used to delete selected data columns. If you select a bookmark Columns, then all the columns which are in the tab will be deleted.

7 The button Retrieve Columns is used to get the data column from the business object.
8 The button Get Columns from Assembly will open the dialog Open Assembly, in which you may choose an assembly file. After selecting the file, press the button Open and, from this file, data columns will be extracted, if they are present there.
9 The panel Columns consists of three fields. In these fields show a list of columns, their properties, and a description of these properties.

Press the Ok button once the fields are filled and parameters are specified. After that, in the data dictionary of the report a description of a new business object will be created, which can be used to create reports. The picture below shows a report built using a business object:



Number	Name	Description
1	Stimulsoft Reports	NET
2	Stimulsoft Reports	WPF

Provide the data to business objects from the data source in .NET

Created business objects that are registered and passed to the report generator, but do not contain the actual data are called a description of business objects. Using the description of the business object, you can create a report template (define the structure and design the report), and then, before building, connect the real data and render a report. This is useful if you want to create reports with the same structure and design, but with different data. Create a structural description of the business object first. Below is a sample code to create a business object class:

C#

```
...
public class MyObject
{
   public class Category
   {
      public int categoryID;
      public int CategoryID
      {
         get
         {
            return categoryID;
         }
      }

      public string categoryName;
```

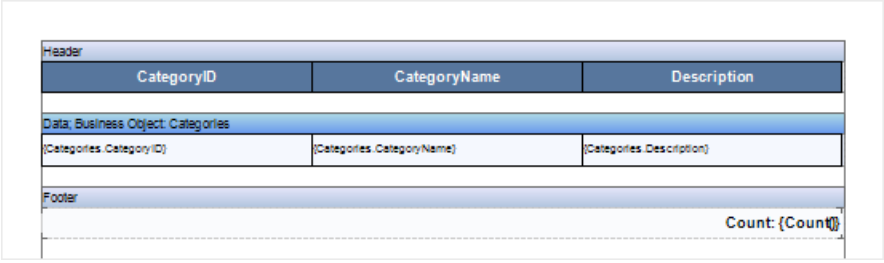
```
    public string CategoryName
    {
      get
      {
        return categoryName;
      }
    }

    public string description;
    public string Description
    {
      get
      {
        return description;
      }
    }

  }

  public Category[] list = null;
  public Category[] Categories
  {
    get
    {
      return list;
    }
  }
}
...
```

You then need to create a new business object class, register and pass it to the report generator. Below is a sample code to create and register a new business object:

C#

```
...
MyObject.Category obj = new MyObject.Category();
int busobjLevel = 1;

StiReport report = new StiReport();
report.RegBusinessObject("Categories", obj);
report.Dictionary.SynchronizeBusinessObjects(busobjLevel);
report.Design();
...
```

Now with help of the created description of the business object, create a report template in the designer. The picture below shows a report template created with the description of the business object:

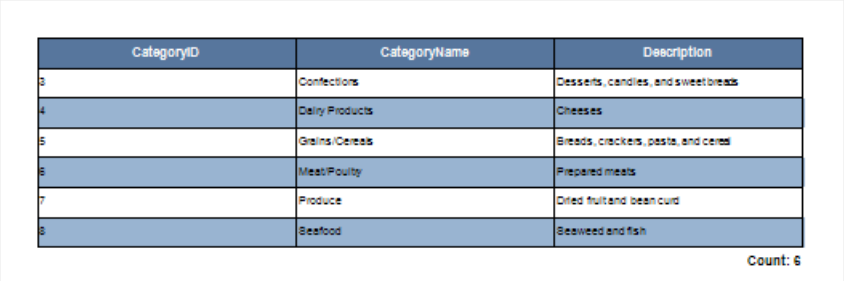


Once a report template is created, you can save it, for example, to the following path D:\\Report.mrt. Because the description of the business object does not contain the actual data, in order to render a report, you will get the real data to business objects, in our example we take the data from the database Northwind. For a start, create a connection to the database in Visual Studio. After that, put the code to obtain data for the business object. Getting real data for the business object occurs immediately before the report. Here is the code to obtain data for the business object:

C#

```
...
int busobjLevel = 1;

StiReport report = new StiReport();
report.Load("D:\\Report.mrt");

using (NorthwindDataContext context = new NorthwindDataContext())
{
   var categories =
      from c in context.Categories
      select new { c.CategoryID, c.CategoryName, c.Description };

   report.RegBusinessObject("Categories", categories);
   report.Show();
}
...
```

After that, the report generator will receive the data for the business object from the specified source, in this case from the database Northwind. Then, the report will be rendered by the existing template. The picture below shows the rendered report:



CategoryID	CategoryName	Description
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

Count: 6

Business objects in Web

Creating, filling, signing and sending business objects to the Web is almost the same as in .NET. First, create a class of the business object that is identical as in .NET. Next, create an object of the business object class, register it manually fill data and pass them. Here are the differences that, instead of the mainreport.Design() method, you should use the StiWebDesigner1.Design(mainreport) method. Also perform synchronization using the mainreport.Dictionary.SynchronizeBusinessObjects(), because in the Web designer it is not possible to create a description of the business object from the data dictionary (the description can only be created from code). Below is a sample code to create, fill, register and pass the business objects:

C#

```
...
MyObject obj = new MyObject();
obj.list = new MyObject.Category[2];

MyObject.Category c1 = new MyObject.Category();
c1.number = 1;
c1.name = "Cat1";
c1.description = "desc for n1";

MyObject.Category c2 = new MyObject.Category();
c2.number = 2;
c2.name = "Cat2";
c2.description = "desc for n2";

obj.list[0] = c1;
obj.list[1] = c2;

int busobjLevel = 1;

StiReport mainreport = new StiReport();
mainreport.RegBusinessObject("MyObject", obj);
mainreport.Dictionary.SynchronizeBusinessObjects(busobjLevel);
StiWebDesigner1.Design(mainreport);
...
```

Just as in .NET, in Web you can create a description of the business objects first, then the report template, and then connect the data source with the real data and render a report. Create a description of the business object. But previously you have to make the class of the business object that is identical to the class of business object in .NET. Here is an example of writing a business object:

C#

```
...
MyObject.Category obj = new MyObject.Category();
int busobjLevel = 1;

StiReport report = new StiReport();
report.RegBusinessObject("Categories", obj);
report.Dictionary.SynchronizeBusinessObjects(busobjLevel);
StiWebDesigner1.Design(report);
...
```

Now with the description created, design a report template identical to .NET. Once a report template is created, you can save it, for example to the following path D:\\Report.mrt. Since the description of the business object does not contain the actual data, in order to build a report, you should get the real data to business objects, in this example, we take the data from the database Northwind. First, create a connection to the database in Visual Studio. After that, write the code to obtain data for the business object. Getting real data for the business object occurs immediately before the report. Here is the code to obtain data for the business object:

C#

```
...
int busobjLevel = 1;

StiReport report = new StiReport();
report.Load("D:\\Report.mrt");

using (NorthwindDataContext context = new NorthwindDataContext())
{
   var categories =
      from c in context.Categories
      select new { c.CategoryID, c.CategoryName, c.Description };

   report.RegBusinessObject("Categories", categories);
   StiWebViewer1.Report = report;
}
...
```

15.1.2 Working with OData Using Business Objects

The protocol **Open Data (OData)** is used to access from different sources, including relational databases, file systems, content management systems and ordinary web sites. **OData** realizes the **CRUD** conception (Create, Read, Update, Delete) in relation to data. In **Visual Studio 2010** and **.NET Framework 4.0** was simplified with support of **OData** using the access technology **Entity Framework**. On the basis of received (using OData protocol) data, it is possible for a user to create reports. Passing data to the report goes through business objects. Let's have an example of retrieving data from the report and passing data to the report:

- Connect the **Stimulsoft** assemblies;
- Add **Service Reference** specifying the address of the entry point to OData-Service. In this case the address is http://services.odata.org/V3/OData/OData.svc;
- Use following code.

C#

```
...
//Connecting to Data Storage
Uri uri = new Uri("http://services.odata.org/V3/OData/OData.svc");
var container = new ServiceReference1.DemoService(uri);

//Creating Query with Selection Parameters
var product = container.Products.Where(p => p.ID < 50).ToList();

//Transffering Data to Report via Business Objects
var report = var StiReport();
report.RegBusinessObject("Products", product);
report.Dictionary.SynchronizeBusinessObjects(2);
report.Design();
...
```

15.2 Report Inheritance

There are two ways of report inheritance:

- Creation of the basic class of a report;
- Creation of the master-report.

In both ways you should create a basic report in the designer that includes all necessary elements. You may add the following components to the basic reports:

- Pages;
- Components;
- Data sources;
- Variables;
- Connections.

To learn more of Basic Approaches.

15.2.1 Basic Approaches

After the report has been created you may either save the report as a special basic class (for this you should use the Save as command) or save the report as a regular report and then use it as a master report. In the first case, you will get the C# or VB.NET class, and will be able to create new reports. For example:

C#

```
...
Reports.Report master = new Reports.Report();
master.RegData(dataSet);
master.Design();
...
```

In order to use the basic report when creating a new report in the designer, you need to add the following string of a code:

C#

```
...
StiReport.ReportType = typeof(Reports.Report);
...
```

Then all new reports will be automatically inherited from the basic class. In the second way you need to use the following code:

C#

```
...
StiReport masterReport = new StiReport();
masterReport.Load("d:\\master-detail.mrt");

StiReport report = new StiReport();
report.RegData(dataSet);

report.MasterReport = masterReport.SaveToString();
report.Design();
...
```

15.3 Right To Left

By default, components are output from left to right. The **Right to Left** property allows changing the mode of showing report items.

- WinForms Viewer,
- WPF Viewer,
- Icons.

15.3.1 WinForms Viewer

There is a capability to change the mode of showing viewer items and order of showing report pages in WinForms. By default, showing of all elements of the viewer and the display order of pages of the report is left to right. How the viewer will look like can depend on the static **RightToLeft** property of the viewer. If the **RightToLeft** property is set to No, then the viewer items and pages of the report in the viewer window are shown from left to right. Code below show how to set the left to right mode of showing viewer items and report pages:

C#

```
...
StiOptions.Viewer.RightToLeft = StiRightToLeftType.No;
...
```

If the **RightToLeft** property is set to **Yes**, then viewer items and report pages in a viewer window are displayed from right to left. The code for setting the right to left mode is shown below:

C#

```
...
StiOptions.Viewer.RightToLeft = StiRightToLeftType.No;
...
```

15.3.2 WPF Designer and Viewer

In the designer and WPF report viewer it is possible to set the mode of showing controls in the right to left direction. By default, all controls are displayed in the left to right order. It is possible to change this. The **FlowDirection** property is used for this. If the property is set to **LeftToRight**, then controls are shown from left to right (see the code below).

XAML

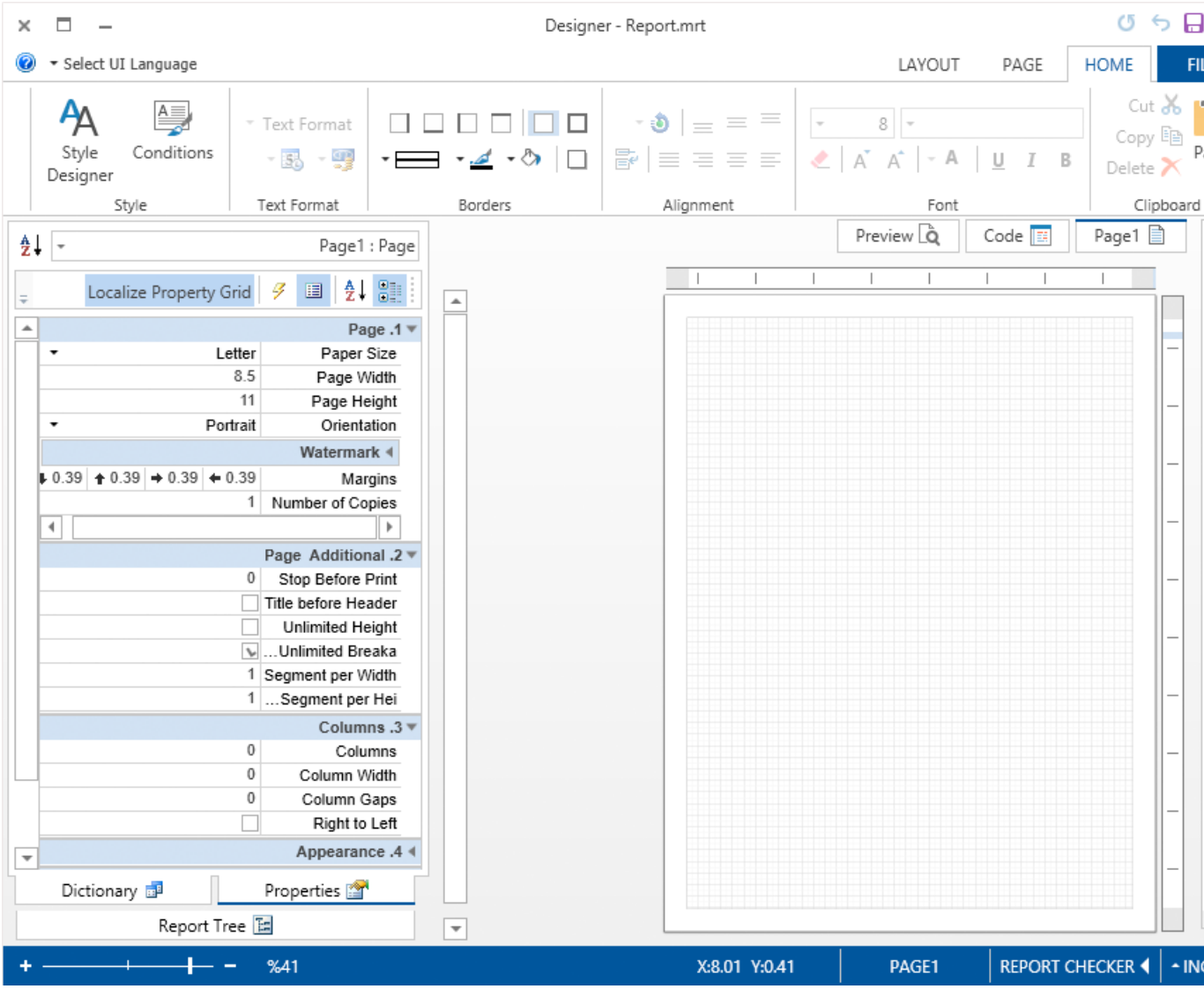
```
...
FlowDirection="LeftToRight"
...
```

The picture below shows the left to right order of showing controls in the designer and viewer. If the **FlowDirection** property is set to **RightToLeft**, then controls are shown in the right to left order. See the code below how to achieve this result.

XAML

```
...
FlowDirection="RightToLeft"
...
```

The picture below shows the right to left order of showing controls in the designer and viewer.



As can be seen from the picture above, the order of showing report pages in the viewer also depends on the value of the **FlowDirection** property.

15.3.3 Icons

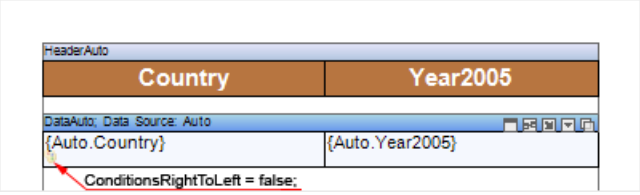
With the help of icons information about the components, settings and tools applied to these components in the reports is displayed. These icons are filtering, conditions and events of the inherited report, dynamic collapsing, dynamic sorting, order, quick information. By default, these icons are displayed in the left to right order in a component, but if necessary, they can be displayed in the right to left order. It is possible using the properties:

- ConditionsRightToLeft,
- EventsRightToLeft,

- InheritedRightToLeft,
- InteractionCollapsingRightToLeft,
- InteractionSortRightToLeft,
- OrderAndQuickInfoRightToLeft,
- FiltersRightToLeft,
- QuickButtonsRightToLeft.

They belong to the **StiOptions.Viewer.Pins** class. Consider these features in more detail:

› The mode of displaying the Conditions icon depends on the value of the **ConditionsRightToLeft** property. For example, if to place a Condition in the DataBand, then the Conditions icon will be displayed by default in the lower left corner of this DataBand, because the **ConditionsRightToLeft** property is set to **false**. The picture below shows an example of a report template with the Conditions icon in the left to right mode:

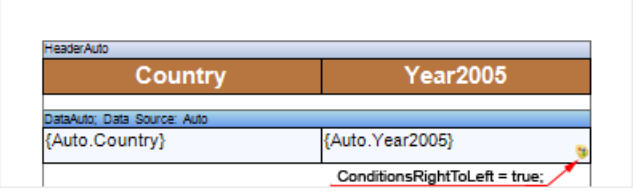


Set the **ConditionsRightToLeft** property to true to change the position of the icon:

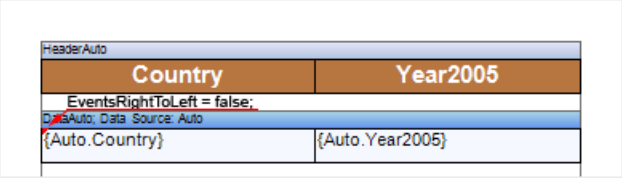
C#

```
...
StiOptions.Viewer.Pins.ConditionsRightToLeft = true;
...
```

And then the icon will be displayed in the "**right to left**" mode, i.e. in the lower right corner of the DataBand. The picture below shows an example of a report template with the Conditions icon in the right to left mode:



› Change the value of the **EventsRightToLeft** property to change the location of the Events icon. For example, if to place an Event in the text component, then the Events bookmark will be displayed by default in the upper left corner of the text component, because the **EventsRightToLeft** property is set to **false**. The picture below shows an example of a report template with the Events icon in the left to right mode:

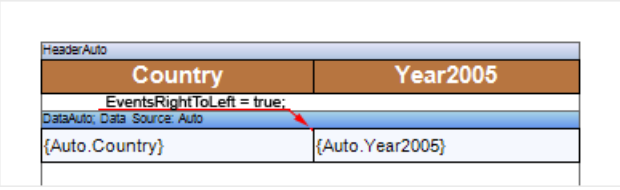


Set the **EventsRightToLeft** property to true to change the location of the **Events** icon.

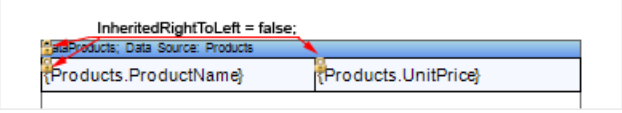
C#

```
...
StiOptions.Viewer.Pins.EventsRightToLeft = true;
...
```

The picture below shows an example of a report template with the **Events** icon in the right to left mode:



› The **InheritedRightToLeft** property is used to change the location of the Inherited icon. By default, this property is set to false, i.e. the icon appears in the left to right mode. The picture below shows an example of a report template with the Inherited icon in the left to right mode:

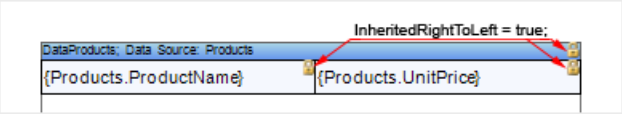


If the **InheritedRightToLeft** property is set to **true**

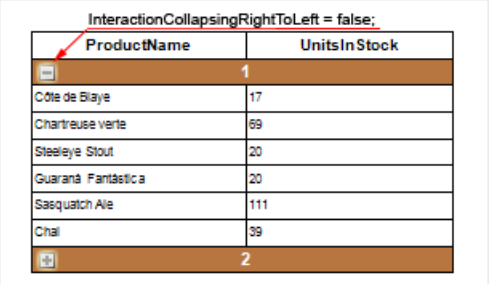
C#

```
...
StiOptions.Viewer.Pins.InheritedRightToLeft = true;
...
```

the Inherited icon will appear in the right to left mode (see the picture below):



› The **InteractionCollapsingRightToLeft** property is used to change the location of the Collapsing icon. By default, this property is set to false, i.e. the icon appears in the left to right mode. The picture below shows an example of a report template with the Collapsing icon in the left to right mode:

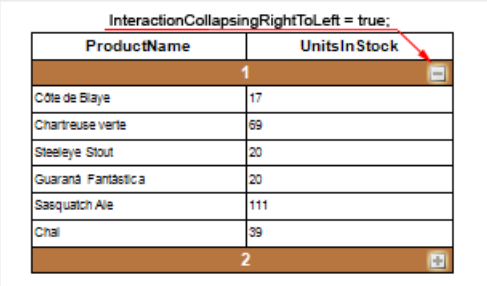


If the **InteractionCollapsingRightToLeft** property is set to **true**

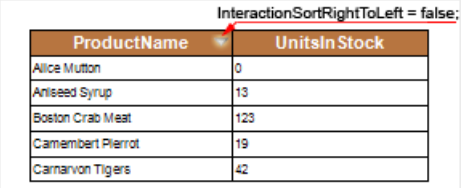
C#

```
...
StiOptions.Viewer.Pins.InteractionCollapsingRightToLeft = true;
...
```

Collapsing icons will appear in the right to left mode (see the picture below):



› The **InteractionSortRightToLeft** property is used to change the location of the **InteractionSort** icon. By default, this property is set to false, i.e. the icon appears in the left to right mode. The picture below shows an example of a report template with the **InteractionSort** icon in the left to right mode:

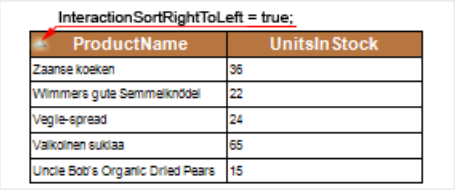


If the **InteractionSortRightToLeft** property is set to **true**

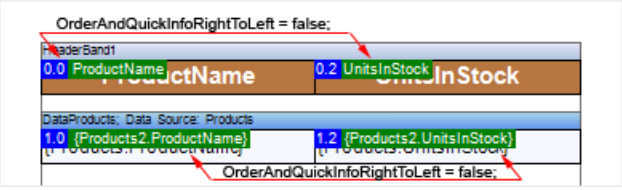
C#

```
...
StiOptions.Viewer.Pins.InteractionSortRightToLeft = true;
...
```

the **InteractionSort** icon will appear in the right to left mode (see the picture below):



› The **OrderAndQuickInfoRightToLeft** property is used to change the location of the **Show Order** icon. By default, this property is set to false, i.e. the icon appears in the left to right mode. The picture below shows an example of a report template with the Show Order icon in the left to right mode:

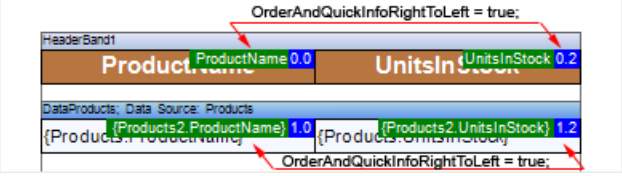


If the **OrderAndQuickInfoRightToLeft** property is set to **true**:

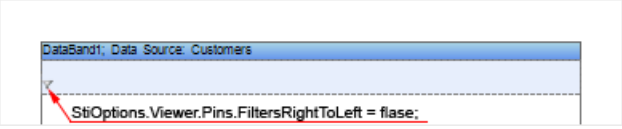
C#

```
...
StiOptions.Viewer.Pins.InteractionSortRightToLeft = true;
...
```

the **Show Order** icon will appear in the right to left mode (see the picture below):



› The **FiltersRightToLeft** property is used to change the location of the Filters icon. By default, this property is set to false, i.e. the icon appears in the left to right mode. The picture below shows an example of a report template with the Filters icon in the left to right mode:

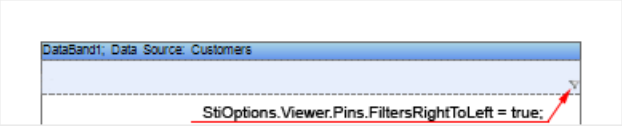


If the **FiltersRightToLeft** property is set to **true**:

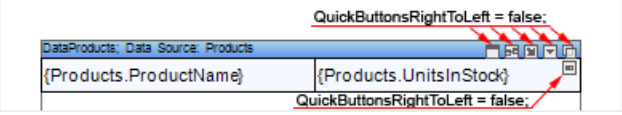
C#

```
...
StiOptions.Viewer.Pins.FiltersRightToLeft = true;
...
```

the **Filters** icon will appear in the right to left mode (see the picture below):



› The **QuickButtonsRightToLeft** property is used to change the location of the **QuickButtons** icon. By default, this property is set to false, i.e. the icon appears in the left to right mode. The picture below shows an example of a report template with the Filters icon in the left to right mode:

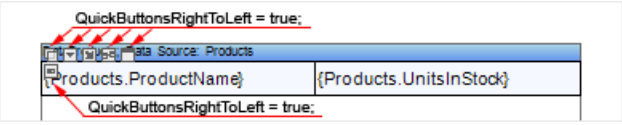


If the **QuickButtonsRightToLeft** property is set to **true**:

C#

```
...
StiOptions.Viewer.Pins.QuickButtonsRightToLeft = true;
...
```

the **QuickButtons** icon will appear in the right to left mode (see the picture below):



15.4 Exports

YouTube

Watch the video tutorials how to export report. Subscribe to the Stimulsoft channel and be the first to watch new video lessons. Questions and suggestions is recommended be left in the comments to the video.

This section describes principles of saving rendered reports to different formats, basic characteristics of methods for export, export optimization guidelines data structure which are used in export methods. Stimulsoft Reports supports great many export formats to save rendered reports. Many clients think that there are too many formats. But when you need to get file of definite format type, write only one string of code and the format is not PDF, HTML or RTF, only Stimulsoft Reports may help. We do not think that too many export formats in our report generator is disadvantage and continually work on adding new formats. The more exports the better, as they say.

- Exports Reports From Code
- Formats with Fixed Page layout
- Web Documents
- Text Formats
- Spreadsheets
- Data
- Images

Available File Formats

The **StiExportFormat** enumeration describes export formats. Brief information of exports is represented below.

Formats	Description
Formats which are used for representing documents and allows for easy viewing and printing	
PDF	export to Adobe PDF.
Microsoft Power Point 2007/2010	export to Microsoft Power Point 2007/2010

XPS	export to Microsoft XPS.
	Web formats
HTML	export to HTML by default. This element duplicates the HTMLTable mode.
HTMLTable	export to HTML using the HTML Table element, to create a report structure.
HTMLSpan	export to HTML using the HTML Span element, to create a report structure.
HTMLDiv	export to HTML using the HTML Div element, to create a report structure.
MHT	export to WebArchive. This format is supported only in Microsoft IE.
	Text formats
Text	export to Text.
RTF	export to Rich Text Format by default. This element duplicates the RTFTable mode.
RTFTable	export to Rich Text Format using the RTF Table element, to create a report structure.
RTFFrame	export to Rich Text Format using the RTF Frame element, to create a report structure.
RTFWinWord	export to Rich Text Format using the Microsoft Word graphic element, to create a report structure.
RTFTabbedText	export to Rich Text Format using the symbols of tabulation, to create a report structure.
Word 2007/2010	export to Microsoft Word 2007/2010. This format is supported starting with Microsoft Office 2007/2010.
ODT	export to the OpenDocument Writer file.
	Spreadsheets
Excel	export to Microsoft Excel. The file is created using the BIFF (Binary Interchange File Format).

ExcelXML	export to Microsoft Excel XML. The file is created using the XML. This format is supported starting with Microsoft Office 2003.
Excel 2007/2010	export to Microsoft Excel 2007/2010. This format is supported starting with Microsoft Office 2007/2010.
ODS	export to OpenDocument Calc file.
Export as Data	
CSV	export to CSV (Comma Separated Value).
DBF	export to dBase/FoxPro.
XML	export to XML as data. This format is a saved DataSet.
DIF	export to DIF (Data Interchange Format).
SYLK	export to SYLK (Symbolic Link).
Export as Image	
ImageGif	export to GIF.
ImageBmp	export to BMP.
ImagePcx	export to PCX.
ImagePng	export to PNG.
ImageTiff	export to TIFF.
ImageJpeg	export to JPEG.
ImageEmf	export to Windows Metafile.

15.4.1 Export Reports From Code

Stimulsoft Reports offers many ways of exporting rendered reports to other formats. Each method of export to other format has several settings. For exporting rendered reports Stimulsoft Reports uses a system of services. This means that all objects which are used in export are represented in the collection of services and when it is necessary to export a report, the report generator searches the appropriate service in the collection of services. There are two ways of exporting rendered formats to other formats from code: using the ExportDocument method of the StiReport class, and using direct creating or getting from a collection of services the required export service.

- ExportDocument Method,
- Export Service.

15.4.1.1 ExportDocument Method

The **ExportDocument** method is a simplified wrapping for report exports. There is no need to get the required export service. All you need is to define the export type, pass parameters of export and define the folder where the file should be saved. For example:

C#

```
...
StiPdfExportSettings pdfSettings = new StiPdfExportSettings();
report.ExportDocument(StiExportFormat.Pdf, "MyReport.Pdf", pdfSettings);
...
```

The following code id used to export reports to PDF. The PDF file will be placed in the MyReport.Pdf. The export parameters can be passed using the **StiPdfExportSettings** object type. This class is described in the description of the PDF format. If there is no need to change export parameters then it is possible to use the short code line:

C#

```
...
report.ExportDocument(StiExportFormat.Pdf, "MyReport.Pdf");
...
```

In this case the export parameters are not passed and the report generator will use parameters which are set by default for each export. Besides, the result of export can be placed in the stream. For example:

C#

```
...
MemoryStream stream = new MemoryStream();
report.ExportDocument(StiExportFormat.Pdf, stream);
...
```

Information

The **ExportDocument** method does not call the Render method automatically. Before calling the **ExportDocument** method it is necessary to render a report or load a previously rendered report.

As you can see, no services in examples were not created and samples contain simple code. All work by creating services and checking parameters can be done using the **ExportDocument** method.

The code above requires connection the following namespaces from assemblies **Stimulsoft.Reports.dll**:

C#

```
...
Stimulsoft.Report
...
```

15.4.1.2 Export Service

The way to create the export service is shown below. See the code:

C#

```
...
StiPdfExportService service = new StiPdfExportService();
StiPdfExportSettings settings = new StiPdfExportSettings();
MemoryStream stream = new MemoryStream();
service.ExportPdf(report, stream, settings);
...
```

If you exported from the WinForms Viewer, then you should notice, than for each export the special form for setting parameters of export is shown. This form can be called from the code. The code below how to do it for the export to the **PDF**:

C#

```
...
service.Export(report, "MyReport.pdf");
...
```

This code will call the dialog form for setting parameters of export. If a user clicks "OK", then the file will be created. If to click the "Cancel" button, then the file creation will be interrupted.

Information

The name of the method for the report export with dialog forms differs from the name of the export method without parameters.

The export service of a report contains yet another ability. The report can be sent via Email. For example:

C#

```
...
bool sendEMail = true;
service.Export(report, "MyReport.pdf", sendEMail);
...
```

This code will call the dialog form for setting parameters of reports, and if a user clicks "OK", then the reporting tool will call the Email client and will create a new Email Email, the exported report will be attached to the Email Email. The code above requires connection of the following names from the **Stimulsoft.Report.dll** assemblies:

C#

```
...
Stimulsoft.Report
Stimulsoft.Report.Export
...
```

All Export Services

The **StiExportFormat** enumeration describes export formats. Brief information of exports is represented below.

Export services to Adobe PDF and Microsoft XPS:

- StiPdfExportService,
- StiXpsExportService

Export services to HTML and MHT:

- StiHtmlExportService
- StiMhtExportService

Export services to text formats:

- StiTxtExportService
- StiRtfExportService
- StiWord2007ExportService
- StiOdtExportService

Export services to Microsoft Excel and Open Document Calc:

- StiExcelXmlExportService
- StiExcelExportService
- StiExcel2007ExportService
- StiOdsExportService

Export services to graphic formats:

- StiBmpExportService
- StiGifExportService
- StiJpegExportService
- StiPcxExportService
- StiPngExportService
- StiTiffExportService
- StiEmfExportService

Export services to data:

- StiCsvExportService
- StiDbfExportService
- StiXmlExportService
- StiDifExportService
- StiSylkExportService

15.4.2 Formats with Fixed Page Layout

Stimulsoft Reports supports two exports with fixed page layout. What is the fixed page layout? This means that all elements of a page can be placed at any part of a page. In this case, if to change a position of one element then other components position will not be changed.

- PDF (Portable Document Format),
- Microsoft Power Point 2007/2010,
- XPS (XML Paper Specification).

15.4.2.1 PDF

PDF (Portable Document Format) – is a file format created by Adobe Systems for document exchange used to create electronic editions using the Adobe Acrobat package. The PDF format is a file text format that is used to publish documents on any platform and OS. The PDF document contains one or more pages. Each page

may contain any components: text, graphic and illustrations, information, that provides navigation across the document.

Export to PDF is based on the "Adobe Portable Document Format, Version 1.3, second edition", using some elements of latest format specifications.

Information

Now, when exporting to PDF, the fields for which **Printable=false** are exported to a separate layer that is not printed in Adobe Acrobat. There is one limitation - layers are not supported in the PDF-A compatibility mode, so the **Printable** property is ignored.

If you print a report to PDF (**Print to PDF**) from a web viewer, then these fields are not exported.

Digital Signature

Digital signature is a requisite of an electronic document used to protect this document from falsification. This document is a result of cryptographic conversion of information using the closed key of the electronic signature and allows identifying the owner of the certificate of the key of the signature. Digital signatures are often used to implement electronic signatures

The **StiPdfExportSettings** class is used to control digital signature. It has the following properties:

C#

```
...
public bool UseDigitalSignature
public bool UseLocalMachineCertificates
public bool GetCertificateFromCryptoUI
public string SubjectNameString
...
```

By default:

C#

```
...
UseDigitalSignature = false;
UseLocalMachineCertificates = true;
GetCertificateFromCryptoUI = true;
SubjectNameString = string.Empty;
...
```

A sample how to use these properties is shown below:

C#

```
...
StiReport report = new StiReport();
report.Load("c:\\test.mrt");
report.Render(false);

StiPdfExportSettings settings = new StiPdfExportSettings();
settings.UseDigitalSignature = true;
settings.GetCertificateFromCryptoUI = false;
settings.UseLocalMachineCertificates = true;
settings.SubjectNameString = "John Smith <johns@google.com>";

report.ExportDocument(StiExportFormat.Pdf, "c:\\test.pdf", settings);
...
```

Encryption

A PDF document can be encoded to protect the content from unauthorized access. A user may set the following parameters of encryption:

- User password;
- Owner password;
- Access permission;
- Key length.

Using the StiPdfExportSettings class it is possible to set the encryption parameters from code. The following properties of this class are used:

C#

```
...
public string PasswordInputUser
public string PasswordInputOwner
public StiUserAccessPrivileges UserAccessPrivileges
public StiPdfEncryptionKeyLength KeyLength
```

```
...
```

The **StiUserAccessPrivileges** enumeration contains the following elements (flags):

- None,
- PrintDocument,
- ModifyContents,
- CopyTextAndGraphics,
- AddOrModifyTextAnnotations,
- All.

The StiPdfEncryptionKeyLength enumeration contains the following elements:

- Bit40,
- Bit128,
- Bit256.

By default the values set as follow:

C#

```
...
PasswordInputUser = string.Empty;
PasswordInputOwner = string.Empty;
UserAccessPrivileges = StiUserAccessPrivileges.All;
KeyLength = StiPdfEncryptionKeyLength.Bit40;
...
```

An example of using:

C#

```
...
StiReport report = new StiReport();
report.Load("c:\\test.mrt");
report.Render(false);

StiPdfExportSettings settings = new StiPdfExportSettings();
settings.PasswordInputUser = "user";
settings.PasswordInputOwner = "owner";
settings.UserAccessPrivileges = StiUserAccessPrivileges.PrintDocument;
settings.KeyLength = StiPdfEncryptionKeyLength.Bit128;

report.ExportDocument(StiExportFormat.Pdf, "c:\\test.pdf", settings);
...
```

Embedded Fonts

By default all embedded fonts are optimized. Characters which are not used in a report are excluded. It allows decreasing the size of a file. But, for correct work of the editable field, the font should be complete. Therefore, for fonts, which are used in editable fields, optimization is not done. This increases the output file size. If Asian languages are used, the file size can be 15-20mb.

If by some reasons the font optimization is not working correct it can be forcibly disabled using the static property:

C#

```
...
StiOptions.Export.Pdf.ReduceFontFileSize = false;
...
```

Editable Fields

To enable the export of editable fields it is necessary to set the static property

C#

```
...
StiOptions.Export.Pdf.AllowEditablePdf = true;
...
```

Editable fields in the PDF-file has two conditions:

- **First** – a condition before editing, it is shown when opening the file. This condition corresponds to the type of a text box in the preview.
- **Second** - the type in the mode of field editing, and after editing. In this condition it is impossible to set the vertical alignment of the text (always Top) and some parameters of a font. Therefore, after editing a field, even if the contents is not changed, the type of this field can be change.

If it is necessary to have the **MultiLine** editable field, then it is necessary to set the **WordWrap** property of the text box to **true**.

Export Settings

The export parameters of the PDF export are described in the **StiPdfExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ImageQuality	float	image quality; may have values from 0.0 (the lowest quality) to 1.0 (the highest quality); by default 0.75
ImageResolution	float	image resolution dpi; can take any value, by default 100
EmbeddedFonts	bool	embed font files into the PDF file; is true then all fonts are embedded into the file and this file will have the same look on any computer (there is no need to embed additional fonts); if false then fonts are not embedded; by default true
StandardPdfFonts	bool	use standard fonts which are embedded in Adobe Acrobat Reader and there is no need to embed them into the file; all fonts are changed on standard fonts (Courier, Helvetica, Times-Roman); by default false
Compressed	bool	compress the PDF file; decreases the file size by compressing the text information (images are always compressed); by default true
UseUnicode	bool	writes a text in the

		Unicode; if false then 190 symbols can be written, and a lot of problems with native language symbols may occur; if true then any symbols can be used; by default true
ExportRtfTextAsImage	bool	export RichText objects as images; if false then export tries to convert RichText objects into PDF primitives; if true the RichText is written as an image; by default false
PasswordInputUser	string	user password (see Encryption); by default empty string
PasswordInputOwner	string	owner password (see Encryption); by default empty string
UserAccessPrivileges	enum	user access privileges (see Encryption); by default StiUserAccessPrivileges.All
KeyLength	enum	key length (see Encryption); by default StiPdfEncryptionKeyLength .Bit40
UseDigitalSignature	bool	use digital signature of a document; by default false
GetCertificateFromCryptoUI	bool	get the certificate from the Crypto interface; if false then a certificate is searched by certificate identifier without using the interface; by default true

SubjectNameString	string	certificate identifier; this is a certificate name (empty string) or a part of a name (substring); by default empty string
UseLocalMachineCertificates	bool	search certificates on the local machine; if false then certificate is searched in the store of the current user; by default false
CreatorString	string	the "Creator" field in the document description (application name that created the original file); if it is not set (empty string) then the StiOptions.Export.Pdf static property is used. CreatorString; by default empty string
KeywordsString	string	the "Keywords" field in the document description (application name that created the original file); if it is not set (empty string) then the StiOptions.Export.Pdf.KeywordsString static property is used; by default empty string
ImageCompressionMethod	enum	image compression method - JPEG (with quality loss) or Flate (without quality loss); by default StiPdfImageCompressionMethod.Jpeg

DitheringType	enum	using this property as a format of monochrome image (with dithering and without) is defined

If the **UseUnicode** is used then, for Acrobat Reader 5.0, it is necessary to use the **Embedded fonts = true**. If the **UseUnicode** + encoding is used, then it is necessary to use the **Embedded fonts = true**.

The following should be done to compress file:

- enable **Compressed**;
- disable **Embedded fonts**;
- if Embedded fonts is required then enable the **ReduceFontFileSize**.

Static Options

Except the **StiPdfExportSettings** class, parameters of export to PDF are also set using the static properties. All properties are described in the table below. To access to export properties it is necessary to add the **StiOptions.Export.Pdf...** prefix. For example, **StiOptions.Export.Pdf.DivideSegmentPages**.

Name	Type	Description
DivideSegmentPages	bool	divide segmented pages into separate pages; if false then are exported "as is" without dividing; by default true
ConvertDigitsToArabic	bool	convert ASCII digits Arabic; by default false
ArabicDigitsType	enum	Select Arabic digits type; by default Standard
ReduceFontFileSize	bool	optimize embedded fonts - eliminate symbols which are not met in a report; if false then fonts are not

Name	Type	Description
		changed; by default true
AllowEditablePdf	bool	export editable fonts as editable PDF objects (in this case fonts which are used in editable fields are not optimized); if false then editable fields are exported as simple text; by default false
AllowImageComparer	bool	use the image comparer, e.g. replace image duplicates (see Common export settings); if false then an image is exported "as is"; by default true
AllowImageTransparency	bool	use transparency in export images; by default true
AllowInheritedPageResources	bool	store resources of pages in the parent dictionary and inherit from it; if false then resources of pages are specified in each page; this property is critical for some programs of PDF files processing; by default true
AllowExtGState	bool	use command to control transparency when creating a document; if false then commands are not used; this property is critical for some programs of PDF files processing; by default true
CreatorString	string	the "Creator" field in document description

Name	Type	Description
		(application name, which created the original document); by default the "Stimulsoft Reports.NET" string
KeywordsString	string	the "Keywords" field in document description (keywords); by default empty string

15.4.2.2 ZUGFeRD

Stimulsoft Company has added support for the format of electronic invoices - ZUGFeRD.

Invoices in the **ZUGFeRD** format pass both human-readable invoices and its structured machine-readable XML based representation. Human-readable representation is encoded in the form of one or more PDF pages of the PDF/A format. XML based representation is embedded in the PDF document as an object in accordance with the specifications of the PDF/A-3 format. In other words, the invoice of the **ZUGFeRD** format contains two separate representations - human-readable in the PDF/A-3 format that is used as a container for the XML representation.

At this moment, you can use the **ZUGFeRD** format only from code. To do this, select the desired format option (V1 or V2) using the **ZUGFeRDComplianceMode** option in the export settings to PDF, using the **ZUGFeRDConformanceLevel** option, select the desired **Conformance Level**, and also load the pre-prepared XML file into the **ZUGFeRDInvoiceData** property. This will automatically add the file to the **EmbeddedFiles** collection with the standard **FileName** and **Description**. If you need to use another **Description**, you yourself can add the file to the **EmbeddedFiles** collection with the desired **FileName** and **Description**.

Pay attention:
- The name of the XML file in different versions of the standard is case-sensitive.
- ConformanceLevel COMFORT in ZUGFeRD 2.0 replaced by EN 16931.

The following is a sample code for exporting a report using the **ZUGFeRD** format:

C#

```
...
FileStream fileStream = new FileStream(@"d:\test.pdf", FileMode.Create);
byte[] buf = File.ReadAllBytes(@"d:\ZUGFeRD-invoice.xml");

//for ZUGFeRD 1.0
var pdfExportSettings = new StiPdfExportSettings()
{
   ZUGFeRDComplianceMode = StiPdfZUGFeRDComplianceMode.V1,
   ZUGFeRDInvoiceData = buf,
   ZUGFeRDConformanceLevel = "COMFORT"     //BASIC, COMFORT, EXTENDED
};

//for ZUGFeRD 2.0
var pdfExportSettings = new StiPdfExportSettings()
{
   ZUGFeRDComplianceMode = StiPdfZUGFeRDComplianceMode.V2,
   ZUGFeRDInvoiceData = buf,
   ZUGFeRDConformanceLevel = "EN 16931"    //BASIC, EN 16931, EXTENDED
};

report.ExportDocument(StiExportFormat.Pdf, fileStream, pdfExportSettings);
fileStream.Close();
...
```

Below is an example of code for exporting a report using the **ZUGFeRD** format, if you need to use an alternative **Description** for the XML file you should to the following:

C#

```
...
FileStream fileStream = new FileStream(@"d:\test.pdf", FileMode.Create);
byte[] buf = File.ReadAllBytes(@"d:\ZUGFeRD-invoice.xml");

//for ZUGFeRD 1.0, Custom settings
var pdfExportSettings = new StiPdfExportSettings();
pdfExportSettings.ZUGFeRDComplianceMode = StiPdfZUGFeRDComplianceMode.V1;
pdfExportSettings.EmbeddedFiles.Add(new StiPdfEmbeddedFileData("ZUGFeRD-
invoice.xml", "ZUGFeRD Invoice", buf));
pdfExportSettings.ZUGFeRDConformanceLevel = "COMFORT";

//for ZUGFeRD 2.0, Custom settings
var pdfExportSettings = new StiPdfExportSettings();
pdfExportSettings.ZUGFeRDComplianceMode = StiPdfZUGFeRDComplianceMode.V2;
pdfExportSettings.EmbeddedFiles.Add(new StiPdfEmbeddedFileData("zugferd-
invoice.xml", "ZUGFeRD Invoice", buf));
pdfExportSettings.ZUGFeRDConformanceLevel = "EN 16931";

report.ExportDocument(StiExportFormat.Pdf, fileStream, pdfExportSettings);
fileStream.Close();
...
```

15.4.2.3 Special Features of PDF/A

PDF/A is an ISO-standardized version of the PDF specialized for use in the archiving and long-term preservation of electronic documents. The PDF/A standard does not define an archiving strategy or the goals of an archiving system. It identifies a "profile" for electronic documents that ensures the documents can be reproduced exactly the same way using various software in years to come. A key element to this reproducibility is the requirement for PDF/A documents to be 100% self-contained. All of the information necessary for displaying the document in the same manner is embedded in the file. This includes, but is not limited to, all content (text, raster images and vector graphics), fonts, and color information. A PDF/A document is not permitted to be reliant on information from external sources.

When the "PDF/A Compliance" mode is enabled, the exported report is subjected to the following restrictions:

- no transparency is allowed (PDF/A-1 only);
- no hyperlinks can be used;
- tooltips are not permitted;
- fonts must always be included;
- document **encryption** is not utilized.

When exporting a report to a PDF document using the PDF/A standard, certain errors may occur. In this chapter, we will discuss some of them.

The "Font not embedded" error

The PDF-A standard mandates the inclusion of fonts within the PDF file. Consequently, when the "PDF/A Compliance" mode is enabled, the fonts will be automatically included in the PDF export, regardless of the "Embedded fonts" parameter in the export settings. However, in applications utilizing .NET Core and JS components, the report engine can only access fonts that have been loaded into the `StiFontCollection`. Therefore, if a particular font has not been loaded, it will not be incorporated into the resulting PDF file. Consequently, the PDF/A compliance check will display an error indicating that the font has not been embedded.

HomeController.cs

```
StiFontCollection.AddFontFile(StiNetCoreHelper.MapPath(this, "Reports/
Font.ttf"));
```

The "Width information for rendered glyphs is inconsistent" error
("Glyph widths in the font dictionary are not consistent with embedded font program widths")
The error message "Width information for rendered glyphs is inconsistent" or "Glyph widths in the font dictionary are not consistent with embedded font program widths" typically occurs when dealing with TrueType or OpenType fonts. These fonts store the contours of symbols as points connected by straight lines or arcs, making them vector fonts. The font file itself contains information about the font, such as:

- Name;
- Coding;
- And a "Widths" table that specifies the width of all the symbols used. This table serves as a cache for viewers, enabling them to quickly process text without calculating the width of characters from the font data each time.

The PDF/A compliance verification process involves multiple stages, one of which is checking the consistency between the data in the Widths table and the actual font data. During this step, the verification utility calculates the width of characters and compares it with the values recorded in the Widths table. According to the standards, a deviation of up to 0.1% is permissible. However, a problem arises due to different graphic libraries calculating character widths differently. As a result, the calculated values may differ, sometimes by 1-2%, which exceeds the allowed deviation of 0.1%. Additionally, different verification utilities may use varying methods to calculate character widths, leading to discrepancies and errors flagged in different files.

Notice

In one instance from our experience, we encountered a scenario where a PDF file underwent verification using Adobe Acrobat Pro successfully, but failed when tested using an online utility. Our graphics library determined the width of a specific character to be 554, a value that was acceptable to Adobe Acrobat. However, upon manually adjusting the number to 553, the file successfully passed the online utility's verification but began to encounter issues with Adobe Acrobat Pro.

While there is no definitive solution to this problem, it is crucial to consider this

particular peculiarity when encountering such an error.

15.4.2.4 Microsoft Power Point

Notice

For desktop versions, there are no specific size restrictions; the size of the opened file is limited by the free memory of the computer. For web versions, there are limitations: the timeout for download/save operations is set to 1 minute, and usually, files larger than 1 gigabyte cannot be saved.

Microsoft PowerPoint is a presentation program developed by Microsoft. It is a part of the Microsoft Office suite. PowerPoint presentations consist of a number of individual pages or "slides". Slides may contain text, graphics, movies, and other objects, which may be arranged on the slide. The presentation can be printed, displayed on a PC, or navigated through at the command of the presenter. In Stimulsoft Reports each report page corresponds to one slide.

Export Settings

The export parameters of the PPT export are described in the **StiPpt2007ExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ImageQuality	float	image quality; may have values from 0.0 (the lowest quality) to 1.0 (the highest quality); by default 0.75
ImageResolution	float	image resolution dpi; can take any value, by default 100

Static Options

Besides the **StiPpt2007ExportSettings** class, the parameters of the export to PPT are also set using the static properties. All properties are described in the table below. To access to export properties it is necessary to add the **StiOptions.Export.Ppt2007...** prefix. For example, **StiOptions.Export.Ppt2007.ReduceFontFileSize**.

Name	Type	Description
AllowImageComparer	bool	use the image comparer, e.g. replace image duplicates (see Common export settings); if false then an image is exported "as is"; by default true

15.4.2.5 XPS

XPS (XML Paper Specification) is the open graphic format of fixed page layout on the base XML (more precisely XAML-based) used to store printed output as electronic documents. This format was developed by Microsoft as alternative to the PDF format.

The XPS document format consists of structured XML markup that defines the layout of a document and the visual appearance of each page, along with rendering rules for distributing, archiving, rendering, processing and printing the documents. The markup language for XPS is a subset of XAML that allows including vector graphic elements, using XAML to mark up the WPF-primitives.

The XPS is a ZIP-archive that contains the files which make up the document. The archive includes page mark up (one file per each page of a document), text, embedded fonts, raster images, 2D vector graphics and other information.

Export Settings

The export parameters of the XPS export are described in the StiXpsExportSettings class. The description of all class properties are in the table below.

Name	Type	Description
ImageQuality	float	image quality; may have

Name	Type	Description
		values from 0.0 (the lowest quality) to 1.0 (the highest quality); by default 0.75
ImageResolution	float	image resolution dpi; can take any value, by default 100

Static Options

Besides the **StiXpsExportSettings** class, the parameters of export to XPS are also set using the static properties. All properties are described in the table below. To access to export properties it is necessary to add the **StiOptions.Export.Xps**... prefix. For example, **StiOptions.Export.Xps.ReduceFontFileSize**.

Name	Type	Description
ReduceFontFileSize	bool	optimize embedded fonts - exclude symbols which are not met in a report; if false then fonts are not changed; by default true
AllowImageComparer	bool	use the image comparer, e.g. replace image duplicates (see Common export settings); if false then an image is exported "as is"; by default true
AllowImageTransparency	bool	use the transparency in exporting images; by default true

15.4.3 Web Documents

There are two formats **HTML** (HyperText Markup Language), **HTML5** and **MHTML** (MIME HTML) are described in this chapter. The first and second formats are used for web page layout. The second format is a web page archive format used to bind resources together with the HTML code into a single file.

- HTML (HyperText Markup Language),
- MHTML (MIME HTML).

15.4.3.1 HTML

HTML (HyperText Markup Language) is the predominant markup language for Web pages. The majority of web pages are created using the HTML language. The HTML language is interpreted by browser and shown as a document. HTML is a tag language of the document layout. It provides a means to describe the structure of text-based information in a document by denoting certain text as links, headings, paragraphs, lists, etc. Elements are the basic structure for HTML markup. Elements have two basic properties: attributes and content. Each attribute and each element's content has certain restrictions that must be followed for a HTML document to be considered valid. An element usually has a start tag (e.g. <element-name>) and an end tag (e.g. </element-name>).

Export Settings

The export parameters of the HTML export are described in the **StiHtmlExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
Zoom	double	zoom factor. By default a value is 1.0 what is equal 100% in export settings window
ImageFormat	ImageFormat	sets an image export format; by default ImageFormat.Png
ExportMode	StiHtmlExportMode	sets the mode of the document export using the div, span or table elements; by default StiHtmlExportMode.Table
ExportQuality	StiHtmlExportQuality	export quality of components size; by

Name	Type	Description
		default StiHtmlExportQuality.High
Encoding	Encoding	file encoding; by default Encoding.UTF8
AddPageBreaks	bool	add page breaks; by default false
BookmarksTreeWidth	int	bookmark column width, in pixels; by default 150
ExportBookmarksMode	StiHtmlExportBookmarksMode	a mode the export a document with bookmarks; by default StiHtmlExportBookmarksMode.All
UseStylesTable	bool	use the Styles table; if false then the style table is empty and all properties of each component will described directly in the style of this component; by default true

Static Options

Except the **StiHtmlExportSettings** class parameters of export to HTML are set using the static properties. All properties are described in the table below. To access to export properties it is necessary to add the **StiOptions.Export.Html...** prefix. For example, **StiOptions.Export.Html.ConvertDigitsToArabic**.

Name	Type	Description
ConvertDigitsToArabic	bool	convert ASCII digits to Arabic digits; by default false
ArabicDigitsType	enum	select Arabic digits type; by default Standard

Name	Type	Description
AllowImageComparer	bool	use the image comparer, e.g. replace image duplicates (see Common export settings); if false then an image is exported "as is"; by default true
ForceWysiwygWordwrap	bool	Forcibly break text in rows as well as in the WYSIWYG mode; by default - false
ReplaceSpecialCharacters	bool	change symbols '<', '>', '&', ' " ' on < > & " by default true

15.4.3.2 MHT

MHTML (MIME HTML) is a web page archive format used to bind resources which are typically represented by external links (such as images, Flash animations, Java applets, audio files) together with HTML code into a single file. This file is a web archive and has the «.mht» extension. The content of a file is written as an Email message using the MIME standard: in the beginning of a file the HTML file is written. Then all resources in the base64 encoding with headers are written. Internet Explorer, Opera, Microsoft Word can work with the MHTML format.

Export Settings

The export parameters of the MHT export are described in the **StiMhtExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
Zoom	double	zoom factor. By default a value is 1.0 what is equal 100% in export settings window
ImageFormat	ImageFormat	sets an image export format; by default

Name	Type	Description
		ImageFormat.Png
ExportMode	StiHtmlExportMode	sets the mode of the document export using the div, span or table elements; by default StiHtmlExportMode.Table
ExportQuality	StiHtmlExportQuality	export quality of components size; by default StiHtmlExportQuality.High
Encoding	Encoding	file encoding; by default Encoding.UTF8
AddPageBreaks	bool	add page breaks; by default false
BookmarksTreeWidth	int	bookmark column width, in pixels; by default 150
ExportBookmarksMode	StiHtmlExportBookmarksMode	a mode the export a document with bookmarks; by default StiHtmlExportBookmarksMode.All

15.4.4 Text Formats

This chapter describes exports formats of text files. In other words the files which are used to create text documents.

- TXT,
- RTF,
- Word 2007/2010,
- ODT.

15.4.4.1 TXT

Text file (TXT) is a kind of computer file that is structured as a sequence of lines. A text file exists within a computer file system. The end of a text file is often denoted by placing one or more special characters, known as an end-of-file marker, after the last line in a text file.

Text files are commonly used for storage of information.

Export Settings

The export parameters of the TXT export are described in the **StiTxtExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
Encoding	Encoding	text file coding; by default Encoding.UTF8
DrawBorder	bool	draw border lines; if false, then borders are not drawn; by default true
BorderType	StiTxtBorderType	a type of a border line; by default StiTxtBorderType.UnicodeSingle
KillSpaceLines	bool	remove all empty rows of a text; by default true
KillSpaceGraphLines	bool	remove all rows of a text which contains only blank spaces and symbols of the vertical border; by default true
PutFeedPageCode	bool	put feed page code after each page; by default true
CutLongLines	bool	cut too long lines of a text which cannot be placed in text boxes; by default true
ZoomX	float	horizontal zoom factor by X axis. By default a value is 1.0 what is equal 100% in export settings window
ZoomY	float	vertical zoom factor by Y

Name	Type	Description
		axis. By default a value is 1.0 what is equal 100% in export settings window

Static Options

Static properties of export to TXT are shown on the table below. To access to export properties it is necessary to add the **StiOptions.Export.Txt...** prefix. For example, **StiOptions.Export.Txt.ColumnWidths**.

Name	Type	Description
ColumnWidths	string	forcibly set the text column width (the list through the semicolon); if a row is empty then the column width is not changed; by default empty string
UseFullTextBoxWidth	bool	use all text box width for a text; in this case if the text is laid on a border, then the border is erased in this place; if false, then when drawing a text, one blank space on the right is always left for correct drawing borders; by default false
UseOldMode	bool	use the old mode of the text export; this property is left for keeping compatibility with old versions; by default false
UseFullVerticalBorder	bool	draw vertical border outside a cell. So a border

Name	Type	Description
		will never be closed with a text; by default true
UseFullHorizontalBorder	bool	draw horizontal border outside a cell. So a border will never be closed with a text; by default true
CheckBoxTextForTrue	string	a text that shows the checkbox true status ; by default "+"
CheckBoxTextForFalse	string	a text that shows the check false status; by default "-"

15.4.4.2 RTF

Rich Text Format (RTF) is a free document file format developed by Microsoft for cross-platform document interchange. The first version of the RTF standard appeared in 1987. Since that time format specification was changed and added. RTF-documents are supported by many text editors.

Export Settings

The export parameters of the RTF export are described in the **StiRtfExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ImageQuality	float	image quality; may have values from 0.0 (the lowest quality) to 1.0 (the highest quality); by default 0.75
ImageResolution	float	image resolution dpi; can take any value, by default 100
UsePageHeadersAndFooters	bool	process headers and footers of a page (see

Name	Type	Description
		Table mode); by default false
ExportMode	enum	select export mode (see Common knowledge); by default StiRtfExportMode.Table
CodePage	int	this property is obsolete and is not used any longer, remained for compatibility with earlier versions

Static Options

Except the **StiRtfExportSettings** class parameters of export to RTF can be set using the static properties. All properties are described in the table below. To access to export properties it is necessary to add the **StiOptions.Export.Rtf...** prefix. For example, **StiOptions.Export.Rtf.UsePageRefField**.

Name	Type	Description
UsePageRefField	bool	when exporting a header with page numbers (for example, the "Anchors" report) the MS-Word "PAGEREF" command should be used for page numbers. Page numbers in the table of contents will be dynamically changed; if false, then numbers of pages will be static; by default true
ConvertDigitsToArabic	bool	convert ASCII digits into Arabic digits; by default false

Name	Type	Description
ArabicDigitsType	enum	select type of Arabic digits; by default Standard
DivideSegmentPages	bool	divide segmented pages into separate pages; if false then are exported "as is" without dividing; by default true
LineHeightExactly	bool	export rows heights of a table "exactly"; if false then the height is exported as "at least"; by default true
RemoveEmptySpaceAtBottom	bool	remove empty space on the bottom of a page; by default true
LineSpacing	double	coefficient of correction of a row height in multilined text fields; by default 0.965
RightMarginCorrection	int	correction of the right margin of a cell; by default 0
SpaceBetweenCharacters	int	sets space between characters of a font in twips; negative value corresponds to condensation; by default -2
UseCanBreakProperty	bool	use the CanBreak property when exporting rows of a table; by default true
DivideBigCells	bool	divide big cells into smaller ones for easier editing and scrolling; by default true

15.4.4.3 Word

Notice

Word can open max. size of the files:
- Plain text: 32 MB;
- Document with images: 512 MB.

Microsoft Word is a text processing software produces by Microsoft. It is a component of the Microsoft Office system. The first version was released for IBM PC's running DOS in 1983. Later there was a release for Apple Macintosh (1984), SCO UNIX, and Microsoft Windows (1989). Microsoft Word is the most popular text processors. Starting with first versions MS Word could write files in binary code with the «.doc» extension. The Word specification was secret and only in 2008 was published. The latest version of Word 2007/2010 "uses by default" the XML based format: Microsoft Office Open XML. For a new format the «.docx» file extension is used. This format is a zip-archive that contains a text as XML, graphics, and other data. When exporting, a report is converted into one table. Such a document is easy to edit.

Export Settings

The export parameters of the Word 2007 export are described in the **StiWord2007ExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ImageQuality	float	image quality; may have values from 0.0 (the lowest quality) to 1.0 (the highest quality); by default 0.75
ImageResolution	float	image resolution dpi; can take any value, by default 100
UsePageHeadersAndFooters	bool	process headers and footers of a page; by

Name	Type	Description
		default false

Static Options

Static properties of export to Word 2007. To access to export properties it is necessary to add the **StiOptions.Export.Word2007...** prefix. For example, **StiOptions.Export.Word2007.DivideSegmentPages**.

Name	Type	Description
DivideSegmentPages	bool	divide segmented pages into separate pages; if false then are exported "as is" without dividing; by default true
AllowImageComparer	bool	use the image comparer, e.g. replace image duplicates (see Common export settings); if false then an image is exported "as is"; by default true
LineHeightExactly	bool	export the rows height of a table "exactly"; if false then the height is exported as "at least"; by default true
RemoveEmptySpaceAtBottom	bool	remove empty space on the bottom of a page; by default true
RightMarginCorrection	int	correction of the right margin of a cell; by default 0
SpaceBetweenCharacters	int	sets the space between characters of a font (in twips); negative value

Name	Type	Description
		corresponds to condensed; by default -2

15.4.4.4 ODT

Open Document Text (**ODT**) is the open document for storing documents of the OpenOffice Writer, which is included into the OpenOffice.org package.

OpenOffice.org is the open package of office applications created as alternative to Microsoft Office. OpenOffice.org was one of the first what supported the new open OpenDocument. Works on Microsoft Windows and UNIX systems: GNU/Linux, Mac OS X, FreeBSD, Solaris, Irix.

OpenDocument Format (ODF) is the open file format for storing office documents, including text documents, spreadsheets, images, data bases, presentations. This format is based on the XML format.

OpenOffice Writer is the text processor and visual HTML editor, included into the OpenOffice. It is open software (LGPL license). Writer is similar to Microsoft Word and has approximately the same functionality. Writer allows saving documents in different formats including Microsoft Word, RTF, XHTML, and OASIS Open Document Format. Starting with the OpenOffice version 2.0, the OpenDocument Format is the default format for saving documents. File have the «.odt» extension.

When exporting the report is converted into a single table. The document is easily editable but some objects can be changed.

Export Settings

The export parameters of the ODT export are described in the **StiOdtExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ImageQuality	float	image quality; may have values from 0.0 (the lowest quality) to1.0 (the highest quality); by default 0.75

Name	Type	Description
ImageResolution	float	image resolution, dot per inch; may have any value, by default 100

Static Options

Static properties of export to ODT. To access to export properties it is necessary to add the **StiOptions.Export.Odt...** prefix. For example, **StiOptions.Export.Odt.DivideSegmentPages**.

Name	Type	Description
DivideSegmentPages	bool	divide segmented pages into separate pages; if false then are exported "as is" without dividing; by default true
AllowImageComparer	bool	use the image comparer, e.g. replace image duplicates (see Common export settings); if false then an image is exported "as is"; by default true
RemoveEmptySpaceAtBottom	bool	remove empty space on the bottom of a page; by default true

15.4.5 Spreadsheets

Notice

For desktop versions, there are no specific size restrictions; the size of the opened file is limited by the free memory of the computer. But for web versions there are restrictions depending on the version of the product and the service used (10mb - 50mb - 250mb).

This group of exports create spreadsheets. They are exports to both different formats of Microsoft Excel and to OpenOffice Calc.

- Excel,
- Excel 2007/2010,
- ODS.

15.4.5.1 Excel

Microsoft Excel is a spreadsheet application written and distributed by Microsoft for Microsoft Windows. It allows using calculation, graphing tools, pivot tables and a macro programming language called VBA. So, it is the most popular table processor available for these platforms since version 5 in 1993.

Microsoft Excel up until Excel 2007 version used a proprietary binary file format called Binary Interchange File Format (BIFF) and **.xls** file extension. Specification was closed but since 2008 it was published. Besides, most of Microsoft Excel can read CSV, DBF, SYLK, DIF, and other formats.

Export Settings

The export parameters of the XLS export are described in the **StiExcelExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ImageQuality	float	image quality; may have values from 0.0 (the lowest quality) to1.0 (the highest quality); by default 0.75
ImageResolution	float	image resolution, dot per inch; may have any value, by default 100
UseOnePageHeaderAndFooter	bool	remove from a report all page headers (except the first one) and all page footers (except the last

Name	Type	Description
		one); by default false
ExportDataOnly	bool	export data only, e.g. all components which are placed on data bands; by default false
ExportPageBreaks	bool	export page breaks; by default false
ExportObjectFormatting	bool	export object formatting; by default true
ExportEachPageToSheet	bool	export each page of a report as a sheet; by default false

The **ExportObjectFormatting** property works only if the ExportDataOnly is set to true.

Static Options

Static properties of export to Excel. To access to export properties it is necessary to add the **StiOptions.Export.Excel...** prefix. For example, **StiOptions.Export.Excel.AllowExportDateTime**.

Name	Type	Description
AllowExportDateTime	bool	export date and time; if false then date and time are exported as text strings; by default false
ColumnsRightToLeft	bool	set the order of columns from right to left; by default false
MaximumSheetHeight	int	maximal number of rows on a sheet; remaining rows are transferred on the next

Name	Type	Description
		sheet; by default 65534
RemoveEmptySpaceAtBottom	bool	remove empty space on the bottom of a page; by default true
ShowGridLines	bool	show grid lines; by default true
DivideBigCells	bool	divide big cells into smaller ones for easier editing and scrolling; by default true

15.4.5.2 Excel 2007/2010

For storing documents as the basic Microsoft Excel format, right up to the Excel 2007 version, used its own binary format of files (BIFF) and the file extension was «.xls». In **Excel 2007/2010**, the basic format is the Microsoft Office Open XML format and stores document in files with the «.**xlsx**» extension. The Excel 2007 is compatible with binary formats such as CSV, DBF, SYLK, DIF, and others.

Export Settings

The export parameters of the Excel 2007 export are described in the **StiExcel2007ExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ImageQuality	float	image quality; may have values from 0.0 (the lowest quality) to1.0 (the highest quality); by default 0.75
ImageResolution	float	image resolution, dot per inch; may have any value, by default 100
UseOnePageHeaderAndFooter	bool	remove from a report all page headers (except the

Name	Type	Description
		first one) and all page footers (except the last one); by default false
ExportDataOnly	bool	export data only, e.g. all components which are placed on data bands; by default false
ExportPageBreaks	bool	export page breaks; by default false
ExportObjectFormatting	bool	export object formatting; by default true
ExportEachPageToSheet	bool	export each page of a report as a sheet; by default false

The **ExportObjectFormatting** property works only if the **ExportDataOnly** is **true**.

Static Options

Static properties of export to Excel 2007.

Name	Type	Description
AllowImageComparer	bool	use the image comparer, e.g. replace image duplicates (see Common export settings); if false then an image is exported "as is"; by default true
ColumnsRightToLeft	bool	set the order of columns from right to left; by default false
MaximumSheetHeight	int	maximal number of rows on a sheet; odd rows are

Name	Type	Description
		moved to the next sheet; by default 1048574
RemoveEmptySpaceAtBottom	bool	remove empty space on the bottom of a page; by default true

15.4.5.3 ODS

Open Document Spreadsheet (**ODS**) is the opened format to store OpenOffice Calc spreadsheet documents, that is included into the OpenOffice.org package.

OpenOffice.org is a free package of office applications developed as alternative to Microsoft Office. The OpenDocument is one of the first what started to support the opened format. it works on Microsoft Windows and UNIX-like systems: GNU/Linux, Mac OS X, FreeBSD, Solaris, Irix.

OpenDocument Format (ODF) — an open document file format for storing and exchanging editable documents including text documents (such as notes, reports, and books), spreadsheets, drawings, databases, presentations. The format is based on the XML-format. The standard was jointly developed by public and various organizations and is available to all and can be used without restrictions.

OpenOffice Calc is the table processor that is included into the OpenOffice and is a free software (LGPL license). Calc is similar to the Microsoft Excel spreadsheet and functionality of these processors is approximately equal. Calc allows you to saving documents to various formats, including Microsoft Excel, CSV, HTML, SXC, DBF, DIF, UOF, SLK, SDC. Starting with version OpenOffice 2.0, for document storage format by default OpenDocument Format, files are saved with the extension «. Ods». Starting with the OpenOffice version 2.0 for storing documents, by default, the OpenDocument Format is used. Files are stored with the «.ods» extension.

Export Settings

The export parameters of the ODS export are described in the **StiOdsExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ImageQuality	float	image quality; may have values from 0.0 (the lowest quality) to1.0 (the highest quality); by default 0.75
ImageResolution	float	image resolution, dot per inch; may have any value, by default 100

Static Options

Static properties of export to ODS. To access to export properties it is necessary to add the **StiOptions.Export.Ods...** prefix. For example, **StiOptions.Export.Ods.AllowImageComparer**.

Name	Type	Description
AllowImageComparer	bool	use the image comparer, e.g. replace image duplicates (see Common export settings); if false then an image is exported "as is"; by default true
DivideSegmentPages	bool	divide segmented pages into separate pages; if false then are exported "as is" without dividing; by default true
RemoveEmptySpaceAtBottom	bool	remove empty space on the bottom of a page; by default true

15.4.6 Data

This is a group of file formats which are used to store table data.

- CSV
- DBF

- XML
- DIF
- SYLK

15.4.6.1 CSV

CSV (Comma Separated Values) is a text format that is used to represent table data. Each string of the file is one row of the table. The values of each column are separated by the delimiter that depends on regional settings. The values that contain reserved characters (such as a comma or a new string) are framed with the double quotes (") symbol; if double quotes are found in the value they are represented as two double quotes in the file.

Information

Only those data (components) can be exported to the **CSV** format which are placed on data bands. If the **SkipColumnHeaders** property is set to false then, additionally, column headers are exported as the first row.

Export Settings

The export parameters of the CSV export are described in the StiCsvExportSettings class. The description of all class properties are in the table below.

Name	Type	Description
Separator	string	sets the symbol-separator of a list that is used when exporting; by default CurrentCulture.TextInfo.ListSeparator
Encoding	Encoding	text file coding; by default Encoding.UTF8
SkipColumnHeaders	bool	skip headers of columns; by default false

Static Options

Static properties of export to CSV. To access to export properties it is necessary to add the **StiOptions.Export.Csv...** prefix. For example, **StiOptions.Export.Csv.ForcedSeparator**.

Name	Type	Description
ForcedSeparator	string	sets the separator forcibly which are used in export; if the empty string is set then the symbol from export settings in used; by default - empty string

15.4.6.2 DBF

The **DBF** (DataBase File) is the format to store data and it is used as the standard way to store and pass information. The DBF file consist of a header section for describing the structure of the data in the file. There are several variations on the .dbf file structure.

Information

Only data can be exported to the DBF format, in other words only the components, which are placed on data bands.

Controlling Exports

The following elements can be specified in the Tag field to control export:

- DataType [: FieldLength [: DecimalPartLength]],
- ExportType : "FieldName",
- Column: "FieldName" "DataString".

Several elements should be separated with the semicolon. The “DataType" element should be only one and should be placed first, other elements – if necessary.

Values of the "DataType" element are shown in the table below. If the data type is

not set, then the string data type is taken by default. The "FieldLength" element sets fixed width of a data field. If the field width is not set, then the width is taken from the table. For the **string** type the default width is the longest **string**. The "DecimalPartLength" element sets the number of characters after comma. If it is not set, then the default number is taken.

Data type	DBF data type (default size)	Description
int	Numeric (15 : 0)	Numeric
long	Numeric (25 : 0)	Numeric
float	Numeric (15 : 5)	Decimal
double	Numeric (20 : 10)	Decimal
string	Character (auto)	Text
date	Date (8)	Date

Sample of using elements are shown in the table below.

Type	Description
string : 25	set the column width (25 characters) and cuts all long strings
float	converts decimal digit with the length 15 characters, 5 characters after comma
float :10	converts decimal digit with the length 10 characters, 5 characters after comma
float :10 : 2	converts decimal digit with the length 10 characters, 2 characters after comma
int :10 : 2	converts integer digit with the length 10 characters; the second parameter is ignored

Information

If the integer part of a digit is long and cannot be placed into the specified field, then it is cut, so data are lost. For example, if the write «-12345,678» in the

«float:8:3» field, then the «2345,678» will be output.

The "**ExportType**" element indicates for which export the field name is set. The values can be used: “dbf”, “csv”, “xml”, “default”. The "FieldName" element indicates the field name in the file (for the DBF the is automatically cut up to 10 characters). The own name can be specified to each type of export. If the name for each export is not specified then the name for the “default” type is taken. For example:

```
DBF : "Describe" ; XML : "Description" ; default: "Default name"
```

The "Column" element indicates that the additional field is added to the exported data. The "FieldName" element indicates the name of a new field. The "DataRow" element indicates the content of a new field and can be expression. For example

```
Column: "SortField" "{Products.Categories.CategoryName}"
```

Export Options

The export parameters of the DBF export are described in the **StiDbfExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
CodePage	StiDbfCodePages	a code page of a file; by default StiDbfCodePages.Default

15.4.6.3 XML

XML (eXtensible Markup Language) is a text format that is used to store structured data (in exchange for existed files of data bases), for exchange of information between programs and also to create on its base the special markup languages (for example, XHTML), sometimes called dictionaries. XML is the hierarchical structure that is used to store any data. Visually this structure can be represented as the tree. XML supports Unicode and other encoding.

Information

Only those data (components) are exported to the XML format which are placed on data bands.

Controlling Exports

The following elements can be specified in the Tag field to control export to XML:

- DataType
- ExportType : "FieldName"
- Column: "FieldName" "DataRow"

Several elements should be separated with the semicolon. The “DataType" element should be only one and should be placed first, other elements – if necessary.

Values of the "DataType" element are shown in the table below. If the data type is not set, then the **string** data type is taken by default.

Data type	Description
int	Numeric
long	Numeric
float	Decimal
double	Decimal
string	Text
date	Date

The "**ExportType**" element indicates for which export the field name is set. The values can be used: “dbf”, “csv”, “xml”, “default”. The "FieldName" element indicates the field name in the file. The own name can be specified to each type of export. If the name for each export is not specified then the name for the “default” type is taken. For example:

```
DBF : "Describe" ; XML : "Description" ; default: "Default name"
```

The "Column" element indicates that additional field is added to the exported data. The "FieldName" element indicates the name of a new field. The "DataRow" element indicates the content of a new field and can be expression. For example:

```
Column: "SortField" "{Products.Categories.CategoryName}"
```

15.4.6.4 DIF

DIF (Data Interchange Format) is a text format that is used to exchange sheets between spreadsheets processors (Microsoft Excel, OpenOffice.org Calc, Gnumeric, StarCalc, Lotus 1-2-3, FileMaker, dBase, Framework, Multiplan, etc). The only limitation of this format is that the DIF format may contain only one sheet in one book.

Export Settings

The export parameters of the DIF export are described in the **StiDifExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ExportDataOnly	bool	export data only. e.g. only components placed on data bands; by default false
Encoding	Encoding	file encoding; by default Encoding.ASCII
UseDefaultSystemEncoding	bool	use the default system encoding; if false then use encoding that is set by the Encoding property; by default true

15.4.6.5 SYLK

SYLK (Symbolic Link) format- this text format is used to exchange data between applications, specifically spreadsheets. Files of SYLK have «.slk» extension. Microsoft does not publish a SYLK specification, therefore work with this format in different applications can be different.

Information

A SYLK file can be written in Unicode and read by some applications but anyway many applications which do support Unicode writes SYLK files into ANSI but not Unicode. Therefore, symbols which do not have representation in the system code page will be written as ('?') symbols.

Export Settings

The export parameters of the SYLK export are described in the **StiSylkExportSettings** class. The description of all class properties are in the table below.

Name	Type	Description
ExportDataOnly	bool	export data only. e.g. only components placed on data bands; by default false
Encoding	Encoding	file encoding; by default Encoding.ASCII
UseDefaultSystemEncoding	bool	use the default system encoding; if false then use encoding that is set by the Encoding property; by default true

15.4.7 Images

Export groups to graphic formats. All graphic formats can be divided in to types: bitmapped images and vector formats. Notice. On the current moment the export of monochrome image is supported only to the BMP, GIF, PCX, PNG, TIFF format. So the DitheringType property works only for these exports.

Export Parameters

All exports of images have the same export settings. They are described in the table below. But each format has its own ExportSettings class. For BMP, GIF, PNG, TIFF, JPEG, PCX, and EMF the following classes are used in exports. The StiBmpExportSettings is used for export to BMP, **StiGifExportSettings** is used for export to GIF, **StiPngExportSettings** is used for export to PNG, **StiTiffExportSettings** is used for export to TIFF, **StiJpegExportSettings** is used for export to JPEG, **StiPcxExportSettings** is used for export to PCX, and **StiEmfExportSettings** is used for export to EMF.

Name	Type	Description
ImageZoom	double	zoom factor. By default a value is 1.0 what is equal 100% in export settings window
CutEdges	bool	cut page edges; by default false
ImageFormat	StiImageFormat	Image format - colored, tint of grey or monochrome; by default StiImageFormat.Color
MultipleFiles	bool	saves pages of a report into separate files; can be used for TIFF only, because it can save some pages into one file, other formats save pages into separate files; by default false
DitheringType	StiMonochromeDitheringType	a type of image dithering to get monochrome image; by default StiMonochromeDitheringType.FloydSteinberg

15.5 Scripts

Stimulsoft Reports supports a choice of languages for report generation.

- Programming Language of Report,
- Report Code.

15.5.1 Programming Language of Report

The report generator uses a single specified programming language to generate the report code and handle report events. If the current programming language of a report does not suit your requirements you can change it. The options are currently C# or VB.NET.

Changing The Language Of The Current Report

To do this select **File | Report Setup**. A new dialog will be displayed

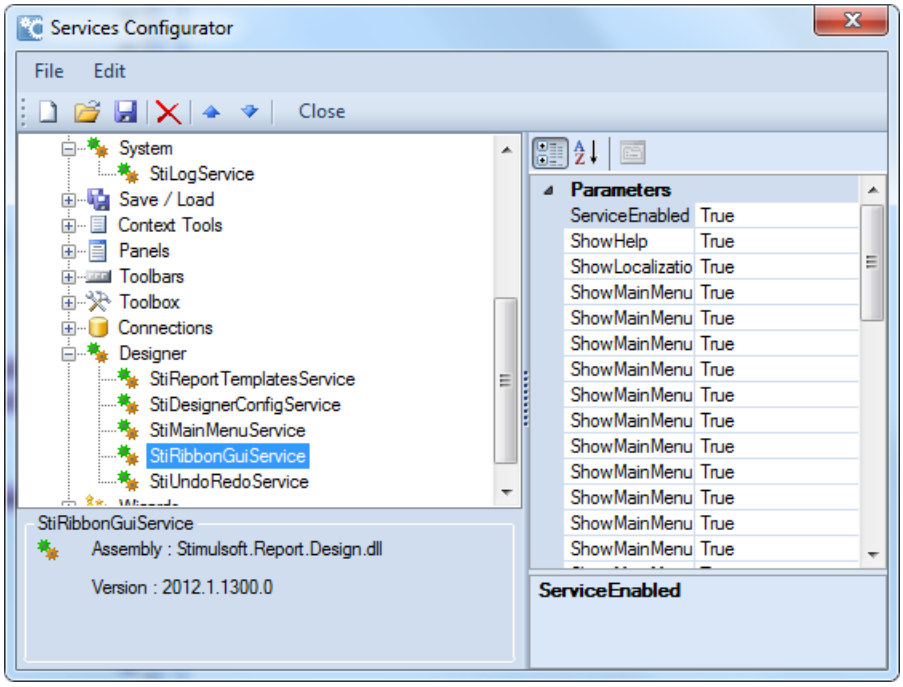
In the **Language** group select a new programming language and press **Ok**. The current programming language will then be changed.

Information

The underlying report code will have to be regenerated for the entire report, and any changes which have been made within the report code will be lost.

It is not convenient to change the programming language each time a report is rendered, so Stimulsoft Reports allows you to set the default programming language used by all new reports.

To do this you should use the Services Configurator utility. All programming languages in Stimulsoft Reports are located under the Languages node. The language which is shown first in the list is the default programming language. For example, on the picture below the C# programming language is the default programming language. To make VB.Net the default programming language simply drag the **StiCSharpLanguage** service down one position with the mouse or use the up and down buttons to re-order the languages.



15.5.2 Report Code

When you create a new report its source code, often called the report script, is generated automatically using either C# or VB.NET programming language depending on the currently selected default. You can use only one of these programming languages at a time

In the report code the structure and initialization of the report class, which itself inherits from the StiReport class, are described. When adding new pages, components or changing any parameters of a report, those changes are automatically recorded within the class. The report class therefore contains a description of all components, data, events, report properties, and data source structures for the report. Any events specified by the user are also added to the report code.

For the ultimate in power and flexibility Stimulsoft Reports allows direct editing of the report code - if you want something not provided by the available properties and features of the designer you can actually code your own features within the report. When writing events or another code in the report, you use the standard syntax of the selected .Net Framework programming language i.e. if the language is set to C# you write code using C# syntax.

Information

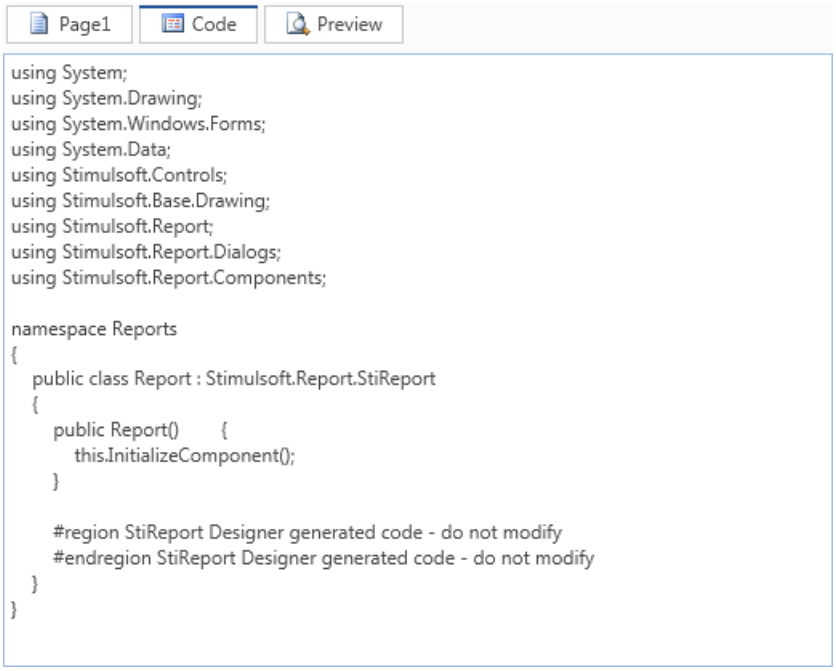
The report code is generated in **C#** or **VB.Net** programming language. All events and any another code in this report must be written in the currently selected language.

When rendering reports, compilation of the report class occurs first. After that the compiled report is executed.

Information

The report code is compiled using the .NET Framework compiler.

To see the report code click the Code tab in the designer. The code will then be displayed:



To edit the code, simply start typing in the appropriate place.

Infromation

Do not change preprocessor directives or automatically updated code.

Whilst Stimulsoft Reports allows you to directly edit the report code, it is important to remember that it is impossible to make changes in the parts of the report code which are automatically updated - such changes will be lost when the next update takes place. The automatically updated report code is enclosed in Region preprocessor directives:

VB.NET

```
...
//At the beginning of the automatically updated code
#region StiReport Designer generated code - do not modify

//Automatically updated code goes here

//the end of the automatically updated code
#endregion StiReport Designer generated code - do not modify
...
```

Any code that you write within the report must be written outside these Regions.

16 PDF Forms

Stimulsoft PDF Forms is a tool for creating, editing, filling, publishing, distributing interactive forms, and collecting results.

16.1 Get Started for ASP.NET Core 3.1

1. Create the **ASP.NET Core Empty** application;
2. Install the **Stimulsoft.PDF.Forms** component into the application from the **NuGet** package manager;
3. In **Startup.cs**, configure the application to use controllers:

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
   services.AddControllers();
```

```
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
   if (env.IsDevelopment())
   {
      app.UseDeveloperExceptionPage();
   }

   app.UseDefaultFiles();
   app.UseStaticFiles();

   app.UseRouting();

   app.UseCors(builder =>
   builder.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod());

   app.UseEndpoints(endpoints =>
   {
      endpoints.MapControllerRoute(
         name: "default",
         pattern: "{controller=Forms}/{action=Action}");
   });
}
```

4. Create the **Controllers** folder;
5. Add **FormsController.cs** to the **Controllers** folder, which will process requests from the **Stimulsoft PDF Forms** component:

FormsController.cs

```
public class FormsController : Controller
{
   [HttpPost]
   public IActionResult Action()
   {
      try
      {
         var data = JObject.Parse(this.HttpContext.Request.Form["data"]);
         var action = data["action"].ToString();
         switch (action)
         {
            case "Initialize":
               var initData = StiWebForm.Initialize(data, null);
               return Json(initData.Content);

            default:
               var result = StiWebForm.ProcessRequest(data);
               return result.ContentType switch
               {
                  "application/pdf" => new FileContentResult(result.Content as
```

```
                byte[], result.ContentType),
                _ => Json(result.Content),
            };
        }
      }
      catch (Exception e)
      {
        return new ContentResult()
        {
          Content = e.Message,
          ContentType = "text/plain"
        };
      }
    }
}
```

6. Inside the application folder, run the command in the terminal to create a new **Angular** application with no `routing` and CSS styling:

console

```
ng new sti-forms-designer --routing false --style css
```

7. Rename the **sti-forms-designer** folder to **ClientApp**;
8. In the **package.json** file, add dependencies for **Stimulsoft PDF Forms** and the necessary components:

package.json

```
"stimulsoft-forms": "^2024.2.4",
"ngx-color-picker": "^13.0.0",
"primeicons": "^6.0.1",
"primeng": "^14.1.2",
"resize-observer-polyfill": "^1.5.1",
"rxjs": "~6.5.0"
```

9. Add styles to the **angular.json** file and improve the `budgets`:

angular.json

```
"build": {
  "options": {
    "styles": [
      "src/styles.css",
      "node_modules/primeicons/primeicons.css",
      "node_modules/primeng/resources/primeng.min.css",
      "node_modules/stimulsoft-forms/theme.css"
    ]},

    "configurations": {
      "production": {
        "budgets": [
          {
            "type": "initial",
            "maximumWarning": "500kb",
            "maximumError": "5mb"
          }]
        }
      }
    }
  }
}
```

10. Update the imports in **ClientApp\src\app\app.module.ts**:

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from "@angular/forms";
import { HttpClientModule } from "@angular/common/http";
import { BrowserAnimationsModule } from "@angular/platform-browser/
animations";
import { DropdownModule } from "primeng/dropdown";
import { StimulsoftFormsModule } from 'stimulsoft-forms';
import { AppComponent } from './app.component';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    BrowserAnimationsModule,
    DropdownModule,
    StimulsoftFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
```

```
})
```

11. Update **ClientApp\src\app\app.component.ts**:

app.component.ts

```
import { Component } from '@angular/core';
import { StimulsoftFormsService } from 'stimulsoft-forms';

@Component({
   selector: 'app-root',
   template: `
      <stimulsoft-forms
         [requestUrl]="'http://localhost:7536/Forms/Action'" /*URL to Forms
         controller, you can find this url in launchSettings.json file of
         the project*/
         [form]="form" /*StiForm object to use to create form object using
         StimulsoftFormsService*/
         [style.width]="'100%'"
         [style.height]="'100%'">
      </stimulsoft-forms>
`
})

export class AppComponent {
   public form!: any;

   constructor(public formService: StimulsoftFormsService) {
      this.form = this.formService.createElement("Form");
   }
}
```

12. Update **ClientApp\src\app\index.html**:

index.html

```
<!doctype html>
<html lang="en" style="position: fixed;width:100%;height:100%">
<head>
   <meta charset="utf-8">
   <title>StiFormsDesigner</title>
   <base href="/">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body style="width:100%;height:100%">
   <app-root></app-root>
```

```
</body>
</html>
```

13. Go to the **ClientApp** folder;
14. Run the command to install **Angular** components:

console

```
npm i --force
```

15. Run the command to build an **Angular** application:

console

```
ng build --output-hashing none
```

16. Create the **wwwroot** folder in the project's root folder;
17. Copy all files from **ClientApp\dist\sti-forms-designer** into **wwwroot**;
18. Run the project.