

USER GUIDE

User Guide

Bold Reports Documentation

Version - v5.4.20 | Release Date - December 22,2023

Overview	80
Overview	80
System Requirements	80
Hardware Environments	80
Development Environments	80
Supported Operating Systems	81
Browser Compatibility	81
See Also	81
Download and installation of Report Viewer SDK	81
Register and download the Report Viewer SDK	81
Online Installation.....	83
Offline Installation	89
Report Viewer SDK samples and demos	93
View the product demos.....	93
JavaScript	93
Angular	94
ASP.NET CORE	95
ASP.NET MVC	96
ASP.NET Web Forms	97
WPF	98
UWP	99
Report Viewer SDK nuget packages and assemblies	100
Script References	100
JavaScript Report Viewer SDK.....	100
Angular Report Viewer SDK	101
ASP.NET Core Report Viewer SDK.....	101
ASP.NET MVC Report Viewer SDK.....	101
ASP.NET WebForms Report Viewer SDK.....	102
WPF Report Viewer SDK	102
UWP Report Viewer SDK.....	102
NPM Packages.....	103
Migrate Reporting Application	104
Migrate JavaScript Report Viewer SDK	104
Migrate Angular Report Viewer SDK.....	105
Migrate ASP.NET Core Report Viewer SDK	105

Migrate ASP.NET MVC Report Viewer SDK	106
Migrate ASP.NET Report Viewer SDK.....	107
Migrate WPF Report Viewer SDK.....	107
Migrate UWP Report Viewer SDK	107
Uninstallation of Reporting Tools	108
Report Viewer SDK licensing overview	109
Report Viewer SDK online licensing overview	109
How to generate Bold Reports online license token	109
How to register the Bold Reports online license token	109
ASP. NET	109
ASP. NET MVC	110
ASP. NET Core	110
WPF	111
UWP	111
Blazor	111
Report Viewer SDK offline licensing overview	112
How to generate and download Bold Reports offline license key.....	113
How to register the Bold Reports offline license key	113
ASP. NET	113
ASP. NET MVC	113
ASP. NET Core	114
WPF	114
UWP	115
Blazor	115
Report Viewer SDK license token errors	116
License token not registered	116
Invalid token	117
License Expired	117
Platform Mismatch	117
Internet Connection Error.....	117
Network Error	118
Could not load Bold.Licensing.dll assembly version	118
FAQ section for Bold Licensing.....	118
How to upgrade from Trial version after purchasing a license.....	118
Where can I get a license token.....	118

See also	118
Does Bold Reports Embedded Reporting Tools or Viewer SDK require additional licensing when deploying an application to Azure App service.....	119
See also	119
Can Syncfusion licenses be used with Bold Reports	119
See also	119
Can the Syncfusion community license be used with Bold Reports	119
See also	119
Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license	120
See also	120
What are the differences in Bold Reports licensing models.....	120
See also	120
How to resolve the license network error	120
See Also.....	121
Report Viewer SDK licensing version 1.2.x	121
Report Viewer SDK licensing overview	121
How to generate Bold Reports license key	121
How to register the Bold Reports license key	122
ASP.NET	122
ASP.NET MVC	122
ASP.NET Core	122
WPF	123
UWP	123
Report Viewer SDK license key errors.....	124
License key not registered	124
Invalid key	124
Trial Expired	124
License Expired	124
Platform Mismatch	125
Version Mismatch	125
Could not load Bold.Licensing.dll assembly version	125
Bold reporting tools for Blazor.....	125
How to best read this user guide	125
Getting help	125

Overview	125
Add Web Report Viewer to a Blazor application	126
Prerequisites	126
Create a Blazor application	126
Create a Web API for report processing	126
Initialize the Report Viewer	130
Run the Application.....	132
See Also	133
Render RDLC report	133
Bind data source in Web API controller.....	133
Extensions	136
See Also	136
Custom Data Extension.....	136
See Also	136
Configure a new data source extension in Blazor Report Viewer	136
Create Blazor Report Viewer Application	136
Install data source extension from NuGet.....	136
Register data source extension.....	138
Run the application.....	139
How to queries for Bold Reports ReportViewer	139
How to resolve Cannot read property enableBlazorMode of undefined error when using with Bold Reports.....	139
See also	140
How to use Bold Reports with Syncfusion Blazor	140
How to resolve circuit shut down issue due to error.....	142
Migrate to Report Viewer v2.0 component.....	142
Bold Report Writer.....	143
Report Writer with Blazor Application.....	143
Prerequisites	143
Create a Blazor application	144
List of dependency libraries	144
Create a PDF Document in Web API	144
Client Side	149
Reporting tools for Angular	152
How to best read this user guide	152

Getting help	152
Reporting tools for Angular	152
How to best read this user guide	152
Getting help	152
System Requirements	152
Supported Operating Systems	153
Development Environments	153
Browser Compatibility	153
See Also	153
Overview	153
Display SSRS RDL report in Bold Reports Angular Report Viewer.....	154
Prerequisites	154
Install the Angular CLI	154
Create a new application	154
Configure the Bold Report Viewer in Angular CLI.....	155
Adding CSS reference.....	157
Adding Report Viewer component	158
Create Web API service.....	160
Adding already created report.....	160
Set report path and Web API service.....	160
Serve the application	161
See Also	162
Load SSRS Report Server reports	162
Network credentials for SSRS	166
Set data source credential for shared data sources	166
Change data source connection string	166
See also	167
Load SharePoint Server reports	167
Forms credential for SharePoint server.....	168
Set data source credential for shared data sources	168
Render RDLC report	169
Bind data source at client side.....	169
Bind data source in Web API controller.....	170
Load report as stream.....	172
View report click	173

Render subreport.....	174
Change subreport path	175
Set subreport parameter	175
Modify subreport data source connection string	176
Set subreport data source	176
Load subreport stream	179
Report parameters.....	180
Set parameter at client side.....	180
Set parameters in Web API Controller.....	181
Set date time display format for date parameter.....	184
Access the hidden or internal parameter information	186
Set the report parameter visibility in Web API Controller.....	187
Report interaction events	187
Report loaded	188
Report error	191
Show error	192
Drill through	193
Hyperlink.....	195
Handle post actions	196
AjaxBeforeLoad.....	196
Add custom header in Ajax request.....	196
Pass custom data in Ajax request	198
AjaxSuccess	200
AjaxError	201
Print report	202
View report in print mode	202
Print in new page	203
Set page orientation and paper size	204
Set report margin.....	205
Set page height and width	207
Print report with images	208
External styles in report printing	209
Show print progress.....	210
Remove empty spaces in printing.....	212
Export report.....	212

Export event handling.....	212
Change Excel and Word export format.....	214
Hide specific export type for report.....	215
PDF export options	216
Export with complex scripts.....	216
PDF conformance.....	216
Add custom PDF fonts.....	216
Word export options.....	217
Word document type.....	217
Word document advance layout for merged cells	218
Protecting Word document from editing	218
Excel export options.....	218
Excel document type.....	219
Excel document advance layout for merged cells	219
Protecting Excel document from editing	219
PowerPoint export options	219
CSV export options.....	220
HTML export options	220
Password protect exported document.....	220
Toolbar customization	221
Hide toolbar items	221
Enable stop option in toolbar	223
Enable export setup option in toolbar.....	224
Enable search text option in toolbar.....	225
Hide toolbar	226
Decide or hide the export option.....	227
Add custom items to the export drop-down	228
Add custom toolbar item.....	230
Add custom item to exiting toolbar group.....	230
Add new toolbar group	232
See also	234
Custom actions.....	234
Send a report as an email attachment.....	234
Add email button in Report Viewer	234
Create custom email action	235

Custom properties	237
See Also	237
Textbox custom properties	237
Show or hide toggle icon in text box report item	237
Table custom properties	239
Limit number of table records on each page	239
Image custom properties	239
Angle	240
Chart custom properties	240
Disable legend item interaction	240
Set maximum size for chart legend	241
Change chart legend shape	242
Show or hide chart legend scroller	243
Set range padding for X-axis and Y-axis	244
Control edge label placement in chart axis	245
Change the drawing style of a chart column or bar series	247
Change width of the column type series in chart	248
Report custom properties	249
Improve performance and handle large amount of data	249
Render large data report faster	249
Reduce memory footprint for large data report	250
Improve report items layout and avoid extra blank pages	250
Handling failure in long-running HTTP requests, report processes, and timeouts	251
Parameter custom properties	252
Show only distinct values in parameter	252
Change the dropdown parameter pop up height value	252
Change the dropdown parameter pop up width value	253
Set the minimum date range for the date report parameter	254
Set the maximum date range for date report parameter	255
Hide the report parameter block	256
Set date and time format for parameter	257
Change time display format for parameter	258
Set time interval in datetime parameter	259
Enable filtering and searching in drop-down report parameter	260
Change date report parameter to display date time picker	261

Export custom properties	263
Improve excel export performance	263
Improve excel export readability	264
Set pdf conformance level	264
Export pdf document with complex script.....	265
Encrypt and secure export documents	265
Protecting Word document from editing	266
Set excel document edit password	267
Set excel document protection.....	268
Localization of Bold Reports Angular Report Viewer.....	269
Responsive layout rendering of Angular Report Viewer	271
Normal layout	272
Responsive layout	272
Limitations	273
RDL specification.....	273
Report layout	273
Expressions.....	273
SSRS.....	273
Samples and demos	273
Locally installed reports	273
Offline demos.....	273
Online demos	274
GitHub demo samples.....	274
Migrate Report Viewer application	274
Server-side migration.....	274
Web API Controller	274
Report export configuration	274
NuGet Packages for Angular	276
Configure NuGet feed URL.....	276
Online NuGet feed URL	276
Offline NuGet feed URL.....	277
Installing NuGet packages.....	278
Install using NuGet Package Manager	278
Install using Package Manager Console	278
install specified package in default project	278

install specified package in default project with specified package source	279
install specified package in specified project.....	279
install specified package in default project	279
install specified package in default project with specified Package Source	279
install specified package in specified project.....	279
Upgrading NuGet packages	279
Upgrading using NuGet Package Manager	279
Upgrading using Package Manger Console	279
Update specific NuGet package in default project	279
Update all the packages in default project	280
Update specified package in default project with specified package source	280
Update specified package in specified project	280
Update specified Bold Reports NuGet package.....	280
Update specified package in default project with specified Package Source.....	280
Update specified package in specified project	280
Upgrading using NuGet CLI	280
update all NuGet packages from config file.....	281
update all NuGet packages from specified Packages Source	281
Update all NuGet packages from config file	281
Update all NuGet packages from specified Packages Source	281
Deployment	281
Frequently asked questions	282
Can you use the Report Viewer component in Angular 9 application.....	282
Is possible to change the culture of the date time parameter	282
Is it possible to hide the parameters in Report Viewer	283
Is it possible to hide the export options in Report Viewer	283
Is it possible to load reports providing parameter values at runtime	283
Angular Report Viewer Reporting Service	283
IRreportController	283
Create ASP.NET Web API service	285
Configure Report Viewer Web API.....	287
ReportHelper.....	287
Add Web API Controller	287
Add routing information	289
Enable Cross-Origin Requests	290

Create ASP.NET Core Web API Service	292
List of dependency Libraries	293
Configure Web API	294
Add Web API Controller	294
Enable Cross-Origin requests	297
How to section for Angular Report Viewer	298
Display SSRS RDL report in Bold Reports Angular Report Viewer	298
Prerequisites	298
Install the Angular CLI	298
Create a new application	299
Configure Bold Report Viewer in Angular CLI	299
Adding CSS reference	301
Adding Report Viewer component	302
Create Web API service	304
Adding already created report	305
Set report path and Web API service	305
Serve the application	306
See Also	306
Create a SSRS RDL report	306
Bold Reports Report Designer	306
Microsoft SQL Report Builder	307
Visual Studio Report Server template	307
Prerequisites	307
Create business object class	307
Add an RDLC report	308
Data source and table configuration wizard	309
How to change the exporting document file name based on parameter	315
How to change the data source dynamically	316
How to disable the vertical scrollbar in parameter panel	316
How to pass multiple values using the custom data	316
Use SASS stylesheet in Angular with Bold Reports Angular Report Viewer Component	319
Prerequisites	319
Install the Angular CLI	319
Create a new application	319
Configure Bold Report Viewer in Angular CLI	320

Adding Bold Reports styles reference.....	321
Adding Report Viewer component	322
Create Web API service.....	324
Adding already created report.....	325
Set report path and Web API service.....	325
Serve the application	326
See Also	326
How to clear cache when closing the Report Viewer	327
TypeScript	327
Web API.....	327
Migrate to Report Viewer v2.0 component.....	328
Replacing CSS reference.....	328
Replacing Report Viewer component.....	329
Breaking Changes.....	330
Breaking Changes.....	330
How to section for Angular Bold Reporting Tools	331
How to resolve ej undefined issue when using the Report Viewer properties	331
How to resolve @angular/core has no export member 00InjectableDeclaration	332
See Also	332
Reporting tools for JavaScript.....	332
How to best read this user guide	332
Getting help	333
Reporting tools for JavaScript.....	333
How to best read this user guide	333
Getting help	333
System Requirements	333
Supported Operating Systems	333
Hardware Environments	333
Development Environments	334
Browser Compatibility	334
See Also	334
Overview	334
Display SSRS RDL report in Bold Reports HTML5 JavaScript Report Viewer.....	334
HTML file creation.....	335
Refer scripts and CSS.....	335

Adding Report Viewer Widget	336
Create Web API service	337
Adding already created report.....	337
Set report path and Web API service.....	337
Preview the report.....	338
See Also	339
Load SSRS Report Server reports	339
Network credentials for SSRS	342
Set data source credential for shared data sources	343
Change data source connection string	344
See also	344
Load SharePoint Server reports	344
Forms credential for SharePoint server	345
Set data source credential for shared data sources	345
Render RDLC report	346
Bind data source at client side	347
Bind data source in Web API controller.....	348
Load report as stream.....	350
View Report Click.....	351
Render subreport.....	351
Change subreport path	352
Set subreport parameter	353
Modify subreport data source connection string.....	353
Set subreport data source	354
Load subreport stream	356
Report parameters.....	357
Set parameter at client	357
Set parameters in Web API Controller.....	358
Get report parameter	359
Set date time display format for date parameter.....	360
Change the Parameter drop-down height and width.....	361
Hide a Parameter scroller	361
Hide a Parameter Pane on load	361
Change the Parameter Item Width and Label Width	362
Access the hidden or internal parameter information	362

Set the report parameter visibility in Web API Controller.....	363
See Also	363
Report interaction events	363
Report loaded	364
Report error	364
Show error	365
Drill through.....	365
Hyperlink.....	365
Handle post actions	366
AjaxBeforeLoad.....	366
Add custom header in Ajax request.....	366
Pass custom data in Ajax request	367
AjaxSuccess	369
AjaxError	370
Change Report Viewer default WEB API action with custom endpoint	371
Change report processing endpoint	371
Change export action endpoint	372
Error logging in JavaScript Report Viewer	372
Print report	376
View report in print mode	376
Print in new page	376
Set page orientation and paper size	377
Set report margin.....	377
Set page height and width	377
Print report with images	378
External styles in report printing	379
Show print progress.....	379
Remove empty spaces in printing.....	380
Export report.....	380
Export event handling.....	380
Change Excel and Word export format.....	381
Hide specific export type for report.....	381
PDF export options	381
Export with complex scripts.....	381
PDF Conformance	382

Add custom PDF fonts.....	382
Word export options.....	383
Word document type.....	383
Word document advance layout for merged cells	383
Protecting Word document from editing	384
Excel export options.....	384
Excel document type.....	384
Excel document advance layout for merged cells	385
Protecting Excel document from editing	385
PowerPoint export options	385
CSV export options.....	385
HTML export options	386
Password protect exported document	386
Toolbar customization	387
Hide toolbar items	387
Enable stop option in toolbar	388
Enable export setup option in toolbar.....	388
Enable search text option in toolbar.....	389
Hide toolbar	389
Decide or hide the export option.....	390
Add custom items to the export drop-down	390
Add custom toolbar item	390
Add custom item to exiting toolbar group.....	391
Add new toolbar group	391
Custom Actions	391
Send a report as an email attachment.....	391
Add email button in Report Viewer	392
Create custom email action	393
Cancel report processing in Report Viewer	395
Add cancel report button.....	395
Custom properties	397
See Also.....	397
Textbox custom properties.....	397
Show or hide toggle icon in text box report item	398
Table custom properties	399

Limit number of table records on each page.....	399
Image custom properties.....	399
Angle	400
Chart custom properties.....	400
Disable legend item interaction.....	400
Set maximum size for chart legend.....	401
Change chart legend shape.....	402
Show or hide chart legend scroller	403
Set range padding for X-axis and Y-axis	404
Control edge label placement in chart axis.....	405
Change the drawing style of a chart column or bar series	407
Change width of the column type series in chart	408
Report custom properties.....	409
Improve performance and handle large amount of data	409
Render large data report faster	409
Reduce memory footprint for large data report	410
Improve report items layout and avoid extra blank pages.....	410
Handling failure in long-running HTTP requests, report processes, and timeouts.....	411
Parameter custom properties.....	412
Show only distinct values in parameter.....	412
Change the dropdown parameter pop up height value	412
Change the dropdown parameter pop up width value	413
Set the minimum date range for the date report parameter.....	414
Set the maximum date range for date report parameter	415
Hide the report parameter block.....	416
Set date and time format for parameter	417
Change time display format for parameter	418
Set time interval in datetime parameter	419
Enable filtering and searching in drop-down report parameter	420
Change date report parameter to display date time picker	421
Show the parameter values in sorted order like Ascending or Descending.....	423
Export custom properties	424
Improve excel export performance	425
Improve excel export readability	425
Set pdf conformance level	426

Export pdf document with complex script.....	427
Encrypt and secure export documents	427
Protecting Word document from editing	428
Set excel document edit password	429
Set excel document protection.....	430
Localization of Bold Reports HTML5 JavaScript Report Viewer.....	431
Responsive layout rendering of HTML5 JavaScript Report Viewer	432
Normal layout	432
Responsive layout	433
Limitations	433
RDL Specification.....	433
Report Layout.....	433
Expressions.....	434
SSRS.....	434
Data Processing Extensions for Report Viewer.....	434
Create a MySQL Data Processing Extension for Report Viewer	434
Configure dependent Assemblies	434
Implementing Data Processing Extensions	434
TestConnection	434
GetSchemaData	435
GetData	435
Deploy Data Processing Extensions in ReportViewer	436
Data Processing Extensions Configuration	436
Configuration with extension assembly.....	437
Create a WebAPI Data Processing Extension for Report Viewer	437
Configure dependent Assemblies	437
Implementing Data Processing Extensions	437
TestConnection	438
GetData	438
Deploy Data Processing Extensions in ReportViewer	439
Data Processing Extensions Configuration	439
Configuration with extension assembly.....	439
Samples and demos	440
Locally installed reports	440
Offline demos.....	440

Online demos	440
GitHub demo samples.....	440
Reporting Service application	440
Migrate Report Viewer application	440
Client-side migration.....	440
Adding data visualization scripts.....	441
Server-side migration.....	442
Web API Controller	442
Report export configuration	443
NuGet Packages for JavaScript.....	444
Configure NuGet feed URL.....	444
Online NuGet feed URL.....	444
Offline NuGet feed URL.....	445
Installing NuGet packages.....	446
Install using NuGet Package Manager	446
Install using Package Manager Console.....	446
install specified package in default project	446
install specified package in default project with specified package source	447
install specified package in specified project.....	447
install specified package in default project	447
install specified package in default project with specified Package Source	447
install specified package in specified project.....	447
Upgrading NuGet packages	447
Upgrading using NuGet Package Manager	447
Upgrading using Package Manger Console.....	447
Update specific NuGet package in default project	447
Update all the packages in default project	448
Update specified package in default project with specified package source	448
Update specified package in specified project	448
Update specified Bold Reporting NuGet package	448
Update specified package in default project with specified Package Source.....	448
Update specified package in specified project	448
Upgrading using NuGet CLI	448
update all NuGet packages from config file.....	449
update all NuGet packages from specified Packages Source	449

Update all NuGet packages from config file	449
Update all NuGet packages from specified Packages Source	449
CDN	449
CDN scripts links.....	449
Bold Reports dependency libraries	449
External dependency libraries	452
CDN style sheet links.....	453
Refer local Scripts and CSS when CDN fails	454
JavaScript Report Viewer Reporting Service	456
IReportController	456
Create ASP.NET Web API service	457
Configure Report Viewer Web API.....	460
ReportHelper.....	460
Add Web API Controller	460
Add routing information	462
Enable Cross-Origin Requests	463
Create ASP.NET Core Web API Service	465
List of dependency Libraries	466
Configure Web API.....	467
Add Web API Controller	467
Enable Cross-Origin requests	470
How to queries for Bold Reports ReportViewer	471
Create a SSRS RDL report	471
Bold Reports Report Designer	472
Microsoft SQL Report Builder	472
Visual Studio Report Server template.....	472
Create a RDLC report using business object data source	472
Prerequisites	472
Create business object class	473
Add an RDLC report.....	474
Data source and table configuration wizard.....	474
Adding data visualization scripts.....	480
How to use JavaScript ReportViewer in `Blazor Web Assembly` App application	481
Refer a scripts and CSS.....	482
Adding a ReportViewer	482

Create a Web API service	484
How to use Bold Reports JavaScript ReportViewer with `Blazor Server` App	485
Refer a scripts and CSS	485
Adding a ReportViewer	486
Create a Web API service	487
How to change the exporting document file name based on parameter	488
How to change the data source dynamically	489
How to disable the vertical scrollbar in parameter panel	489
How to customize the Boolean parameter UI with parameter pane	490
See Also	492
How to add null value within DatePicker for DateTime Parameter	492
See Also	493
How to group the values in parameter drop-down	493
See Also	496
How to set Maximum and Minimum value for DateTime Parameter	496
See Also	496
How to pass authorization header for exporting request	496
See also	503
How to pass multiple values using custom data	503
How to clear cache when closing the Report Viewer	506
Javascript	506
Web API	506
Migrate to Report Viewer v2.0 component	507
JavaScript Report Viewer API reference	509
Members	509
Methods	530
destroy()	530
exportReport()	530
fitToPage()	531
fitToPageHeight()	531
fitToPageWidth()	531
getDataSetNames()	532
getParameters()	532
gotoFirstPage()	532
gotoLastPage()	532

gotoNextPage()	533
gotoPageIndex()	533
gotoPreviousPage()	533
print()	534
printLayout()	534
refresh()	534
setParameterBlockVisibility()	535
cancelRendering()	535
Events	535
drillThrough	535
renderingBegin	536
renderingComplete	537
reportError	537
reportExport	538
reportLoaded	538
showError	539
viewReportClick	540
ajaxBeforeLoad	540
ajaxSuccess	541
ajaxError	541
toolbarRendering	542
exportProgressChanged	542
printProgressChanged	543
exportItemClick	544
toolBarItemClick	544
hyperlink	545
reportPrint	546
beforeParameterAdd	546
JavaScript Report Viewer Properties	547
pageSettings	547
Properties	547
parameters `array`	552
Properties	552
exportSettings	556
Properties	556

toolbarSettings.....	563
Properties.....	563
parameterSettings	572
Properties.....	572
dataSources `array`	582
Properties.....	582
customBrandSettings	584
Properties.....	584
Frequently asked questions	587
How can improve the performance and handle the large amounts of data with Report Viewer.....	587
See Also	587
Is Report Viewer compatible with latest version of JQuery library	588
Is the back action from drillthrough report will load the report again with Report Viewer	588
Is possible to change the culture of the date time parameter	588
Is it possible to hide the parameters in Report Viewer	589
Is it possible to hide the export options in Report Viewer	589
Is it possible to load reports providing parameter values at runtime	589
How to Add Web Report Viewer to a Vue application	589
Prerequisites	589
Install the Vue CLI	589
Create a new Vue application	589
Configure Bold Report Viewer in Vue CLI	589
Adding Report Viewer component	590
Adding scripts reference	590
Run the Application.....	594
Bold reporting tools for React.....	595
How to best read this user guide	595
Getting help	595
Bold reporting tools for React.....	595
How to best read this user guide	595
Getting help	595
System Requirements	595
Supported Operating Systems	595
Development Environments	595
Browser Compatibility	595

See Also	596
Overview	596
Add Web Report Viewer to a React application	596
Prerequisites	596
Install the Create React App package	596
Create a new React application	597
Install the Create React Class	597
Install the Bold Reports React package.....	597
Adding scripts reference	597
Adding Report Viewer component	598
Create a Web API service	599
Set a Web API service URL	599
Run the Application.....	600
Deploying the application in production	601
Add Web Report Viewer with Typescript React application.....	601
Prerequisites	601
Install the Create React App Package	601
Create a new React Typescript Application	602
Install Create React Class	602
Install Bold Reports React package	602
Install JQuery package	602
Adding Scripts reference.....	602
Adding Report Viewer component	603
Create a Web API service	604
Set Web API service URL	605
Run the Application.....	606
Deploying the application in production	606
Load SSRS Report Server reports	607
Network credentials for SSRS	609
Set data source credential for shared data sources	610
Change data source connection string	610
See also	611
Add Web Report Viewer to a React Boilerplate application.....	611
Prerequisites	611
React Boilerplate Application	611

Install the Bold Reports React package.....	611
Adding global scripts reference	612
Configuring webpack	612
Adding Report Viewer component	613
Create a Web API service	614
Set a Web API service URL	615
Run the Application.....	616
Deploying the application in production	616
Add Web Report Viewer to a React Boilerplate application.....	616
Prerequisites	616
React Boilerplate TypeScript Application	617
Install the Bold Reports React package.....	617
Adding global scripts reference	617
Configuring webpack	618
Adding Report Viewer component	620
Create a Web API service	621
Set a Web API service URL	621
Run the Application.....	622
Deploying the application in production	623
Render RDLC report	623
Bind data source at client side	623
Bind data source in Web API controller.....	624
Load report as stream	626
View report click	627
Render subreport.....	628
Change subreport path	629
Set subreport parameter	629
Modify subreport data source connection string.....	629
Set subreport data source	630
Load subreport stream	633
Report parameters.....	633
Set parameter at client side.....	633
Set parameters in Web API Controller.....	634
Get report parameter	635
Access the hidden or internal parameter information	636

Set the report parameter visibility in Web API Controller.....	637
Report interaction events	638
Report loaded	638
Report error	641
Show error	642
Drill through.....	643
Hyperlink.....	643
Handle post actions	644
AjaxBeforeLoad.....	644
Add custom header in Ajax request.....	644
Pass custom data in Ajax request	646
AjaxSuccess	647
AjaxError	648
Print report	649
View report in print mode	649
Print in new page	649
Set page orientation and paper size	650
Set report margin.....	650
Set page height and width	651
Print report with images	652
External styles in report printing	653
Show print progress.....	653
Remove empty spaces in printing.....	654
Export report.....	654
Export event handling.....	654
Change Excel and Word export format.....	655
Hide specific export type for report.....	656
PDF export options	657
Export with complex scripts.....	657
PDF conformance.....	657
Add custom PDF fonts.....	657
Word export options.....	658
Word document type.....	658
Word document advance layout for merged cells	658
Protecting Word document from editing	659

Excel export options.....	659
Excel document type.....	659
Excel document advance layout for merged cells	660
Protecting Excel document from editing	660
PowerPoint export options.....	660
CSV export options.....	660
HTML export options	661
Password protect exported document	661
Toolbar customization	662
Hide toolbar items	662
Enable stop option in toolbar	664
Enable export setup option in toolbar.....	664
Enable search text option in toolbar.....	665
Hide toolbar	665
Decide or hide the export option.....	666
Add custom items to the export drop-down	667
Add custom toolbar item	668
Add custom item to exiting toolbar group.....	668
Add new toolbar group	669
Custom actions.....	671
Send a report as an email attachment.....	671
Add email button in Report Viewer	671
Create custom email action	672
Custom properties	674
See Also	674
Textbox custom properties	674
Show or hide toggle icon in text box report item.....	674
Table custom properties	676
Limit number of table records on each page.....	676
Image custom properties.....	676
Angle	677
Chart custom properties	677
Disable legend item interaction.....	677
Set maximum size for chart legend.....	678
Change chart legend shape.....	679

Show or hide chart legend scroller	680
Set range padding for X-axis and Y-axis	681
Control edge label placement in chart axis.....	682
Change the drawing style of a chart column or bar series	684
Change width of the column type series in chart	685
Report custom properties.....	686
Improve performance and handle large amount of data	686
Render large data report faster	686
Reduce memory footprint for large data report	687
Improve report items layout and avoid extra blank pages.....	687
Handling failure in long-running HTTP requests, report processes, and timeouts.....	688
Parameter custom properties.....	689
Show only distinct values in parameter.....	689
Change the dropdown parameter pop up height value	689
Change the dropdown parameter pop up width value	690
Set the minimum date range for the date report parameter.....	691
Set the maximum date range for date report parameter	692
Hide the report parameter block.....	693
Set date and time format for parameter	694
Change time display format for parameter	695
Set time interval in datetime parameter	696
Enable filtering and searching in drop-down report parameter	697
Change date report parameter to display date time picker	698
Export custom properties	700
Improve excel export performance	700
Improve excel export readability	701
Set pdf conformance level	701
Export pdf document with complex script.....	702
Encrypt and secure export documents	702
Protecting Word document from editing	703
Set excel document edit password	704
Set excel document protection.....	705
Responsive layout rendering of React Report Viewer	706
Normal layout	706
Responsive layout	707

Localization of Bold Reports React Report Viewer	707
React Report Viewer Reporting Service	709
IReportController	709
Create ASP.NET Web API service	711
Configure Report Viewer Web API.....	713
ReportHelper.....	713
Add Web API Controller	713
Add routing information	715
Enable Cross-Origin Requests	716
Create ASP.NET Core Web API Service	718
List of dependency Libraries	719
Configure Web API	720
Add Web API Controller	720
Enable Cross-Origin requests	723
How to queries for Bold Reports ReportViewer	724
How to pass multiple values using custom data.....	724
How to clear cache when closing the Report Viewer	726
JavaScript	726
Web API.....	727
Migrate to Report Viewer v2.0 component.....	728
Replacing Report Viewer component	728
Frequently asked questions	729
Is it possible to hide the parameters in Report Viewer	729
Is it possible to hide the export options in Report Viewer	729
How to section for React Bold Reporting Tools	729
How to resolve the Javascript memory heap out issue that occurs while building the React application	729
See also	730
Reporting tools for ASP.NET Core	730
How to best read this user guide	730
Getting help	730
Reporting tools for ASP.NET Core	730
How to best read this user guide	730
Getting help	731
System Requirements	731

Supported Operating Systems	731
Framework	731
Hardware Environments	731
Development Environments	731
Browser Compatibility	731
See Also	731
Overview	731
Display SSRS RDL report in Bold Reports ASP.NET Core Report Viewer	732
Create an ASP.NET Core application	732
List of dependency libraries	733
Refer scripts and CSS.....	734
Tag helper	735
Configure Script Manager	735
Initialize Report Viewer.....	735
Add already created reports	736
Configure Web API	736
Add Web API Controller	736
Set report path and service URL	740
Preview the report	740
See Also	741
Display SSRS RDL report in an ASP.NET Core Razor Pages.....	741
Prerequisites	742
Create an ASP.NET Core application	742
List of dependency libraries	743
Refer scripts and CSS.....	743
Tag helper	745
Configure Script Manager	745
Initialize Report Viewer.....	745
Add already created reports	746
Configure Web API	746
Add Web API Controller	746
Web API with ASP.NET Core Web Application.....	750
Set report path and service URL	751
Preview the report	751
See Also	752

Getting started for MAC system	752
Prerequisites	752
Create ASP.NET core web application	752
List of dependency libraries	756
Refer scripts and CSS.....	758
Tag helper	759
Configure Script Manager	759
Initialize Report Viewer.....	759
Add already created reports	760
Configure Web API	760
Add Web API Controller	760
Enable cross-origin requests	763
Set report path and service URL	764
Preview the report	765
See Also	765
Load SSRS Report Server reports	765
Network credentials for SSRS	767
Set data source credential to shared data sources.....	768
Change data source connection string	768
See also	769
Load SharePoint Server reports	769
Forms credential for SharePoint Server.....	769
Set data source credential to shared data sources.....	769
Render RDLC report	770
Bind data source in Web API controller.....	770
Bind data source with razor view.....	773
View report click	775
Render subreport.....	775
Change subreport path	778
Set subreport parameter	779
Modify subreport data source connection string	779
Set subreport data source	780
Load subreport stream	782
Report parameters.....	783
Set a parameter with razor view.....	783

Set parameters in Web API Controller	784
Get report parameter	784
Set date time display format for date parameter.....	786
Change the Parameter drop-down height and width.....	787
Hide a Parameter scroller	788
Hide a Parameter Pane on load	788
Change the Parameter Item Width and Label Width	788
Access the hidden or internal parameter information	789
Set the report parameter visibility in Web API Controller.....	789
IReportController	790
Extensions	792
See Also	792
Custom Data Extension.....	792
See Also	792
Configure a new data source extension in ASP.NET Core Report Viewer	792
Create ASP.NET Core Report Viewer Application	792
Install data source extension from NuGet	792
Register data source extension.....	794
Run the application.....	795
Report interaction events	795
Report loaded	795
Report error	796
Show error	797
Drill through.....	797
Hyperlink.....	798
Handle post actions	798
AjaxBeforeLoad.....	798
Add custom header to Ajax request	798
Pass custom data in Ajax request	800
AjaxSuccess	801
AjaxError	802
Change Report Viewer default WEB API action with custom endpoint	802
Change report processing endpoint	802
Change export action endpoint	803
Error logging in ASP.NET Core Report Viewer	804

Print report	807
View report in print mode	807
Print in new page	807
Set page orientation and paper size	808
Set report margin.....	808
Set page height and width	809
Print report with images	809
External styles in report printing	810
Show print progress	810
Remove empty spaces in printing.....	812
Export report.....	812
Export event handling.....	812
Change Excel and Word export format.....	813
Hide specific export type for report.....	814
PDF export options	814
Export with complex scripts.....	814
PDF conformance.....	815
Add custom fonts to PDF document.....	815
Word export options.....	816
Word document type.....	816
Word document advance layout for merged cells	816
Protecting Word document from editing	817
Excel export options.....	817
Excel document type.....	817
Excel document advance layout for merged cells	818
Protecting Excel document from editing	818
CSV export options.....	818
Password protect exported document	819
Change image quality in export	820
Custom Actions	820
Add email button	820
Create custom email action	822
Custom properties	824
See Also	824
Textbox custom properties	824

Show or hide toggle icon in text box report item	824
Table custom properties	825
Limit number of table records on each page	825
Image custom properties	826
Angle	826
Chart custom properties	827
Disable legend item interaction	827
Set maximum size for chart legend	827
Change chart legend shape	828
Show or hide chart legend scroller	829
Set range padding for X-axis and Y-axis	830
Control edge label placement in chart axis	832
Change the drawing style of a chart column or bar series	833
Change width of the column type series in chart	834
report custom properties	836
Improve performance and handle large amount of data	836
Render large data report faster	836
Reduce memory footprint for large data report	837
Improve report items layout and avoid extra blank pages	837
Handling failure in long-running HTTP requests, report processes, and timeouts	838
Parameter custom properties	838
Show only distinct values in parameter	838
Change the dropdown parameter pop up height value	839
Change the dropdown parameter pop up width value	840
Set the minimum date range for the date report parameter	841
Set the maximum date range for date report parameter	842
Hide the report parameter block	843
Set date and time format for parameter	844
Change time display format for parameter	845
Set time interval in datetime parameter	846
Enable filtering and searching in drop-down report parameter	847
Change date report parameter to display date time picker	848
Export custom properties	850
Improve excel export performance	850
Improve excel export readability	851

Set pdf conformance level	851
Export pdf document with complex script.....	852
Encrypt and secure export documents	852
Protecting Word document from editing	853
Set excel document edit password	854
Set excel document protection.....	855
Toolbar customization	856
Hide toolbar items	856
Enable stop option in toolbar	857
Enable export setup option in toolbar	857
Enable search text option in toolbar.....	858
Hide toolbar	858
Decide or hide the export option.....	859
Add custom items to the export drop-down	860
Add custom toolbar item	861
Add custom item to exiting toolbar group.....	862
Add new toolbar group	864
Localization of Bold Reports ASP.NET Core Report Viewer	865
Responsive layout rendering of ASP.NET Core Report Viewer	867
Normal layout	867
Responsive layout	868
Limitations	868
RDL specification.....	868
Report layout	868
Expressions.....	868
SSRS.....	868
Image format	868
Samples and Demos.....	868
Locally installed reports	868
Offline demos.....	869
Online demos	869
GitHub demo samples.....	869
Migrate Report Viewer application	869
Client-side migration.....	869
Scripts and CSS references.....	869

Adding data visualization scripts.....	871
Tag helper	872
Control initialization.....	872
Server-side migration.....	873
Web API Controller	873
NuGet Packages for ASP.NET Core.....	873
Configure NuGet feed URL.....	873
Online NuGet feed URL	873
Offline NuGet feed URL.....	874
Installing NuGet packages.....	875
Install using NuGet Package Manager	875
Install using Package Manager Console	875
install specified package in default project	876
install specified package in default project with specified package source	876
install specified package in specified project.....	876
install specified package in default project	876
install specified package in default project with specified Package Source	876
install specified package in specified project.....	876
Upgrading NuGet packages	876
Upgrading using NuGet Package Manager	876
Upgrading using Package Manger Console	876
Update specific NuGet package in default project	877
Update all the packages in default project.....	877
Update specified package in default project with specified package source	877
Update specified package in specified project	877
Update specified Bold Reporting NuGet package	877
Update specified package in default project with specified Package Source.....	877
Update specified package in specified project	877
Upgrading using NuGet CLI	877
update all NuGet packages from config file.....	878
update all NuGet packages from specified Packages Source	878
Update all NuGet packages from config file	878
Update all NuGet packages from specified Packages Source	878
Report page layouting and rendering.....	878
Report measurement options.....	878

Improve text rendering in view and export	878
Improve report items layouting and pagination.....	879
How to Create RDL/RDLC Report	879
Display ssrs rdl report in ASP.NET Core 2.1 application using Report Viewer	880
Create ASP.NET Core application	880
List of dependency libraries	881
Refer scripts and CSS.....	881
Tag helper	882
Configure Script Manager	882
Initialize Report Viewer.....	883
Add already created reports	883
Configure Web API	883
Add Web API Controller	883
Enable cross-origin requests	887
Set report path and service URL	888
Preview the report	889
See Also	889
Display ssrs rdl report in ASP.NET Core 3.1 application using Report Viewer	889
Create ASP.NET Core application	890
List of dependency libraries	890
Refer scripts and CSS.....	891
Tag helper	892
Configure Script Manager	892
Initialize Report Viewer.....	893
Add already created reports	893
Configure Web API	893
Add Web API Controller	893
Enable cross-origin requests	897
Set report path and service URL	898
Preview the report	899
See Also	899
Display SSRS RDL report in Bold Reports ASP.NET Core Report Viewer	899
Create an ASP.NET Core application	900
List of dependency libraries	900
Refer scripts and CSS.....	901

Tag helper	902
Configure Script Manager	902
Initialize Report Viewer.....	903
Add already created reports	903
Configure Web API.....	903
Add Web API Controller	903
Enable cross-origin requests	907
Set report path and service URL	908
Preview the report	909
See Also	909
Create a SSRS RDL report	910
Bold Reports Report Designer	910
Microsoft SQL Report Builder	910
Visual Studio Report Server template.....	910
Create a RDLC report using business object data source	910
Prerequisites	911
Create business object class	911
Add an RDLC report.....	912
Data source and table configuration wizard.....	913
Render data visualization report items.....	919
How to use the custom code with Report Viewer	920
Create a custom code class file	921
Create a assembly reference and add it to Report Viewer.....	921
Display ssrs rdl report in Bold Reports ASP.NET Core Report Viewer	922
Create ASP.NET Core application.....	922
Enable Docker support.....	922
install System.Drawing native dependencies	922
List of dependency libraries	923
Refer scripts and CSS.....	923
Tag helper	924
Configure Script Manager	925
Initialize Report Viewer.....	925
Add already created reports	925
Configure Web API	925
Add Web API Controller	926

Enable cross-origin requests	929
Set report path and service URL	930
Preview the report	930
See Also	931
How to change the exporting document file name based on parameter	931
How to change the data source dynamically	932
How to generate the RDL and RDLC reports programmatically in the report viewer	933
Initialize a report definition	933
Create a Data source for report	934
Create a Dataset for the report	935
Create a report page header	937
Create a report page footer	938
Initialize a report body section	939
Create a Table for the report	942
Create a Chart for the report	944
Render the generated report in ReportViewer	950
How to pass multiple values using custom data	951
How to resolve InvalidOperationException when creating a new application	953
How to load the report from the database	953
From byte array	953
From base64String	953
From string	954
How to clear cache when closing the Report Viewer	954
CSHTML	954
Web API	955
Migrate to Report Viewer v2.0 component	955
Frequently asked questions	957
Does Report Viewer supports .NET Core on Linux Docker	957
Is possible to change the culture of the date time parameter	957
Is it possible to hide the parameters in Report Viewer	958
Is it possible to hide the export options in Report Viewer	958
Is it possible to load reports providing parameter values at runtime	958
Refer Bold Reports Scripts and Themes from BoldReports.Javascript NuGet in .NET Core application ..	958
Find and install the package	958
Installed Location	959

Copy and refer Scripts and CSS files into project.....	960
Bold Report Writer.....	960
Export SSRS RDL Report in Bold Reports ASP.NET Core Report Writer	961
Create an ASP.NET Core application.....	961
List of dependency libraries.....	962
Server side Report Writer changes	963
Client side changes.....	966
Export RDLC Report.....	968
Bind data source in Web API controller.....	968
Export SSRS Report Server Report	973
Network credentials for SSRS	975
Set data source credential to shared data sources.....	975
Export SharePoint Server Report.....	977
Forms credential for SharePoint Server.....	978
Set data source credential to shared data sources.....	978
Export Subreport.....	980
Export RDL Subreport	980
Export RDLC subreport.....	984
Export Parameter Report.....	986
Set report parameters to export file.....	986
Report Writer Events	989
Subreport Processing	989
Report Error	990
Error logging in ASP.NET Core Report Writer	990
Encrypt and Secure Documents.....	994
Password Protected PDF document	994
Password Protected Word document.....	994
Password Protected Excel document	994
Advance Layouts For Merged Cells.....	995
Word document advance layout for merged cells	995
Excel document advance layout for merged cells	995
Change the Default File Format	995
Change the Word document type	996
Change the Excel document type	996
Change the CSV document type	996

PDF Settings	996
Export with complex scripts.....	996
PDF conformance.....	997
Add custom fonts to PDF document.....	997
Password Protected PDF document	998
Excel Settings	998
Excel document type.....	998
Excel document advance layout for merged cells	998
Protecting Excel document from editing	999
Change Excel export format.....	999
Password Protected Excel document	999
Word Settings	999
Word document type.....	999
Word document advance layout for merged cells	1000
Protecting Word document from editing	1000
Password Protected Word document.....	1000
CSV Settings	1001
Export Data Visualization in Core Environment.....	1001
`External Browser - Puppeteer`	1001
`PhantomJS - Classic Tool`	1005
Limitations	1008
Linux	1008
Custom code	1008
Data source	1008
PDF fonts.....	1008
Excel export.....	1008
Image format	1008
How to section for Report Writer component	1009
How to use custom code with Report Writer	1009
Create a custom code class file.....	1009
Create a assembly reference and add it to Report Writer	1009
How to generate and export RDL and RDLC reports programmatically	1010
Initialize a report definition	1010
Create a Data source for report.....	1011
Create a Dataset for the report	1012

Create a report page header.....	1014
Create a report page footer	1015
Initialize a report body section	1016
Create a Table for the report	1019
Create a Chart for the report	1021
Export the generated report in ReportWriter	1027
How to combine the exported PDF with another PDF file	1027
How to load the report from the database	1029
From byte array.....	1030
From base64String	1030
From string.....	1030
Reporting tools for ASP.NET MVC.....	1030
How to best read this user guide	1030
Getting help	1031
Reporting tools for ASP.NET MVC.....	1031
How to best read this user guide	1031
Getting help	1031
System Requirements	1031
Supported Operating Systems	1031
Hardware Environments	1031
Development Environments	1031
Framework.....	1031
Browser Compatibility	1032
See Also	1032
Overview	1032
Display SSRS RDL report in Bold Reports ASP.NET MVC Report Viewer.....	1032
Create an ASP.NET MVC 5 application.....	1032
Configure Report Viewer in an application.....	1034
ReportHelper.....	1034
Add Web API Controller	1035
Add routing information	1036
Set report path and service URL	1037
Preview the report.....	1037
See Also	1038
Load SSRS Report Server reports	1038

Network credentials for SSRS	1040
Set data source credential to shared data sources.....	1041
Change data source connection string	1041
See also	1042
Load SharePoint Server reports	1042
Forms credential for SharePoint Server.....	1042
Set data source credential to shared data sources.....	1042
Render RDLC report	1043
Bind data source at client side	1043
Bind data source in Web API controller.....	1044
Load report as stream	1046
Render subreport.....	1046
Change subreport path	1047
Set subreport parameter	1047
Modify subreport data source connection string.....	1048
Set subreport data source	1048
Load subreport stream	1051
Report parameters.....	1052
Set parameter at client	1052
Set parameters in Web API Controller.....	1052
Get report parameter	1053
Set date time display format for date parameter.....	1054
Change the Parameter drop-down height and width.....	1056
Hide a Parameter scroller	1056
Hide a Parameter Pane on load	1056
Change the Parameter Item Width and Label Width	1057
Access the hidden or internal parameter information	1057
Set the report parameter visibility in Web API Controller.....	1057
IReportController	1058
Extensions	1059
See Also	1059
Custom Data Extension.....	1060
See Also	1060
Configure a new data source extension in ASP.NET MVC Report Viewer	1060
Create ASP.NET MVC Report Viewer Application	1060

Install data source extension from NuGet	1060
Register data source extension.....	1062
Run the application	1062
Report interaction events	1063
Report loaded	1063
Report error	1064
Show error	1065
Drill through	1065
Hyperlink	1066
Handle post actions	1066
AjaxBeforeLoad	1067
Add custom header to Ajax request	1067
Pass custom data in Ajax request	1068
AjaxSuccess	1070
AjaxError	1070
Change Report Viewer default WEB API action with custom endpoint	1071
Change report processing endpoint	1071
Change export action endpoint	1072
Error logging in ASP.NET MVC Report Viewer	1072
Print report	1076
View report in print mode	1076
Print in new page	1076
Set page orientation and paper size	1076
Set report margin.....	1077
Set page height and width	1077
Print report with images	1078
External styles in report printing	1078
Show print progress.....	1079
Remove empty spaces in printing.....	1080
Export report.....	1080
Export event handling.....	1080
Change Excel and Word export format.....	1082
Hide specific export type for report.....	1082
PDF export options	1082
Export with complex scripts.....	1082

PDF conformance.....	1083
Add custom PDF fonts.....	1083
Word export options.....	1084
Word document type.....	1084
Word document advance layout for merged cells	1084
Protecting Word document from editing	1085
Excel export options.....	1085
Excel document type.....	1085
Excel document advance layout for merged cells	1086
Protecting Excel document from editing	1086
CSV export options.....	1086
Password protect exported document	1087
Change file name in export	1088
Change image quality in export	1088
Custom Actions	1089
Add email button	1089
Create custom email action	1091
Custom properties	1093
See Also	1093
Textbox custom properties	1093
Show or hide toggle icon in text box report item	1093
Table custom properties	1094
Limit number of table records on each page.....	1094
Image custom properties.....	1095
Angle	1095
Chart custom properties	1096
Disable legend item interaction.....	1096
Set maximum size for chart legend.....	1096
Change chart legend shape.....	1097
Show or hide chart legend scroller	1098
Set range padding for X-axis and Y-axis	1099
Control edge label placement in chart axis.....	1101
Change the drawing style of a chart column or bar series	1102
Change width of the column type series in chart	1103
Report custom properties.....	1105

Improve performance and handle large amount of data	1105
Render large data report faster	1105
Reduce memory footprint for large data report	1106
Improve report items layout and avoid extra blank pages.....	1106
Handling failure in long-running HTTP requests, report processes, and timeouts.....	1107
Parameter custom properties.....	1107
Show only distinct values in parameter.....	1107
Change the dropdown parameter pop up height value	1108
Change the dropdown parameter pop up width value	1109
Set the minimum date range for the date report parameter.....	1110
Set the maximum date range for date report parameter	1111
Hide the report parameter block.....	1112
Set date and time format for parameter.....	1113
Change time display format for parameter	1114
Set time interval in datetime parameter	1115
Enable filtering and searching in drop-down report parameter	1116
Change date report parameter to display date time picker	1117
Export custom properties	1119
Improve excel export performance	1119
Improve excel export readability	1120
Set pdf conformance level	1120
Export pdf document with complex script.....	1121
Encrypt and secure export documents.....	1121
Protecting Word document from editing	1122
Set excel document edit password.....	1123
Set excel document protection.....	1124
Toolbar customization	1125
Hide toolbar items	1125
Enable stop option in toolbar	1125
Enable export setup option in toolbar.....	1126
Enable search text option in toolbar.....	1126
Hide toolbar	1127
Decide or hide the export option.....	1127
Add custom items to the export drop-down	1127
Add custom toolbar item	1129

Add custom item to exiting toolbar group.....	1131
Add new toolbar group	1133
Localization of Bold Reports ASP.NET MVC Report Viewer	1134
Responsive layout rendering of ASP.NET MVC Report Viewer.....	1136
Normal layout	1137
Responsive layout	1137
Limitations	1138
RDL specification.....	1138
Report layout	1138
Expressions.....	1138
SSRS.....	1138
HTML Formatted Data with Report	1138
Supported HTML Tags.....	1138
Limitations of Cascading Style Sheet Attributes	1138
Samples and Demos.....	1139
Locally installed reports	1139
Offline demos.....	1139
Online demos	1139
GitHub demo samples.....	1139
Migrate Report Viewer application	1139
Client-side migration.....	1139
Scripts and CSS references.....	1139
Adding data visualization scripts.....	1141
Control initialization.....	1142
Server-side migration.....	1143
Web API Controller	1143
Report export configuration	1144
NuGet Packages for ASP.NET MVC	1145
Online NuGet feed URL	1145
Offline NuGet feed URL.....	1146
Installing NuGet packages.....	1147
Install using NuGet Package Manager	1147
Install using Package Manager Console	1147
install specified package in default project	1148
install specified package in default project with specified package source	1148

install specified package in specified project.....	1148
install specified package in default project	1148
install specified package in default project with specified Package Source	1148
install specified package in specified project.....	1148
Upgrading NuGet packages	1148
Upgrading using NuGet Package Manager	1148
Upgrading using Package Manger Console	1148
Update specific NuGet package in default project	1149
Update all the packages in default project	1149
Update specified package in default project with specified package source	1149
Update specified package in specified project	1149
Update specified Bold Reporting NuGet package	1149
Update specified package in default project with specified Package Source.....	1149
Update specified package in specified project	1149
Upgrading using NuGet CLI	1149
update all NuGet packages from config file.....	1150
update all NuGet packages from specified Packages Source	1150
Update all NuGet packages from config file	1150
Update all NuGet packages from specified Packages Source	1150
Frequently asked questions	1150
Is possible to change the culture of the date time parameter	1150
Is it possible to hide the parameters in Report Viewer	1151
Is it possible to hide the export options in Report Viewer	1151
Is it possible to load reports providing parameter values at runtime	1151
How to Create RDL/RDLC Report	1151
Create a SSRS RDL report	1151
Bold Reports Report Designer	1151
Microsoft SQL Report Builder	1152
Visual Studio Report Server template.....	1152
Create a RDLC report using business object data source	1152
Prerequisites	1152
Create business object class	1152
Add an RDLC report.....	1153
Data source and table configuration wizard.....	1154
Adding data visualization scripts.....	1160

How to change the exporting document file name based on parameter	1161
How to change the data source dynamically	1162
How to disable the vertical scrollbar in parameter panel	1163
How to generate the RDL and RDLC reports programmatically in the report viewer	1163
Initialize a report definition	1163
Create a Data source for report	1164
Create a Dataset for the report	1165
Create a report page header	1167
Create a report page footer	1168
Initialize a report body section	1169
Create a Table for the report	1172
Create a Chart for the report	1174
Render the generated report in ReportViewer	1180
How to use Report Viewer in partial view	1181
How to pass multiple values using custom data	1182
How to load the report from the database	1185
From byte array	1185
From base64String	1185
From string	1185
How to clear cache when closing the Report Viewer	1186
CSHTML	1186
Web API	1186
Migrate to Report Viewer v2.0 component	1187
Bold Report Writer	1188
Export SSRS RDL Report in Bold Reports ASP.NET MVC Report Writer	1188
Create an ASP.NET MVC application	1188
List of dependency libraries	1189
Server side Report Writer changes	1190
Client side changes	1193
Export RDLC Report	1195
Bind data source in Web API controller	1195
Export SSRS Report Server Report	1199
Network credentials for SSRS	1202
Set data source credential to shared data sources	1202
Export SharePoint Server Report	1204

Forms credential for SharePoint Server.....	1205
Set data source credential to shared data sources.....	1205
Export Subreport.....	1207
Export RDL Subreport	1207
Export RDLC subreport.....	1210
Export Parameter Report.....	1211
Set report parameters to export file.....	1212
Report Writer Events	1214
Subreport Processing	1214
Report Error	1215
Error logging in ASP.NET MVC Report Writer	1216
Encrypt and Secure Documents.....	1219
Password Protected PDF document	1219
Password Protected Word document.....	1219
Password Protected Excel document	1219
Advance Layouts For Merged Cells.....	1220
Word document advance layout for merged cells	1220
Excel document advance layout for merged cells	1220
Change the Default File Format	1221
Change the Word document type	1221
Change the Excel document type	1221
Change the CSV document type	1221
PDF Settings	1222
Export with complex scripts.....	1222
PDF conformance.....	1222
Embedding Custom PDF fonts	1222
Password Protected PDF document	1223
Excel Settings	1223
Excel document type.....	1223
Excel document advance layout for merged cells	1223
Protecting Excel document from editing	1224
Change Excel export format.....	1224
Password Protected Excel document	1224
Word Settings	1224
Word document type.....	1224

Word document advance layout for merged cells	1225
Protecting Word document from editing	1225
Password Protected Word document.....	1225
CSV Export Settings.....	1226
Export Data Visualization in MVC Environment.....	1226
`External Browser - Puppeteer`	1226
`PhantomJS - Classic Tool`	1230
Limitations	1232
RDL Specification support	1232
Layout Process	1233
Unsupported expression.....	1233
HTML Formatted Data with Report	1233
Supported HTML Tags.....	1233
Limitations of Cascading Style Sheet Attributes	1233
How to section for Report Writer component	1233
How to generate and export RDL and RDLC reports programmatically	1234
Initialize a report definition	1234
Create a Data source for report.....	1234
Create a Dataset for the report	1235
Create a report page header.....	1237
Create a report page footer	1238
Initialize a report body section	1239
Create a Table for the report	1242
Create a Chart for the report	1245
Export the generated report in ReportWriter	1251
How to load the report from the database	1251
From byte array.....	1251
From base64String.....	1251
From string.....	1251
Reporting tools for ASP.NET Web Forms	1252
How to best read this user guide	1252
Getting help	1252
Reporting tools for ASP.NET Web Forms	1252
How to best read this user guide	1252
Getting help	1252

System Requirements	1253
Supported Operating Systems	1253
Hardware Environments	1253
Development Environments	1253
Framework	1253
Browser Compatibility	1253
See Also	1253
Overview	1253
Display SSRS RDL report in Bold Reports ASP.NET Web Forms Report Viewer	1254
Create an ASP.NET Web Forms application	1254
Configure Report Viewer in an application	1255
ReportHelper	1256
Add Web API Controller	1256
Add routing information	1258
Set report path and service URL	1258
Preview the report	1259
See Also	1259
Load SSRS Report Server reports	1259
Network credentials for SSRS	1262
Set data source credential to shared data sources	1263
Change data source connection string	1263
See also	1263
Load SharePoint Server reports	1264
Forms credential for SharePoint Server	1264
Set data source credential to shared data sources	1264
Render RDLC report	1265
Bind data source at client	1265
Bind data source in Web API controller	1266
Render subreport	1268
Change subreport path	1269
Set subreport parameter	1269
Modify subreport data source connection string	1270
Set subreport data source	1270
Load subreport stream	1273
Report parameters	1273

Set parameter at client	1273
Set parameters in Web API Controller	1274
Get report parameter	1275
Set date time display format for date parameter.....	1276
Hide a Parameter scroller	1278
Hide a Parameter Pane on load	1278
Change the Parameter Item Width and Label Width	1279
Access the hidden or internal parameter information	1279
Set the report parameter visibility in Web API Controller.....	1279
Extensions	1280
See Also	1280
Custom Data Extension	1280
See Also	1280
Configure a new data source extension in ASP.NET Web Forms Report Viewer	1280
Create ASP.NET Web Forms Report Viewer Application	1280
Install data source extension from NuGet.....	1281
Register data source extension.....	1282
Run the application.....	1283
Report interaction events	1283
Report loaded	1283
Report error	1285
Show error	1285
Drill through.....	1286
Hyperlink.....	1287
IReportController	1287
Handle post actions	1289
AjaxBeforeLoad.....	1289
Add custom header to Ajax request	1289
Pass custom data in Ajax request	1290
AjaxSuccess	1292
AjaxError	1293
Change Report Viewer default WEB API action with custom endpoint	1294
Change report processing endpoint	1294
Change export action endpoint	1294
Error logging in ASP.NET Web Forms Report Viewer	1295

Print report	1298
View report in print mode	1299
Print in new page	1299
Set page orientation and paper size	1299
Set report margin.....	1300
Set page height and width	1301
Print report with images	1302
External styles in report printing	1302
Show print progress	1303
Remove empty spaces in printing.....	1304
Export report.....	1304
Export event handling.....	1304
Change Excel and Word export format.....	1306
Hide specific export type for report.....	1307
PDF export options	1307
Export with complex scripts.....	1307
PDF conformance.....	1308
Add custom PDF fonts.....	1308
Word export options.....	1309
Word document type.....	1309
Word document advance layout for merged cells	1309
Protecting Word document from editing	1310
Excel export options.....	1310
Excel document type.....	1310
Excel document advance layout for merged cells	1311
Protecting Excel document from editing	1311
CSV export options.....	1311
Password protect exported document	1312
Toolbar customization	1313
Hide toolbar items	1313
Enable stop option in toolbar	1314
Enable export setup option in toolbar.....	1315
Enable search text option in toolbar.....	1315
Hide toolbar	1316
Decide or hide the export option.....	1317

Add custom items to the export drop-down	1317
Add custom toolbar item	1319
Add custom item to exiting toolbar group.....	1320
Add new toolbar group	1321
Custom Actions	1322
Add email button	1322
Create custom email action	1324
Custom properties	1326
See Also	1326
Textbox custom properties	1326
Show or hide toggle icon in text box report item	1326
Table custom properties	1327
Limit number of table records on each page.....	1327
Image custom properties	1328
Angle	1328
Chart custom properties	1329
Disable legend item interaction.....	1329
Set maximum size for chart legend.....	1329
Change chart legend shape.....	1330
Show or hide chart legend scroller	1331
Set range padding for X-axis and Y-axis	1332
Control edge label placement in chart axis.....	1334
Change the drawing style of a chart column or bar series	1335
Change width of the column type series in chart	1336
Report custom properties.....	1338
Improve performance and handle large amount of data	1338
Render large data report faster	1338
Reduce memory footprint for large data report	1339
Improve report items layout and avoid extra blank pages.....	1339
Handling failure in long-running HTTP requests, report processes, and timeouts.....	1340
Parameter custom properties.....	1340
Show only distinct values in parameter.....	1340
Change the dropdown parameter pop up height value	1341
Change the dropdown parameter pop up width value	1342
Set the minimum date range for the date report parameter	1343

Set the maximum date range for date report parameter	1344
Hide the report parameter block	1345
Set date and time format for parameter	1346
Change time display format for parameter	1347
Set time interval in datetime parameter	1348
Enable filtering and searching in drop-down report parameter	1349
Change date report parameter to display date time picker	1350
Export custom properties	1352
Improve excel export performance	1352
Improve excel export readability	1353
Set pdf conformance level	1353
Export pdf document with complex script.....	1354
Encrypt and secure export documents	1354
Protecting Word document from editing	1355
Set excel document edit password	1356
Set excel document protection.....	1357
Localization of Bold Reports ASP.NET Web Forms Report Viewer	1358
Responsive layout rendering of ASP.NET Report Viewer	1359
Normal layout	1359
Responsive layout	1360
Limitations	1360
RDL specification.....	1360
Report layout	1360
Expressions.....	1361
SSRS.....	1361
HTML Formatted Data with Report	1361
Supported HTML Tags.....	1361
Limitations of Cascading Style Sheet Attributes	1361
Samples and Demos.....	1361
Locally installed reports	1362
Offline demos.....	1362
Online demos	1362
GitHub demo samples.....	1362
Migrate Report Viewer application	1362
Client-side migration.....	1362

Scripts and CSS references.....	1362
Adding data visualization scripts.....	1363
Control initialization.....	1364
Server-side migration.....	1365
Web API Controller	1365
Report export configuration	1365
NuGet Packages for ASP.NET Web Forms.....	1367
Configure NuGet feed URL.....	1367
Online NuGet feed URL	1367
Offline NuGet feed URL.....	1368
Installing NuGet packages.....	1369
Install using NuGet Package Manager	1369
Install using Package Manager Console.....	1369
install specified package in default project	1370
install specified package in default project with specified package source	1370
install specified package in specified project.....	1370
install specified package in default project	1370
install specified package in default project with specified Package Source	1370
install specified package in specified project.....	1370
Upgrading NuGet packages	1370
Upgrading using NuGet Package Manager	1370
Upgrading using Package Manger Console.....	1370
Update specific NuGet package in default project	1371
Update all the packages in default project.....	1371
Update specified package in default project with specified package source	1371
Update specified package in specified project	1371
Update specified Bold Reporting NuGet package	1371
Update specified package in default project with specified Package Source.....	1371
Update specified package in specified project	1371
Upgrading using NuGet CLI	1371
update all NuGet packages from config file.....	1372
update all NuGet packages from specified Packages Source	1372
Update all NuGet packages from config file	1372
Update all NuGet packages from specified Packages Source	1372
Frequently asked questions.....	1372

Is possible to change the culture of the date time parameter	1372
Is it possible to hide the parameters in Report Viewer	1373
Is it possible to hide the export options in Report Viewer	1373
Is it possible to load reports providing parameter values at runtime	1373
How to Create RDL/RDLC Report.....	1373
Create a SSRS RDL report.....	1373
Bold Reports Report Designer	1373
Microsoft SQL Report Builder	1374
Visual Studio Report Server template.....	1374
Create a RDLC report using business object data source	1374
Prerequisites	1374
Create business object class	1374
Add an RDLC report.....	1375
Data source and table configuration wizard.....	1376
Adding data visualization scripts.....	1382
How to change the exporting document file name based on parameter	1383
How to change the data source dynamically.....	1384
How to generate the RDL and RDLC reports programmatically in the report viewer	1384
Initialize a report definition	1385
Create a Data source for report.....	1385
Create a Dataset for the report	1386
Create a report page header.....	1388
Create a report page footer	1389
Initialize a report body section	1390
Create a Table for the report	1393
Create a Chart for the report	1396
Render the generated report in ReportViewer	1402
How to disable the vertical scrollbar in parameter panel	1402
How to pass multiple values using custom data.....	1402
How to load the report from the database	1405
From byte array.....	1405
From base64String.....	1405
From string.....	1405
How to clear cache when closing the Report Viewer	1406
CSHTML.....	1406

Web API.....	1406
Bold Report Writer.....	1407
Export SSRS RDL Report in Bold Reports ASP.NET Report Writer	1407
Create an ASP.NET application	1407
List of dependency libraries	1408
Server side Report Writer changes	1409
Client side changes.....	1410
Export RDLC Report.....	1412
Bind data source in Web API controller.....	1412
Export SSRS Report Server Report	1416
Network credentials for SSRS	1418
Set data source credential to shared data sources.....	1419
Export SharePoint Server Report	1420
Forms credential for SharePoint Server.....	1421
Set data source credential to shared data sources.....	1421
Export Subreport.....	1423
Export RDL Subreport	1423
Export RDLC subreport.....	1425
Export Parameter Report.....	1427
Set report parameters to export file.....	1427
Report Writer Events	1429
Subreport Processing	1429
Report Error	1430
Error logging in ASP.NET Web Forms Report Writer	1431
Encrypt and Secure Documents.....	1434
Password Protected PDF document	1434
Password Protected Word document.....	1434
Password Protected Excel document	1434
Advance Layouts For Merged Cells.....	1435
Word document advance layout for merged cells	1435
Excel document advance layout for merged cells	1435
Change the Default File Format	1436
Change the Word document type	1436
Change the Excel document type	1436
PDF Settings	1436

Export with complex scripts.....	1436
PDF conformance.....	1437
Embedding Custom PDF fonts	1437
Password Protected PDF document	1437
Excel Settings	1438
Excel document type.....	1438
Excel document advance layout for merged cells	1438
Protecting Excel document from editing	1438
Change Excel export format.....	1438
Password Protected Excel document	1439
Word Settings	1439
Word document type.....	1439
Word document advance layout for merged cells	1439
Protecting Word document from editing	1440
Password Protected Word document.....	1440
Export Data Visualization in Web Forms Environment.....	1440
`External Browser - Puppeteer`	1440
`PhantomJS - Classic Tool`	1444
Limitations	1446
RDL Specification support	1446
Layout Process	1446
Unsupported expression.....	1446
HTML Formatted Data with Report	1447
Supported HTML Tags.....	1447
Limitations of Cascading Style Sheet Attributes	1447
How to section for Report Writer component	1447
How to generate and export RDL and RDLC reports programmatically	1447
Initialize a report definition	1447
Create a Data source for report.....	1448
Create a Dataset for the report	1449
Create a report page header.....	1451
Create a report page footer	1452
Initialize a report body section	1453
Create a Table for the report	1456
Create a Chart for the report	1458

Export the generated report in ReportWriter	1464
How to load the report from the database	1465
From byte array.....	1465
From base64String.....	1465
From string.....	1465
Reporting tools for WPF.....	1465
How to best read this user guide	1466
Getting help	1466
Reporting tools for WPF.....	1466
How to best read this user guide	1466
Getting help	1466
System Requirements	1466
Supported Operating Systems	1466
Hardware Environments	1466
Development Environments	1466
Framework	1467
See Also	1467
Overview	1467
Display SSRS RDL report in Bold Reports WPF Report Viewer.....	1467
Create the application project	1467
Configure Report Viewer in an application.....	1468
Initialize Report Viewer.....	1468
Add already created reports	1469
Set report path.....	1470
Preview the report.....	1470
See Also	1471
Display SSRS RDL report in WPF .NET Core app using Report Viewer	1471
Create the application project	1471
Configure Report Viewer in an application.....	1472
Initialize Report Viewer.....	1472
Add already created reports	1473
Set report path.....	1474
Preview the report.....	1475
See Also	1475
Load SSRS rdl reports	1475

Set data source credential for shared data sources	1475
Render linked reports	1476
Load SharePoint integrated reports	1477
Set data source credential for shared data sources	1477
Render RDLC report	1478
Load report as stream	1480
Render subreport.....	1481
Load subreport stream	1482
Report parameters.....	1484
Set report parameter	1484
Get report parameter	1485
Report interaction events	1485
Report loaded	1486
Report error	1486
Drill through.....	1487
Hyperlink.....	1487
Error logging in WPF Report Viewer	1488
Print report	1490
View report in print mode	1491
Remove empty spaces in printing.....	1491
Export report.....	1491
PDF export options	1491
Export with complex scripts.....	1491
PDF Conformance	1491
Word export options.....	1492
Word document type.....	1492
Word document advance layout for merged cells	1492
Protecting Word document from editing	1492
Excel export options.....	1493
Excel document type.....	1493
Excel document advance layout for merged cells	1493
Protecting Excel document from editing	1493
PowerPoint export options.....	1493
CSV export options.....	1494
HTML export options	1494

Password protect exported document	1494
Change file name in export	1495
Change image quality in export	1495
Localization of Bold Report Viewer.....	1496
Add the Resource file for the different cultures.....	1496
Assign the values to each culture using key	1496
Assign a Current UI Culture to the application	1497
Set Report Viewer properties	1497
Limitations	1498
RDL specification.....	1498
Report layout	1498
Expressions.....	1498
SSRS.....	1498
Samples and Demos.....	1499
Locally installed reports	1499
Offline demos.....	1499
Migrate Report Viewer application	1499
Client-side migration.....	1499
Control initialization.....	1499
NuGet Packages for WPF	1500
Configure NuGet feed URL.....	1500
Online NuGet feed URL	1500
Installing NuGet packages.....	1500
Install using NuGet Package Manager	1500
Install using Package Manager Console	1501
install specified package in default project	1501
install specified package in default project with specified package source	1501
install specified package in specified project.....	1501
install specified package in default project	1501
install specified package in default project with specified Package Source	1501
install specified package in specified project.....	1501
Upgrading NuGet packages	1501
Upgrading using NuGet Package Manager	1501
Upgrading using Package Manger Console	1502
Update specific NuGet package in default project	1502

Update all the packages in default project	1502
Update specified package in default project with specified package source	1502
Update specified package in specified project	1502
Update specified Bold Reporting NuGet package	1502
Update specified package in default project with specified Package Source.....	1502
Update specified package in specified project	1502
Upgrading using NuGet CLI	1503
update all NuGet packages from config file.....	1503
update all NuGet packages from specified Packages Source	1503
Update all NuGet packages from config file	1503
Update all NuGet packages from specified Packages Source	1503
Custom properties	1503
See Also	1504
Table custom properties	1504
Limit number of table records on each page.....	1504
Report custom properties.....	1504
Improve performance and handle large amount of data	1504
Render large data report faster	1504
Reduce memory footprint for large data report	1505
Improve report items layout and avoid extra blank pages.....	1505
Handling failure in long-running HTTP requests, report processes, and timeouts.....	1506
Parameter custom properties.....	1507
Show only distinct values in parameter.....	1507
Export custom properties	1508
Improve excel export performance	1508
Improve excel export readability	1508
Set pdf conformance level	1509
Export pdf document with complex script.....	1510
Encrypt and secure export documents	1510
Protecting Word document from editing	1511
Set excel document edit password	1512
Set excel document protection.....	1513
How to Create RDL/RDLC Report	1514
Create a SSRS RDL report	1514
Bold Reports Report Designer	1514

Microsoft SQL Report Builder	1515
Visual Studio Report Server template.....	1515
Create a RDLC report using business object data source	1515
Prerequisites	1515
Create business object class	1515
Add an RDLC report.....	1516
Data source and table configuration wizard.....	1517
Frequently asked questions	1523
Bold Report Writer.....	1523
Export SSRS RDL Report in Bold Reports WPF Report Writer.....	1524
Create WPF application.....	1524
List of dependency libraries	1524
Add already created reports	1525
Initialize Report Writer	1525
Set report path.....	1526
Frequently asked questions	1528
How to print the report silently without preview the report in Report Viewer	1529
Reporting tools for UWP	1530
How to best read this user guide	1530
Getting help	1530
Reporting tools for UWP	1530
How to best read this user guide	1530
Getting help	1530
System Requirements	1530
Supported Operating Systems	1531
Hardware Environments	1531
Development Environments.....	1531
Framework	1531
See Also	1531
Overview	1531
Display SSRS RDL report in Bold Reports UWP Report Viewer	1531
Create the application project	1532
Configure Report Viewer in an application.....	1533
Initialize Report Viewer.....	1533
Create Web API service.....	1534

Adding already created report.....	1534
Set report path and service URL	1534
Preview the report	1535
See Also	1535
Load SSRS Report Server reports	1536
Set data source credential for shared data sources	1536
Render linked reports	1537
Load SharePoint Server reports	1537
Set data source credential for shared data sources	1537
Render RDLC report	1538
Render subreport.....	1540
Load subreport stream	1541
Report parameters.....	1543
Set report parameter	1543
Set report parameter in Web API	1544
Get report parameter	1544
Report interaction events	1546
Report loaded	1546
Report error	1546
Drill through	1547
Error logging in UWP Report Viewer	1548
Print report	1550
Remove empty spaces in printing.....	1550
Export report.....	1551
PDF export options	1551
Export with complex scripts.....	1551
PDF Conformance	1551
Add custom PDF fonts.....	1551
Word export options.....	1552
Word document type.....	1552
Word document advance layout for merged cells	1552
Protecting Word document from editing	1552
Excel export options.....	1553
Excel document type.....	1553
Excel document advance layout for merged cells	1553

Protecting Excel document from editing	1553
PowerPoint export options	1554
CSV export options.....	1554
HTML export options	1554
Password protect exported document	1554
Change file name in export	1555
Change image quality in export	1555
Toolbar customization	1556
Decide or hide the export option.....	1556
Handle post actions	1556
RenderingBegin.....	1556
RenderingCompleted	1556
ReportServiceRequestBegin.....	1557
EncryptCredentials.....	1557
Localization of Bold Report Viewer.....	1557
Add Resources file for the different cultures.....	1558
Assign the value to each culture using key	1558
Assign a Current UI Culture to the application	1559
Set Report Viewer properties	1559
Limitations	1560
RDL specification.....	1560
Report layout	1560
Expressions.....	1560
SSRS.....	1560
Samples and Demos.....	1560
Locally installed reports	1560
Offline demos.....	1560
Migrate Report Viewer application	1561
Client-side migration.....	1561
Control initialization.....	1561
Server-side migration.....	1561
Web API Controller	1562
Report export configuration	1562
NuGet Packages for UWP.....	1563
Configure NuGet feed URL.....	1563

Online NuGet feed URL	1563
Installing NuGet packages.....	1564
Install using NuGet Package Manager	1564
Install using Package Manager Console	1564
install specified package in default project	1565
install specified package in default project with specified package source	1565
install specified package in specified project.....	1565
install specified package in default project	1565
install specified package in default project with specified Package Source	1565
install specified package in specified project.....	1565
Upgrading NuGet packages	1565
Upgrading using NuGet Package Manager	1565
Upgrading using Package Manger Console.....	1565
Update specific NuGet package in default project	1566
Update all the packages in default project	1566
Update specified package in default project with specified package source	1566
Update specified package in specified project	1566
Update specified Bold Reporting NuGet package	1566
Update specified package in default project with specified Package Source.....	1566
Update specified package in specified project	1566
Upgrading using NuGet CLI	1566
update all NuGet packages from config file.....	1567
update all NuGet packages from specified Packages Source	1567
Update all NuGet packages from config file	1567
Update all NuGet packages from specified Packages Source	1567
Frequently asked questions	1567
Custom properties	1567
See Also	1567
Table custom properties	1568
Limit number of table records on each page.....	1568
Report custom properties.....	1568
Improve performance and handle large amount of data	1568
Render large data report faster	1568
Reduce memory footprint for large data report	1569
Improve report items layout and avoid extra blank pages.....	1569

Handling failure in long-running HTTP requests, report processes, and timeouts.....	1570
Parameter custom properties.....	1571
Show only distinct values in parameter.....	1571
Export custom properties	1572
Improve excel export performance	1572
Improve excel export readability	1572
Set pdf conformance level	1573
Export pdf document with complex script.....	1574
Encrypt and secure export documents.....	1574
Protecting Word document from editing	1575
Set excel document edit password.....	1576
Set excel document protection.....	1577
UWP Report Viewer Reporting Service.....	1578
Create ASP.NET Web API service	1578
Configure Report Viewer Web API.....	1581
ReportHelper.....	1581
Add routing information	1583
Enable Cross-Origin Requests	1584
Create ASP.NET Core Web API Service	1586
List of dependency Libraries	1587
Configure Web API	1588
Add Web API Controller	1588
Enable Cross-Origin requests	1591
How to Create RDL/RDLC Report.....	1592
Create a SSRS RDL report	1592
Bold Reports Report Designer	1592
Microsoft SQL Report Builder	1593
Visual Studio Report Server template.....	1593
Create a RDLC report using business object data source	1593
Prerequisites	1593
Create business object class	1593
Add an RDLC report.....	1594
Data source and table configuration wizard.....	1595
Resolve does not contains a definition issue	1601
Bold Report Writer.....	1601

Export SSRS RDL Report in Bold Reports UWP Report Writer	1602
Create UWP application.....	1602
Configure Report Writer in an application.....	1603
Adding already created report.....	1603
Initialize Report Writer	1603
Set Report Path	1604
How to queries for Bold Reports ReportViewer	1609
How to dispose the Web ReportViewer object	1610
How to Apply Bold Reports Custom NuGet Packages	1610
Apply Custom NuGet packages using the Visual Studio	1610
Replaced the assemblies manually	1612
How to add Report Viewer in JSP	1614
How to use JSON data as Report Data	1614
How to resolve the unsupported media type error on loading image in ASP.NET Core	1619
See also	1619
How to resolve the image rendering and exporting issue with ASP.NET Core Authentication middleware	1619
How to set the parameter in Web API Controller.....	1621
How to manage reports within application using Bold Reports Report Designer	1621
How to manage the reports with database using Bold Reports Report Designer	1623
How to resolve the Bold Reporting components undefined issue	1625
Script reference.....	1625
Using Bold Reports along with EJ1 components.....	1626
Jquery script reference order	1626
How to resolve the Syncfusion components undefined issue when using the Bold Reporting components	1626
Script reference.....	1626
Using Bold Reports along with EJ2 components.....	1626
How to add a WebAPI Data Processing Extension for Report Designer	1627
How to resolve the image rendering and exporting issue with the ASP.NET MVC Authentication filter	1628
How to change the data source dynamically.....	1629
How to change the data source dynamically using Report Serializer.....	1630
How to connect the Report Server Report Service with application API.....	1631
See also	1631
How to customize the save and open button in the Bold Reports Report Designer.....	1631

Hiding the existing button and add the new buttons.....	1631
Overriding the actions when clicking the inbuilt button	1633
How to pass JSON data in the Bold Reports Report Viewer and Report Writer	1635
Report Viewer	1635
Report Writer	1635
How to serialize the report using Report Serializer	1635
How to customize parameter setting by parameter name	1636
How to use the Report Viewer with camel case serializer settings application	1637
How to have the save alone with report designer.....	1638
How to show alert for users to save the changes before closing the designer	1639
How to add the custom http headers for the Bold Reports Report Viewer and Report Designer	1639
How to generate a PDF from the report in console app.....	1640
How to change the parameter data type	1640
How to configure PostgreSQL Data Processing Extension for Report Viewer	1642
Install PostgreSQL data source extension NuGet	1642
Register PostgreSQL data source extension in application startup	1643
ASP.NET	1643
ASP.NET Core	1644
How to configure WebAPI Data Processing Extension for Report Viewer	1644
Install WebAPI data source extension NuGet	1644
Register WebAPI data source extension in application startup	1645
ASP.NET	1645
ASP.NET Core	1646
How to resolve the Multiple actions were found that matches the request issue when using ASP.NET MVC application.....	1646
How to resolve the image not rendered issue in .NET 6.0 on Linux Environment	1647
Frequently asked questions	1648
Troubleshooting Embedded Reporting Tools	1649
Controlled folder access and the protected file usage	1651
Steps to allow Bold Reports Embedded Reporting Tools to access protected folders.....	1652
References	1657
Silent installation	1657
Is Bold Reports Embedded Reporting Tools or Viewer SDK requires additional licensing when running application with Azure App service.....	1658
How to provide the permission for user to access the SSRS Report Server reports	1658

Site Setting of Report Server	1658
Permission of user.....	1658
Folder	1659
Permission for Folder	1659
Permission of user with Folder	1659
How to use the ReportViewer along with EJ2 controls	1660
Styles	1660
Scripts.....	1660
Script compatibility for ASP.NET Core	1662
Using EJ2 components with JavaScript	1662
See also	1662
How to use the Bold Reports with ASP.NET Core 3.x	1663
What are all the SSRS versions are supported in Bold Reports	1663
Centralized report authoring	1664
Is Bold Reports have a centralized report authoring system.....	1664
Get the user from database and pass the user as parameter for filtering the data from database	1664
Does Bold Reports Embedded Reporting Tools support .NET Core and Docker on Linux.....	1665
install System.Drawing native dependencies	1666
For .NET 6.0 or above version on Linux	1666
How to use the Bold Reports with ASP.NET Core 3.1	1666
Does Bold Reports have any licensing procedure for deployment.....	1667
Is it possible to create the RDLC reports using the business object data source in Report Designer	1667
Can you use the Report Designer component from Syncfusion community license	1667
How to add a header authorization for report service.....	1668
Can the Bold Reports be used with ASP.NET Core on Linux and macOS.....	1668
Linux.....	1668
For .NET 6.0 or above version on Linux	1669
macOS	1669
Is it possible to use the .NET class objects in ReportViewer.....	1669
See Also	1670
Is theme studio available for Bold Reports to generate the custom theme	1671
Is it possible to create a WPF .Net Core application with Bold Report Viewer	1671
Why should you migrate to Bold Reports from Syncfusion Report Platform.....	1672
See also	1672
Is it possible to load the million records report with Report Viewer and Report Writer	1672

Can specify ReportServer URL in Web API controller	1672
Difference between Report Service URL and Report Server URL	1673
Report Service URL.....	1673
Report Server URL.....	1673
How to use the Bold Reports along with EJ1 controls	1673
Can use Java Spring Framework for Bold Reports	1674
What are all the supported HTML tags in Report Viewer.....	1674
Supported HTML Tags.....	1674
Limitations of Cascading Style Sheet Attributes	1674
How to hide the license watermarks	1675
See Also	1675
Unobtrusive.....	1675
How to use report viewer and designer with unobtrusive JavaScript Enabled	1675
How to get a dataset name in Web API Service	1676
ASP.NET	1676
ASP.NET Core	1677
Where can i find the installation error and debug log files	1678
Is it possible to showing the selected all text instead of showing all values from a parameter	1678
How to avoid the extra blank pages in print and print preview	1679
Horizontal usable area	1680
Vertical usable area	1680
How to use a frame border-image for PDF exported documentation	1680
What are the differences in Embedded Reporting and Report Viewer SDK	1683
See also	1683
Does Bold Report Viewer use SSRS Report processing.....	1683
See also	1684
Why should not use Produces attribute in web API controller	1684
How to convert Crystal Reports to RDL for Bold Reports	1685
Is it possible to use a data table in Bold Reports	1685
Why http authorization headers are not being set for all requests	1686
CDN links for Localization and Culture.....	1689
ar-AE.....	1689
cs-CZ.....	1689
da-DK.....	1689
de-DE.....	1689

en-GB	1689
en-US.....	1690
es-ES.....	1690
fa-IR.....	1690
fi-FI	1690
fr-FR.....	1690
he-IL	1690
hr-HR	1690
hu-HU	1691
it-IT	1691
ja-JP	1691
ko-KR	1691
ms-MY	1691
nb-NO.....	1691
nl-NL.....	1691
pl-PL	1692
pt-PT.....	1692
ro-RO.....	1692
ru-RU	1692
sk-SK.....	1692
sv-SE	1692
tr-TR	1692
vi-VN.....	1693
zh-CN.....	1693
How to resolve the Routing issue in ASP.NET MVC application	1693
How to print the report using the external button	1694
See also	1694
How to deploy PhantomJS WebKit manually	1694
Overview	1695
Key features	1695
Troubleshoot installation or upgrade problems	1695
Allow controlled folder access and the protected file usage	1695
Steps to allow Bold Reports Report Viewer SDK to access protected folders.....	1695
References	1701
Troubleshooting Report Viewer SDK sample browsers	1701

RV0001	1703
Description	1703
Solution	1703
RV0009	1703
Description	1703
Solution	1703
RV0010	1704
Description	1704
Solution	1704
RV0013	1704
Description	1704
Solution	1704
RV0014	1704
Description	1704
Solution	1704
RV0023	1705
Description	1705
Solution	1705
RV0024	1705
Description	1705
Solution	1705
RV0034	1705
Description	1705
Solution	1705
RV0051	1705
Description	1705
Solution	1705
RV0062	1706
Description	1706
Solution	1706
RV0072	1706
Description	1706
Solution	1706
RV0073	1706
Description	1706

Solution	1706
RV0075	1706
Description	1706
Solution	1706
RV0083	1706
Description	1707
Solution	1707
RV0084	1707
Description	1707
Solution	1707
RV0089	1707
Description	1707
Solution	1707
RV0152	1707
Description	1707
Solution	1707
RV0156	1707
Description	1707
Solution	1707
RV0164	1708
Description	1708
Solution	1708
RV0167	1708
Description	1708
Solution	1708
RV0175	1708
Description	1708
Solution	1708
RV0293	1708
Description	1708
Solution	1708
RV0331	1708
Description	1708
Solution	1709
RV0334	1709

Description	1709
Solution	1709
RV0347	1709
Description	1709
Solution	1709
RV0362	1709
Description	1709
Solution	1709
RV0394	1709
Description	1709
Solution	1710
RV0444	1710
Description	1710
Solution	1710
RV0445	1710
Description	1710
Solution	1710
RV0446	1710
Description	1710
Solution	1710
RV0451	1710
Description	1710
Solution	1710
RV0488	1710
Description	1710
Solution	1711
RV0534	1711
Description	1711
Solution	1711
RV0535	1711
Description	1711
Solution	1711
RV0537	1711
Description	1711
Solution	1711

RV0538.....	1711
Description	1711
Solution	1712
RV0539.....	1712
Description	1712
Solution	1712
RV0540.....	1712
Description	1712
Solution	1712
RV0542.....	1712
Description	1712
Solution	1712
RV0543.....	1712
Description	1712
Solution	1712
RV0544.....	1712
Description	1713
Solution	1713
RV0546.....	1713
Description	1713
Solution	1713
RV0547.....	1713
Description	1713
Solution	1713
RV0548.....	1713
Description	1713
Solution	1713
RV0549.....	1713
Description	1713
Solution	1713
RV0552.....	1713
Description	1713
Solution	1714
Bold Reports Theme Studio	1714
Create a new theme.....	1714

Choose an existing Bold Reports theme	1714
Filter Bold Reports control.....	1715
Customize property	1716
Download the customized theme.....	1718
Import the customized theme	1719
Use the downloaded CSS in the application	1720
Referring minified CSS.....	1720
Referring unminified CSS	1721
Overview	1722
Overview	1722
System Requirements	1722
Hardware Environments	1722
Development Environments	1722
See Also	1723
Download and installation of Bold Reports Source Code Add-On.....	1723
Register and Download	1723
Step-by-Step Installation.....	1724
See Also	1730
Generate Nuget	1730
Build	1730
Folder Structure	1731
Generate Nuget-packages	1731

Overview

This section describes the components, nuget packages, assemblies, samples, and demos included in Report Viewer SDK installer.

[Prerequisites](#)

Describes the system requirements of Report Viewer SDK.

[Download and installation](#)

Provides information on login, download, and installation of Report Viewer SDK.

[Sample and demos in installation](#)

Explains about the Report Viewer SDK demos and view their source code applications.

[Nugets and assemblies](#)

Provides the list of Nugets and assemblies provided in Report Viewer SDK.

Overview

This section describes the components, nuget packages, assemblies, samples, and demos included in Report Viewer SDK installer.

[Prerequisites](#)

Describes the system requirements of Report Viewer SDK.

[Download and installation](#)

Provides information on login, download, and installation of Report Viewer SDK.

[Sample and demos in installation](#)

Explains about the Report Viewer SDK demos and view their source code applications.

[Nugets and assemblies](#)

Provides the list of Nugets and assemblies provided in Report Viewer SDK.

System Requirements

This topic describes the software and hardware requirements to run Report Viewer SDK.

Hardware Environments

The following hardware environments are necessary to run the Report Viewer SDK.

- 2.4 GHZ or faster, 32 bit or 64 bit processor.
- 8 GB RAM for 32 bit or 2 GB RAM for 64 bit.
- 1.68 GB Hard Disk space (Installation files).

Development Environments

The following development environments are necessary to run the Report Viewer SDK.

- [Microsoft Visual Studio 2010](#) or [later](#)
- Internet Information Services (IIS) 7.0+

Supported Operating Systems

- Windows 7+
- Windows Server 2008 R2+

Browser Compatibility

- IE 9+
- Microsoft Edge
- Mozilla Firefox 22+
- Chrome 17+
- Opera 12+
- Safari 5+

See Also

- [Licensing procedure for deployment](#)

Download and installation of Report Viewer SDK

This section briefly describes the steps involved in the download and installation of the Report Viewer SDK setup.

Register and download the Report Viewer SDK

1. Go to the Report Viewer SDK [download](#) page.
2. Fill the form input details to register a new account or sign in with an existing account.


Get Started with Bold Reports
Free 15-day trial. No credit card required.

Already have a Bold ID or Syncfusion account? [Sign in](#)

First name Last name


Please enter first name

Business email address

Password 

Company name

Phone number

Business ID 

Site name: .boldreports.com

☐ I have read and agree to Syncfusion's [privacy policy](#), [cookie policy](#), and Bold Reports [terms of use](#). [Read More](#).

Start Creating Reports

3. Once the portal has been created, you will be redirected to your downloads page.

Bold Reports by Syncfusion

Products

- Cloud Reporting
- Enterprise Reporting
- Embedded Reporting
- Report Viewer SDK**
- Data Integration
- Standalone Report Designer



Support

Report Viewer SDK

The Report Viewer component includes reporting tools such as the Report Viewer and Report Writer for viewing reports. It does not include a server.

[Request more licenses](#)

Download and install Report Viewer SDK Released on November 10, 2021 | Version: 3.5.23

Report Viewer SDK	 EXE 625.00 MB Download Now Checksum Value	 ZIP 625.00 MB Download Now Checksum Value
--------------------------	---	---

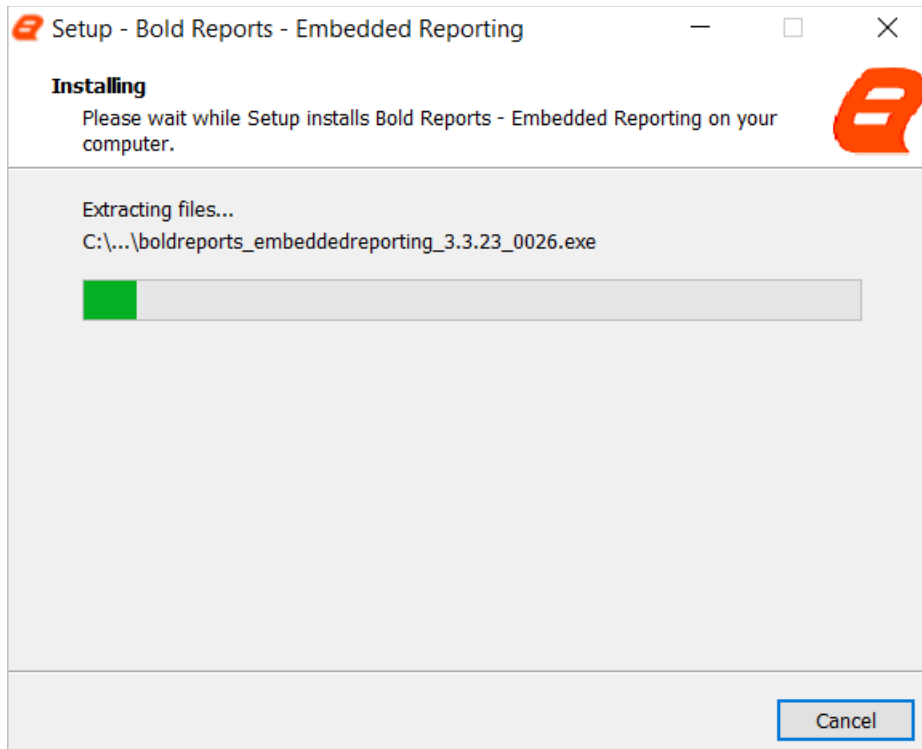
Subscriptions [>](#)

4. Select the file type to download by clicking the download button.



Online Installation

1. Once the setup is downloaded, run the installer from the saved location by clicking the **Run** button or by double-clicking the EXE file.



2. Sign-in with your registered e-mail address to unlock the setup.

Bold Reports

Bold Reports - Embedded Reporting

Version: 3.3.23 Created on: 11/30/2021

Thank you for choosing Bold Reports - Embedded Reporting. Please enter the required information to proceed further.

Login to install Offline install

Email

Password

[Forgot password?](#)

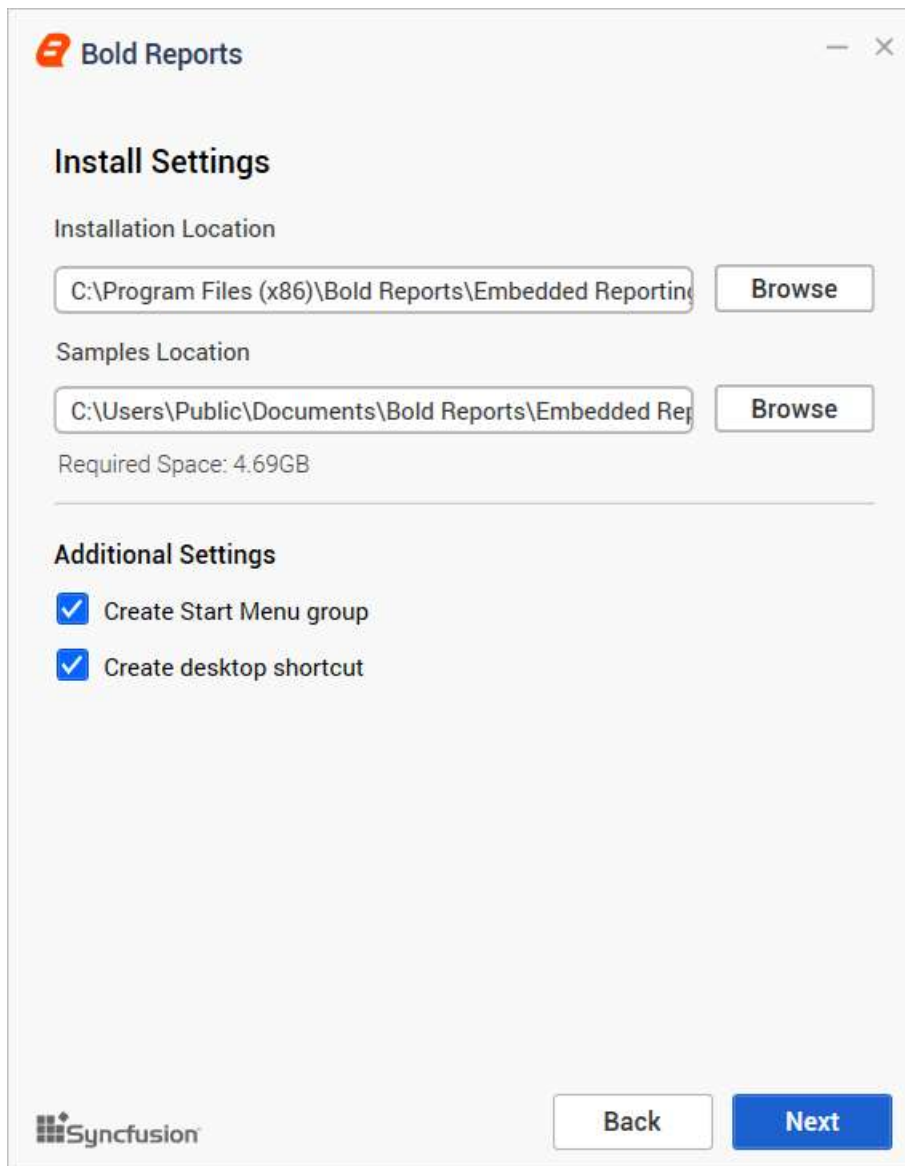
or Sign in with

Microsoft ADFS

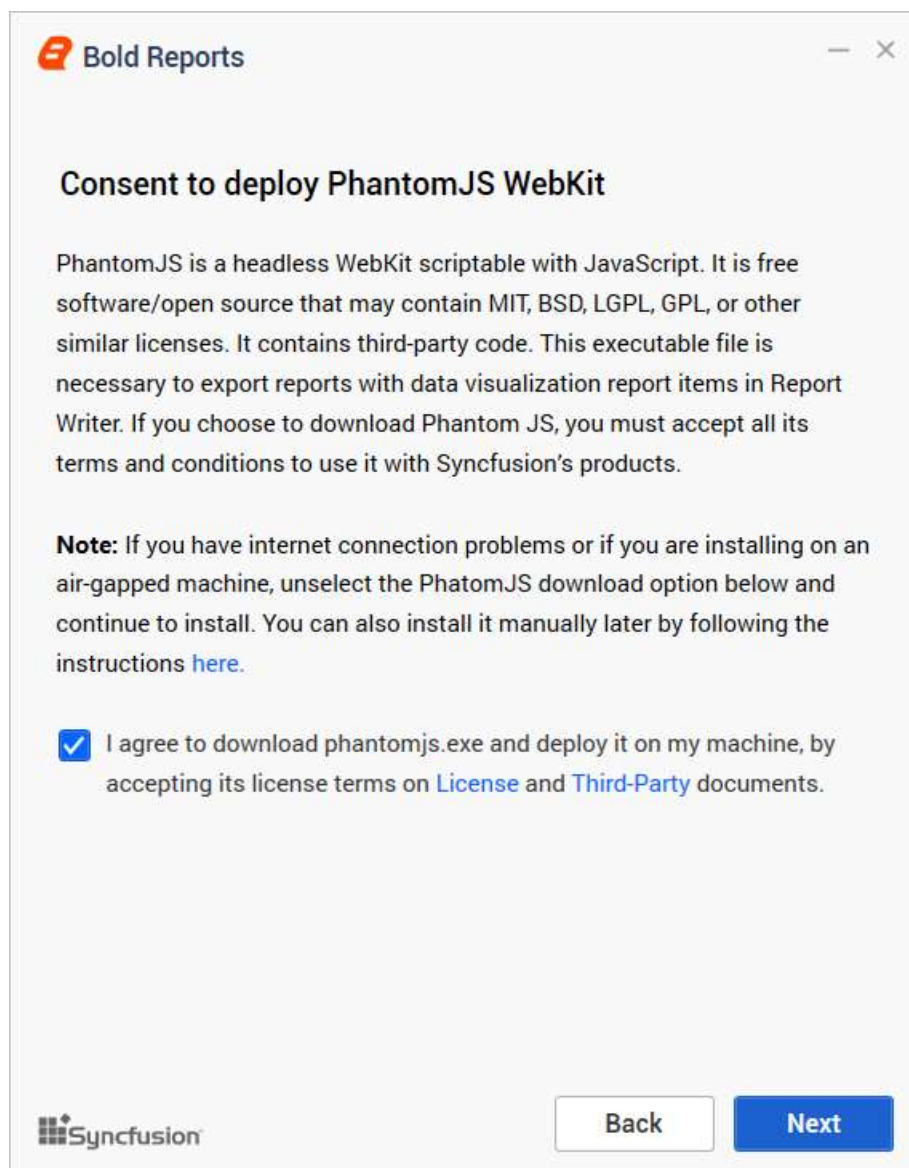
☐ I agree to the [License Terms and Conditions](#)

Syncfusion Next

3. Accept the Report Viewer SDK License Agreement by clicking the **License Terms and Conditions**.
4. Choose the location where you would like to install the Report Viewer SDK setup and samples. If you want to perform additional tasks like start menu and desktop shortcut creation, you can check the options. Otherwise, you can uncheck it and click **Next**.



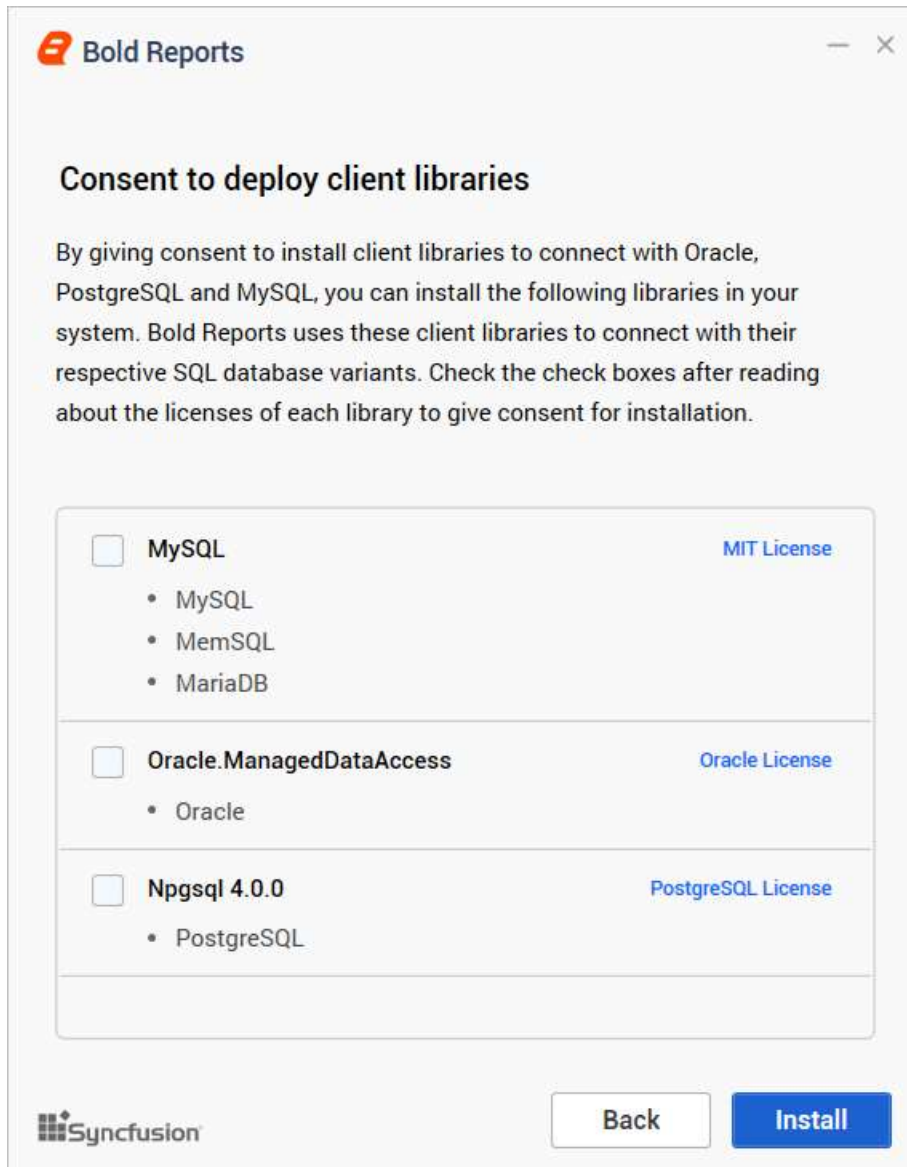
5. Read and accept the license and third-party terms and conditions by checking the options for installing PhantomJS and click **Next**.



PhantomJS is a headless WebKit scriptable with JavaScript. This is free software or open source that may contain MIT, BSD, LGPL, GPL, or other similar licenses. It contains third-party code. This executable file is necessary to export reports with data visualization report items in Report Writer. If you choose to download Phantom JS, you must accept all its terms and conditions to use it with Syncfusion's products.

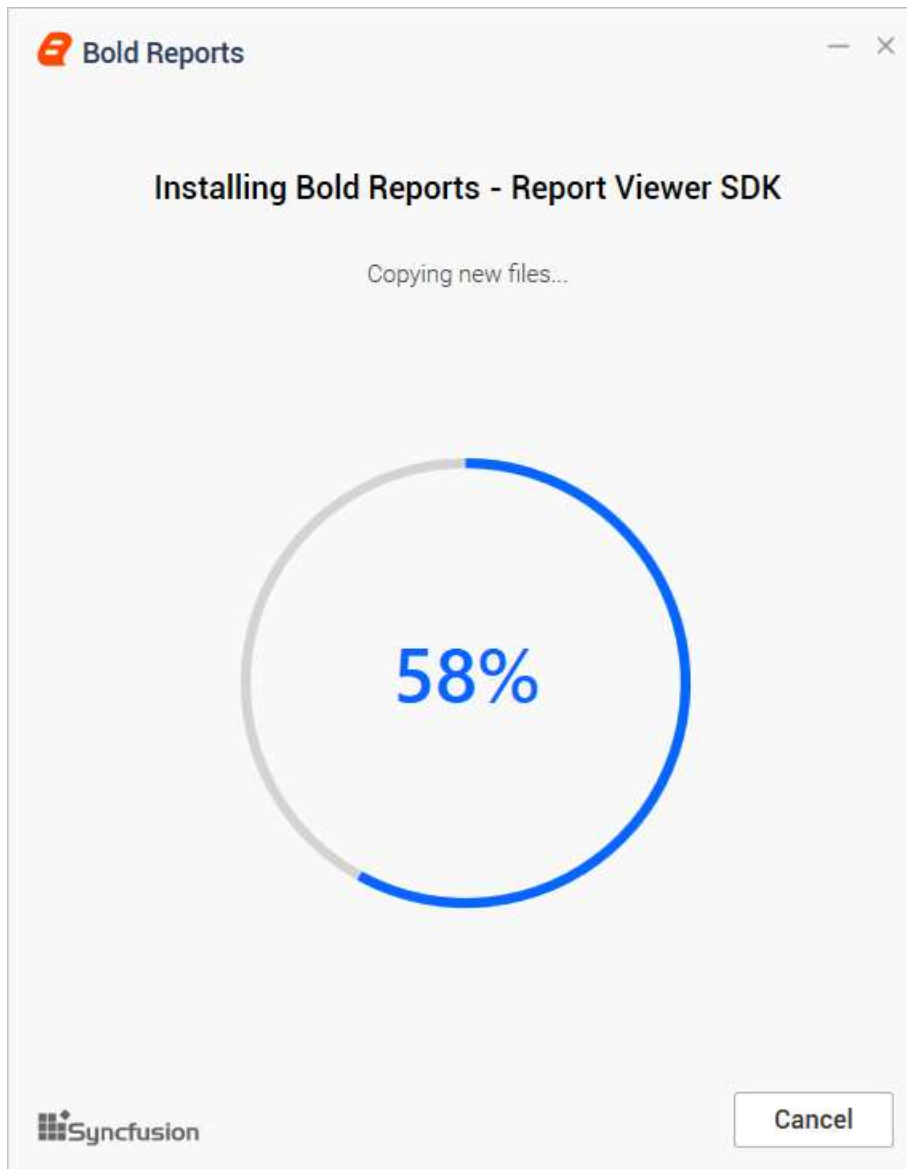
If you have internet connection problems or if you are installing on an air-gapped machine, unselect the PhantomJS download option and continue to install. You can also install it manually later.

6. Read and accept the license to ship the selected client libraries and click **Install**.

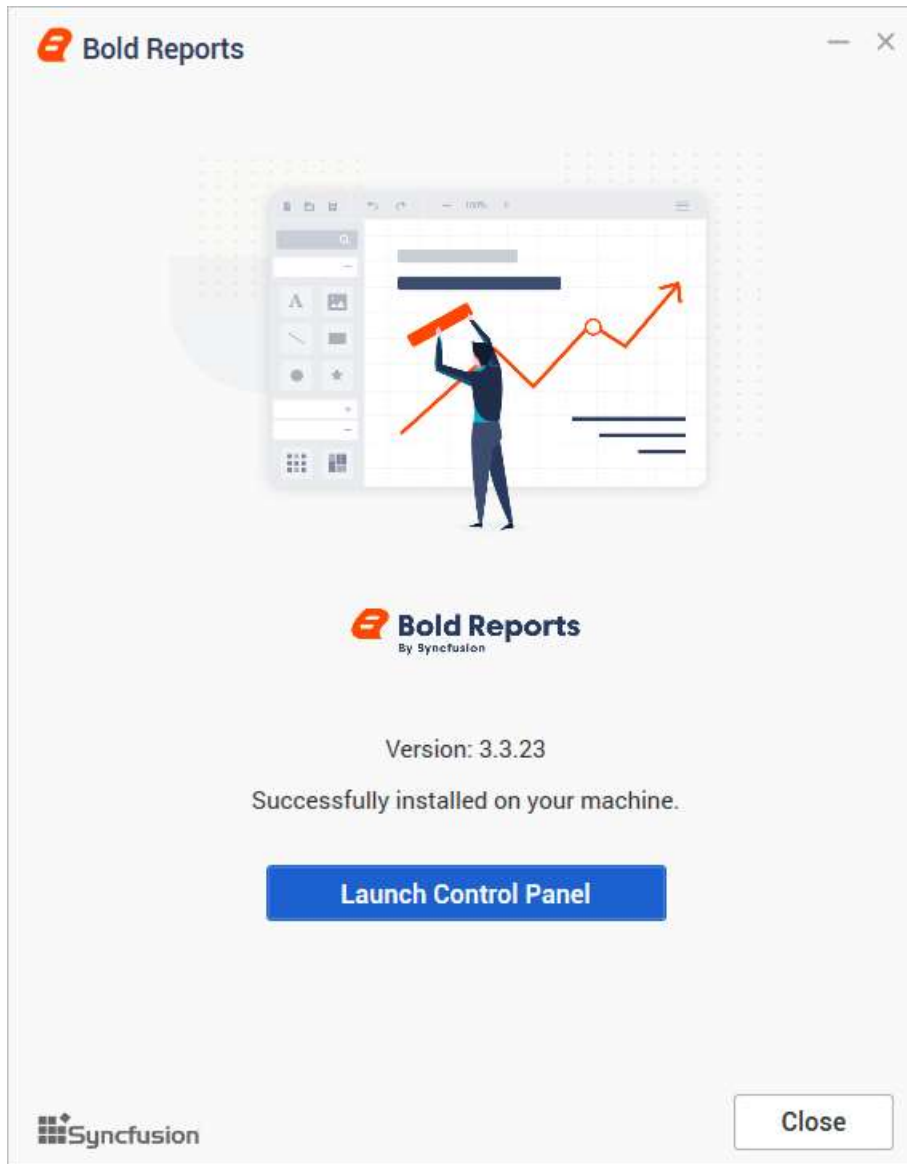


Bold Reports Embedded Edition uses client libraries such as Oracle, PostgreSQL, and MySQL to connect with their respective SQL database variants. Check the license of each library to give consent for installation. The selected client libraries alone will be shipped with the product.

7. Now the installation begins. You can cancel the installation at anytime by pressing **Cancel**, if you prefer.

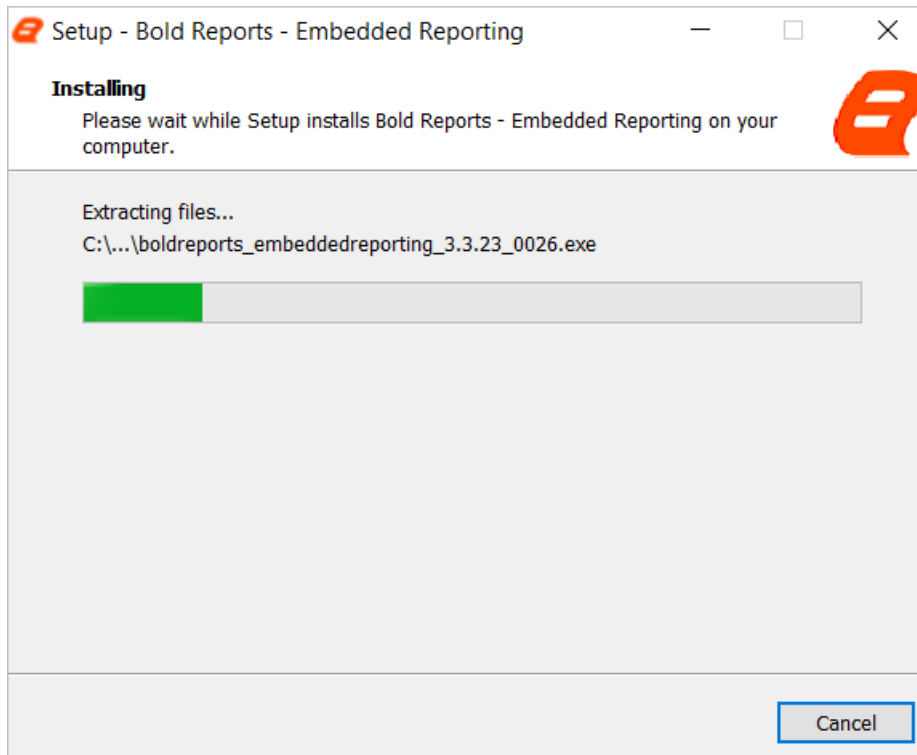


8. On successful installation, the below screen appears. Click the **Launch Control Panel** button to run the Sample Browser or Explore Samples.

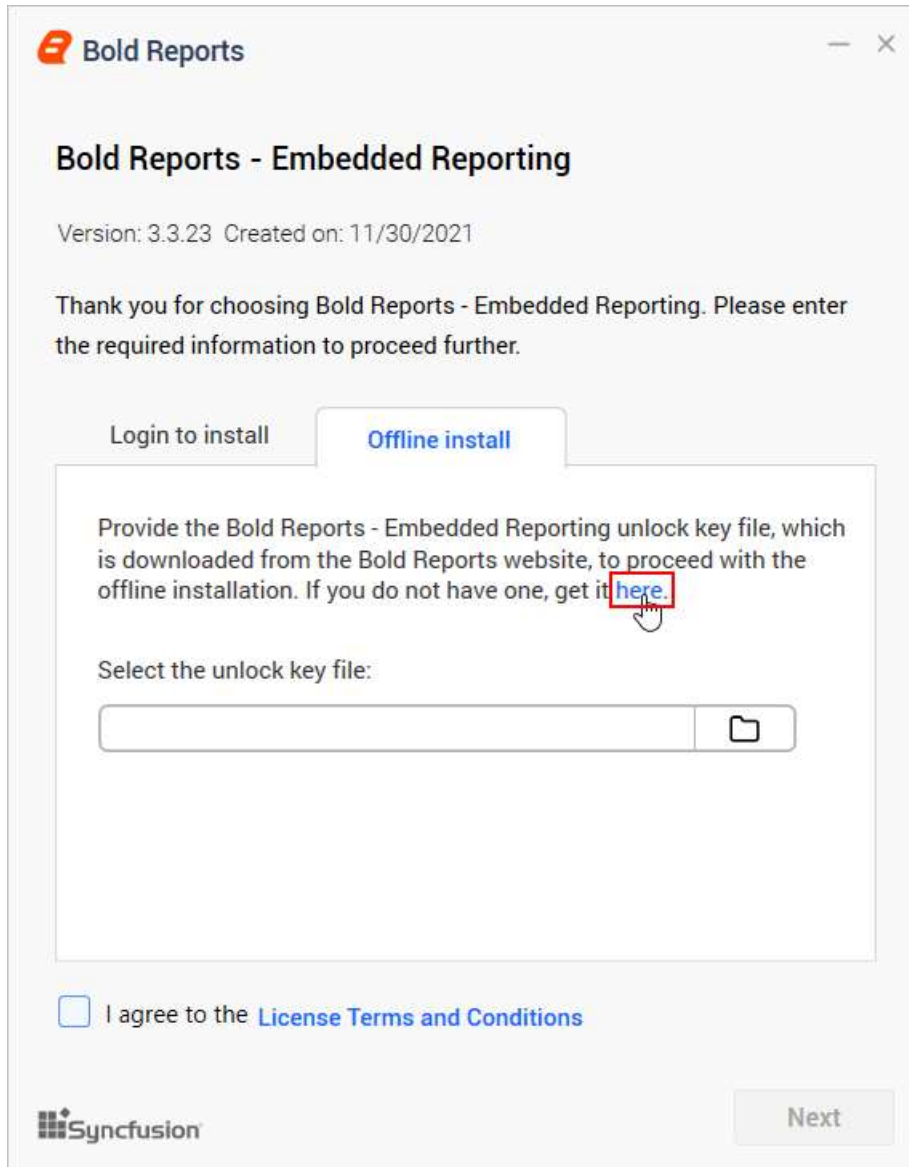


Offline Installation

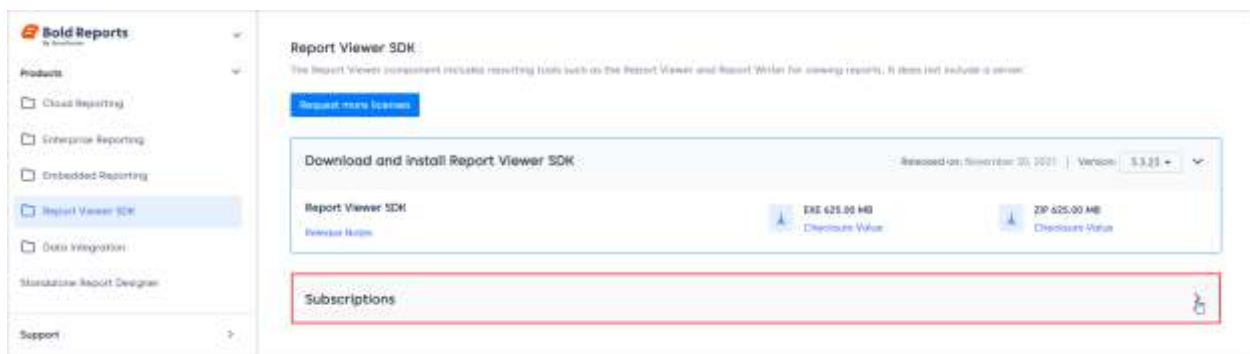
1. Once the setup is downloaded, run the installer from the saved location by clicking the **Run** button or by double-clicking the EXE file.



2. For offline installation, unlock the setup using an offline key. If you don't have the setup unlock key, you can download it by clicking the [here](#) option as shown in the below image.



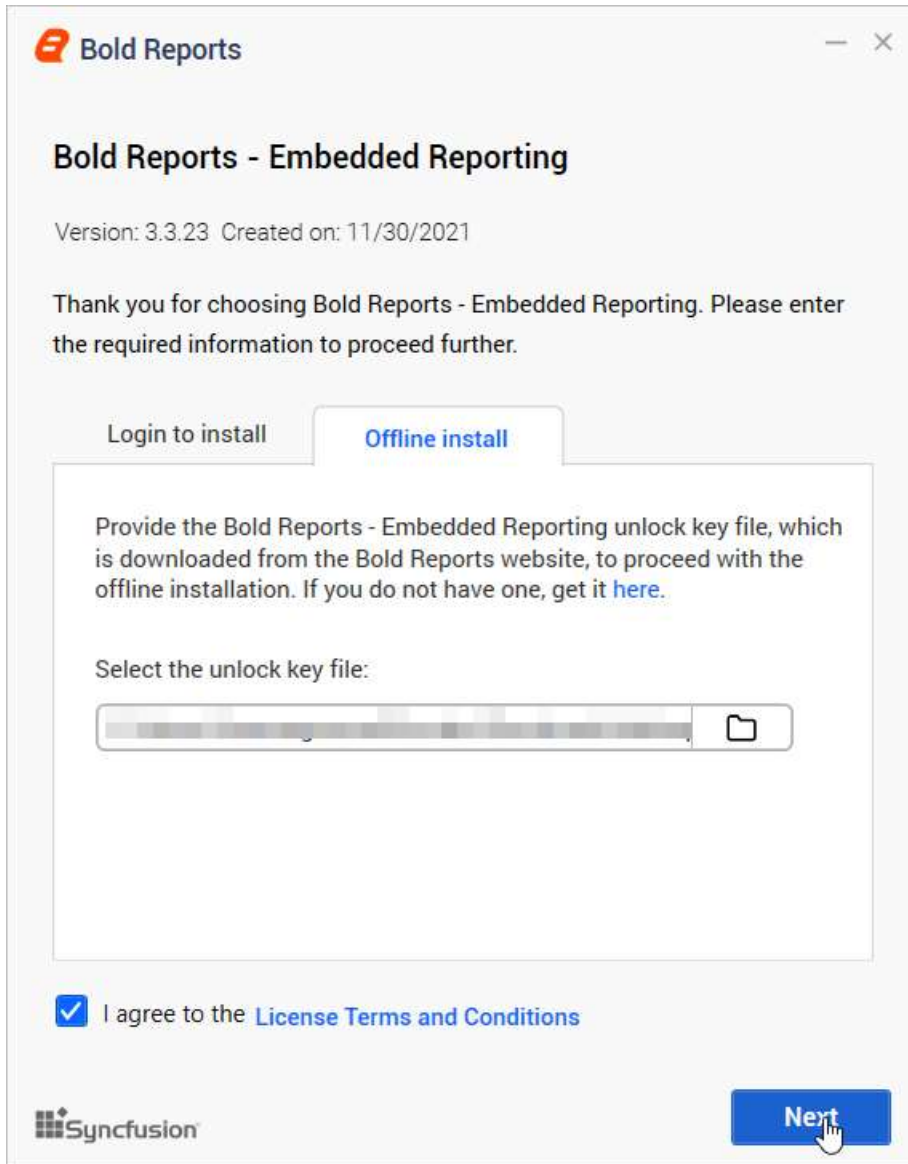
3. It will redirect to the Report Viewer SDK page and then expand the **Subscriptions** section.



4. Then, click **Get Unlock Key File**. A file will be downloaded with the file extension **.lic**.



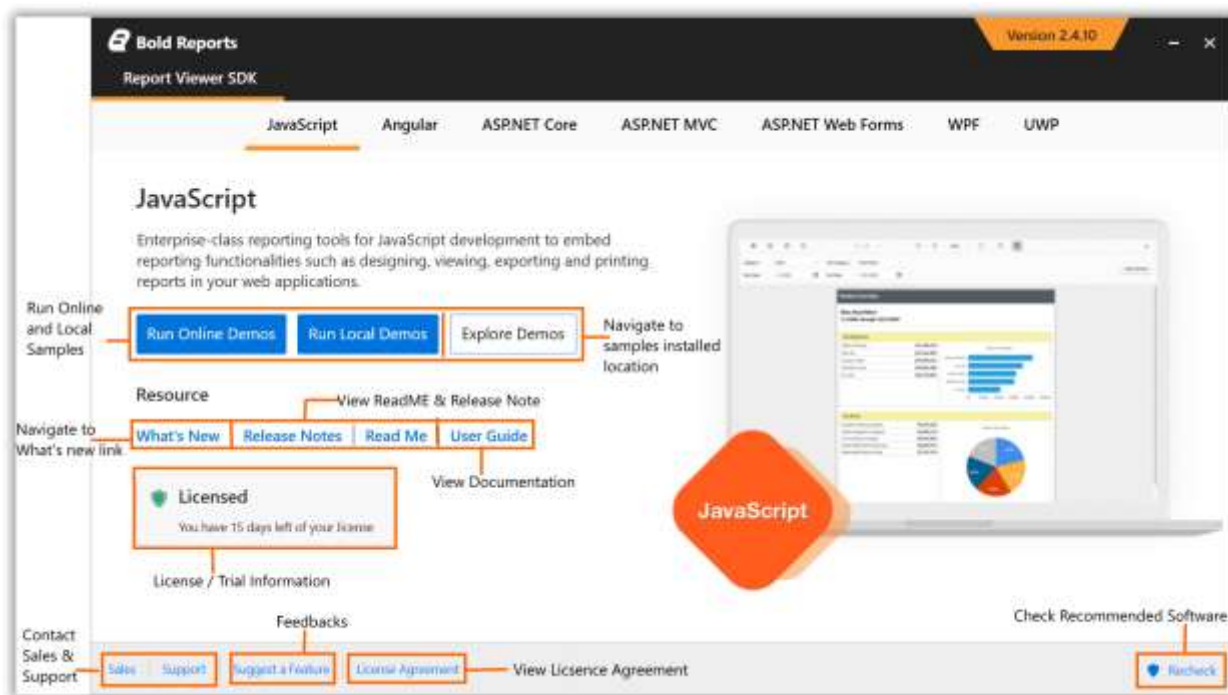
5. To unlock the setup, upload the downloaded .lic file. Then, enable the license terms and conditions checkbox and click **Next**.



6. Then, follow the steps from 4 to 8 as in the [Online Installation](#).

Report Viewer SDK samples and demos

Report Viewer SDK provides control panel to access samples and demos of Reporting components for the JavaScript, Angular, ASP.NET CORE, ASP.NET, ASP.NET MVC, JSP, PHP, WPF and UWP platforms. To launch the Reporting Tools control panel, click the Desktop Shortcut or Start Menu -> Report Viewer SDK.



View the product demos

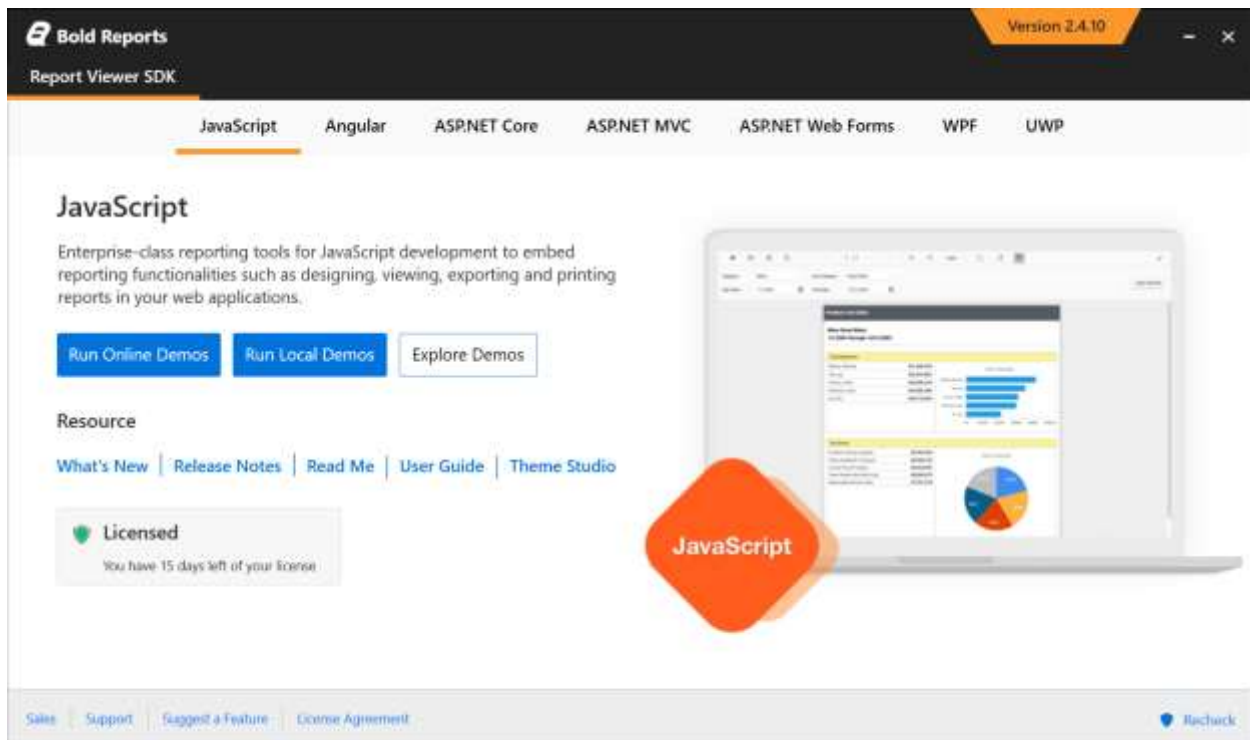
You can access the Report Viewer SDK demos using the following steps.

1. In the Report Viewer SDK control panel, choose the desired platform (JavaScript, Angular etc.).
2. Click any of the following options to know more about the selected platform.
 - Run Local Demos - To run the locally installed samples.
 - Explore Demos - Opens the local installed location.
 - What's New - To view the **What's New** content.
 - Release Notes - To view the **Release Notes** content.
 - Read Me -To view the **Read Me** content.
 - User Guide - To view the user guide documentation for the respective platform product.

JavaScript

To preview the JavaScript demos, follow these steps:

1. Choose JavaScript platform in Report Viewer SDK control panel.



2. Click **Run Local Demos** to view the locally installed demos.
3. You can explore the demo sample application from the following installed location by clicking **Explore Demos** option.

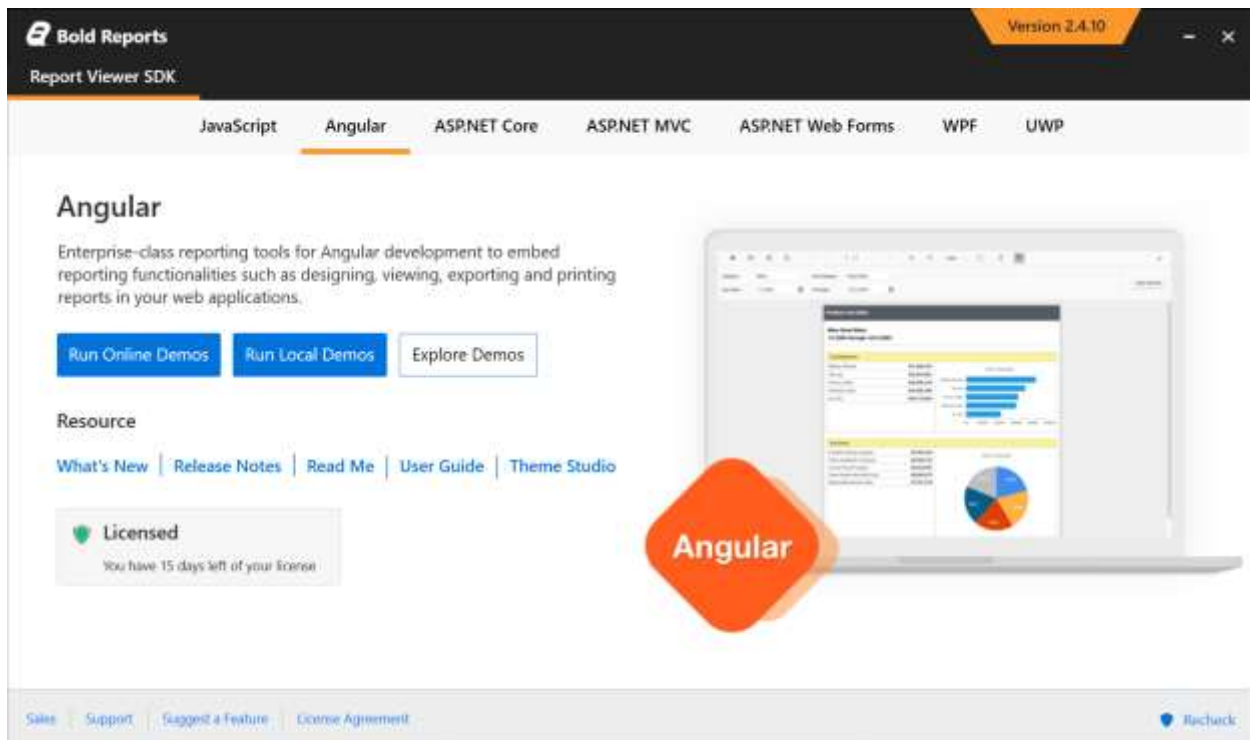
%localappdata%\Bold Reports\Embedded Reporting\Samples\JavaScript

4. You can view the online demos from [here](#).

Angular

To preview the Angular demos, follow these steps:

1. Choose Angular platform in Report Viewer SDK control panel.



2. Click **Run Local Demos** to view the locally installed demos.
3. You can explore the demo sample application from the following installed location by clicking **Explore Demos** option.

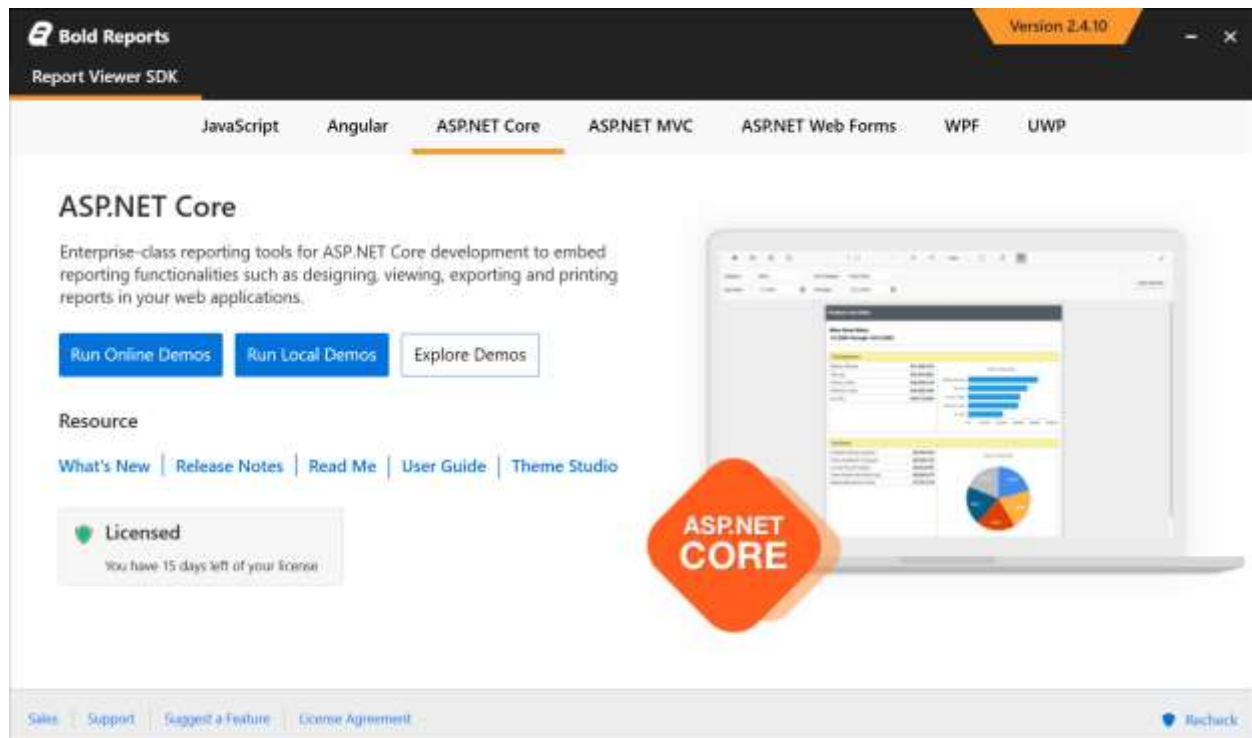
%localappdata%\Bold Reports\Embedded Reporting\Samples\Angular

4. You can view the online demos from [here](#).

ASP.NET CORE

To preview the ASP.NET Core demos, follow these steps:

1. Choose ASP.NET Core platform in Report Viewer SDK control panel.



2. Click **Run Local Demos** to view the locally installed demos.
3. You can explore the demo sample application from the following installed location by clicking **Explore Demos** option.

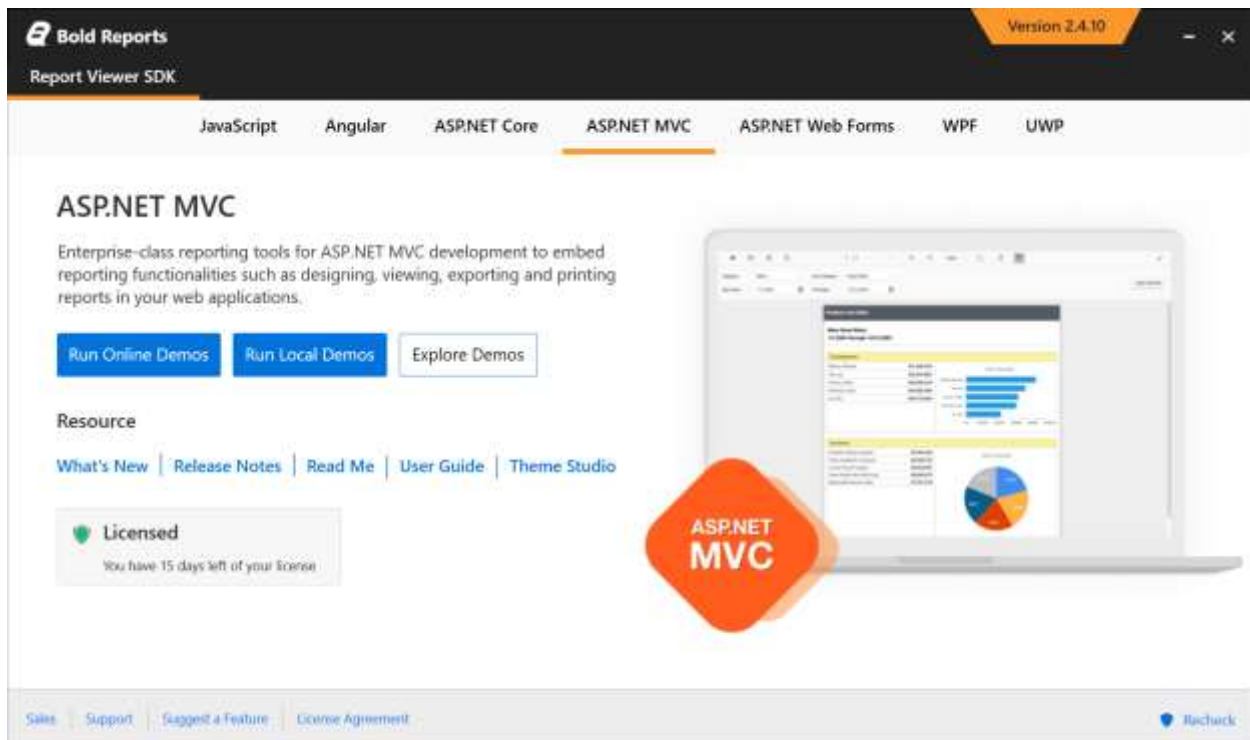
%localappdata%\Bold Reports\Embedded Reporting\Samples\ASP.NET Core

4. You can view the online demos from [here](#).

ASP.NET MVC

To preview the ASP.NET MVC demos, follow these steps:

1. Choose ASP.NET MVC platform in Report Viewer SDK control panel.



2. Click **Run Local Demos** to view the locally installed demos.
3. You can explore the demo sample application from the following installed location by clicking **Explore Demos** option.

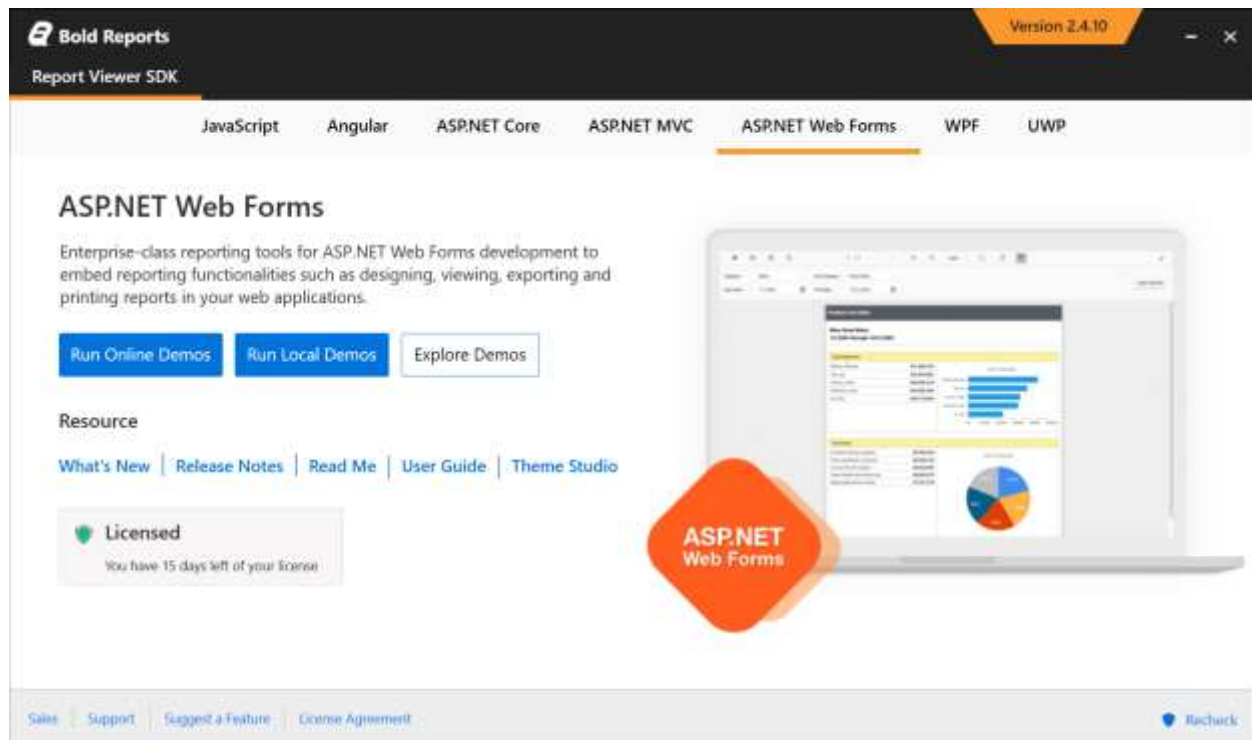
`%localappdata%\Bold Reports\Embedded Reporting\Samples\ASP.NET MVC`

4. You can view the online demos from [here](#).

ASP.NET Web Forms

To preview the ASP.NET Web Forms demos, follow these steps:

1. Choose ASP.NET Web Forms platform in Report Viewer SDK control panel.



2. Click **Run Local Demos** to view the locally installed demos.
3. You can explore the demo sample application from the following installed location by clicking **Explore Demos** option.

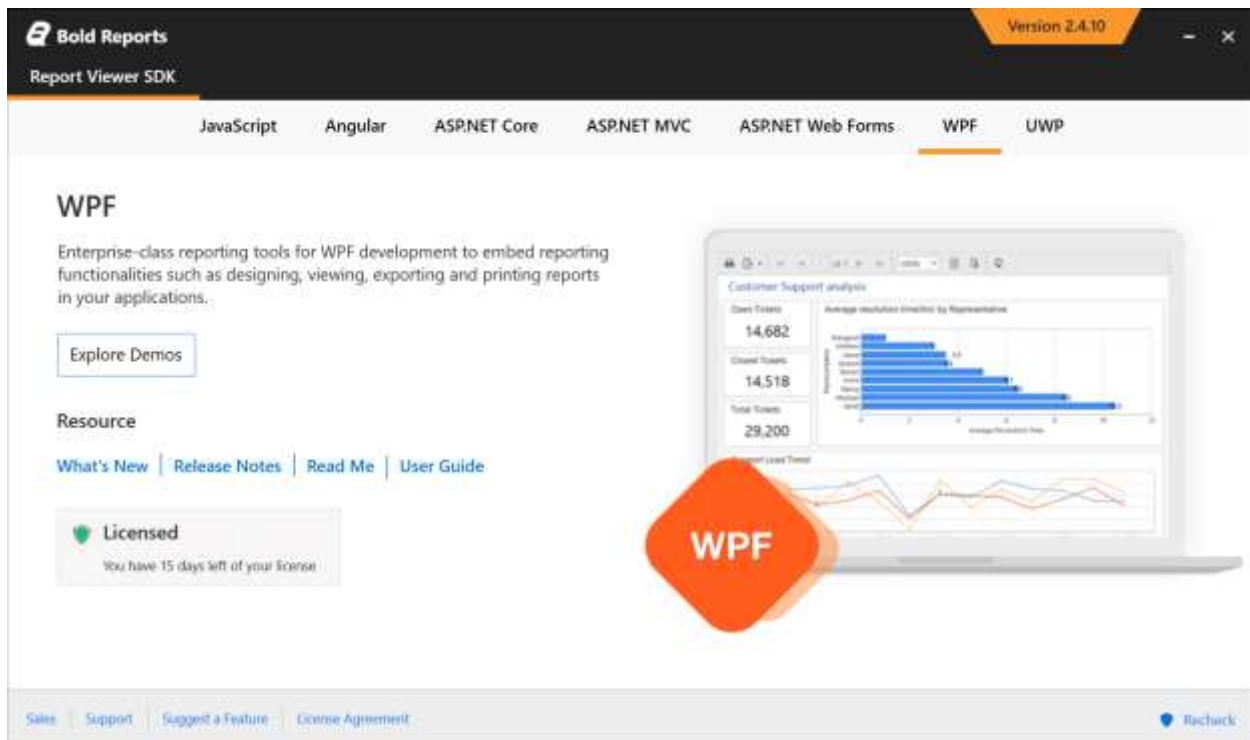
%localappdata%\Bold Reports\Embedded Reporting\Samples\ASP.NET

4. You can view the online demos from [here](#).

WPF

To preview the WPF demos, follow these steps:

1. Choose WPF platform in Report Viewer SDK control panel.



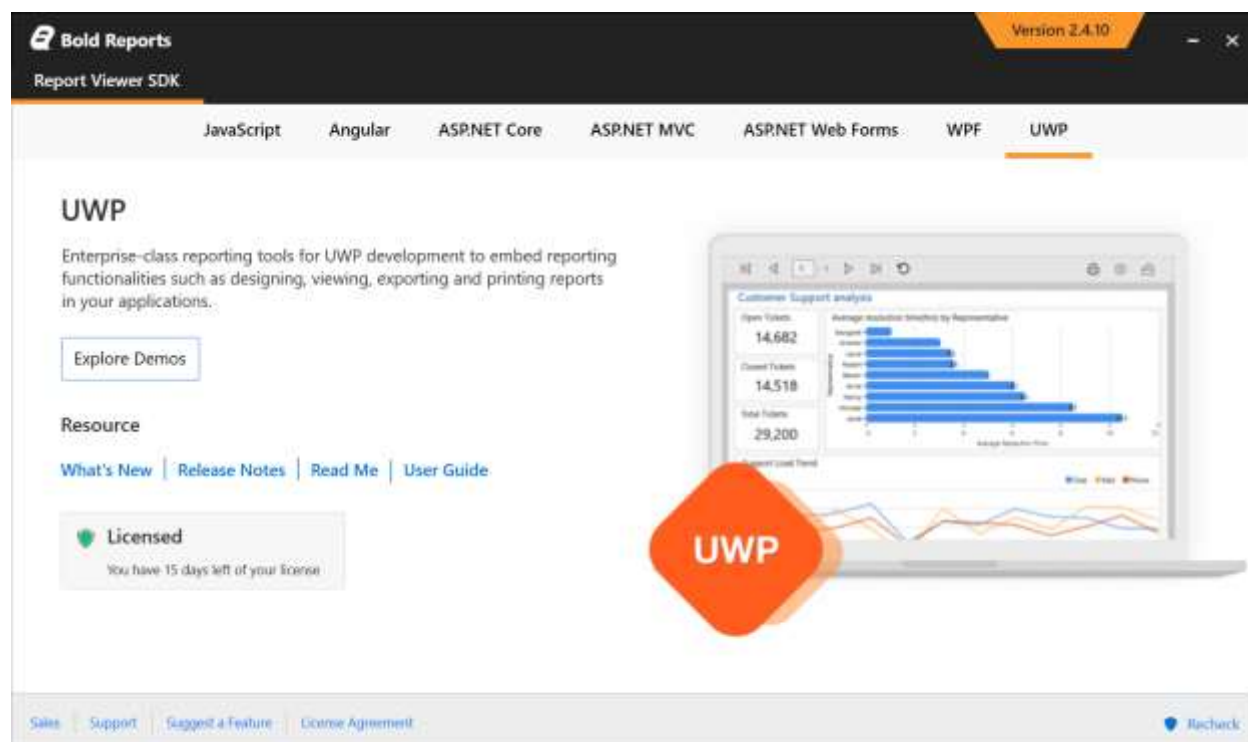
2. You can explore the demo sample application from the following installed location by clicking **Explore Demos** option to preview the WPF reporting demos.

%localappdata%\Bold Reports\Embedded Reporting\Samples\WPF

UWP

To preview the UWP demos, follow these steps:

1. Choose UWP platform in Report Viewer SDK control panel.



2. You can explore the demo sample application from the following installed location by clicking **Explore Demos** option to preview the UWP reporting demos.

%localappdata%\Bold Reports\Embedded Reporting\Samples\UWP

Report Viewer SDK nuget packages and assemblies

Report Viewer SDK provides nuget packages and assemblies to embed reporting functionalities into your Blazor, JavaScript, Angular, React, ASP.NET CORE, ASP.NET, ASP.NET MVC, WPF and UWP platform applications.

Refer to the following Report Viewer SDK location to add assemblies and nuget packages into your application without using online nuget package.

Report Viewer SDK Nuget packages Location:

C:\Program Files (x86)\Bold Reports\Embedded Reporting\Nuget Packages

Report Viewer SDK Assemblies location:

C:\Program Files (x86)\Bold Reports\Embedded Reporting\Assemblies

Script References

Report viewer themes and script files available from the Report Viewer SDK build installation location, C:\Program Files (x86)\Bold Reports\Embedded Reporting\Javascript\assets.

JavaScript Report Viewer SDK

The following table JavaScript Report Viewer SDK nuget packages and assemblies details.

Purpose	Bold Reports Package	Description
---------	----------------------	-------------

Client Side Script and Styles	<u>BoldReports.JavaScript</u>	BoldReports.JavaScript package contain Report Viewer HTML5 and JavaScript components for web development.
Server Side Helper	<u>BoldReports.Web</u>	BoldReports.Web is a server-side helper package to build Web API service for Report Viewer components.

Assembly location: C:\Program Files (x86)\Bold Reports\Embedded Reporting\Assemblies

Angular Report Viewer SDK

The following table Angular Report Viewer SDK nuget packages and assemblies details.

Purpose	Bold Reports Package	Description
Server Side Helper	<u>BoldReports.Web</u>	BoldReports.Web is a server-side helper package to build Web API service for Report Viewer components.

Assembly location: C:\Program Files (x86)\Bold Reports\Embedded Reporting\Assemblies

ASP.NET Core Report Viewer SDK

The following table ASP.NET Core Report Viewer SDK nuget packages and assemblies details.

Purpose	Bold Reports Package	Description
Client Side Tag Helper	<u>BoldReports.AspNet.Core</u>	BoldReports.AspNet.Core package contain runtime assemblies for building line-of-business applications that can run on Windows, Linux, and Mac. It contains HTML Helpers for Report Viewer components to build server-side dynamic websites based on Microsoft's ASP.NET Core.
Server Side Helper	<u>BoldReports.Net.Core</u>	BoldReports.Net.Core is a server-side helper package to build ASP.NET Core Web API service that is used for Report Viewer controls with ASP.NET Core, Angular, and JavaScript platforms.

Assembly location: C:\Program Files (x86)\Bold Reports\Embedded Reporting\Assemblies

ASP.NET MVC Report Viewer SDK

The following table ASP.NET MVC Report Viewer SDK nuget packages and assemblies details.

Purpose	Bold Reports Package	Description
Client Side MVC Helper	<u>BoldReports.Mvc4</u> <u>BoldReports.Mvc5</u>	BoldReports.Mvc4, andBoldReports.Mvc5 packages contains HTML Helpers for Report Viewer and Report Writer components to build server-side dynamic websites based on Microsoft's ASP.NET MVC.

Server Side Helper	<u>BoldReports.Web</u>	BoldReports.Web is a server-side helper package to build Web API service for Report Viewer components.
--------------------	------------------------	--

Assembly location: C:\Program Files (x86)\Bold Reports\Embedded Reporting\Assemblies

ASP.NET WebForms Report Viewer SDK

The following table ASP.NET WebForms Report Viewer SDK nuget packages and assemblies details.

Purpose	Bold Reports Package	Description
Client Side Tag Helper	<u>BoldReports.Webforms</u>	TheBoldReports.WebForms package contains HTML Helpers for Report Viewer component to build server-side dynamic websites based on Microsoft's ASP.NET Web Forms.
Server Side Helper	<u>BoldReports.Web</u>	BoldReports.Web is a server-side helper package to build Web API service for Report Viewer components.

Assembly location: C:\Program Files (x86)\Bold Reports\Embedded Reporting\Assemblies

WPF Report Viewer SDK

The following table WPF Report Viewer SDK nuget packages and assemblies details.

Purpose	Bold Reports Package	Description
Components Packages	<u>BoldReports.Wpf</u>	BoldReports.Wpf is a .NET Report Viewer, Report Writer control package to view and export RDL/RDLC reports within a .NET application.

Assembly location: C:\Program Files (x86)\Bold Reports\Embedded Reporting\Assemblies

UWP Report Viewer SDK

The following table UWP Report Viewer SDK nuget packages and assemblies details.

Purpose	Bold Reports Package	Description
Components Packages	<u>BoldReports.UWP</u>	BoldReports.UWP is a Report Viewer control package to display RDL/RDLC reports within the Universal Windows Platform application. It builds the server-side implementations for Report Viewer component.
Server Side Helper	<u>BoldReports.Web</u>	BoldReports.Web is a server-side helper package to build Web API service for Report Viewer components.

Assembly location: C:\Program Files (x86)\Bold Reports\Embedded Reporting\Assemblies for Universal Windows

NPM Packages

The Report Viewer SDK components NPM package details are tabulated below:

Platform	Bold Reports Package
Javascript	BoldReports.JavaScript
Extensions	BoldReports.JavaScript.Extensions
Globalize	BoldReports.Global

The above Javascript NPM packages contain the following scripts and styles required for Reporting components.

Reporting Component	Styles	Scripts	Usage
Report Viewer	<code>bold.reports.all.min.css</code>	<code>bold.reports.common.min.js</code> <code>bold.reports.widgets.min.js</code> <code>ej.chart.min.js</code> <code>ej2-base.min.js</code> <code>ej2-data.min.js</code> <code>ej2-pdf-export.min.js</code> <code>ej2-svg-base.min.js</code> <code>ej2-lineargauge.min.js</code> <code>ej2-circulargauge.min.js</code> <code>ej.map.min.js</code> <code>bold.report-viewer.min.js</code>	<code>bold.reports.all.min.css</code> - It contains the styles and css references of reporting dependent components. <code>bold.reports.common.min.js</code> - Common script for reporting widgets. <code>bold.reports.widgets.min.js</code> - It contains the scripts of dependent Syncfusion controls that are used for Report Viewer. <code>ej.chart.min.js</code> - Renders the chart item. Add this script only if your report contains the chart report item. <code>ej2-base.min.js</code> - Renders the gauge item. Add this script only if your report contains the gauge report item. <code>ej2-data.min.js</code> - Renders the gauge item. Add this script only if your report contains the gauge report item.

			<p><code>ej2-pdf-export.min.js</code> - Renders the gauge item. Add this script only if your report contains the gauge report item.</p> <p><code>ej2-svg-base.min.js</code> - Renders the gauge item. Add this script only if your report contains the gauge report item.</p> <p><code>ej2-lineargauge.min.js</code> - Renders the linear gauge item. Add this script only if your report contains the linear gauge report item.</p> <p><code>ej2-circulargauge.min.js</code> - Renders the circular gauge item. Add this script only if your report contains the circular gauge report item.</p> <p><code>ej.map.min.js</code> - Renders the map item. Add this script only if your report contains the map report item.</p> <p><code>bold.report-viewer.min.js</code> - Renders the Syncfusion Report Viewer widget.</p>
--	--	--	--

Migrate Reporting Application

This section provides step-by-step instructions for migrating Report Viewer SDK such as Report Viewer and Report Writer from Syncfusion Essential Studio release version to Bold Reports version:

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Migrate JavaScript Report Viewer SDK

The following provides step-by-step instructions for migrating your JavaScript reporting application.

Purpose	Essential Studio Packages	Bold Reports Package	Description
---------	---------------------------	----------------------	-------------

Client Side Script and Styles	<code>Syncfusion.JavaScript.ReportDesigner</code>	<code>BoldReports.JavaScript</code>	BoldReports.JavaScript package contain Report Viewer HTML5 and JavaScript components for web development.
	<code>Syncfusion.JavaScript</code>		
Server Side Helper	<code>Syncfusion.Web.ReportDesigner</code>	<code>BoldReports.Web</code>	BoldReports.Web is a server-side helper package to build Web API service for Report Viewer components.
	<code>Syncfusion.Web.ReportViewer</code>		

Refer the JavaScript Report Viewer SDK migration steps [here](#).

Migrate Angular Report Viewer SDK

The following provides step-by-step instructions for migrating your Angular reporting application.

Essential Studio Packages	Bold Reports Package	Description
<code>syncfusion-javascript</code>	@boldreports/javascript-reporting-controls	JavaScript NPM packages contain the Reporting components scripts and styles.
<code>ej-angular2</code>	@boldreports/angular-reporting-components	Angular NPM packages contain the Reporting components scripts and styles.
<code>syncfusion-ej-global</code>	@boldreports/global	Globalize NPM packages contain the Reporting components globalize scripts.

Purpose	Essential Studio Packages	Bold Reports Package	Description
Server Side Helper	<code>Syncfusion.Web.ReportDesigner</code>	<code>BoldReports.Web</code>	BoldReports.Web is a server-side helper package to build Web API service for Report Viewer components.
	<code>Syncfusion.Web.ReportViewer</code>		

Refer the Angular Report Viewer SDK migration steps [here](#).

Migrate ASP.NET Core Report Viewer SDK

The following provides step-by-step instructions for migrating your ASP.NET Core reporting application.

Purpose	Essential Studio Packages	Bold Reports Package	Description
Client Side Tag Helper	<code>Syncfusion.EJ.AspNet.Core</code>	<u><code>BoldReports.AspNet.Core</code></u>	<code>BoldReports.AspNet.Core</code> package contains runtime assemblies for building line-of-business applications that can run on Windows, Linux, and Mac. It contains HTML Helpers for Report Viewer components to build server-side dynamic websites based on Microsoft's ASP.NET Core.
Server Side Helper	<code>Syncfusion.EJ.ReportDesigner.AspNet.Core</code> <code>Syncfusion.EJ.ReportViewer.AspNet.Core</code> <code>Syncfusion.Report.Net.Core</code>	<u><code>BoldReports.Net.Core</code></u>	<code>BoldReports.Net.Core</code> is a server-side helper package to build ASP.NET Core Web API service that is used for Report Viewer controls with ASP.NET Core.

Refer the ASP.NET Core Report Viewer SDK migration steps [here](#).

Migrate ASP.NET MVC Report Viewer SDK

The following provides step-by-step instructions for migrating your ASP.NET MVC reporting application.

Purpose	Essential Studio Packages	Bold Reports Package	Description
Client Side MVC Helper	<code>Syncfusion.AspNet.Mvc4</code> <code>Syncfusion.AspNet.Mvc5</code>	<u><code>BoldReports.Mvc4</code></u> <u><code>BoldReports.Mvc5</code></u>	<code>TheBoldReports.Mvc4</code> , and <code>BoldReports.Mvc5</code> packages contains HTML Helpers for Report Viewer components to build server-side dynamic websites based on Microsoft's ASP.NET MVC.
Server Side Helper	<code>Syncfusion.ReportDesigner.AspNet.Mvc</code> <code>Syncfusion.ReportViewer.AspNet.Mvc</code>	<u><code>BoldReports.Web</code></u>	<code>BoldReports.Web</code> is a server-side helper package to build Web

			API service for Report Viewer components.
--	--	--	---

Refer the ASP.NET MVC Report Viewer SDK migration steps [here](#).

Migrate ASP.NET Report Viewer SDK

The following provides step-by-step instructions for migrating your ASP.NET reporting application.

Purpose	Essential Studio Packages	Bold Reports Package	Description
Client Side Tag Helper	<code>Syncfusion.AspNet</code>	<u><code>BoldReports.Webforms</code></u>	The <code>BoldReports.WebForms</code> package contains HTML Helpers for Report Viewer components to build server-side dynamic websites based on Microsoft's ASP.NET Web Forms.
Server Side Helper	<code>Syncfusion.ReportDesigner.AspNet</code> <code>Syncfusion.ReportViewer.AspNet</code>	<u><code>BoldReports.Web</code></u>	<code>BoldReports.Web</code> is a server-side helper package to build Web API service for Report Viewer components.

Refer the ASP.NET Report Viewer SDK migration steps [here](#).

Migrate WPF Report Viewer SDK

The following provides step-by-step instructions for migrating your WPF reporting application.

Platform	Purpose	Essential Studio Packages	Bold Reports Package	Description
WPF	Components Packages	<code>Syncfusion.ReportControls.WPF</code> <code>Syncfusion.ReportViewer.WPF</code> <code>Syncfusion.ReportWriter.Base</code>	<u><code>BoldReports.Wpf</code></u>	<code>BoldReports.Wpf</code> is a .NET Report Viewer, Report Writer control package to view and export RDL/RDLC reports within a .NET application.

Refer the WPF Report Viewer SDK migration steps [here](#).

Migrate UWP Report Viewer SDK

The following provides step-by-step instructions for migrating your UWP reporting application.

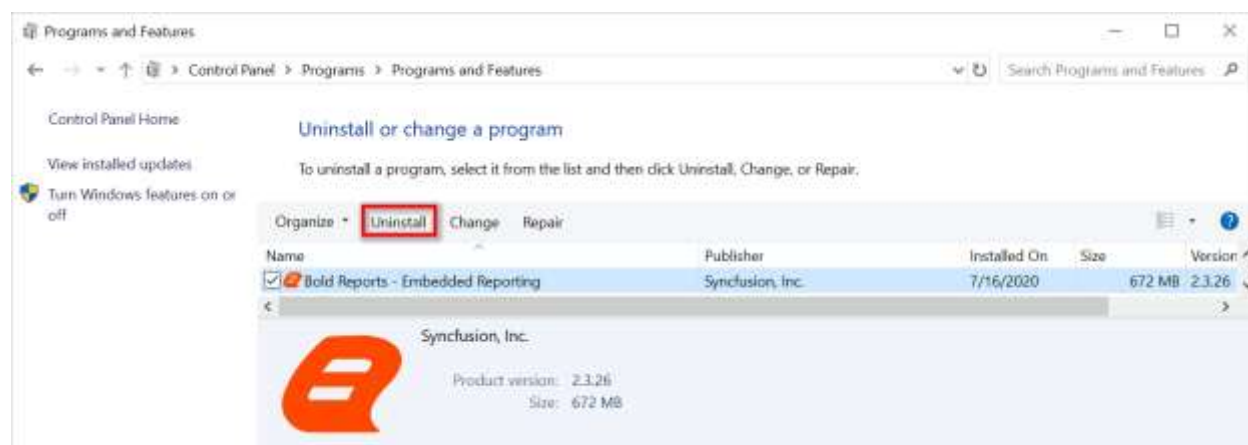
Purpose	Essential Studio Packages	Bold Reports Package	Description
Components Packages	<code>Syncfusion.SfReportViewer.UWP</code>	<u><code>BoldReports.UWP</code></u>	BoldReports.UWP is a Report Viewer control package to display RDL/RDLC reports within the Universal Windows Platform application. It builds the server-side implementations for Report Viewer component.
Server Side Helper	<code>Syncfusion.Web.ReportViewer</code>	<u><code>BoldReports.Web</code></u>	BoldReports.Web is a server-side helper package to build Web API service for Report Viewer components.

Refer the UWP Report Viewer SDK migration steps [here](#).

Uninstallation of Reporting Tools

You can uninstall the Report Viewer SDK from the control panel through the following steps.

1. On the Start menu, click Control Panel.
2. In the Control Panel, click Programs and Features.
3. Search Bold Reporting Tools in the Name list and click it.
4. Click **Uninstall**.



5. Click **Yes** to confirm the uninstallation of Reporting Tools.

6. The Report Viewer SDK setup will be uninstalled and re-direct to the Report Viewer SDK [Uninstallation page](#).

Report Viewer SDK licensing overview

Starting with the version 2.x , Syncfusion introduced a new licensing system for the Bold Reports. You have to include the license token in your projects in order to use the [Bold Reports](#) assemblies. Note that the license token is different from the setup unlock key and that needs to be generated separately from the [Bold Reports](#) website. The below licensing error will be displayed if the license token is missing in your projects.

This application was built using a trial version of Bold Reports. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period.

From version v2.4, we provide offline license key support

- [Generate and register Bold Reports online license token](#)
- [Generate and register Bold Reports offline license key file](#)

Report Viewer SDK online licensing overview

Starting with the version 2.x , Syncfusion introduced a new licensing system for the Bold Reports. You have to include the license token in your projects in order to use the [Bold Reports](#) assemblies. Note that the license token is different from the setup unlock key and that needs to be generated separately from the [Bold Reports](#) website. The below licensing error will be displayed if the license token is missing in your projects.

This application was built using a trial version of Bold Reports. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period.

How to generate Bold Reports online license token

License tokens can be generated from the [Downloads](#) section of the [Bold Reports](#) site.

Bold Reports license tokens are version specific. So, you should use the corresponding version license tokens in the projects.

How to register the Bold Reports online license token

The generated online license tokens is just a string that needs to be registered before any Bold Reports control is initiated. The following code is used to register the license.

```
`csharp
```

```
Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR ONLINE LICENSE TOKEN");
```

```
`
```

ASP. NET

Register the online license token in Application_Start method of `Global.asax.cs/Global.asax`

```
`csharp
```

```
void Application_Start(object sender, EventArgs e)
```

```
{  
//Register Bold license  
Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR ONLINE LICENSE TOKEN");  
// Code that runs on application startup  
RouteConfig.RegisterRoutes(RouteTable.Routes);  
BundleConfig.RegisterBundles(BundleTable.Bundles);  
}  
`
```

ASP. NET MVC

Register the online license token in Application_Start method of `Global.asax.cs`

```
`csharp  
void Application_Start(object sender, EventArgs e)  
{  
//Register Bold license  
Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR ONLINE LICENSE TOKEN");  
// Code that runs on application startup  
RouteConfig.RegisterRoutes(RouteTable.Routes);  
BundleConfig.RegisterBundles(BundleTable.Bundles);  
}  
`
```

To get start quickly, you can refer this video:

youtube: <https://youtu.be/KRPKcHZQhQ0>

ASP. NET Core

Register the online license token in `Configure` method of `Startup.cs`

```
`csharp  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, ILoggerFactory loggerFactory)  
{  
//Register Bold license  
Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR ONLINE LICENSE TOKEN");  
}  
`
```

To get start quickly, you can refer this video:

youtube: <https://youtu.be/KLcx0VODXTE>

WPF

Register the online license token in App constructor of `App.xaml.cs` in C#. If App constructor not available in `App.xaml.cs`, create the `App()` constructor in `App.xaml.cs` and register the online license token inside the constructor.

```
`csharp
public partial class App : Application
{
    public App()
    {
        //Register Bold license
        Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR ONLINE LICENSE TOKEN");
    }
}
```

To get start quickly, you can refer this video:

youtube: <https://youtu.be/J4W9xtxceJU>

UWP

Register the online license token in `App.xaml.cs` constructor before `InitializeComponent()` in C#. If App constructor not available in `App.xaml.cs`, create the `App()` constructor in `App.xaml.cs` and register the online license token inside the constructor.

```
`csharp
public App()
{
    //Register Bold license
    Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR ONLINE LICENSE TOKEN");
    this.InitializeComponent();
    this.Suspending += OnSuspending;
}
`
```

To get start quickly, you can refer this video:

youtube: <https://youtu.be/nW9E0F2cN10>

Blazor

For `Server side` application. Register the online license token in `Configure` method of `Startup.cs`

```
`csharp
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

```
{
//Register Bold license
Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR ONLINE LICENSE TOKEN");
if (env.IsDevelopment())
{
app.UseDeveloperExceptionPage();
}
else
{
app.UseExceptionHandler("/Error");
// The default HSTS value is 30 days. You may want to change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.
app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseEndpoints(endpoints =>
{
endpoints.MapControllers();
endpoints.MapBlazorHub();
endpoints.MapFallbackToPage("/_Host");
});
}
```

To get start quickly, you can refer this video:

youtube: <https://youtu.be/e4EXJIUmCj8>

Report Viewer SDK offline licensing overview

Starting with the version **2.4.X**, Syncfusion introduced offline licensing system too for the Bold Reports. You have to include the license key file (**.lic**) in your projects in order to use the [Bold Reports](#) assemblies. The below licensing error will be displayed if the license token is missing in your projects.

This application was built using a trial version of Bold Reports. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period.

Report Viewer SDK offline licensing overview **How** to generate and download Bold Reports offline license key

How to generate and download Bold Reports offline license key

The License key file(`.lic`) can be generated from the [Downloads](#) section of the [Bold Reports](#) site.

Bold Reports License key file(`.lic`) are version specific. So, you should use the corresponding version License key file(`.lic`) in the projects.

How to register the Bold Reports offline license key

The generated offline license key file(`.lic`) is just a string that needs to be registered before any Bold Reports control is initiated. The following code is used to register the license.

```
`csharp
string licenseKey = File.ReadAllText(licenseFilePath);
Bold.Licensing.BoldLicenseProvider.RegisterLicense(licenseKey, isOfflineValidation: true);
`
```

ASP. NET

Register the license key in Application_Start method of `Global.asax.cs/Global.asax`

```
`csharp
void Application_Start(object sender, EventArgs e)
{
    //Register Bold licenseKey
    string licenseKey = File.ReadAllText(@"boldreports_licensekey.lic"); // Replace it with actual license key
    file path
    BoldLicenseProvider.RegisterLicense(licenseKey, isOfflineValidation: true);
    // Code that runs on application startup
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
}
`
```

ASP. NET MVC

Register the license key in Application_Start method of `Global.asax.cs`

```
`csharp
void Application_Start(object sender, EventArgs e)
{
    //Register Bold licenseKey
    string licenseKey = File.ReadAllText(@"boldreports_licensekey.lic"); // Replace it with actual license key
    file path
    BoldLicenseProvider.RegisterLicense(licenseKey, isOfflineValidation: true);
    // Code that runs on application startup
}
```

```
RouteConfig.RegisterRoutes(RouteTable.Routes);
BundleConfig.RegisterBundles(BundleTable.Bundles);
}
`
```

To get start quickly, you can refer this video:

youtube: https://youtu.be/fjI_wdKzY0

ASP. NET Core

Register the license key in `Configure` method of `Startup.cs`

```
`csharp
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, ILoggerFactory loggerFactory)
{
    //Register Bold licenseKey
    string licenseKey = File.ReadAllText(@"boldreports_licensekey.lic"); // Replace it with actual license key
    file path
    BoldLicenseProvider.RegisterLicense(licenseKey, isOfflineValidation: true);
}
`
```

To get start quickly, you can refer this video:

youtube: <https://youtu.be/IliCfvenSzo>

WPF

Register the license key in App constructor of `App.xaml.cs` in C#. If App constructor not available in `App.xaml.cs`, create the `App()` constructor in `App.xaml.cs` and register the license key inside the constructor.

```
`csharp
public partial class App : Application
{
    public App()
    {
        //Register Bold licenseKey
        string licenseKey = File.ReadAllText(@"boldreports_licensekey.lic"); // Replace it with actual license key
        file path
        Bold.Licensing.BoldLicenseProvider.RegisterLicense(licenseKey, isOfflineValidation:true);
    }
}
`
```

To get start quickly, you can refer this video:

youtube: <https://youtu.be/5oYBgwqBw1M>

UWP

Register the license key in `App.xaml.cs` constructor before `InitializeComponent()` in C#. If App constructor not available in `App.xaml.cs`, create the `App()` constructor in `App.xaml.cs`, Add the `boldreports_licensekey.lic` file as `embedded resource` in the project and register the license key inside the constructor.

```
`csharp
public App()
{
    var assembly= typeof(App).GetTypeInfo().Assembly;
    var fileStream=assembly.GetManifestResourceStream("OfflineExport.boldreports_licensekey.lic");
    if (fileStream!= null && fileStream.Length >0)
    {
        using(var reader= new StreamReader(fileStream))
        {
            //Register Bold licenseKey
            string licenseKey = reader.ReadLineAsync().Result;
            BoldLicenseProvider.RegisterLicenseProvider.RegisterLicense(licenseKey, isOfflineValidation: true);
        }
    }
    this.InitializeComponent();
    this.Suspending += OnSuspending;
}
`
```

To get start quickly, you can refer this video:

youtube: <https://youtu.be/GOxP-JysXck>

Blazor

For `Server side` application. Register the license key in `Configure` method of `Startup.cs`

```
`csharp
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    //Register Bold licenseKey
    string licensekey = System.IO.File.ReadAllText(@"boldreports_licensekey.lic");// Replace it with actual
    license key file path
}
```

```
Bold.Licensing.BoldLicenseProvider.RegisterLicense(licensekey, isOfflineValidation: true);
if (env.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see
    https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
    endpoints.MapBlazorHub();
    endpoints.MapFallbackToPage("/_Host");
});
}
```

To get start quickly, you can refer this video:

youtube: <https://youtu.be/OP1VmlabyFs>

Report Viewer SDK license token errors

Licensing error popup is displayed with various messages under different circumstances. Here are some ways to resolve different issues.

License token not registered

The following error message will be shown if Bold license token was not registered in your application.

Error message:

This application was built using a trial version of Bold Reports. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period. Please refer to this

<https://help.boldreports.com/report-viewer-sdk/licensing/online-license-token/> help topic for more information.

Solution:

Generate a valid license token from [here](#) for a specific version.

Invalid token

If the application is registered with an invalid token, another version of license token, or another platform's license token, the following error message will pop up when launching the application.

Error Message:

The included Bold license is invalid. Please refer to this help topic <https://help.boldreports.com/report-viewer-sdk/licensing/licensing-errors/#invalid-token> for more information.

Solution:

Generate a valid license token from [here](#) for a specific version.

License Expired

The following error message will be shown if the license token has expired.

Error Message:

Your Bold license has expired. Please refer to this help topic <https://help.boldreports.com/report-viewer-sdk/licensing/licensing-errors/#license-expired> for more information.

Solution:

Purchase from [here](#) to get Bold license.

Platform Mismatch

If the application is registered with another platform's license token, the following error message will pop up when launching the application.

Error Message:

The included Bold license is invalid (Platform mismatch). Please refer to this help topic <https://help.boldreports.com/report-viewer-sdk/licensing/licensing-errors/#platform-mismatch> for more information.

Solution:

Generate a valid license token from [here](#) platform.

Internet Connection Error

We need internet connection to validate the registered license. If internet is not connected in the machine, the following error message will pop up when launching the application.

Error Message:

We couldn't connect to internet to validate your license. Please check your internet connection.

Solution:

Connect the internet and try running the application again.

Network Error

If the server is not reachable to validate the license token, the following error message will pop up when launching the application.

Error Message:

Unable to validate your license. Please try again after some time or check your firewall/proxy settings. Refer to the help topic <https://help.boldreports.com/report-viewer-sdk/licensing/licensing-errors/#network-error>.

Solution:

Please ensure that you are connected to the internet. If that does not resolve the problem, check your proxy settings.

Could not load Bold.Licensing.dll assembly version

Make sure that all the referenced Bold assemblies are of the same version. Try cleaning and rebuilding the application to resolve assembly conflict issues.

FAQ section for Bold Licensing

We are going to discuss about most frequently asked questions of Bold Licensing in this section. We discussed the below topics in this section,

- [How to upgrade from Trial version after purchasing a license?](#)
- [Where can I get a license token?](#)
- [Bold Licensing before v2.x](#)
- [Does Bold Reports Embedded Reporting Tools or Viewer SDK require additional licensing when deploying an application to Azure App service](#)
- [What are the differences in Bold Reports licensing models](#)
- [Can Syncfusion licenses be used with Bold Reports](#)
- [Can the Syncfusion community license be used with Bold Reports](#)
- [Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license](#)
- [How to resolve the license network error?](#)

How to upgrade from Trial version after purchasing a license

Replace the currently used trial license token with a paid license token that can be generated from the [License & Downloads](#) section of our Bold Reports website. Refer to [this](#) topic for more information regarding registering the license in the application.

Where can I get a license token

License tokens can be generated from the [License & Downloads](#) section of the Bold Reports website. Refer to [this](#) topic for more information to generate the license token in the application.

See also

[Where should I replace the Bold Reports license token in Bold Reports Azure App Service](#)

Does Bold Reports Embedded Reporting Tools or Viewer SDK require additional licensing when deploying an application to Azure App service

See also

Does Bold Reports Embedded Reporting Tools or Viewer SDK require additional licensing when deploying an application to Azure App service

No, if you already have Bold Reports for your team, you do not need to buy an additional license for deploying your applications to Azure App service. You should register the license token on initial startup of your application as explained in the following documentation.

[License Token](#)

See also

- [What are the differences in Bold Reports licensing models](#)
- [Can Syncfusion licenses be used with Bold Reports](#)
- [Can the Syncfusion community license be used with Bold Reports](#)
- [Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license](#)

Can Syncfusion licenses be used with Bold Reports

Yes, you can use Syncfusion licenses with Bold Reports, but with access only to Report Viewer SDK. You should login with Syncfusion account to access Report Viewer SDK. To know more about licensing, please refer to [What are the differences in Bold Reports licensing models](#).

See also

- [Can the Syncfusion community license be used with Bold Reports](#)
- [Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license](#)
- [Does Bold Reports Embedded Reporting Tools or Viewer SDK require additional licensing when deploying an application to Azure App service](#)

Can the Syncfusion community license be used with Bold Reports

Yes, you can use the Syncfusion community license with Bold Reports, with access only to Report Viewer SDK. You should login with Syncfusion account to access Report Viewer SDK. To know more about licensing, check [What are the differences in Bold Reports licensing models](#).

See also

- [Can Syncfusion licenses be used with Bold Reports](#)
- [Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license](#)
- [Does Bold Reports Embedded Reporting Tools or Viewer SDK require additional licensing when deploying an application to Azure App service](#)

Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license
See also

Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license

Yes, you can access the Report Viewer component from Bold Reports using Syncfusion Community license. You should login with Syncfusion account to access the component. For more details about the licensing, refer to [What are the differences in Bold Reports licensing models](#).

See also

- [Can the Syncfusion community license be used with Bold Reports](#)

What are the differences in Bold Reports licensing models

Bold Reports has different restrictions when using reporting tools and solutions with licensing models. Refer to the following table more details.

License Models		Cloud Reporting		Enterprise Reporting	
Embedded Reporting		Report Viewer SDK			
----- -----		----- -----		----- -----	
Bold Reports Cloud Report Server		Yes	No	No	
No					
Bold Reports Enterprise Report Server		No	Yes	Yes	
No					
Report Designer for application		No	No	Yes	
No					
Report Viewer for application		No	No	Yes	
Yes					
Report Writer for application (Export Reports to different formats without view)		No		No	
Yes		Yes			
Standalone Report Designer for creating Reports		Yes	Yes		
Yes		Yes			

See also

- [Can Syncfusion licenses be used with Bold Reports](#)
- [Can the Syncfusion community license be used with Bold Reports](#)
- [Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license](#)
- [Does Bold Reports Embedded Reporting Tools or Viewer SDK require additional licensing when deploying an application to Azure App service](#)

How to resolve the license network error

The network license error watermarks will be shown based on license validation in reporting components as per [licensing](#). For the online license validation process, you will pass the license token to our servers (<https://websiteapi.boldbi.com/> and <https://websiteapiv2.boldbi.com/>), and validate your

subscription. In order to execute license validation., you should ensure that your firewall is not blocking the following sites from your network.

Sites	IP Address
-----	-----
https://websiteapi.boldbi.com/	159.65.153.192
https://websiteapiv2.boldbi.com/	139.59.56.9

Since it requires to connect to Bold Report license server it requires internet connection for the license validation process. If you don't have the network problem, then you should ensure the license token registered, as properly with your application as explained in below documentation.

[Register license token for an application](#)

Since, the process of our license validation will be at the startup of every application, you have to register the license in the startup of application. The internet license validation will be processed in the place of registering the license token only.

See Also

[How to upgrade from Trial version after purchasing a license](#)

[Where can I get a license token](#)

[Embedded reporting tools licensing version 1.2.x](#)

Report Viewer SDK licensing version 1.2.x

If you are using [Bold Reports](#) lesser than the version 2.x then you need to register the license key with your application. In this section we discussed about the below topics,

- [How to register a license key](#)
- [Licensing errors while using v1.2.x of Bold Reports.](#)

This section specifically for Bold Reports version less than 2.x. If you are using higher version of [Bold Reports](#), we recommend you to refer the section [licensing tokens](#) to register your application with Bold Licensing.

Report Viewer SDK licensing overview

Starting with the version 1.2.x, Syncfusion introduced a new licensing system for the Bold Reports. You have to include the license key in your projects in order to use the Bold Reports assemblies. Note that the license key is different from the setup unlock key and that needs to be generated separately from the Bold Reports website. The below licensing error will be displayed if the license key is missing in your projects.

This application was built using a trial version of Bold Reports. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period.

[How to generate Bold Reports license key](#)

License keys can be generated from the [Downloads](#) section of the [Bold Reports](#) site.

Bold Reports license keys are version specific. So, you should use the corresponding version license key in the projects.

How to register the Bold Reports license key

The generated license key is just a string that needs to be registered before any Bold Reports control is initiated. The following code is used to register the license.

```
`csharp
Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
`
```

ASP.NET

Register the license key in Application_Start method of `Global.asax.cs/Global.asax`

```
`csharp
void Application_Start(object sender, EventArgs e)
{
    //Register Bold license
    Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
    // Code that runs on application startup
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
}
`
```

ASP.NET MVC

Register the license key in Application_Start method of `Global.asax.cs`

```
`csharp
void Application_Start(object sender, EventArgs e)
{
    //Register Bold license
    Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
    // Code that runs on application startup
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
}
`
```

ASP.NET Core

Register the license key in `Configure` method of `Startup.cs`

```
`csharp
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, ILoggerFactory loggerFactory)
{
    //Register Bold license
    Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();
}
`
```

WPF

Register the license key in App constructor of App.xaml.cs in C#. If App constructor not available in App.xaml.cs, create the App() constructor in App.xaml.cs and register the license key inside the constructor.

```
`csharp
public partial class App : Application
{
    public App()
    {
        //Register Bold license
        Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
    }
}
`
```

UWP

Register the license key in App.xaml.cs constructor before InitializeComponent() in C#. If App constructor not available in App.xaml.cs, create the App() constructor in App.xaml.cs and register the license key inside the constructor.

```
`csharp
public App()
{
    //Register Bold license
    Bold.Licensing.BoldLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
    this.InitializeComponent();
    this.Suspending += OnSuspending;
}
`
```

Report Viewer SDK license key errors

Licensing error popup is displayed with various messages under different circumstances. Here are some ways to resolve different issues.

License key not registered

The following error message will be shown if Bold license key was not registered in your application.

Error message:

This application was built using a trial version of Bold Reports. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period. Please refer to this <https://help.boldreports.com/report-viewer-sdk/licensing/faq/v1.x/> help topic for more information.

Solution:

Generate a valid license key from [here](#) for a specific version.

Invalid key

If the application is registered with an invalid key, another version of license key, or another platform's license key, the following error message will pop up when launching the application.

Error Message:

The included Bold license is invalid. Please refer to this help topic <https://help.boldreports.com/report-viewer-sdk/licensing/faq/v1.x/errors/#invalid-key> for more information.

Solution:

Generate a valid license key from [here](#) for a specific version.

Trial Expired

The following error message will be shown if the trial key has expired after 30 days.

Error Message:

Your Bold trial license has expired. Please refer to this help topic <https://help.boldreports.com/report-viewer-sdk/licensing/faq/v1.x/errors/#trial-expired> for more information.

Solution:

Purchase from [here](#) to get a valid Bold license.

License Expired

The following error message will be shown if the license key has expired.

Error Message:

Your Bold license has expired. Please refer to this help topic <https://help.boldreports.com/report-viewer-sdk/licensing/faq/v1.x/errors/#license-expired> for more information.

Solution:

Purchase from [here](#) to get Bold license.

Platform Mismatch

If the application is registered with another platform's license key, the following error message will pop up when launching the application.

Error Message:

The included Bold license is invalid (Platform mismatch). Please refer to this help topic <https://help.boldreports.com/report-viewer-sdk/licensing/faq/v1.x/errors/#platform-mismatch> for more information.

Solution:

Generate a valid license key from [here](#) platform.

Version Mismatch

If the application is registered with another version's license key, the following error message will pop up when launching the application.

Error Message:

The included Bold license ({Registered Version}) is invalid for version {Required version}. Please refer to this help topic <https://help.boldreports.com/report-viewer-sdk/licensing/faq/v1.x/errors/#version-mismatch> for more information.

Solution:

Generate a valid license key from [here](#) platform.

Could not load Bold.Licensing.dll assembly version

Make sure that all the referenced Bold assemblies are of the same version. Try cleaning and rebuilding the application to resolve assembly conflict issues.

Bold reporting tools for Blazor

Enterprise-class reporting tools for Blazor development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your Blazor applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

Overview

The Report Viewer is a visualization control used to display SSRS, RDL, RDLC, and Bold Report Server reports within web applications. It allows you to view RDL/RDLC reports with or without using SSRS or Bold Report Server. You can bind data sources, parameters, and render reports with all major capabilities of RDL reporting and export the report to PDF, Excel, CSV, Word, PowerPoint, and HTML formats. Some of the key features are:

- Renders interactive reports with drill down, drill through, hyperlinks, and interactive sorting.
- Easily customize each element of the Report Viewer and provide events for report processing customization.
- Supports jQuery, Angular, React, Blazor, ASP.NET Core, ASP.NET MVC, ASP.NET WebForms, WPF, and UWP.

Introducing the newly redesigned Report Viewer! We have given it a refreshing new look that embraces sleek and modern design, aligning perfectly with the latest web design trends. It allows seamless integration into your application. For detailed guidance on the migration process, we recommend you to refer our [Report Viewer v2.0 migration document](#).

Add Web Report Viewer to a Blazor application

This section explains the steps required to add a web Report Viewer to a Blazor application.

To get start quickly with Report Viewer, you can check on this video:

youtube: <https://youtu.be/nyaEEQzqRew>

Prerequisites

Before getting started with a bold web report viewer, make sure your development environment includes the following requirements.

- [Visual Studio 2019](#) with ASP.NET and Web Development workloads.
- [.NET Core 3.1](#) Framework.

Create a Blazor application

To get started, create a Blazor Server App template application and name it **BlazorReportingTools** using Visual Studio 2019. If this template is not available, refer to this [link](#) to configure the system environment.

The source code for this Blazor reporting components app is available on [GitHub](#).

Create a Web API for report processing

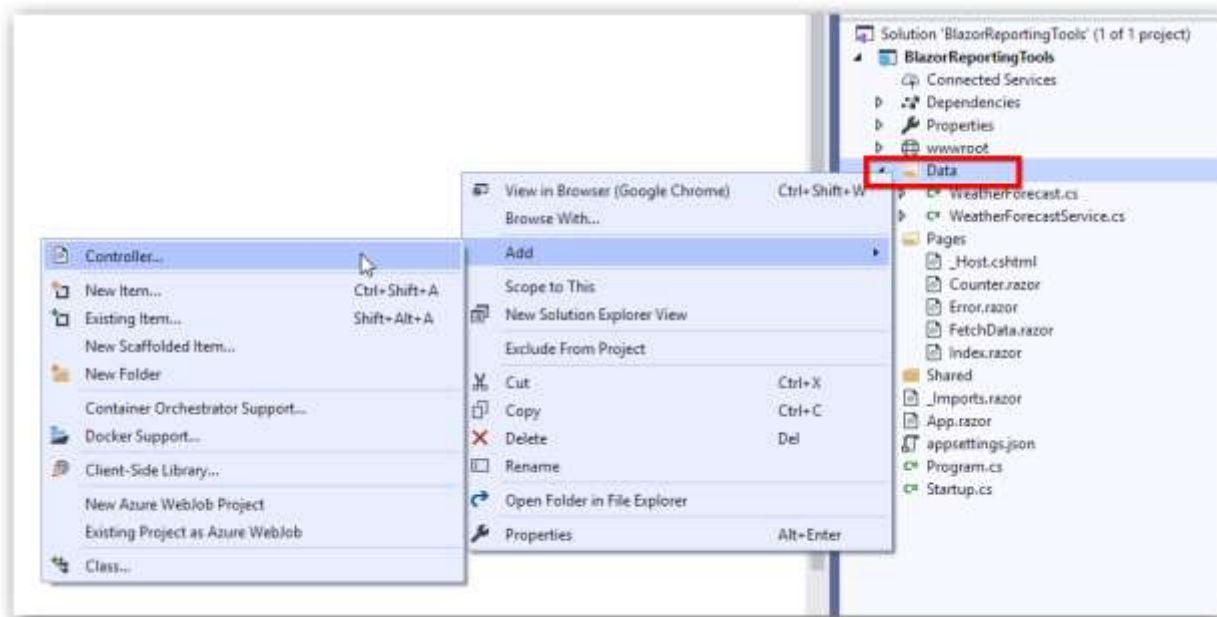
In this section, we are going to create a Web API controller that will be used to process the provided RDL reports.

- Install the following [NuGet packages](#) in the created Blazor application. These are used for processing the RDL reports.

Package	Purpose
BoldReports.Net.Core	Creates Web API service is used to process the reports.
System.Data.SqlClient	Should be referenced in the project when the RDL report renders visual data from the SQL Server or SQL Azure data source based on the RDL design. The package version should be higher than 4.1.0.

Package	Purpose
Microsoft.AspNetCore.Mvc.NewtonsoftJson	ASP.NET Core MVC features that use Newtonsoft.Json. Includes input and output formatter for JSON and JSON Patch. The package version should be higher than 3.1.2.

- Create an empty API controller by right-clicking the **Data** folder, choosing **Add > Controller**, and then naming it **BoldReportsAPIController.cs**



- We are going to implement the **IReportController** interface from the namespace **BoldReports.Web.ReportViewer**, which is used to process the reports in the **wwwroot/resources** folder. Refer to the following code sample for RDL report processing.

```
`csharp
using BoldReports.Web.ReportViewer;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Caching.Memory;
namespace BlazorReportingTools.Data
{
    [Route("api/{controller}/{action}/{id?}")]
    public class BoldReportsAPIController : ControllerBase, IReportController
    {
        // Report viewer requires a memory cache to store the information of consecutive client requests and
```

```
// the rendered report viewer in the server.
private IMemoryCache _cache;
// IWebHostEnvironment used with sample to get the application data from wwwroot.
private IWebHostEnvironment _hostingEnvironment;
public BoldReportsApiController(IMemoryCache memoryCache, IWebHostEnvironment
hostingEnvironment)
{
    _cache = memoryCache;
    _hostingEnvironment = hostingEnvironment;
}
//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
    return ReportHelper.GetResource(resource, this, _cache);
}
// Method will be called to initialize the report information to load the report with ReportHelper for
processing.
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from the application folder
    wwwroot\Resources. sales-order-detail.rdl should be in the wwwroot\Resources application folder.
    System.IO.FileStream inputStream = new System.IO.FileStream(basePath + @"\resources\" +
reportOption.ReportModel.ReportPath + ".rdl", System.IO.FileMode.Open, System.IO.FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}
```

// Method will be called when report is loaded internally to start the layout process with ReportHelper.

[NonAction]

public void OnReportLoaded(ReportViewerOptions reportOption)

{
}

[HttpPost]

public object PostFormReportAction()

{

return ReportHelper.ProcessReport(null, this, _cache);
}

// Post action to process the report from the server based on json parameters and send the result back to the client.

[HttpPost]

public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)

{

return ReportHelper.ProcessReport(jsonArray, this, this._cache);
}
}
}
`

- To request the report processing unit properly, we have changed the router API attribute to include the controller and action names using `[Route("api/{controller}/{action}/{id?}")]`.
- To invoke this Web API with the controller and action, include that information in the `endPoint` routing in the `Startup.cs` file.

`csharp

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

endpoints.MapBlazorHub();

endpoints.MapFallbackToPage("/_Host");

});
`

Initialize the Report Viewer

In this section, we are going to integrate the Bold Reports JavaScript controls by creating an interop file to initialize the report viewer with basic parameters.

- Create a `Data/BoldReportOptions.cs` class with the following code to hold the RDL report rendering properties.

[Data/BoldReportOptions.cs]

```
`csharp
namespace BlazorReportingTools.Data
{
    public class BoldReportViewerOptions
    {
        public string ReportName { get; set; }
        public string ServiceURL { get; set; }
    }
}
```

- Create a `boldreports-interop.js` file inside the `wwwroot/scripts` folder and use the following code snippet to invoke the Bold Report Viewer JavaScript control.

```
`csharp
// Interop file to render the Bold Report Viewer component with properties.
window.BoldReports = {
    RenderViewer: function (elementID, reportViewerOptions) {
        $("#" + elementID).boldReportViewer({
            reportPath: reportViewerOptions.reportName,
            reportServiceUrl: reportViewerOptions.serviceURL
        });
    }
}
```

- Create a folder named `resources` inside the `wwwroot` folder in your application to store the RDL reports. Now, add any RDL reports that have already been created to the newly created folder.

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded [here](#). You can add the reports from the Bold Reports installation location. For more information, refer to the [samples and demos](#) section of the Bold Reports documentation.

- Reference the following online CDN links along with the `boldreports-interop.js` interop file in the head section of `Pages/_Host.cshtml` to use our JavaScript reporting controls in the Blazor application.

```
`html
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<!--Used to render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<!--Used to render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
<!-- Report Viewer component script-->
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Used to render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
<!-- Blazor interop file -->
<script src="~/scripts/boldreports-interop.js"></script>
`
```

- Inject [IJSRuntime](#) and invoke this JavaScript interop with the `sales-order-detail.rdl` report and the created `BoldReportsAPI` URL in the `Pages/Index.razor` file to visualize the report using our viewer.

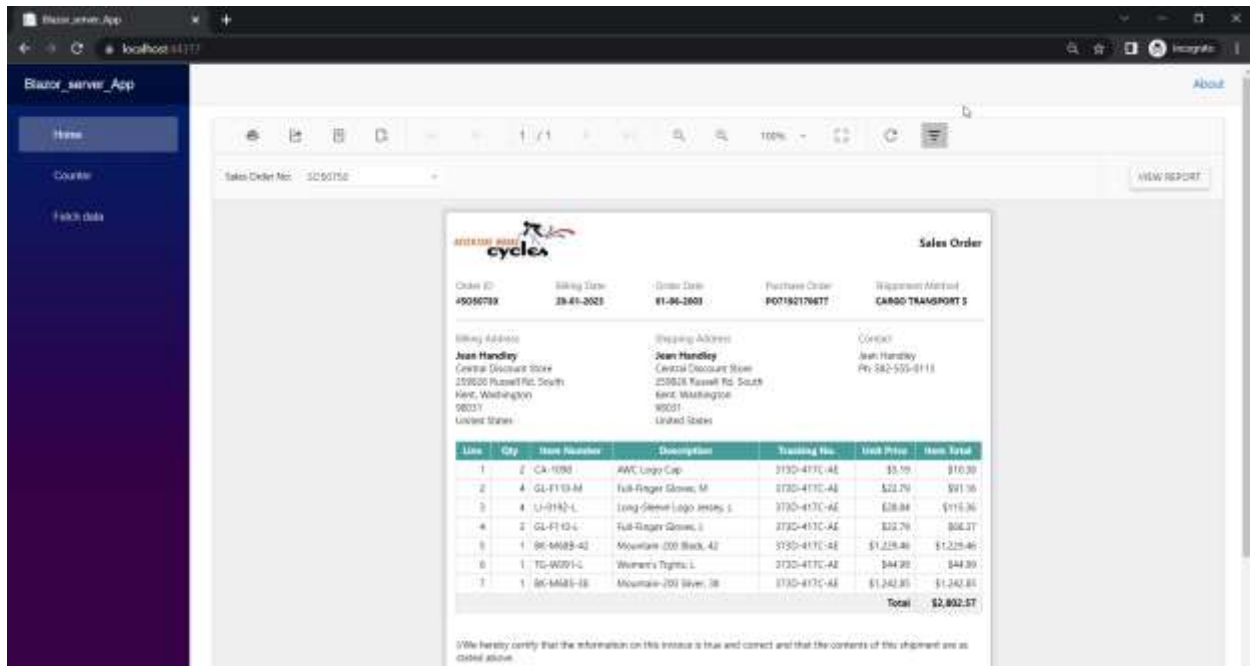
[Pages/Index.razor]

```
`csharp
@page "/"
@using Microsoft.JSInterop
@using Microsoft.AspNetCore.Components
@inject IJSRuntime JSRuntime
@using BlazorReportingTools.Data;
<div id="report-viewer" style="width: 100%;height: 950px"></div>
@code {
    // ReportViewer options
    BoldReportViewerOptions viewerOptions = new BoldReportViewerOptions();
    // Used to render the Bold Report Viewer component in Blazor page.
    public async void RenderReportViewer()
    {
        viewerOptions.ReportName = "sales-order-detail";
        viewerOptions.ServiceURL = "/api/BoldReportsAPI";
        await JSRuntime.InvokeVoidAsync("BoldReports.RenderViewer", "report-viewer", viewerOptions);
    }
    // Initial rendering of Bold Report Viewer
    protected override void OnAfterRender(bool firstRender)
    {
        RenderReportViewer();
    }
}
```

Here, we have created and used **BoldReportViewerOptions** to pass the parameters for report rendering. In the future, if we need any additional parameters, we can just include them in the **BoldReportsOptions.cs** file and use it to render the reports.

Run the Application

Click the **Run** or **F5** button to launch the application. The report will be rendered and displayed as shown in the following screenshot.



Note: You can refer to our feature tour page for the [Blazor Report Viewer](#) to see its innovative features. Additionally, you can view our [Blazor Report Viewer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

See Also
[Bold Reports Licensing](#)

Render RDLC report

The data binding support allows you to view the RDLC reports that exist on the local file system with JSON array and custom business object data collection. The following steps demonstrates how to render a RDLC report with JSON array and custom business object data collection.

Add the RDLC report **Product List.rdlc** from Bold Reports installation location to your application **wwwroot/Resources** folder. For more information, see [Samples and demos](#).

Bind data source in Web API controller

The following steps help you to configure the Web API to render the RDLC report with business object data collection.

1. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
```

```
public double Price { get; set; }
public string Category { get; set; }
public string Ingredients { get; set; }
public string ProductImage { get; set; }
public static IList GetData()
{
    List<ProductList> datas = new List<ProductList>();
    ProductList data = null;
    data = new ProductList()
    {
        ProductName = "Baked Chicken and Cheese",
        OrderId = "323B60",
        Price = 55,
        Category = "Non-Veg",
        Ingredients = "grilled chicken, corn and olives.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Delite",
        OrderId = "323B61",
        Price = 100,
        Category = "Non-Veg",
        Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Tikka",
        OrderId = "323B62",
        Price = 64,
```

```

Category = "Non-Veg",
Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
ProductImage = ""
};
datas.Add(data);
return datas;
}
}
`

```

2. Set the value of the `ProcessingMode` property to `ProcessingMode.Local` in the RDLC report location.
3. To load a report as a stream, create a report stream using the `FileStream` class, and assign the report stream to the `Stream` property.
4. Bind the business object data values collection by adding a new item to the `DataSources` as in the following code snippet.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    reportOption.ReportModel.ProcessingMode = ProcessingMode.Local;
    FileStream inputStream = new FileStream(basePath + @"\resources\" +
    reportOption.ReportModel.ReportPath + ".rdlc", System.IO.FileMode.Open, System.IO.FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
    Value = ProductList.GetData() });
}
`

```

Here, the `Name` is case sensitive and it should be same as in the data source name in the report definition.

The **Value** accepts IList, DataSet, and DataTable inputs.

5. Open the **Pages/Index.razor** page and set Report name.

```
`csharp
// ReportViewer options
BoldReportViewerOptions viewerOptions = new BoldReportViewerOptions();
// Used to render the Bold Report Viewer component in Blazor page.
public async void RenderReportViewer()
{
    viewerOptions.ReportName = "Product List";
    viewerOptions.ServiceURL = "/api/BoldReportsAPI";
    await JSRuntime.InvokeVoidAsync("BoldReports.RenderViewer", "report-viewer", viewerOptions);
}
`
```

Extensions

See Also

- [Custom Data Extension](#)

Custom Data Extension

This section explains the steps required to create and load data extensions in Web Blazor Application

See Also

- [Configure Data Extension](#)

Configure a new data source extension in Blazor Report Viewer

The Blazor Report Viewer provides SQL, ODBC and OLEDB data sources as built in support data sources. Other data source like Web API, JSON, XML, and OData are provided as extension data sources.

This documentation provides step by step procedure to register and connect with new extension data sources in Report Viewer Application.

Create Blazor Report Viewer Application

Refer [Getting Started](#) and create a Blazor Report Viewer Application.

Install data source extension from NuGet

Based on the required data connector install the respective NuGet package to the application. The NuGet packages name for each data connectors are provided in below table,

Data source	Package Name	Assembly Name

Configure a new data source extension in Blazor Report Viewer **Install** data source extension from NuGet

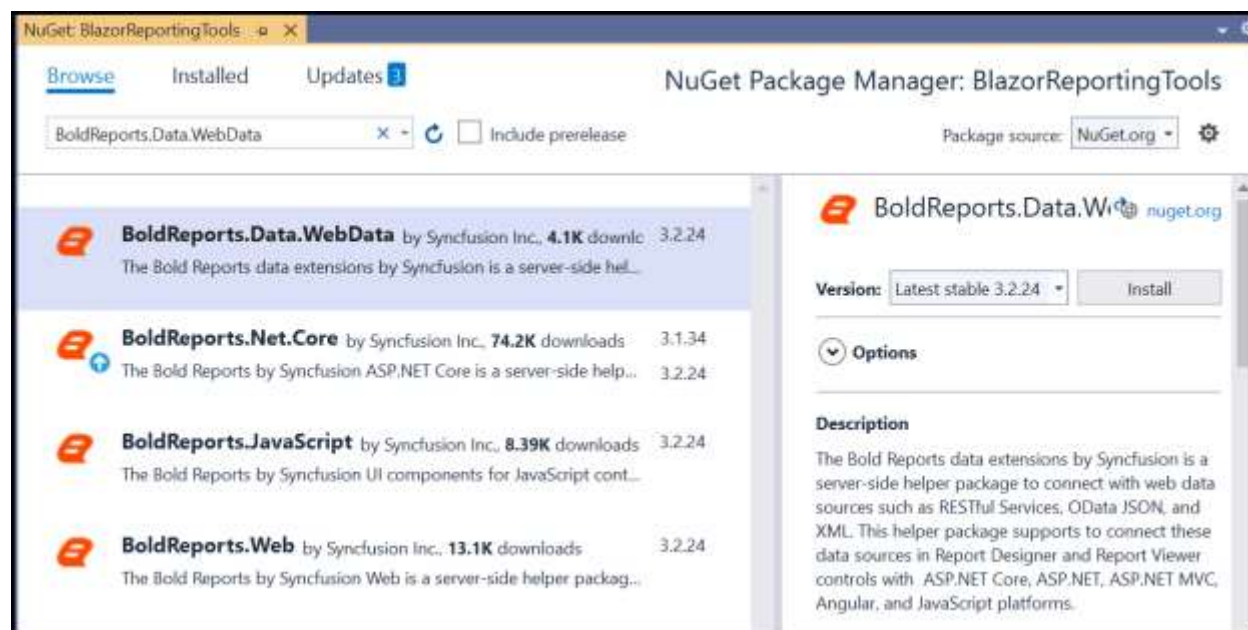
|-----|-----|-----|
|Web data sources(WebAPI, JSON, XML, and OData) | BoldReports.Data.WebData |
BoldReports.Data.WebData.dll |
PostgreSQL data sources	BoldReports.Data.PostgreSQL	BoldReports.Data.PostgreSQL.dll
CSV data sources	BoldReports.Data.Csv	BoldReports.Data.Csv.dll
Excel data sources	BoldReports.Data.Excel	BoldReports.Data.Excel.dll
MySQL data sources	BoldReports.Data.MySQL	BoldReports.Data.MySQL.dll
Oracle data sources	BoldReports.Data.Oracle	BoldReports.Data.Oracle.dll

For example, to register and load web data sources in the application install *BoldReports.Data.WebData* package.

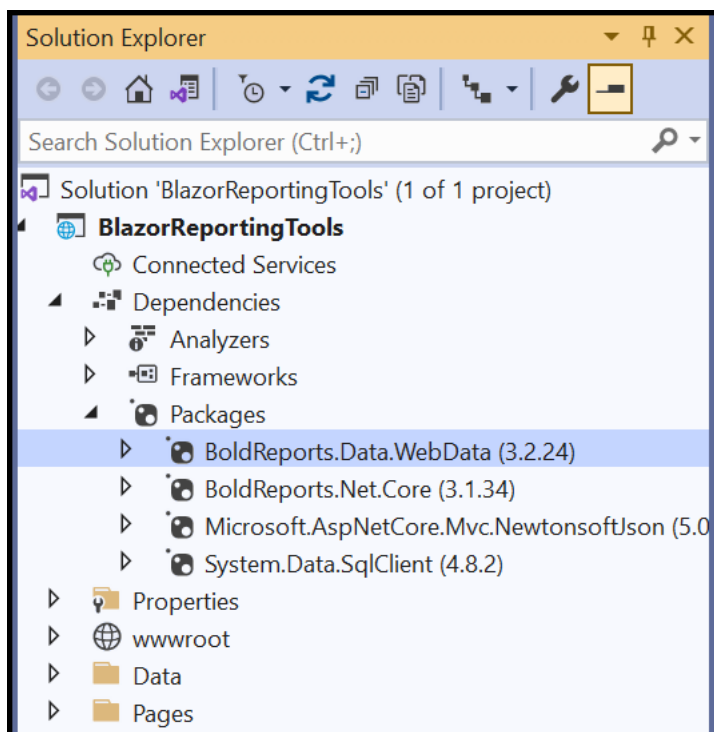
Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring NuGet packages.

Search for **BoldReports.Data.WebData** NuGet package, and install it in your application.



BoldReports.Data.WebData will install into your application. Click OK. Now, the assembly will be added in the respective project references.



Register data source extension

1. Open the code-behind file `Startup.cs` and add the following using statement.

```
`csharp
using BoldReports.Web;
`
```

2. Then add the following code to register extension assembly in `Startup` method.

```
`csharp
public Startup(IConfiguration configuration)
{
    //Use the below code to register extensions assembly into report viewer
    ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {
        "BoldReports.Data.WebData" });

    //To register multiple data extensions, provide the assembly name's as list of strings. For example:
    "ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {
        "BoldReports.Data.WebData", "BoldReports.Data.Excel"};

    //Incase the data source extensions fails to register or any error occurs replace the code as below,
    "ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string>
    {System.IO.Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory) +
    "BoldReports.Data.WebData.dll" });"
```

```
...
...
}
`
```

Run the application

1. Run the Blazor Report Viewer Application
2. Now we can able to see the company sales report rendered with JSON data

Components	2002				2003			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Bottom Brackets							\$17,454	\$13,339
Brakes							\$26,659	\$18,571
Chains							\$3,476	\$2,218
Cranksets							\$80,244	\$44,127
Derailleurs							\$25,385	\$18,974
Forks			\$26,167	\$23,543	\$9,914	\$16,345		
Handlebars			\$35,341	\$18,309	\$6,275	\$17,194	\$43,624	\$21,676
Headsets			\$19,702	\$16,382	\$10,960	\$14,102		
Mountain Frames	\$127,558	\$220,935	\$608,353	\$443,599	\$236,070	\$440,261	\$827,288	\$565,230
Pedals							\$54,185	\$39,901
Road Frames	\$47,486	\$155,311	\$657,715	\$458,059	\$132,692	\$457,089	\$755,821	\$286,592
Saddles							\$24,908	\$12,940
Touring Frames							\$666,977	\$367,783
Wheels			\$288,628	\$163,822	\$63,186	\$163,930	\$1,678	\$83
Total	\$68,458	\$191,979	\$1,514,448	\$1,013,455	\$354,448	\$1,133,455	\$1,678	\$83

Download the custom data extension configured Report Viewer sample from the link [Sample](#)

How to queries for Bold Reports ReportViewer

This section helps to get the answer for the frequently asked how to queries in Bold Reports ReportViewer.

- [How to resolve Cannot read property enableBlazorMode of undefined error?](#)
- [How to use Bold Reports with Syncfusion Blazor?](#)
- [How to resolve Circuit has been shut down due to error?](#)

How to resolve Cannot read property enableBlazorMode of undefined error when using with Bold Reports

When using the Bold Reporting components with Syncfusion Blazor components. You have to refer the Syncfusion Blazor script in `_Host.cshtml` file. Follow these steps in your application to avoid this issue.

Refer the Syncfusion Blazor script in `_Host.cshtml` file.

```
`html
<head>
```

```
<script src="_content/Syncfusion.Blazor/scripts/syncfusion-blazor.min.js"></script>
</head>
`
```

To use manual scripts in the application, register the Blazor service in `~/Startup.cs` file by using the true parameter mentioned as follows.

```
`csharp
public class Startup
{
public void ConfigureServices(IServiceCollection services)
{
....
....
services.AddSyncfusionBlazor(true);
}
}
`
```

[See also](#)

- [How to use Bold Reports with Syncfusion Blazor?](#)

How to use Bold Reports with Syncfusion Blazor

When using the Bold Reporting components with Syncfusion Blazor components. You have to refer the Syncfusion Blazor script in `_Host.cshtml` file. Follow these steps in your application.

Refer the Syncfusion Blazor script in `_Host.cshtml` file.

```
`html
<head>
<script src="_content/Syncfusion.Blazor/scripts/syncfusion-blazor.min.js"></script>
</head>
`
```

To use manual scripts in the application, register the Blazor service in `~/Startup.cs` file by using the true parameter mentioned as follows.

```
`csharp
public class Startup
{
public void ConfigureServices(IServiceCollection services)
```



```
{
....
....
services.AddSyncfusionBlazor();
}
}
,
```

For Style, you need to refer the compatibility styles of EJ2 from Bold Reports and Bold Reports styles should be referred before EJ2 styles.

```
`html
<link href="https://cdn.syncfusion.com/ej2/styles/compatibility/material.css" rel="stylesheet" />
<link href="https://cdn.boldreports.com/5.4.20/content/bold.widgets.core.compatibility.min.css"
rel="stylesheet" />
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.theme.compatibility.min.css"
rel="stylesheet" />
,
```

You should refer EJ2 scripts before Bold Reports scripts as follows.

```
`html
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>BoldReports ReportViewer and EJ2 controls</title>
@ Syncfusion Essential JS 2 Styles @
<link rel="stylesheet" href="https://cdn.syncfusion.com/ej2/styles/compatibility/material.css" />
@ BoldReports Styles @
<link href="https://cdn.boldreports.com/5.4.20/content/bold.widgets.core.compatibility.min.css"
rel="stylesheet" />
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.theme.compatibility.min.css"
rel="stylesheet" />
@Default scripts@
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-easing/1.3/jquery.easing.min.js"
type="text/javascript"></script>
<script src="https://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
@ Syncfusion Essential JS 2 Scripts @
```

```

<script src="_content/Syncfusion.Blazor/scripts/syncfusion-blazor.min.js"></script>
@ BoldReports Scripts @
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Used to render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
</head>
`

```

How to resolve circuit shut down issue due to error

This is a Blazor [framework error](#). Suggest you to upgrade the ASP.NET Core to the [latest version](#) of NET Core 3.1 or migrate your application to .NET 5 to resolve this issue.

Migrate to Report Viewer v2.0 component

This section provides simple step-by-step instructions to update your existing Report Viewer application to our latest modern v2.0 scripts, styles and components.

1. Open the `Pages/_Host.cshtml` page that contains Report Viewer and its scripts.
2. Remove the following older scripts and CSS references in your `Pages/_Host.cshtml` page.
 - o `bold.reports.all.min.css`
 - o `jquery.min.js`
 - o `ej2-base.min.js`
 - o `ej2-data.min.js`
 - o `ej2-pdf-export.min.js`
 - o `ej2-svg-base.min.js`
 - o `ej2-lineargauge.min.js`
 - o `ej2-circulargauge.min.js`
 - o `ej2-maps.min.js`
 - o `ej.chart.min.js`
 - o `bold.reports.common.min.js`
 - o `bold.reports.widgets.min.js`
 - o `bold.report-viewer.min.js`
3. Add the following v2.0 scripts and CSS references in your `_Layout.cshtml` page.

```
`html
```

```
<!-- Report Viewer component styles -->
```

```

<link href="https://cdn.boldreports.com/5.4.20/content/v2.0/tailwind-light/bold.report-viewer.min.css"
rel="stylesheet" />
<script src="https://cdn.jsdelivr.net/npm/jquery/3.6.0/jquery.min.js"></script>
<!-- Report Viewer component dependent script -->
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.widgets.min.js"></script>
<!-- Report Viewer component script -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/bold.report-viewer.min.js"></script>
<!-- Blazor interop file -->
<script src="~/scripts/boldreports-interop.js"></script>

```

Bold Report Writer

Report Writer is a class library that enables the user to render reports defined in Microsoft's RDL format as PDF, Word, Excel or CSV documents.

The important features of Report Writer are listed as follows:

- RDL Specification - Supports RDL specification from RDL XML schema 2008 version. You can find the list of RDL schema versions from [here](#).
- Data sources - You can use advanced database servers data sources in Report Writer (SQL and Oracle).
- Charts - Show all basic types of charts that are available in Microsoft RDL reports.
- Tablix - Shows the summaries and simple tables.
- Gauge - Shows measurement values by using the expression values.
- Textbox - Shows textbox data with expression support.
- Export - Export report as PDF, Word, Excel, and CSV.
- Report parameter - Views the report based on the report parameter value.

Report Writer with Blazor Application

This section explains the steps required to add a web Report Writer to a Blazor application.

Prerequisites

Before getting started with the Bold Web Report Writer, make sure that your development environment includes the following requirements.

- [Visual Studio 2022](#) with ASP.NET and Web Development workloads.
- [.NET Core 6.0](#) Framework.

Create a Blazor application

To get started, create a Blazor Server App template application and name it **BlazorReportingTools** using Visual Studio 2022. If this template is not available, then refer to this [link](#) to configure the system environment.

The source code for this Blazor reporting components app is available on [GitHub](#).

List of dependency libraries

1. In the Solution Explorer tab, right-click the project or solution , and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.Net.Core** and **System.Data.SqlClient** packages, and install them in your Core application. The following table provides details about the packages and their usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Exports the report to a PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core**, and **Syncfusion.XlsIO.Net.Core** packages.

Syncfusion.Pdf.Net.Core | Exports the report to a PDF.

Syncfusion.DocIO.Net.Core | Exports the report to a Word.

Syncfusion.XlsIO.Net.Core | Exports the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is the base library of the **Syncfusion.XlsIO.Net.Core** package.

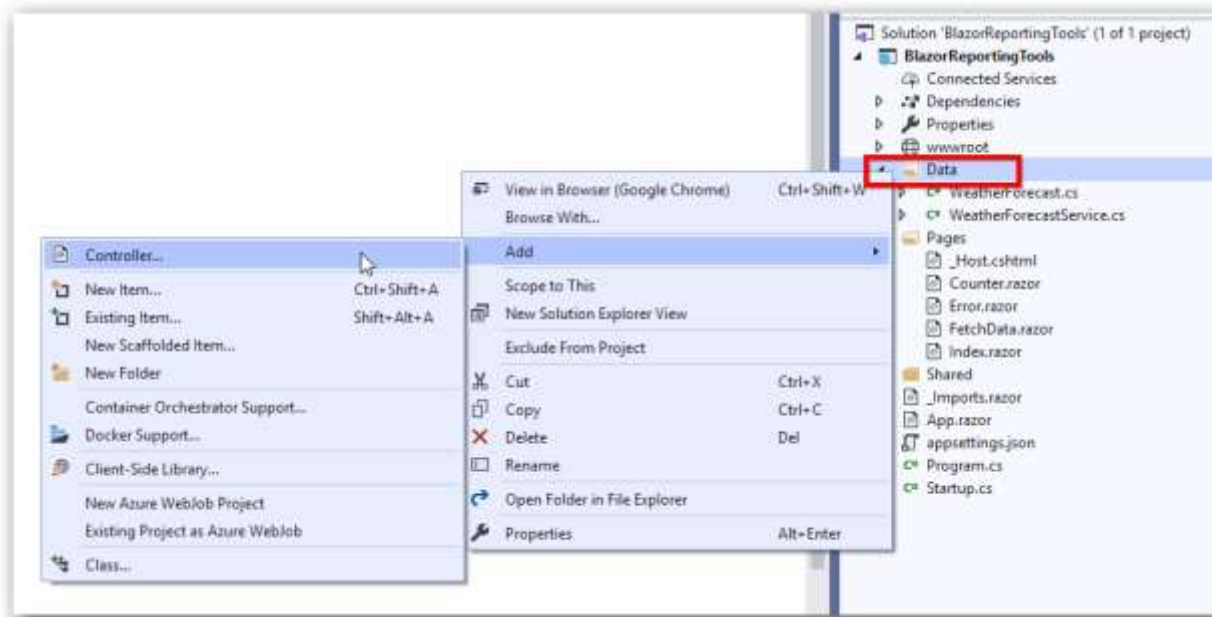
Newtonsoft.Json | Serializes and deserializes data for the Report Writer. It is a mandatory package for Report Writer, and the package version should be 10.0.1 or higher.

In this tutorial, the **sales-order-detail.rdl** report is used, and it can be downloaded [here](#). You can get the reports from the Bold Reports installation location. For more information, refer to the [samples and demos](#) section.

Create a PDF Document in Web API

In this section, we are going to create a Web API controller that will be used to export the provided RDL reports.

- Create a folder **Resources** into the **wwwroot** folder in your application. Copy and paste the sample RDL reports into the **Resources** folder.
- Create an empty API controller by right-clicking the **Data** folder, choosing **Add > Controller**, and then naming it **BoldReportWriterController.cs**.



- Create local variables inside the **BoldReportWriterController** class.

```
`csharp
// IWebHostEnvironment used with sample to get the application data from wwwroot.
private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
// IWebHostEnvironment initialized with controller to get the data from application data folder.
public BoldReportWriterController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment
hostingEnvironment)
{
    _hostingEnvironment = hostingEnvironment;
}
`
```

- Open the **BoldReportWriterController.cs** file in your application and add the **Export()** function to load the report as a stream. Refer to the following code snippet.

```
`csharp
[HttpGet]
public IActionResult Export(string writerFormat)
{
    // Here, we have loaded the sales-order-detail sample report from application the folder
    wwwroot\Resources.
```

```

FileStream inputStream = new FileStream(_hostingEnvironment.WebRootPath + @"\Resources\sales-
order-detail.rdl", FileMode.Open, FileAccess.Read);
MemoryStream reportStream = new MemoryStream();
inputStream.CopyTo(reportStream);
reportStream.Position = 0;
inputStream.Close();
.....
}
`

```

- Initialize the Report Writer instance with the report stream and set the specified export format and file name for the export document.

```

`csharp
public IActionResult Export(string writerFormat)
{
.....
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
string fileName = null;
WriterFormat format;
string type = null;
fileName = "sales-order-detail.pdf";
type = "pdf";
format = WriterFormat.PDF;
}
`

```

- You can use the **Save** method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as a stream.

```

`csharp
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;

```

```
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
`
```

- Refer to the following complete code snippet, to get the exported file stream.

```
`csharp
[Route("api/{controller}/{action}/{id?}")]
public class BoldReportWriterController : ControllerBase
{
    // IWebHostEnvironment used with sample to get the applicationdata from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // IWebHostEnvironment initialized with controller to get the data from application data folder.
    public BoldReportWriterController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment
    hostingEnvironment)
    {
        _hostingEnvironment = hostingEnvironment;
    }
    [HttpGet]
    public IActionResult Export(string writerFormat)
    {
        // Here, we have loaded the sales-order-detail sample report from application the folder
        wwwroot\Resources.

        FileStream inputStream = new FileStream(_hostingEnvironment.WebRootPath + @"\Resources\sales-
        order-detail.rdl", FileMode.Open, FileAccess.Read);
        MemoryStream reportStream = new MemoryStream();
        inputStream.CopyTo(reportStream);
        reportStream.Position = 0;
        inputStream.Close();
        BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
        string fileName = null;
        WriterFormat format;
        string type = null;
        if (writerFormat == "PDF")
```

```
{
fileName = "sales-order-detail.pdf";
type = "pdf";
format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
fileName = "sales-order-detail.doc";
type = "doc";
format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
fileName = "sales-order-detail.csv";
type = "csv";
format = WriterFormat.CSV;
}
else
{
fileName = "sales-order-detail.xls";
type = "xls";
format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}
}
```


Client Side

1. Use the following code snippet in the `_Host.cshtml` file to invoke the Web API from the client side.

```
`html
<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
<script>
function downloadFile() {
var writerformat = getWriterFormat();
location.href = 'api/BoldReportWriter/Export?writerFormat=' + writerformat;
};
function getWriterFormat() {
var formatType = "";
if ($('#rbtnPDF').is(':checked')) {
formatType = $('#rbtnPDF').val();
} else if ($('#rbtnWord').is(':checked')) {
formatType = $('#rbtnWord').val();
} else if ($('#rbtnxls').is(':checked')) {
formatType = $('#rbtnxls').val();
} else if ($('#rbtnCSV').is(':checked')) {
formatType = $('#rbtnCSV').val();
}
return formatType;
}
</script>
```

2. You can add the export file types to the home page to choose the format you want to export in the Report Writer. Copy and paste the following code snippet into the `Index.razor` file in your application.

```
`html
@page "/"
```

```
@using Microsoft.JSInterop
@using Microsoft.AspNetCore.Components
@inject IJSRuntime JSRuntime
<div class="Common">
<div class="tablediv">
<div class="rowdiv">
<label id="design">
Choose the any one of the format to export the document in Report Writer.
<br />
<br />
</label>
</div>
<div class="rowdiv">
<div class="celldiv" style="padding:10px">
<label>
<strong> Save As :</strong>
</label>
<input id="rbtnPDF" type="radio" name="writerFormat" value="PDF" checked="checked" style="margin-left: 15px" />
<label for="rbtnPDF" style="padding:0px 5px 0px 2px">
PDF
</label>
<input id="rbtnWord" type="radio" name="writerFormat" value="Word" style="margin-left: 15px" />
<label for="rbtnWord" style="padding:0px 5px 0px 2px">
Word
</label>
<input id="rbtnxls" type="radio" name="writerFormat" value="xls" style="margin-left: 15px" />
<label for="rbtnxls" style="padding:0px 5px 0px 2px">
Excel
</label>
<input id="rbtnCSV" type="radio" name="writerFormat" value="CSV" style="margin-left: 15px" />
<label for="rbtnCSV" style="padding:0px 25px 0px 2px ">
CSV
```

```
</label>
<input class="buttonStyle" type="submit" name="button" value="Generate" style="width:150px;"
@onclick="PDFReport" />
</div>
</div>
</div>
</div>
@code {
public async void PDFReport()
{
JSRuntime.InvokeVoidAsync("downloadFile");
}
}
`
```

3. Ensure that the application configuration is set correctly in the `Program.cs` file,

```
`csharp
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
var app = builder.Build();
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
app.UseExceptionHandler("/Error");
app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");
```

```
app.Run();
```

```
,
```

4. Now, execute the application and export the report to the specified export format in your Report Writer application.

Reporting tools for Angular

Enterprise-class reporting tools for Angular development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your Angular applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

Reporting tools for Angular

Enterprise-class reporting tools for Angular development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your Angular applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

System Requirements

This topic describes the software and hardware requirements for setting up the development environment of Bold Reports Angular.

Supported Operating Systems

- Windows 7+, 8+
- Windows Server 2008 R2+

Development Environments

By using the following IDEs, you can develop the Bold Reports Angular:

- [Microsoft Visual Studio Code](#)
- Internet Information Services (IIS) 7.0+
- [Node JS](#) (version 8.x or 10.x)
- [NPM](#) (v3.x.x or higher)
- Angular 4+

Browser Compatibility

- IE 9+
- Microsoft Edge
- Mozilla Firefox 22+
- Chrome 17+
- Opera 12+
- Safari 5+

See Also

- [Licensing procedure for deployment](#)

Overview

The Report Viewer is a visualization control used to display SSRS, RDL, RDLC, and Bold Report Server reports within web applications. It allows you to view RDL/RDLC reports with or without using SSRS or Bold Report Server. You can bind data sources, parameters, and render reports with all major capabilities of RDL reporting and export the report to PDF, Excel, CSV, Word, PowerPoint, and HTML formats. Some of the key features are:

- Renders interactive reports with drill down, drill through, hyperlinks, and interactive sorting.
- Easily customize each element of the Report Viewer and provide events for report processing customization.
- Supports jQuery, Angular, React, Blazor, ASP.NET Core, ASP.NET MVC, ASP.NET WebForms, WPF, and UWP.

Introducing the newly redesigned Report Viewer! We have given it a refreshing new look that embraces sleek and modern design, aligning perfectly with the latest web design trends. It allows seamless integration into your application. For detailed guidance on the migration process, we recommend you to refer our [Report Viewer v2.0 migration document](#).

Display SSRS RDL report in Bold Reports Angular Report Viewer

This section explains the steps required to create your first Angular reporting application in Angular CLI to display an already created SSRS RDL report in the Bold Reports Angular Report Viewer without using a Report Server, refer to the following steps.

If you are using lower version of [Bold Reports Angular](#) (< v2.2.28), then refer the [Getting Started for earlier version](#).

To get start quickly with Report Viewer, you can check on this video:

youtube: <https://youtu.be/MZOW6HkpMi4>

Prerequisites

Before you begin, make sure your development environment includes the following:

- [Node JS](#) (version 8.x or 10.x and above)
- [NPM](#) (v3.x.x or higher)

Install the Angular CLI

Angular provides the easiest way to set Angular CLI projects using the [Angular CLI](#) tool. To install the CLI application globally on your machine, run the following command in the Command Prompt.

```
`typescript
npm install -g @angular/cli@latest
`
```

To learn more about `angular-cli` commands, click [here](#).

Create a new application

To create a new Angular application, run the following command in the Command Prompt.

```
`typescript
ng new project-name
E.g : ng new reportviewerapp
`
```

The `ng new` command prompts you for information about features to include in the initial app project. Accept the defaults by pressing the Enter or Return key.

```

E:\Report viewer demo>ng new reportviewerapp
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS [ http://sass-lang.com/documentation/file.SASS_REFERENCE.html#syntax ]
  Sass [ http://sass-lang.com/documentation/file.INDENTED_SYNTAX.html       ]
  Less  [ http://lesscss.org                                                 ]
  Stylus [ http://stylus-lang.com                                           ]
  
```

Configure the Bold Report Viewer in Angular CLI

Reporting tools packages are distributed in NPM package as [@boldreports/angular-reporting-components](https://www.npmjs.com/package/@boldreports/angular-reporting-components).

1. To configure the Report Viewer component, change the directory to your application's root folder.

```

`typescript
cd project-name
E.g: cd reportviewerapp
`
  
```

2. Run the following commands to install the [Bold Reports Angular](https://www.npmjs.com/package/@boldreports/angular-reporting-components) library.

Angular version	NPM package installation
-----	-----
12 or greater than 12	npm install @boldreports/angular-reporting-components@latest --save-dev
less than 12	npm install @boldreports/angular-reporting-components@5.1.27 --save-dev

Note: It is important to exercise caution during the package installation process to avoid potential complications with package compilation.

3. Also, Install the [Bold Reports typings](https://www.npmjs.com/package/@boldreports/angular-reporting-components) by executing the below command.

```

`typescript
  
```

```
npm install --save-dev @boldreports/types
```

```
,
```

4. Register the `@bold-reports/types` under the `typeRoots` and add the typings `jquery` and `reports.all` to the `tsconfig.app.json` file.

```
`js
{
...
...
"compilerOptions": {
...
...
"typeRoots": [
"node_modules/@types",
"node_modules/@boldreports/types"
],
"types": [
"jquery",
"reports.all"
]
},
...
...
}
```

```
,
```

5. The Report Viewer requires a `window.jQuery` object to render the component.
 - For Angular version 12 or higher, create a file `src/globals.ts` and import the `jQuery` as shown in the below code snippet.

```
`js
import * as jquery from 'jquery';
let windowInstance = (window as { [key: string]: any });
windowInstance['jQuery'] = jquery;
windowInstance['$'] = jquery;
```


,

- In the `src/app/app.module.ts` file, import the `src/globals.ts` file prior to importing the **Bold Reports** viewer or designer component in the `app.module.ts`.

```
`js
```

```
import './globals';
```

,

- For **Angular version lesser than 12**, import `jQuery` into the `src/polyfills.ts` file as shown in the following code snippet.

```
`js
```

```
import * as jquery from 'jquery';
```

```
let windowInstance = (window as { [key: string]: any });
```

```
windowInstance['jQuery'] = jquery;
```

```
windowInstance['$'] = jquery;
```

,

Adding CSS reference

Add the Report Viewer component style (`bold.reports.all.min.css`) as given in the `angular.json` file within the `projectname > styles` section (For example, `reportviewerapp > styles`).

If you are using an Angular 6 or lower version project, add the changes to the `angular-cli.json` file.

```
`js
```

```
{
```

```
"$schema": "./node_modules/@angular/cli/lib/config/schema.json",
```

```
"project": {
```

```
"name": "reportviewerapp"
```

```
},
```

```
"reportviewerapp": [
```

```
{
```

```
"root": "src",
```

```
"outDir": "dist",
```

```
...
```

```
...
```

```
"styles": [
```

```
"styles.css",
```

```
"/node_modules/@boldreports/javascript-reporting-
controls/Content/material/bold.reports.all.min.css"
```

```
],
```

```
"scripts": [],
```

```
...
```

```
...
```

```
}
```

```
`
```

In the previous code, the `material` theme is used. You can modify the theme based on your application, refer the following syntax: `./node_modules/@boldreports/javascript-reporting-controls/Content/[theme-name]/bold.reports.all.min.css`

Adding Report Viewer component

To add the Report Viewer component, refer to the following steps:

1. Open the `app.module.ts` file.
2. You can replace the following code snippet in the `app.module.ts` file.

```
`typescript
```

```
import { BrowserModule } from '@angular/platform-browser';
```

```
import { NgModule } from '@angular/core';
```

```
import { BoldReportViewerModule } from '@boldreports/angular-reporting-components';
```

```
import { AppComponent } from './app.component';
```

```
// Report viewer
```

```
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
```

```
// data-visualization
```

```
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
```

```
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
```

```
@NgModule({
```

```
declarations: [
```

```
AppComponent
```

```
],
```

```
imports: [
```

```
BrowserModule,
```

```
BoldReportViewerModule
```

```
],
```

```
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
`
```

3. Open the `index.html` file and refer to the following scripts in the `<head>` tag.

```
`html
<!-- Data-Visualization -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
`
```

4. Open the `app.component.html` file and initialize the Report Viewer.
5. You can replace the following code snippet in the `app.component.html` file.

```
`javascript
<bold-reportviewer id="reportViewer_Control" style="width: 100%;height: 950px">
</bold-reportviewer>
`
```

6. Open the `app.component.ts` and replace the following code example.

```
`typescript
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
```

```
})  
export class AppComponent {  
  title = 'reportviewerapp';  
  public serviceUrl: string;  
  public reportPath: string;  
  constructor() {  
    // Initialize the Report Viewer properties here.  
  }  
}  
`
```

If you have faced the issue `'ej' is not defined` after the above configuration in the Angular CLI latest version 7, refer to the following code snippet in your application where you have rendered Syncfusion Components (model file) to resolve the issue.

```
`typescript  
/// <reference types="reports.all" />  
`
```

Create Web API service

The Report Viewer requires a Web API service to process the report files. You can skip this step and use the online [Web API services](#) to preview the already available reports or you should create any one of the following Web API services:

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

If you are looking to load the report directly from the SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#).

Adding already created report

If you have created a new service, you can add the reports from the Bold Reports installation location. For more information, refer to the [samples and demos](#) section.

1. Create a folder `Resources` in your Web API application to store the RDL reports and add already created reports to it.
2. Add already created reports to the newly created folder.

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded at this [link](#).

Refer to the [create RDL report](#) section for creating new reports.

Set report path and Web API service

To set the report path and the Web API service, open the `app.component.ts` file and add the codes as shown in the `constructor`.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'jsreport-sample';
  public serviceUrl: string;
  public reportPath: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = '~/Resources/docs/sales-order-detail.rdl';
  }
}
```

In the above code, the `reportServiceUrl` is used from online URL. You can host the Bold Reports service at any Azure, AWS, or own domain URL and use it in the Report Viewer. You can view the already created Web API service from the [Reporting Service](#) git hub location.

Open the `app.component.html` to set the `reportPath` and `reportServiceUrl` properties of the Report Viewer as in the following.

```
`javascript
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [reportPath] =
"reportPath" style="width: 100%;height: 950px">
</bold-reportviewer>
```

Serve the application

To serve the application, follow these steps:

1. Navigate to the root of the application and run the application using the following command.

```
`typescript
ng serve
```

2. Navigate to the appropriate port `http://localhost:4200` in the browser.
3. Click the **view** option to view the demo.

```
{% tab demoPath="angular-reporting/report-viewer/getting-started"
files="app.component.ts,app.component.html,app.module.ts" %}

{% endtab %}
```

Note: You can refer to our feature tour page for the [Angular Report Viewer](#) to see its innovative features. Additionally, you can view our [Angular Report Viewer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

See Also

[Create RDLC report](#)

[Render RDLC reports](#)

[Preview report in print mode](#)

[Set data source credential for shared data sources](#)

[Change data source connection string](#)

[Production deployment](#)

[List of SSRS server versions are supported in Bold Reports](#)

Load SSRS Report Server reports

Report Viewer has support to load RDL reports from SSRS Report Server. To render SSRS Reports, set the `reportServerUrl`, `reportPath`, and `reportServiceUrl` properties as shown in the following steps.

1. To create your first Angular reporting application in Angular CLI, refer to the [Getting-started](#) section.

If you need to know about the difference between `reportServiceUrl` and `reportServerUrl`, then refer to [Difference between Report Service URL and Report Server URL](#).

2. Set the `reportServerUrl` API on Bold Report Viewer with `WebServiceURL`. Open the `App.js` or `app.component.ts` and replace the following code example.

```
`html
```

```
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [processingMode] =
"Remote" [reportServerUrl] = "serverUrl">
```

```
</bold-reportviewer>
```

```
,
```

```
`typescript
```

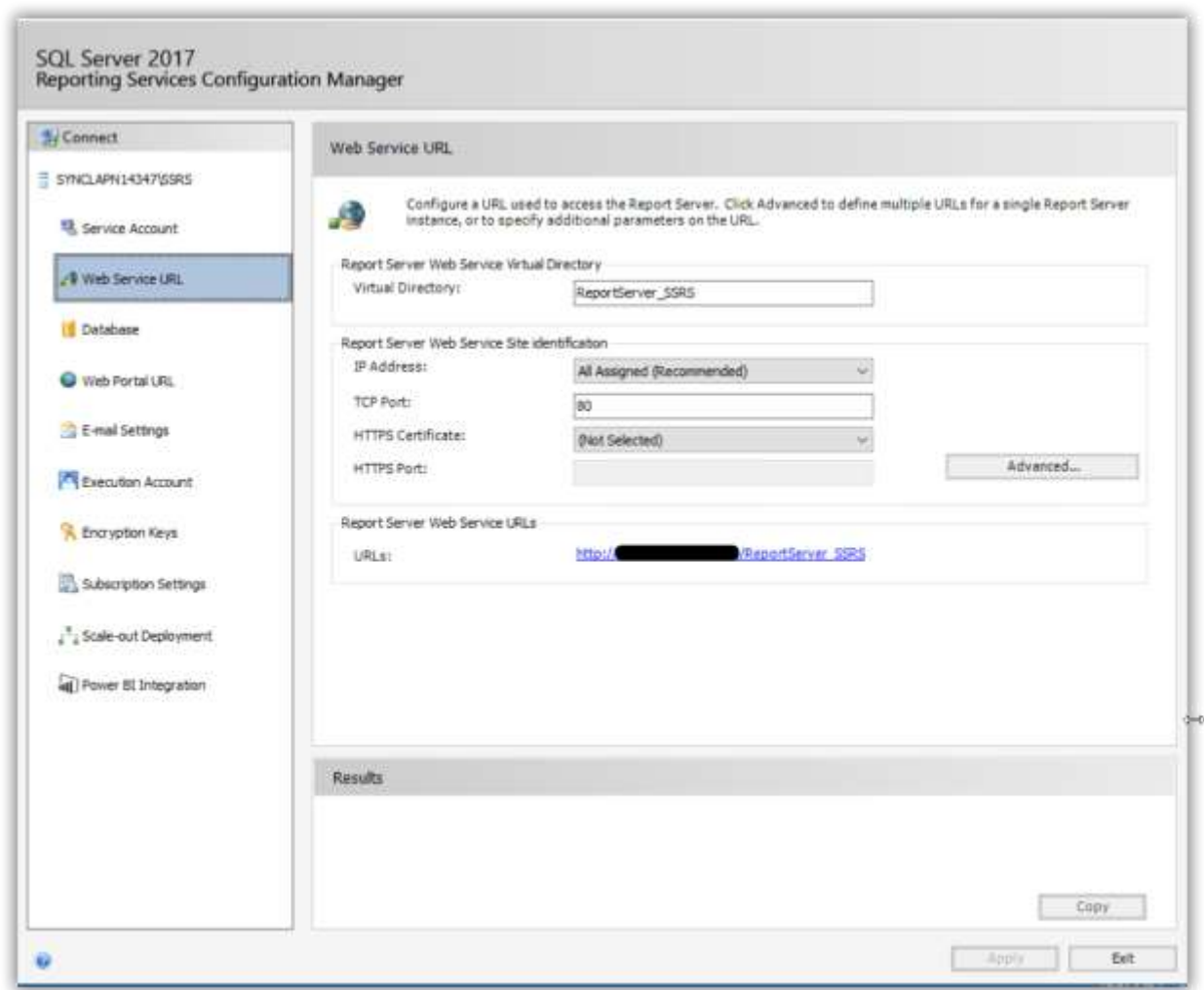
```
import { Component } from '@angular/core';
```

```
@Component({
```

```
selector: 'ej-app',
templateUrl: 'src/reportviewer/reportviewer.component.html',
styleUrls: ['src/reportviewer/reportviewer.component.css']
})
export class ReportViewerComponent {
  public serviceUrl: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https:localhost:7941/api/ReportViewer';
    this.serverUrl = 'http://<servername>/Reports_SSRS';
  }
}
```

The Web Service URL should be set as `reportServerUrl` in the report viewer configuration. The Web Service URL can be found from the Reporting Services Configuration manager under the `Web Service`

URL section, as shown in the following image.



3. Set the report path for loading the reports from the SSRS Report Server. The report path should be in the format of `/folder name/report name`. Open the `app.component.ts` and replace the following code example.

```
`html
```

```
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [processingMode] = "Remote" [reportServerUrl] = "serverUrl" [reportPath]="reportPath">
```

```
</bold-reportviewer>
```

```
,
```

```
`typescript
```

```
import { Component } from '@angular/core';
```

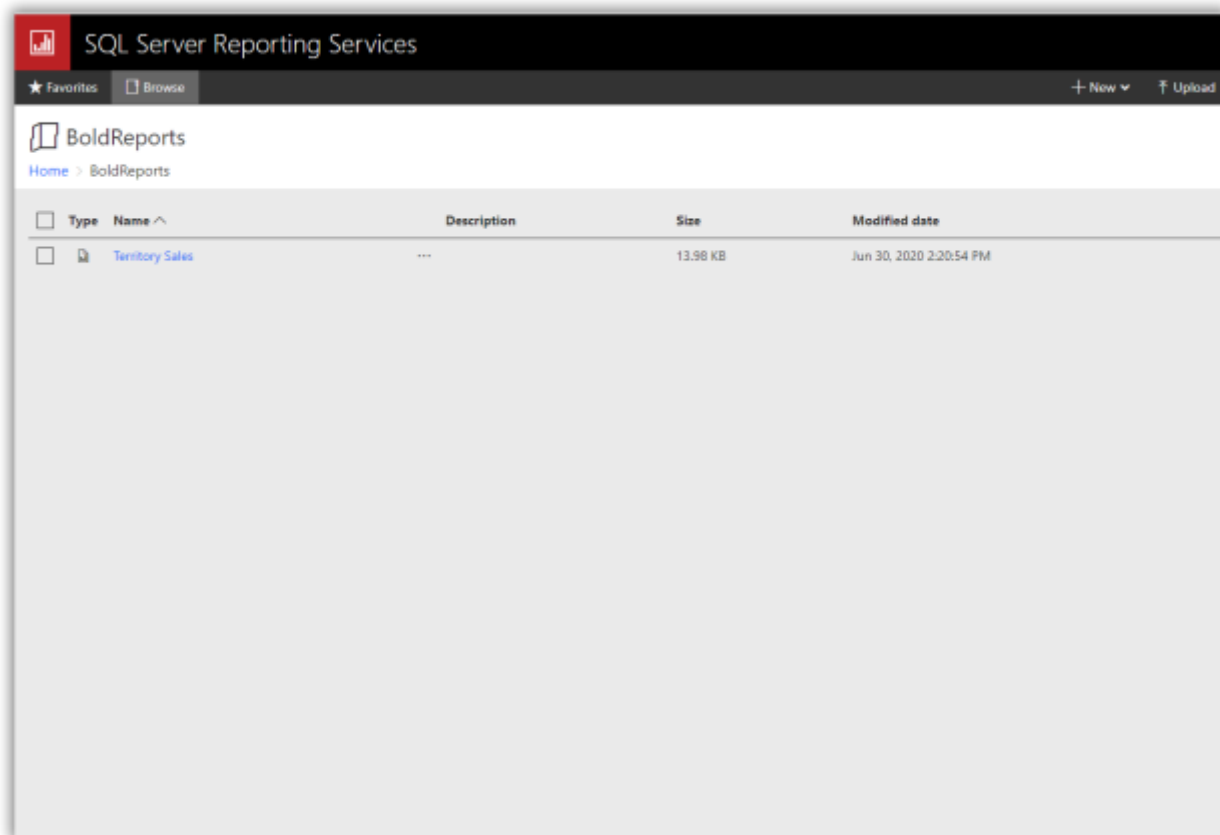
```
@Component({
```

```
selector: 'ej-app',
```



```
templateUrl: 'src/reportviewer/reportviewer.component.html',
styleUrls: ['src/reportviewer/reportviewer.component.css']
})
export class ReportViewerComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https:localhost:7941/api/ReportViewer';
    this.serverUrl = 'http://<servername>/Reports_SSRS';
    this.reportPath = '/BoldReports/Territory Sales';
  }
}
```

The report path can be found from the SSRS Report Server by navigating to the path of the report to be loaded, as shown in the following image.



Network credentials for SSRS

The network credentials are required to connect with the specified SSRS Report Server using the Report Viewer. Specify the `ReportServerCredential` property in the Web API Controller `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server credential
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
    "RDLReport1");
}
```

If you are facing problem to access the SSRS Report server reports, you can refer [How to provide the permission for user to access the SSRS Report Server reports](#).

Set data source credential for shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS server. If the report has any data source that uses credentials to connect with the database, then you should specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server and data source credentials
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
    "RDLReport1");
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "<username>", "<password>"));
}
```

Data source credentials should be added to the shared data sources that do not have credentials in the connection strings.

Change data source connection string

You can change the connection string of a report data source before it is loaded in the Report Viewer. The `DataSourceCredentials` class provides the option to set and update the modified connection string as in the following code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "<username>", "<password>", "Data
    Source=<instancename>;Initial Catalog=<database>;"));
}
`
```

The previous code shows an option to change the connection string only, but the class provides multiple options to change data source information. To learn more about this, refer to this [DataSourceCredentials](#) class.

See also

[Does Bold Report Viewer use SSRS Report processing?](#)

Load SharePoint Server reports

To render SharePoint server reports, set the `reportServerUrl`, `reportPath` and `reportServiceUrl` properties as shown in the following code snippet.

```
`html
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [processingMode] =
"Remote" [reportServerUrl] = "serverUrl" [reportPath] = "reportPath">
</bold-reportviewer>
`
```

```
`typescript
import { Component } from '@angular/core';
@Component({
    selector: 'ej-app',
    templateUrl: 'src/reportviewer/reportviewer.component.html',
    styleUrls: ['src/reportviewer/reportviewer.component.css']
})
export class ReportViewerComponent {
    public serviceUrl: string;
    public reportPath: string;
    public serverUrl: string;
    constructor() {
        this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    }
}
```

```

this.serverUrl = 'http://<servername>/reportserver$instanceName';
this.reportPath = 'http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl';
}
}
`

```

In SharePoint integrated mode, the `reportServerUrl` will be same as your site URL. The `reportPath` is relative to the Report Server URL with the file extension.

Forms credential for SharePoint server

The Forms credentials are required to connect with the specified SharePoint integrated SSRS Report Server using the Report Viewer. Specify the `ReportServerFormsCredential` property in the Web API Controller `OnInitReportOptions` method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add ReportServerFormsCredential for server
    reportOption.ReportModel.ReportServerFormsCredential = new
    BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");
}
`

```

Set data source credential for shared data sources

The shared data source credentials can be added to the `DataSourceCredentials` property to connect with the database.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add ReportServerFormsCredential and data source credentials
    reportOption.ReportModel.ReportServerFormsCredential = new
    BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
}
`

```

Data source credentials should be added to shared data sources that do not have credentials in the connection strings.

Build and run the application.

Render RDLC report

The data binding support allows you view the RDLC reports that exist on the local file system with JSON array and custom business object data collection. The following steps demonstrates how to render an RDLC report with JSON array and custom business object data collection.

Add the RDLC report `AreaCharts.rdlc` from the Bold Reports installation location to your application `App_Data` folder. For more information, refer to [Samples and demos](#).

Bind data source at client side

To bind the data source at client side, follow these steps;

- Set the RDLC report path to the `reportPath` property.
- Assign the `processingMode` property to `ProcessingMode.Local`.
- Bind the JSON array collection to the `dataSources` property as shown in following code.

```
`html
<bold-reportviewer id = "reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[dataSources] = "reportData"
[reportPath] = "reportPath"
[processingMode] = "processingMode"
style = "width: 100%;height: 950px">
</bold-reportviewer>
`

`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: 'src/reportviewer/reportviewer.component.html',
  styleUrls: ['src/reportviewer/reportviewer.component.css']
})
export class ReportViewerComponent {
  public serviceUrl: string;
  public reportPath: string;
  public reportData: any;
  public processingMode: string;
```

```

constructor() {
this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
this.reportPath = 'AreaCharts.rdlc';
this.processingMode = "Local";
this.reportData = [{
value: [
{ SalesPersonID: 281, FullName: 'Ito', Title: 'Sales Representative', SalesTerritory: 'South West', Y2002: 0,
Y2003: 28000, Y2004: 3018725 },
{ SalesPersonID: 282, FullName: 'Saraiva', Title: 'Sales Representative', SalesTerritory: 'Canada', Y2002:
25000, Y2003: 14000, Y2004: 3189356 },
{ SalesPersonID: 283, FullName: 'Cambell', Title: 'Sales Representative', SalesTerritory: 'North West',
Y2002: 12000, Y2003: 13000, Y2004: 1930885 }
],
name: 'AdventureWorksXMLDataSet'
}];
}
}
`

```

- Build and run the application.

Bind data source in Web API controller

The following steps help you configure the Web API to render the RDLC report with business object data collection.

1. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```

`csharp
public class ProductList
{
public string ProductName { get; set; }
public string OrderId { get; set; }
public double Price { get; set; }
public string Category { get; set; }
public string Ingredients { get; set; }
public string ProductImage { get; set; }
}

```

```
public static IList GetData()
{
    List<ProductList> datas = new List<ProductList>();
    ProductList data = null;
    data = new ProductList()
    {
        ProductName = "Baked Chicken and Cheese",
        OrderId = "323B60",
        Price = 55,
        Category = "Non-Veg",
        Ingredients = "grilled chicken, corn and olives.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Delite",
        OrderId = "323B61",
        Price = 100,
        Category = "Non-Veg",
        Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Tikka",
        OrderId = "323B62",
        Price = 64,
        Category = "Non-Veg",
        Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
        ProductImage = ""
    };
};
```

```

datas.Add(data);
return datas;
}
}
`

```

2. Set the value of the `ProcessingMode` property to `ProcessingMode.Local` and `ReportPath` in the RDLC report location.
3. Bind the business object data values collection by adding a new item to the `DataSources` as shown in the following code snippet.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ReportPath =
    System.Web.Hosting.HostingEnvironment.MapPath(@"~/App_Data/Product List.rdlc");
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
    Value = ProductList.GetData() });
}
`

```

Here the `Name` is case sensitive and it should be same as in the data source name in the report definition.

The `Value` accepts `ICollection`, `DataSet`, and `DataTable` inputs.

Load report as stream

To load report as a stream, create a report stream using the `FileStream` class and assign the report stream to the `Stream` property.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string filePath = System.Web.Hosting.HostingEnvironment.MapPath(@"~/App_Data/Product List.rdlc");
    ;
    // Opens the report from application App_Data folder using FileStream
    FileStream reportStream = new FileStream(filePath, FileMode.Open, FileAccess.Read);
    reportOption.ReportModel.Stream = reportStream;
}
`

```



```
reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
Value = ProductList.GetData() });
}
,
```

In the previous code, the `Product List.rdlc` report is loaded from the `App_Data` folder location.

View report click

You can get the user selected parameter details when users clicks the `ViewReport` button in the parameter block. The `viewReportClick` event allows you handle the `ViewReport` button click at client side as shown in the following code.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[locale]= Remote
(viewReportClick) = "viewReportClick($event)">
</bold-reportviewer>
,

`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
```

```

}
viewReportClick(event) {
var reportParams = [];
reportParams.push({ name: 'ReportParameter1', labels: ['SO50756'], values: ['SO50756'] });
event.model.parameters = reportParams;
}
}
,

```

The model property in the event argument has the details of current processing report model.

Render subreport

You can display another report inside the body of a main report using the Report Viewer. The following steps helps you to customize the subreport properties such as data source, report path, and parameters.

- Add the subreport and main reports to your application `App_Data` folder. In this tutorial, the already created reports are used. Refer to the [Create RDL Report section](#) or [Create RDLC Report section](#) section.

Download the `SideBySideMainReport.rdl` and `SideBySideSubReport.rdl` reports from [here](#). You can add the report from the Bold Reports installation location. For more information, refer to [Samples and demos](#). The reports used from the installed location requires the `NorthwindIO_Reports.sdf` database to run, so add it to your application.

- Set the `reportPath` and `reportServiceUrl` properties of the Report Viewer as shown in following code snippet.

```

`html
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [processingMode] =
"Remote" [reportServerUrl] = "serverUrl" [reportPath] = "reportPath">
</bold-reportviewer>
,
`ts
import { Component } from '@angular/core';
@Component({
selector: 'ej-app',
templateUrl: 'src/reportviewer/reportviewer.component.html',
styleUrls: ['src/reportviewer/reportviewer.component.css']
})

```

```

export class ReportViewerComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'SideBySideMainReport.rdl';
  }
}

```

- Build and run the application.

Change subreport path

To change the subreport file path, set the **ReportPath** property of **SubReportModel** in the **OnInitReportOptions** method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
  if (reportOption.SubReportModel != null)
  {
    reportOption.SubReportModel.ReportPath =
      System.Web.Hosting.HostingEnvironment.MapPath(@"~/App_Data/" +
        reportOption.SubReportModel.ReportPath);
  }
}

```

Set subreport parameter

You can change the parameter default values of a subreport in the **OnReportLoaded** method of the Web API Controller as given in the following code snippet.

```

`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
  if (reportOption.SubReportModel != null)

```

```

{
reportOption.SubReportModel.Parameters = new BoldReports.Web.ReportParameterInfoCollection();
reportOption.SubReportModel.Parameters.Add(new BoldReports.Web.ReportParameterInfo()
{
Name = "SalesPersonID",
Values = new List<string>() { "2" }
});
}
}
`

```

Modify subreport data source connection string

You can change the credential and connection information of the data sources used in the subreport using the SubReportModel in the `OnInitReportOptions` method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
if (reportOption.SubReportModel != null)
{
reportOption.SubReportModel.DataSourceCredentials = new
List<BoldReports.Web.DataSourceCredentials>();
reportOption.SubReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>;user id=<username>;password=<password>"));
}
}
`

```

Set subreport data source

You can bind local business object data source collection only for RDLC reports. To specify data source of a RDLC subreport, set the `ReportDataSource` property in the `OnReportLoaded` method.

The RDL report has the connection information in report definition itself, so no need to bind the data source.

1. Add the RDLC subreport and main reports to your application `App_Data` folder. You can downloaded it from [here](#).
2. Set the `reportPath` and `reportServiceUrl` properties of the Report Viewer as shown in following code snippet.

```
`html
```

```
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [processingMode] =
"Local" [reportServerUrl] = "serverUrl" [reportPath] = "reportPath">
</bold-reportviewer>
```

```
,
```

```
`ts
```

```
import { Component } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: 'src/reportviewer/reportviewer.component.html',
  styleUrls: ['src/reportviewer/reportviewer.component.css']
})
export class ReportViewerComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'Product List Main.rdlc';
  }
}
,
```

- Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
```

```
public class ProductList
{
  public string ProductName { get; set; }
  public string OrderId { get; set; }
  public double Price { get; set; }
  public string Category { get; set; }
  public string Ingredients { get; set; }
  public string ProductImage { get; set; }
```

```
public static IList GetData()
{
    List<ProductList> datas = new List<ProductList>();
    ProductList data = null;
    data = new ProductList()
    {
        ProductName = "Baked Chicken and Cheese",
        OrderId = "323B60",
        Price = 55,
        Category = "Non-Veg",
        Ingredients = "grilled chicken, corn and olives.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Delite",
        OrderId = "323B61",
        Price = 100,
        Category = "Non-Veg",
        Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Tikka",
        OrderId = "323B62",
        Price = 64,
        Category = "Non-Veg",
        Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
        ProductImage = ""
    };
};
```

```

datas.Add(data);
return datas;
}
}
`

```

- Bind the business object data values collection to subreport by adding a new item to the **DataSources** as shown in the following code snippet.

```

`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'Product List.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
"list", Value = ProductList.GetData() });
    }
}
`

```

The data source name is case sensitive, and it should be same as in the report definition.

Load subreport stream

To load subreport as stream, set the **Stream** property in the **OnInitReportOptions** method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        // Opens the report from application App_Data folder using FileStream and loads the sub report stream.
        FileStream reportStream = new
        FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"~/App_Data/" +
reportOption.SubReportModel.ReportPath), FileMode.Open, FileAccess.Read);
        reportOption.SubReportModel.Stream = reportStream;
    }
}
`

```

```

}
}
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'Product List.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
        "list", Value = ProductList.GetData() });
    }
}

```

Report parameters

Provides property options to pass or set report parameters default values at run time using the `parameters` property. You can set the report parameters while creating the Report Viewer control in a script or in the Web API Controller.

In this tutorial, the `Sales Order Detail.rdl` report is used, and it can be downloaded from [here](#).

Set parameter at client side

The `parameters` property takes the JSON array value input with parameter details.

- Set the default value data to the `values` property and name of the report parameter to the `name` property.

The parameter name is case sensitive, and it should be same as available in the report definition.

The following code example illustrates how to set a report parameter in the script.

``html`

```

<bold-reportviewer id="reportViewer_Control" style="width: 100%;height: 950px" [reportServiceUrl] =
"serviceUrl" [processingMode] = "Remote" [reportPath] = "reportPath" [parameters] = "parameters">
</bold-reportviewer>

```

``typescript`

```

import { Component } from '@angular/core';
@Component({

```



```

selector: 'ej-app',
templateUrl: 'src/reportviewer/reportviewer.component.html',
styleUrls: ['src/reportviewer/reportviewer.component.css']
})
export class ReportViewerComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public parameters: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'Sales Order Detail.rdl';
    this.parameters = [{
      name: 'SalesOrderNumber',
      labels: ['SO50751'],
      values: [SO50751],
      nullable: false
    }];
  }
}

```

- Build and run the application.

Set parameters in Web API Controller

To set parameter default value in the Web API Controller, use the following code in the `OnReportLoaded` method.

```

`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
  List<BoldReports.Web.ReportParameter> userParameters = new
  List<BoldReports.Web.ReportParameter>();
  userParameters.Add(new BoldReports.Web.ReportParameter()
  {

```

```

Name = "SalesOrderNumber",
Values = new List<string>() { "SO50756" }
});
reportOption.ReportModel.Parameters = userParameters;
}
`

```

The Report Parameters name should be case sensitive

Get report parameter

The **ReportHelper** class provides methods that help you to get the report parameters used in the report. The following helper methods are used to get parameter with or without values.

Methods | Description

minDateTime | Specify minimum range value of a date parameter

maxDateTime | Specify maximum range value of a date parameter

`javascript

```

<bold-reportviewer id="reportViewer_Control" style="width: 100%;height: 950px" [reportServiceUrl] =
"serviceUrl"

```

```

[reportPath] = "reportPath" [parameterSettings] = "parameterSettings">

```

```

</bold-reportviewer>
`

```

`typescript

```

export class AppComponent {

```

```

  public serviceUrl: string;

```

```

  public reportPath: string;

```

```

  public parameterSettings: any;

```

```

  constructor() {

```

```

    this.serviceUrl = "https://demos.boldreports.com/services/api/ReportViewer";

```

```

    this.reportPath = "~/Resources/demos/Report/product-line-sales.rdl";

```

```

    this.parameterSettings = {

```

```

      minDateTime: new Date("4/5/2003"),

```

```

      maxDateTime: new Date("6/15/2003"),

```

```

    };

```

```

  }

```

```

}
`

```

The above code sets date range for all the date parameters used in the report.

To set different date range for each date parameter used in the report, register the event `beforeParameterAdd` and specify range based on parameter name as in below code sample.

```
`html
<bold-reportviewer id="reportViewer_Control" style="width: 100%;height: 950px"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
(beforeParameterAdd) = "onBeforeParameterAdd($event)">
</bold-reportviewer>
`

`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'product-line-sales.rdl';
  }
  onBeforeParameterAdd(event) {
    event.parameterSettings.dateTimePickerType = "DateTime";
    if (event.parameterModel.Name === "StartDate") {
      event.parameterSettings.minDateTime = new Date("4/5/2003 5:00:00 AM");
      event.parameterSettings.maxDateTime = new Date("4/15/2003 5:00:00 AM");
    }
  }
}
```

```

if (event.parameterModel.Name === "EndDate") {
    event.parameterSettings.minDateTime = new Date("5/10/2003 5:00:00 AM");
    event.parameterSettings.maxDateTime = new Date("5/20/2003 5:00:00 AM");
}
}
}
,

```

Set date time display format for date parameter

The properties `dateTimeFormat` and `timeDisplayFormat` in the `parameterSettings` are used to set date and time format to be displayed in the `DateTimePicker` control in a report.

Format | Display in DateTimePicker

Short Date and Time - d/M/yy h:mm tt | 9/12/2014 2:04 PM

Medium Date - d MMM yy h:mm tt | 12 Sep 14 2:04: PM

Full Date and short time - dddd, MMMM dd, yyyy HH:mm tt | Friday, September 12,2014 2:04 PM

Full Date and Long Time - dddd, MMMM dd, yyyy HH:mm:ss tt | Friday, September 12,2014 2:04:00 PM

UTC - yyyy-MM-dThh:mm:ssz | 2014-09-12T2:04:00+5

`javascript

```

<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl"
[reportPath] = "reportPath" [parameterSettings] = "parameterSettings">
</bold-reportviewer>
,

```

`typescript

```

export class AppComponent {
    public serviceUrl: string;
    public reportPath: string;
    public parameterSettings: any;
    constructor() {
        this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
        this.reportPath = 'product-line-sales.rdl';
        this.parameterSettings = {
            dateTimePickerType: "DateTime",
            dateTimeFormat: "MM/dd/yyyy h:mm tt",

```

```
timeDisplayFormat: "HH:mm",
timeInterval: 60,
};
}
}
,
```

The above code sets date and time value to be display for all the date parameters used in the report.

To set different date and time value to be display for each date parameter used in the report, register the event `beforeParameterAdd` and specify date and time value based on parameter name as in below code sample.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
(beforeParameterAdd) = "onBeforeParameterAdd($event)">
</bold-reportviewer>
,
```

```
`typescript
import { Component, ViewChild } from "@angular/core";
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = '~/Resources/demos/Report/product-line-sales.rdl';
  }
}
```

```

onBeforeParameterAdd(event) {
event.parameterSettings.dateTimePickerType = "DateTime";
if (event.parameterModel.Name === "StartDate") {
event.parameterSettings.dateTimeFormat = "MM/dd/yyyy h:mm tt";
event.parameterSettings.timeDisplayFormat = "HH:mm";
event.parameterSettings.timeInterval = 60;
}
if (event.parameterModel.Name === "EndDate") {
event.parameterSettings.dateTimeFormat = "MM/dd/yyyy h:mm tt";
event.parameterSettings.timeDisplayFormat = "HH:mm";
event.parameterSettings.timeInterval = 60;
}
}
}
,

```

Access the hidden or internal parameter information

The `accessInternalValue` property in the `parameterSettings` helps you to expose the `hidden` or `internal` report parameter information used in report to the user.

`javascript

```

<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl"
[reportPath] = "reportPath" [parameterSettings] = "parameterSettings">
</bold-reportviewer>
,

```

`typescript

```

export class AppComponent {
public parameterSettings: any;
constructor() {
this.parameterSettings= { accessInternalValue: true}
}
}
,

```

Set the report parameter visibility in Web API Controller

The `Hidden` property of `ReportParameter` allows you to show or hide the parameter at the top of the report viewer panel. The following code example shows hiding a report parameter in the Web API controller's `OnReportLoaded` method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    var reportParameters = ReportHelper.GetParameters(jsonArray, this);
    List<BoldReports.Web.ReportParameter> modifiedParameters = new
    List<BoldReports.Web.ReportParameter>();
    if (reportParameters != null)
    {
        foreach (var rptParameter in reportParameters)
        {
            modifiedParameters.Add(new BoldReports.Web.ReportParameter()
            {
                Name = rptParameter.Name,
                Hidden = true
            });
        }
        reportOption.ReportModel.Parameters = modifiedParameters;
    }
}
```

Report interaction events

You can handle the report interaction events with reports using the following events.

- `ReportLoaded`
- `ReportError`
- `ShowError`
- `Drill through`
- `Hyperlink`

Report loaded

The `reportLoaded` event occurs after the report is loaded and it is ready to start the processing. You can handle the event and specify the data source and parameters at client side. The following sample code loads a report by assigning the report data source input in the `reportLoaded` event.

In this tutorial, the `IndicatorReport.rdlc` report is used. You can add the report from the Bold Reports installation location. For more information, refer to [Samples and demos](#).

`html

```
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [processingMode] =
"Local" [reportPath] = "reportPath" (reportLoaded) = "reportLoaded($event)">
```

```
</bold-reportviewer>
```

,

`typescript

```
import { Component } from '@angular/core';
```

```
@Component({
```

```
  selector: 'ej-app',
```

```
  templateUrl: 'src/reportviewer/reportviewer.component.html',
```

```
  styleUrls: ['src/reportviewer/reportviewer.component.css']
```

```
})
```

```
export class ReportViewerComponent {
```

```
  public serviceUrl: string;
```

```
  public reportPath: string;
```

```
  public serverUrl: string;
```

```
  constructor() {
```

```
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
```

```
    this.reportPath = 'IndicatorReport.rdlc';
```

```
  }
```

```
  reportLoaded(event) {
```

```
    let desc2013 = [
```

```
    {
```

```
      No: 1, Name: "Carlos Slim", NetWorth: 73.0, Age: 73, CitizenShip: "Mexico", Source: "Telmex,America
Movil, Grupo Carso", RankingStatus: 50, ProfitStatus: 75
```

```
    },
```

```
    {
```

```
      No: 2, Name: "Bill Gates", NetWorth: 67.0, Age: 57, CitizenShip: "United States", Source: "Microsoft",
RankingStatus: 50, ProfitStatus: 75
```



```

},
{
No: 3, Name: "Amancio Ortega", NetWorth: 57.0, Age: 57, CitizenShip: "Spain", Source: "Inditex Group",
RankingStatus: 75, ProfitStatus: 75
},
{
No: 4, Name: "Warren Buffett", NetWorth: 53.0, Age: 82, CitizenShip: "United States", Source:
"Berkshire Hathaway", RankingStatus: 25, ProfitStatus: 75
},
{
No: 5, Name: "Larry Ellison", NetWorth: 43.0, Age: 68, CitizenShip: "United States", Source: "Oracle
Corporation", RankingStatus: 75, ProfitStatus: 75
},
{
No: 6, Name: "Charles Koch", NetWorth: 34.0, Age: 77, CitizenShip: "United States", Source: "Koch
Industries", RankingStatus: 75, ProfitStatus: 75
},
{
No: 7, Name: "David Koch", NetWorth: 34.0, Age: 72, CitizenShip: "United States", Source: "Koch
Industries", RankingStatus: 75, ProfitStatus: 75
},
{
No: 8, Name: "Li Ka-shing", NetWorth: 32.0, Age: 84, CitizenShip: "Hong Kong/ Canada", Source:
"Cheung Kong Holdings", RankingStatus: 75, ProfitStatus: 75
},
{
No: 9, Name: "Liliane Bettencourt", NetWorth: 30.0, Age: 90, CitizenShip: "France", Source: "L'Oreal",
RankingStatus: 75, ProfitStatus: 75
},
{
No: 10, Name: "Bernard Arnault", NetWorth: 29.0, Age: 63, CitizenShip: "France", Source: "LVMH Moet
Hennessy Louis Vuitton", RankingStatus: 25, ProfitStatus: 25
}};
let desc2012 = [
{

```

No: 1, Name: "Carlos Slim", NetWorth: 69.0, Age: 72, CitizenShip: "Mexico", Source: "Telmex,America Movil, Grupo Carso", RankingStatus: 50, ProfitStatus: 25

},

{

No: 2, Name: "Bill Gates", NetWorth: 61.0, Age: 56, CitizenShip: "United States", Source: "Microsoft", RankingStatus: 50, ProfitStatus: 75

},

{

No: 3, Name: "Warren Buffett", NetWorth: 44.0, Age: 81, CitizenShip: "United States", Source: "Berkshire Hathaway", RankingStatus: 50, ProfitStatus: 25

},

{

No: 4, Name: "Bernard Arnault", NetWorth: 41.0, Age: 63, CitizenShip: "France", Source: "LVMH Moet Hennessy Louis Vuitton", RankingStatus: 50, ProfitStatus: 75

},

{

No: 5, Name: "Amancio Ortega", NetWorth: 37.5, Age: 75, CitizenShip: "Spain", Source: "Inditex Group", RankingStatus: 75, ProfitStatus: 75

},

{

No: 6, Name: "Larry Ellison", NetWorth: 36.0, Age: 67, CitizenShip: "United States", Source: "Oracle Corporation", RankingStatus: 25, ProfitStatus: 75

},

{

No: 7, Name: "Eike Batista", NetWorth: 30.0, Age: 55, CitizenShip: "Brazil", Source: "EBX Group", RankingStatus: 75, ProfitStatus: 75

},

{

No: 8, Name: "Stefan Persson", NetWorth: 26.5, Age: 64, CitizenShip: "Sweden", Source: "H&M", RankingStatus: 75, ProfitStatus: 75

},

{

No: 9, Name: "Li Ka-shing", NetWorth: 25.0, Age: 83, CitizenShip: "Hong Kong/ Canada", Source: "Cheung Kong Holdings", RankingStatus: 75, ProfitStatus: 75

},

{

No: 10, Name: "Karl Albrecht", NetWorth: 25.4, Age: 92, CitizenShip: "Germany", Source: "Aldi",
RankingStatus: 75, ProfitStatus: 25

```

});
let description = [
{
Status: 25, Description: "Has not changed from the previous ranking."
},
{
Status: 50, Description: "Has increased from the previous ranking."
},
{
Status: 75, Description: "Has decreased from the previous ranking."
}
];
event.model.dataSources = [{
value: ej.DataManager(desc2013).executeLocal(ej.Query()),
name: "DataSet1"
}, {
value: ej.DataManager(desc2012).executeLocal(ej.Query()),
name: "DataSet2"
}, {
value: ej.DataManager(description).executeLocal(ej.Query()),
name: "DataSet3"
}
];
}
}
`

```

Report error

When an error occurs in the report processing, it raises the `reportError` event. You can handle the event and show the details in your custom dialog instead of component default error detail interface.

```
`html
```

```

<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"

```

```

[reportPath] = "reportPath"
(reportError) = "onReportError($event)">
</bold-reportviewer>
`
`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'Sales Order Detail.rdl';
  }
  onReportError(event) {
    alert(event.errmsg);
    event.cancel = true;
  }
}
`

```

To suppress the default error dialog, set the cancel argument to true.

Show error

The **showError** event is invoked whenever users click a report item that contains an error in processing. It provides detailed information about the cause of the error. You can hide the default dialog and show your customized dialog.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"

```

```

[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
(showError) = "onShowError($event)">
</bold-reportviewer>
`
`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'Sales Order Detail.rdl';
  }
  onShowError(event) {
    alert("Error code : " + event.errorCode + "\n" +
      "Error Detail : " + event.errorDetail + "\n" +
      "Error Message : " + event.errorMessage);
    event.cancel = true;
  }
}
`
`

```

Drill through

When a drill through item is selected in a report, it invokes the **drillThrough** event. You can change the drill through arguments such as report parameter and data sources. The following sample code can be

used to change the drill through report name and set the parameter value before the drill through report is rendered.

```
`html
```

```
<bold-reportviewer id="reportViewer_Control"
```

```
[reportServiceUrl] = "serviceUrl"
```

```
[processingMode] = "Remote"
```

```
[reportServerUrl] = "serverUrl"
```

```
[reportPath] = "reportPath"
```

```
(drillThrough) = "onDrillThrough($event)">
```

```
</bold-reportviewer>
```

```
,
```

```
`typescript
```

```
import { Component, ViewChild } from '@angular/core';
```

```
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
```

```
@Component({
```

```
  selector: 'ej-app',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  public serviceUrl: string;
```

```
  public reportPath: string;
```

```
  public serverUrl: string;
```

```
  constructor() {
```

```
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
```

```
    this.reportPath = 'SalesPersonDetails.rdl';
```

```
  }
```

```
  onDrillThrough(event) {
```

```
    event.actionInfo.ReportName = "PersonalDetails";
```

```
    event.actionInfo.Parameters = [{
```

```
      name: 'SalesOrderNumber',
```

```
      labels: ['SO50751'],
```

```
      values: [SO50751],
```

```
nullable: false
```

```
}};
```

```
}
```

```
}
```

```
,
```

Hyperlink

The **hyperlink** event occurs when users click a hyperlink in a report, before the hyperlink is followed. The following sample code redirects to a new custom URL and cancels the component default action.

```
`cshhtml
```

```
<bold-reportviewer id="reportViewer_Control"
```

```
[reportServiceUrl] = "serviceUrl"
```

```
[processingMode] = "Remote"
```

```
[reportServerUrl] = "serverUrl"
```

```
[reportPath] = "reportPath"
```

```
(hyperlink) = "onHyperLink($event)">
```

```
</bold-reportviewer>
```

```
,
```

```
`typescript
```

```
import { Component, ViewChild } from '@angular/core';
```

```
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
```

```
@Component({
```

```
  selector: 'ej-app',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  public serviceUrl: string;
```

```
  public reportPath: string;
```

```
  public serverUrl: string;
```

```
  constructor() {
```

```
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
```

```
    this.reportPath = 'Customer Support Analysis (Random data).rdl';
```

```
  }
```

```
  onHyperLink(event) {
```

```

event.cancel = true;
//You can modify the URL here
window.open(event.actionInfo.Hyperlink);
}
}
`

```

Handle post actions

Report processing actions are sent in an Ajax request to exchange data with the Web API service. You can handle post actions event to customize the Ajax requests.

- AjaxBeforeLoad
- AjaxSuccess
- AjaxError

AjaxBeforeLoad

This event can be triggered before an Ajax request is sent to the Report Viewer Web API service. It allows you to set additional headers and custom data in the Ajax request. The following code sample demonstrates adding custom authorization header and passing default parameter values to service.

Add custom header in Ajax request

Initialize the `ajaxBeforeLoad` event in the script and add the authorization token to the `headers` property.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
(ajaxBeforeLoad) = "onAjaxRequest($event)">
</bold-reportviewer>
`

`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',

```



```

styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'Sales Order Detail.rdl';
  }
  onAjaxRequest(event) {
    event.headers.push({ Key: 'Authorization', Value: 'demo@123' });
  }
}

```

In this tutorial, the **Sales Order Detail.rdl** report is used, and it can be downloaded from [here](#).

Get the custom header value from the **HttpContext** header collection using the key name specified at client side.

```

`csharp
string authenticationHeader;
public object PostReportAction(Dictionary<string, object> jsonResult)
{
  if (jsonResult != null)
  {
    //Get client side custom ajax header and store in local variable
    authenticationHeader = HttpContext.Current.Request.Headers["Authorization"];
    //Perform your custom validation here
    if (authenticationHeader == "")
    {
      return new Exception("Authentication failed!!!");
    }
    else
    {

```

```

return ReportHelper.ProcessReport(jsonResult, this);
}
}
return null;
}
,

```

Perform your own action to validate the header values.

Pass custom data in Ajax request

Use the `data` property to set custom data to the server in the Ajax request. In the following code sample, parameter values are passed to the server side.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
(ajaxBeforeLoad) = "onAjaxRequest($event)">
</bold-reportviewer>
,

`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
selector: 'ej-app',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
public serviceUrl: string;
public reportPath: string;
public serverUrl: string;
constructor() {
this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
this.reportPath = 'Sales Order Detail.rdl';

```

```

}
onAjaxRequest(event) {
//Passing custom data to server
var customerID = "CI0021";
event.data = customerID;
}
}
`

```

The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates to change the datasource connection strings based on Customer ID in the `OnInitReportOptions` method.

```

`csharp
string customerID = null;
public object PostReportAction(Dictionary<string, object> jsonResult)
{
if (jsonResult != null)
{
if (jsonResult.ContainsKey("customData"))
{
//Get client side custom data and store in local variable.
customerID = jsonResult["customData"].ToString();
}
}
return ReportHelper.ProcessReport(jsonResult, this);
}
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
if (customerID != null)
{
if(customerID.Contains("CI0021"))
{
//If you are changing the connection string based on customer id then could you please change the
connection string as below.

```

```
//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=NorthwindSdf;"));

}

else if(customerID.Contains("CI0022"))
{
//If you are changing the connection string based on customer id then could you please change the
connection string as below.

//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=NorthwindSdf;"));

}
}
}
`
```

AjaxSuccess

To perform custom action or show user defined message, use the `ajaxSuccess` event on the successful Ajax request.

```
`html

<bold-reportviewer id="reportViewer_Control"

[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
(ajaxSuccess) = "onAjaxSuccess($event)">
</bold-reportviewer>
`
```

```
`typescript

import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
selector: 'ej-app',
```

```

templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'Sales Order Detail.rdl';
  }
  onAjaxSuccess(event) {
    //Perform your custom success message here
    alert("Ajax request success!!!");
  }
}

```

AjaxError

The `ajaxError` event is called, if an error occurred with the Ajax request. You can display the customized error details in the event method.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
(ajaxError) = "onAjaxFailure($event)">
</bold-reportviewer>

```

```

`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',

```

```

templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'Sales Order Detail.rdl';
  }
  onAjaxFailure(event) {
    alert("Status: " + event.status + "\n" +
      "Error: " + event.responseText);
  }
}
`

```

You can never have both an error and a success callback with a request.

Print report

The Report Viewer provides print report option in the toolbar to print a copy of the report. The Page Setup dialog allows you to set the paper size or other page setup properties. To see print margins, click **Print Layout** on the toolbar.

You can set values in the Page Setup dialog box for current session only. When you close the report and reopen it, it will have the default values again. The default values for the Page Setup dialog is based on the report properties set in the design view.

View report in print mode

Print margins are displayed in the Print Layout only. To view the report in print mode by default, set the `printMode` property to true.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"=
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"

```

```
[printMode] = "isPrintMode">
</bold-reportviewer>
`
`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public isPrintMode: boolean;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.isPrintMode = true;
  }
}
`
```

By default, the Report Viewer renders the report in normal layout in which the print margins are not displayed.

[Print in new page](#)

To open the print in a new tab of the current browser, set the `printOption` property to `NewTab`. By default, it shows the print dialog in the same page.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
```

```
[printOption] = "printOption">
</bold-reportviewer>
`
`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public printOption: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.printOption = ej.ReportViewer.PrintOptions.NewTab;
  }
}
`
```

The pop-up blocker should be enabled for the page to open the print view in a new tab.

Set page orientation and paper size

You can specify the print page paper size and orientation at client-side to change the page setup properties by setting the `pageSettings` property.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[printMode] = "isPrintMode"
```



```
[pageSettings] = "pageSettings">
```

```
</bold-reportviewer>
```

```
,
```

```
`typescript
```

```
import { Component, ViewChild } from '@angular/core';
```

```
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
```

```
@Component({
```

```
  selector: 'ej-app',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  public serviceUrl: string;
```

```
  public reportPath: string;
```

```
  public serverUrl: string;
```

```
  public pageSettings: any;
```

```
  public isPrintMode: boolean;
```

```
  constructor() {
```

```
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
```

```
    this.reportPath = 'GroupingAgg.rdl';
```

```
    this.isPrintMode = true;
```

```
    this.pageSettings = {
```

```
      orientation: ej.ReportViewer.Orientation.Portrait,
```

```
      paperSize: ej.ReportViewer.PaperSize.A3,
```

```
    };
```

```
  }
```

```
}
```

```
,
```

Set report margin

To set margin values to the report page setup, use the **margins** property and specify the value to top, right, bottom, and left.

```
`html
```

```
<bold-reportviewer id="reportViewer_Control"
```

```
  [reportServiceUrl] = "serviceUrl"
```

```
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[printMode] = "isPrintMode"
[pageSettings] = "pageSettings">
</bold-reportviewer>
`
```

```
`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public pageSettings: any;
  public isPrintMode: boolean;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'GroupingAgg.rdl';
    this.isPrintMode = true;
    this.pageSettings = {
      margins: {
        top: 0.5,
        right: 0.25,
        bottom: 0.25,
        left: 0.25
      }
    };
  }
};
```

```

}
}
`

```

The values set in the margin property is considered as inches input.

Set page height and width

To set the height and width values to the report page setup, use the `height` and `width` properties.

```
`html
```

```

<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[printMode] = "isPrintMode"
[pageSettings] = "pageSettings">
</bold-reportviewer>
`

```

```
`typescript
```

```

import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public pageSettings: any;
  public isPrintMode: boolean;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'GroupingAgg.rdl';
  }
}

```

```

this.isPrintMode = true;
this.pageSettings = {
  height: 2.69,
  width: 28.27
};
}
}
`

```

The values set in the height and width properties is considered as inches input.

Print report with images

When the report has more images, the browser will send the report stream to the print dialog before the images are completely loaded. To load the print report stream with complete images, set the `EmbedImageData` property to true in `OnInitReportOptions` as shown in the following code.

```

`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
  reportOption.ReportModel.EmbedImageData = true;
}
`

```

Replace the following code sample in client-side HTML file.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath">
</bold-reportviewer>
`

`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',

```

```

styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'GroupingAgg.rdl';
  }
}

```

In this tutorial, the **Product Details.rdl** report is used, and it can be downloaded from [here](#).

External styles in report printing

While printing a report, the external styles used in the application overrides the printable page style and prints output with incorrect alignments. To avoid the external script overriding, set the **isStyleLoad** property to false, which will print the page using only the Report Viewer styles.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[locale]= "Remote"
(reportPrint) = "onReportPrint($event)">
</bold-reportviewer>

```

```

`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']

```

```

})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'Product Details.rdl';
  }
  onReportPrint(event) {
    event.isStyleLoad = false;
  }
}

```

Show print progress

The Report Viewer provides events that help you show the progress information, when the printing process takes a long time to complete.

To show print progress, follow these steps:

1. Set the `printProgressChanged` in Report Viewer initialization.
2. Implement the function and add code samples to show a custom message based on the print progress status as shown in the following code snippet.

```

`html
<ej-ReportViewer id="ReportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[locale]= "Remote"
(printProgressChanged) = "onPrintProgressChanged($event)">
</ej-ReportViewer>
`
`typescript
import { Component, ViewChild } from '@angular/core';

```

```
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = 'Product Details.rdl';
  }
  onPrintProgressChanged(event) {
    if (event.stage === "beginPrint") {
      console.log(event.stage);
      //$('#viewer').ejWaitingPopup({ showOnInit: true, cssClass: "customStyle", text: "Preparing print data.. Please wait..." });
    }
    if (event.stage === "printStarted") {
      console.log(event.stage);
      //var popupObj1 = $('#viewer').data('ejWaitingPopup');
      //popupObj1.hide();
    }
    else if (event.stage === "preparation") {
      console.log(event.stage);
      if (event.preparationStage === "dataPreparation") {
        console.log(event.preparationStage);
        console.log(event.totalPages);
        console.log(event.currentPage);
        //if (event.totalPages > 1 && event.currentPage > 1) {
        //  var progressPercentage = Math.floor((event.currentPage / event.totalPages) * 100);
```

```
// if (progressPercentage > 0) {
//     var popupObj2 = $('#viewer').data('ejWaitingPopup');
//     popupObj2.setModel({ text: "Preparing print data.." + progressPercentage + " % completed..
// Please wait..." });
// }
//}
}
}
event.handled = true;
}
}
`
```

Remove empty spaces in printing

The extra blank page is created when the body of your report is too wide for your page. To make the report appear on a single page, all the content within the report body must fit on the physical page, and the body width should be as the following formula:

Body Width <= Page Width - (Left Margin + Right Margin)

For more details to remove the empty pages in the report while designing, refer to the knowledge base article of [report page sizing](#).

Export report

The Report Viewer provides events and properties to control and customize the report exporting functionality.

Export event handling

You can show the progress information, when the exporting process takes long time to complete using the `exportProgressChanged` event.

1. Set the `exportProgressChanged` event in Report Viewer initialization.
2. Implement the function and replace the following code samples to get the log message based on the progress stage.

`html

```
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
(exportProgressChanged) = "onExportProgressChanged($event)">
```



```
</bold-reportviewer>
```

```
,
```

```
`typescript
```

```
import { Component, ViewChild } from '@angular/core';
```

```
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
```

```
@Component({
```

```
  selector: 'ej-app',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  public serviceUrl: string;
```

```
  public reportPath: string;
```

```
  public serverUrl: string;
```

```
  constructor() {
```

```
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
```

```
    this.reportPath = 'GroupingAgg.rdl';
```

```
  }
```

```
  onExportProgressChanged(event) {
```

```
    if (event.stage === "beginExport") {
```

```
      console.log(event.stage);
```

```
    }
```

```
    else if (event.stage === "exportStarted") {
```

```
      console.log(event.stage);
```

```
    }
```

```
    else if (event.stage === "preparation") {
```

```
      console.log(event.stage);
```

```
      console.log(event.format);
```

```
      console.log(event.preparationStage);
```

```
    }
```

```
    event.handled = true;
```

```
  }
```

```
}
```

Change Excel and Word export format

You can change the default file format to any other file format using the `excelFormat` and `wordFormat` properties. The following code sample changes the default versions.

`html

```
<bold-reportviewer id="reportViewer_Control"
```

```
[reportServiceUrl] = "serviceUrl"
```

```
[processingMode] = "Remote"
```

```
[reportServerUrl] = "serverUrl"
```

```
[reportPath] = "reportPath"
```

```
[exportSettings] = "exportSettings">
```

```
</bold-reportviewer>
```

`

`typescript

```
import { Component, ViewChild } from '@angular/core';
```

```
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
```

```
@Component({
```

```
  selector: 'ej-app',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  public serviceUrl: string;
```

```
  public reportPath: string;
```

```
  public serverUrl: string;
```

```
  public exportSettings: any;
```

```
  constructor() {
```

```
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
```

```
    this.reportPath = 'GroupingAgg.rdl';
```

```
    this.exportSettings = {
```

```
      excelFormat: ej.ReportViewer.ExcelFormats.Excel2013,
```

```
      wordFormat: ej.ReportViewer.WordFormats.Word2013
```

```
    }
```

```
  }
```

```

}
`

```

Hide specific export type for report

Show or hide the default export types available in the component using the `exportOptions` property. The following code hides the HTML export type from the default export options.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[exportSettings] = "exportSettings">
</bold-reportviewer>
`

`typescript
import { Component, ViewChild } from '@angular/core';
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public exportSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.exportSettings = {
      exportOptions: ej.ReportViewer.ExportOptions.All & ~ej.ReportViewer.ExportOptions.HTML &
      ~ej.ReportViewer.ExportOptions.Word
    }
  }
}

```

```
}  
}  
,
```

PDF export options

The `PDFOptions` provides properties to manage PDF export behaviors. You should set the properties in the `OnInitReportOptions` method of the Web API service.

Export with complex scripts

To export reports with the complex scripts, set the `ComplexScript` property of `PDFOptions` instance to true.

```
`csharp  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()  
{  
    EnableComplexScript = true  
};  
}
```

PDF conformance

You can export the report as a PDF/A-1b document by specifying the `PdfConformanceLevel.Pdf_A1B` conformance level in the `PdfConformanceLevel` property.

```
`csharp  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()  
{  
    PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B  
};  
}
```

Add custom PDF fonts

You can add custom fonts to the PDF exported document by adding the font streams to `Fonts` collection in `PDFOptions` instance.

To add custom fonts to the PDF exported document, follow these steps:

1. Add the font .ttf files into your application App_Data folder.
2. In the Solution Explorer, open the properties of the font file and set the property Copy to Output Directory as Copy always.
3. Initialize the Font collection and add the font stream to it.

The key value provided in the font collection should be same as in the report item font property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
    {
        //Load Missing font stream
        Fonts = new Dictionary<string, System.IO.Stream>
        {
            { "Segoe UI",
              System.IO.File.OpenRead(System.Web.Hosting.HostingEnvironment.MapPath(@"~/AppData/fontsymbols.ttf")) }
        }
    };
}
```

If any fonts used in the report definition is not installed or available in the local system, then you should load the font stream. In the above code, font_symbols font stream is loaded to export the Sales Order Detail.rdl report as symbols.

Word export options

The WordOptions provides properties to manage Word document export behaviors.

Word document type

You can save the report to the required document version by setting the FormatType property.

```
`csharp
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
    FormatType = BoldReports.Writer.WordFormatType.Docx
};
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
    LayoutOption = BoldReports.Writer.WordLayoutOptions.TopLevel,
    ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
    {
        Bottom = 0.5f,
        Top = 0.5f
    }
};
`
```

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior.

The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, added an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without password. The following are the types of protection:

- `AllowOnlyComments`: Adds or modifies only the comments in the Word document.
- `AllowOnlyFormFields`: Modifies the form field values in the Word document.
- `AllowOnlyRevisions`: Accepts or rejects the revisions in the Word document.
- `AllowOnlyReading`: Allows you to view the content only in the Word document.
- `NoProtection`: Accesses or edits the Word document contents as normally.

```
`csharp
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
    ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading
};
`
```

Excel export options

The `ExcelOptions` provides properties to manage Excel document export behaviors.

Excel document type

You can save the report to the required excel version by setting the `ExcelSaveType` property.

```
`csharp
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013
};
`
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` to `IgnoreCellMerge`.

```
`csharp
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
};
`
```

Protecting Excel document from editing

You can restrict the Excel document from editing either by providing the `ExcelSheetProtection` or enabling the `ReadOnlyRecommended` properties.

```
`csharp
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    ReadOnlyRecommended = true,
    ExcelSheetProtection = ExcelSheetProtection.DeletingColumns
};
`
```

PowerPoint export options

You can save the report to the required PowerPoint version by setting the `FormatType` property.

```
`csharp
reportOption.ReportModel.PPTOptions = new BoldReports.Writer.PPTOptions()
{
    FormatType = BoldReports.Writer.PPTSaveType.PowerPoint2013
};
`
```

`

CSV export options

The `CsvOptions` allows you change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document.

```
`csharp
reportOption.ReportModel.CsvOptions = new BoldReports.Writer.CsvOptions()
{
    Encoding = System.Text.Encoding.Default,
    FieldDelimiter = ",",
    UseFormattedValues = false,
    Qualifier = "#",
    RecordDelimiter = "@",
    SuppressLineBreaks = true,
    FileExtension = ".txt"
};
`
```

HTML export options

You can hide the separator added at the end of each page by setting the `HidePageSeparator` property to true.

```
`csharp
reportOption.ReportModel.HTMLOptions = new BoldReports.Writer.HTMLOptions()
{
    HidePageSeparator = true
};
`
```

Password protect exported document

Allows you protect the exported document such as PDF, Microsoft Word, Microsoft Excel, and PowerPoint from unauthorized users by encrypting the document using encryption password. The following code snippet demonstrates how to encrypt the exported document with user defined password.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //PDF encryption
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions();
}
```



```

reportOption.ReportModel.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity()
{
    UserPassword = "password"
};
//Word encryption
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
    EncryptionPassword = "password"
};
//Excel encryption
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    PasswordToModify = "password",
    PasswordToOpen = "password"
};
//PPT encryption
reportOption.ReportModel.PPTOptions = new BoldReports.Writer.PPTOptions()
{
    EncryptionPassword = "password"
};
}
`

```

Password protection is not supported for HTML export format.

Toolbar customization

You can hide the component toolbar to show customized user interface or to customize the toolbar icons and element's appearances using the templates and Report Viewer toolbar customization properties.

In this tutorial, the `Sales Order Detail.rdl` report is used, and it can be downloaded from [here](#). You can add the reports from the Bold Reports installation location. For more information, refer to [Samples and demos](#).

Hide toolbar items

To hide toolbar items, set the `toolbarSettings` property. The following code can be used to remove the parameter option from the toolbar and hide the parameter block.

Similarly, you can show or hide all other toolbar options with the help of [toolbarSettings.items](#) enum.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[toolbarSettings] = "toolbarSettings">
</bold-reportviewer>
`
`ts
import { Component, ViewChild } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public toolbarSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.toolbarSettings = {
      items: ~ej.ReportViewer.ToolbarItems.Parameters,
    }
  }
}
`

```

The following code sample hides the print options from the toolbar items.

```

`html
<bold-reportviewer id="reportViewer_Control"

```

```
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[toolbarSettings] = "toolbarSettings">
</bold-reportviewer>
`
`ts
import { Component, ViewChild } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public toolbarSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.toolbarSettings = {
      items: ~ej.ReportViewer.ToolbarItems.Print,
    }
  }
}
```

Enable stop option in toolbar

To enable stop option in toolbar, set the `toolbarSettings.items` property to `ej.ReportViewer.ToolbarItems.All`. The following code can be used to enable stop option in toolbar.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
```

```
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[toolbarSettings] = "toolbarSettings">
</bold-reportviewer>
`
`ts
import { Component, ViewChild } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public toolbarSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.toolbarSettings = {
      items: ej.ReportViewer.ToolbarItems.All,
    }
  }
}
```

Enable export setup option in toolbar

To enable export setup option in toolbar, set the `toolbarSettings.items` property to `ej.ReportViewer.ToolbarItems.All`. The following code can be used to enable export setup option in toolbar.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
```

```
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[toolbarSettings] = "toolbarSettings">
</bold-reportviewer>
`
`ts
import { Component, ViewChild } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public toolbarSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.toolbarSettings = {
      items: ej.ReportViewer.ToolbarItems.All
    }
  }
}
```

Enable search text option in toolbar

To enable search text option in toolbar, set the `toolbarSettings.items` property to `ej.ReportViewer.ToolbarItems.All`. The following code can be used to enable search text option in toolbar.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
```

```

[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[toolbarSettings] = "toolbarSettings">
</bold-reportviewer>
`
`ts
import { Component, ViewChild } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public toolbarSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.toolbarSettings = {
      items: ej.ReportViewer.ToolbarItems.All
    }
  }
}
`

```

Hide toolbar

To hide the Report Viewer toolbar, set the `showToolbar` property to false.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"

```

```

[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[toolbarSettings] = "toolbarSettings">
</bold-reportviewer>
`
`ts
import { Component, ViewChild } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public toolbarSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.toolbarSettings = {
      showToolbar: false,
    }
  }
}
`

```

Decide or hide the export option

The Report Viewer provides the `exportOptions` property to show or hide the default export types available in the component. The following code hides the HTML export type from the default export options.

```

`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"

```

```
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[exportSettings] = "exportSettings">
</bold-reportviewer>
`
`ts
import { Component, ViewChild } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public exportSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.exportSettings = {
      exportOptions: ej.ReportViewer.ExportOptions.All & ~ej.ReportViewer.ExportOptions.HTML
    }
  }
}
```

[Add custom items to the export drop-down](#)

To add custom items to the export drop-down available in the Report Viewer toolbar, use the `customItems` property and provide the JSON array of collection input with the `index`, `cssClass` name, and `value` properties. Register the `exportItemClick` event to handle the custom item actions as given in following code snippet.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
```



```
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[exportSettings] = "exportSettings"
(exportItemClick) = "onExportItemClick($event)">
</bold-reportviewer>
`ts
import { Component, ViewChild } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public exportSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.exportSettings = {
      customItems: [{
        index: 2,
        cssClass: "",
        value: 'Text File'
      },
      {
        index: 4,
        cssClass: "",
        value: 'DOT'
      }
    ]
  }
}
```

```
};
}
//Export click event handler
onExportItemClick(event) {
  if (event.value === "Text File") {
    //Implement the code to export report as Text
    alert("Text File export option clicked");
  } else if (event.value === "DOT") {
    //Implement the code to export report as DOT
    alert("DOT export option clicked");
  }
}
}
```

Add custom toolbar item

You can add custom items to Report Viewer toolbar using the `toolbarSettings` property. You should register the `toolBarItemClick` event to handle the newly added custom items action.

Add custom item to exiting toolbar group

To add a custom item to existing toolbar group use the `customItems` property in `toolbarSettings` and provide the JSON array of collection input with the `groupIndex`, `index`, `itemType`, `cssClass` name, and `tooltip` properties as given in following code snippet.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[toolbarSettings] = "toolbarSettings"
(toolBarItemClick) = "onToolBarItemClick($event)">
</bold-reportviewer>
`ts
import { Component, ViewChild } from '@angular/core';
@Component({
```

```
selector: 'ej-app',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public toolbarSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
    this.toolbarSettings = {
      showToolbar: true,
      items: ~ej.ReportViewer.ToolbarItems.Print,
      customItems: [{
        groupId: 1,
        index: 1,
        type: 'Default',
        cssClass: "e-icon e-mail e-reportviewer-icon",
        id: 'E-Mail',
        tooltip: {
          header: 'E-Mail',
          content: 'Send rendered report as mail attachment'
        }
      }]
    }
  }
  //Toolbar click event handler
  onToolBarItemClick(event) {
    if (event.value === "CustomItem") {
      //Implement the code to CustomItem toolbar option
      alert("CustomItem toolbar option Clicked");
    }
  }
}
```

```
}
}
}
`
```

Add new toolbar group

To add a new toolbar group and custom items to it, use the `customGroups` property in the `toolbarSettings` and provide the JSON array of collection input with the `groupIndex` and `items` properties. The `items` should have the `itemType`, `cssClass`, and `tooltip` properties as given in following code snippet.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
[toolbarSettings] = "toolbarSettings"
(toolBarItemClick) = "onToolBarItemClick($event)">
</bold-reportviewer>
`

`ts
import { Component, ViewChild } from '@angular/core';
@Component({
  selector: 'ej-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public toolbarSettings: any;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'GroupingAgg.rdl';
```

```
this.toolbarSettings = {
  showToolbar: true,
  items: ~ej.ReportViewer.ToolbarItems.Print,
  customGroups: [{
    items: [{
      type: 'Default',
      cssClass: "e-icon e-mail e-reportviewer-icon CustomGroup",
      id: 'CustomGroup',
      tooltip: { header: 'CustomGroup', content: 'toolbargroups' }
    },
    {
      type: 'Default',
      cssClass: "e-icon e-mail e-reportviewer-icon subCustomGroup",
      id: 'subCustomGroup',
      tooltip: { header: 'subCustomGroup', content: 'subtoolbargroups' }
    }
  ],
  groupIndex: 3
}]
}
}

//Toolbar click event handler
onToolBarItemClick(event) {
  if (event.value === "CustomGroup") {
    //Implement the code to CustomGroup toolbar option
    alert("CustomGroup toolbar option clicked");
  }
  if (event.value === "subCustomGroup") {
    //Implement the code to subCustomGroup toolbar option
    alert("SubCustomGroup toolbar option clicked");
  }
}
}
```

See also

- [Disable the vertical scrollbar in parameter panel](#)
- [How to resolve ej undefined issue when using the Report Viewer properties](#)

Custom actions

This section explains you the steps required to add user defined buttons in Report Viewer toolbar and invoke custom actions.

Send a report as an email attachment

This topic describes steps required to create custom email option and share a report as an email attachment to other users.

Add email button in Report Viewer

1. Create an email button in the toolbar using the `customItems` property with the values such as `groupIndex`, `index`, `itemType`, `cssClass`, and `tooltip`. The `toolBarItemClick` event triggers when you click the email button.
2. Access the Report Viewer model and create a JSON array for sending requests to the Web API server. You can use the following codes to create an event with custom action.

```
`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportsWebApi",
reportPath: '~/App_Data/Sales Order Detail.rdl',
toolbarSettings: {
showToolbar: true,
items: ej.ReportViewer.ToolbarItems.All & ~ej.ReportViewer.ToolbarItems.Print,
customItems: [{
groupIndex: 1,
index: 1,
type: 'Default',
cssClass: "e-icon e-mail e-reportviewer-icon",
id: 'E-Mail',
tooltip: {
header: 'E-Mail',
content: 'Send rendered report as mail attachment'
}
}]
})
}
```

```

    }
  },
  toolBarItemClick: 'onToolBarItemClick'
});

//Toolbar click event handler
function onToolBarItemClick(args) {
  if (args.value == "E-Mail") {
    var proxy = $('#viewer').data('boldReportViewer');
    var Report = proxy.model.reportPath;
    var lastIndex = Report.lastIndexOf("/");
    var reportName = Report.substring(lastIndex + 1);
    var requrl = proxy.model.reportServiceUrl + '/SendEmail';
    var _json = {
      exportType: "PDF", reportViewerToken: proxy._reportViewerToken, ReportName: reportName
    };
    $.ajax({
      type: "POST",
      contentType: "application/json; charset=utf-8",
      url: requrl,
      data: JSON.stringify(_json),
      dataType: "json",
      crossDomain: true
    })
  }
}
</script>

```

Create custom email action

To create custom email action, follow these steps:

1. Create a new action method **SendEmail** in the Web API service.
2. Export the report to the required type using the **ReportHelper.GetReport** method to send a report stream as an attachment.

The following code sample exports the report to stream and send it as an attachment to a specified mail address. In the code, the `SmtpClient` method is used to send the report as an email attachment.

```
`csharp
public object SendEmail(Dictionary<string, object> jsonResult)
{
    string _token = jsonResult["reportViewerToken"].ToString();
    var stream = ReportHelper.GetReport(_token, jsonResult["exportType"].ToString());
    stream.Position = 0;
    if (!ComposeEmail(stream, jsonResult["reportName"].ToString()))
    {
        return "Mail not sent !!!";
    }
    return "Mail Sent !!!";
}

public bool ComposeEmail(Stream stream, string reportName)
{
    try
    {
        MailMessage mail = new MailMessage();
        SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
        mail.IsBodyHtml = true;
        mail.From = new MailAddress("xx@gmail.com");
        mail.To.Add("xx@gmail.com");
        mail.Subject = "Report Name : " + reportName;
        stream.Position = 0;
        if (stream != null)
        {
            ContentType ct = new ContentType();
            ct.Name = reportName + DateTime.Now.ToString() + ".pdf";
            System.Net.Mail.Attachment attachment = new System.Net.Mail.Attachment(stream, ct);
            mail.Attachments.Add(attachment);
        }
        SmtpServer.Port = 587;
```



```
SmtpServer.Credentials = new System.Net.NetworkCredential("xx@gmail.com", "xx");
SmtpServer.EnableSsl = true;
SmtpServer.Send(mail);
return true;
}
catch (Exception ex)
{
return ex.ToString();
}
return false;
}
`
```

In the above code sample, the report is exported to PDF format and sent to users using the `SmtpClient` method.

3. Build and run the application.

Custom properties

The custom properties helps you to include additional features that are not natively supported in the RDL reporting. This topic explains the list of custom properties supported in Angular Report Viewer.

See Also

- [Textbox Custom Properties](#)
- [Table Custom Properties](#)
- [Image Custom Properties](#)
- [Chart Custom Properties](#)
- [Report Custom Properties](#)
- [Parameter Custom Properties](#)
- [Export Custom Properties](#)

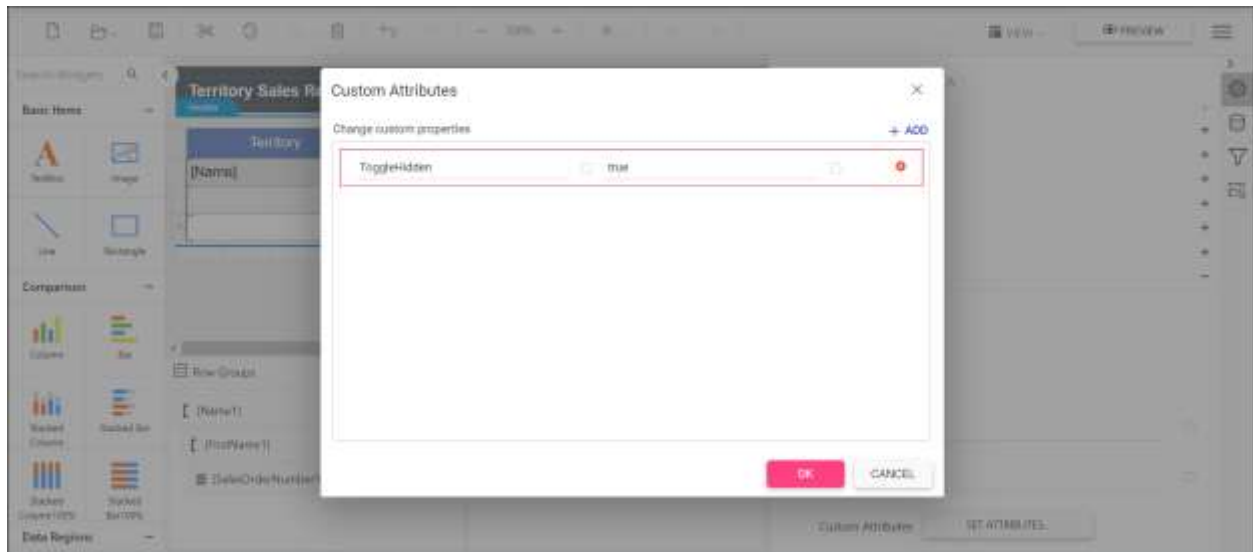
Textbox custom properties

This topic explains about the list of textbox report item custom properties that are supported to render in Angular Report Viewer.

Show or hide toggle icon in text box report item

The `ToggleHidden` custom property is used to show or hide the toggle icon in the textbox.

You can set the `ToggleHidden` property value, as shown in the below.



Before setting the toggle hidden property, the default value will be displayed as below.

The screenshot shows a preview of a 'Territory Sales Report'. The report is displayed as a table with the following data:

Territory	Sales Person	Order Number	Total Sales
Australia	Lynn Tsotlias		\$1,943,016
Canada	Garnett Vargas		\$12,808,458
Canada	José Saravia		\$4,840,689
Canada			\$7,967,769
Central	Jillian Carson		\$13,434,510
France			\$13,434,510
France	Ranjit Varkey		\$6,083,691
France	Chudakatil		\$6,083,691
Germany			\$2,476,530
Germany	Rachel Valdez		\$2,476,530
Northeast			\$12,433,503
Northeast	Michael Blythe		\$12,433,503
Northwest			\$6,305,407

Preview the report and the see the toggle icon is hidden in the report.



Territory	Sales Person	Order Number	Total Sales
Australia	Lynn Tsiftas		\$1,943,016
Canada	Garrett Vargas		\$12,808,458
	José Saravia		\$4,840,689
Central	Jillian Carson		\$7,967,769
France	Ranjit Varkey Chudakatti		\$13,434,510
Germany	Rachel Valdez		\$13,434,510
Northeast	Michael Blythe		\$6,083,691
Northwest			\$6,083,691
			\$2,476,530
			\$2,476,530
			\$12,433,503
			\$12,433,503
			\$6,306,407

Table custom properties

This topic explains about the list of table report item custom properties that are supported to render in Angular Report Viewer.

Limit number of table records on each page

The `RowsPerPage` custom property is used to specify the number of table records to display on each page. It supports integer data value greater than zero.

This property is ignored when table rows heights higher than current page size. Increase the report page height or reduce `RowsPerPage` count that fits within the page.

You can set the `RowsPerPage` property value, as shown in the below.

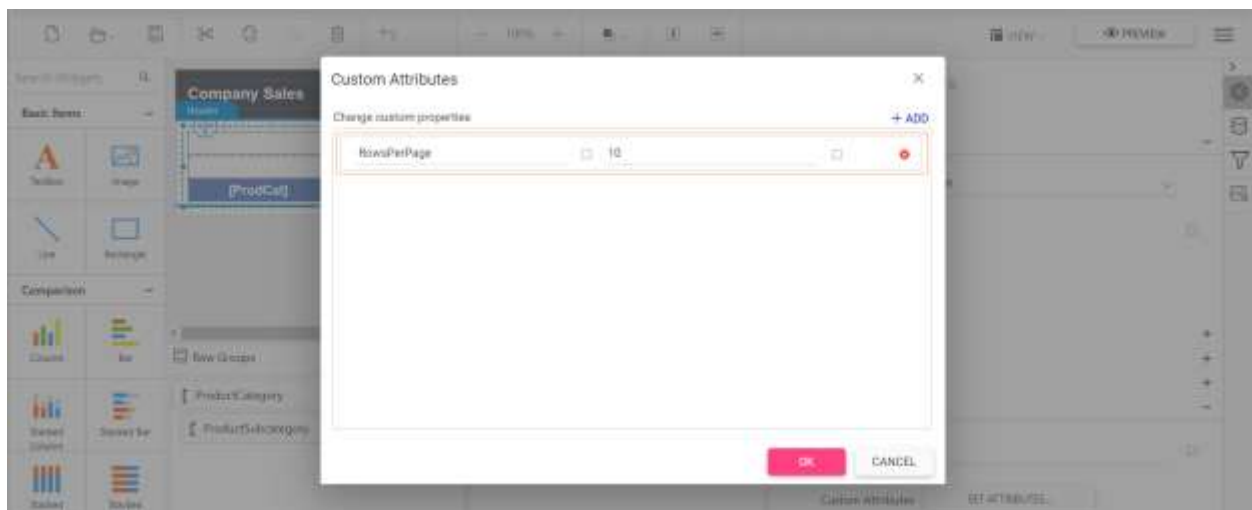
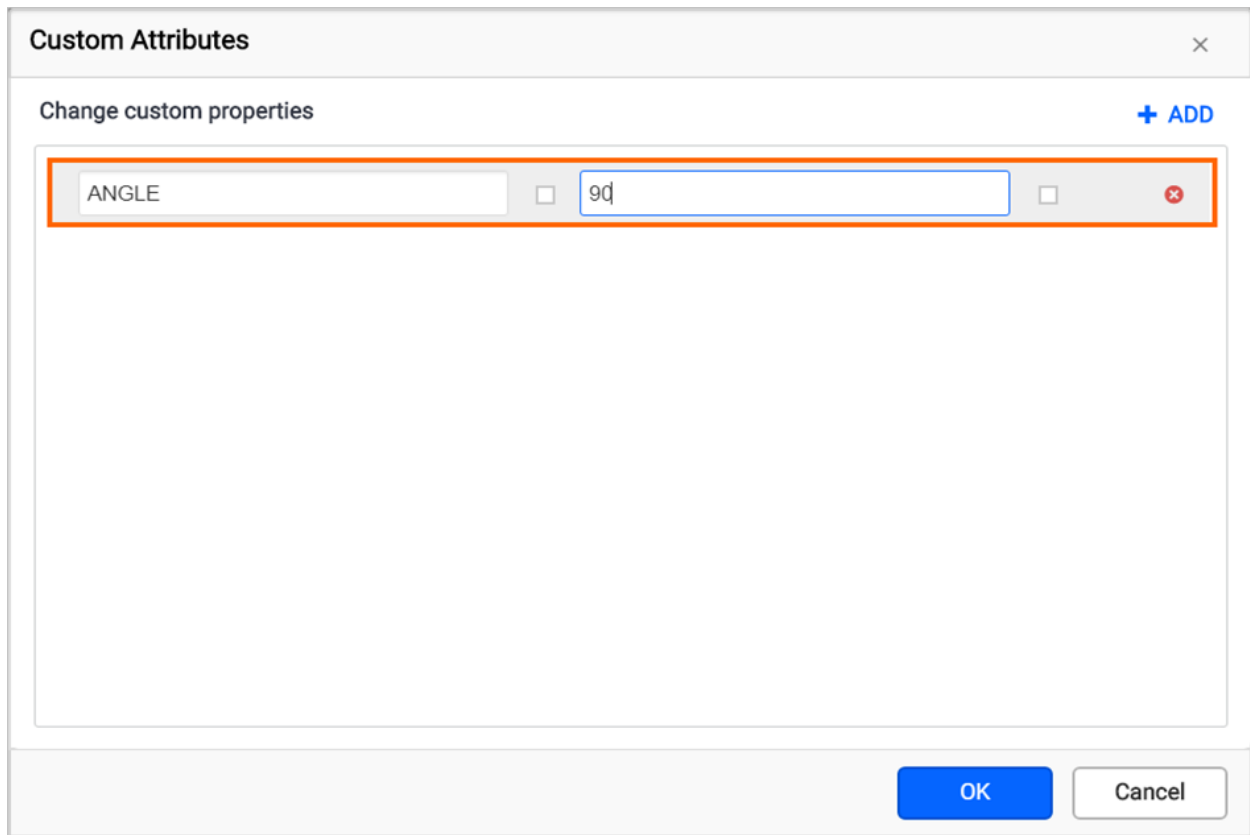


Image custom properties

This topic explains about the list of image custom properties that are supported to render in the Angular Report Viewer.

Angle

Set **Angle** custom property to image report item to rotate the image on a specified angle. It supports the angle values 0, 90, 180, and 270. You can set the property value, as shown in the below.



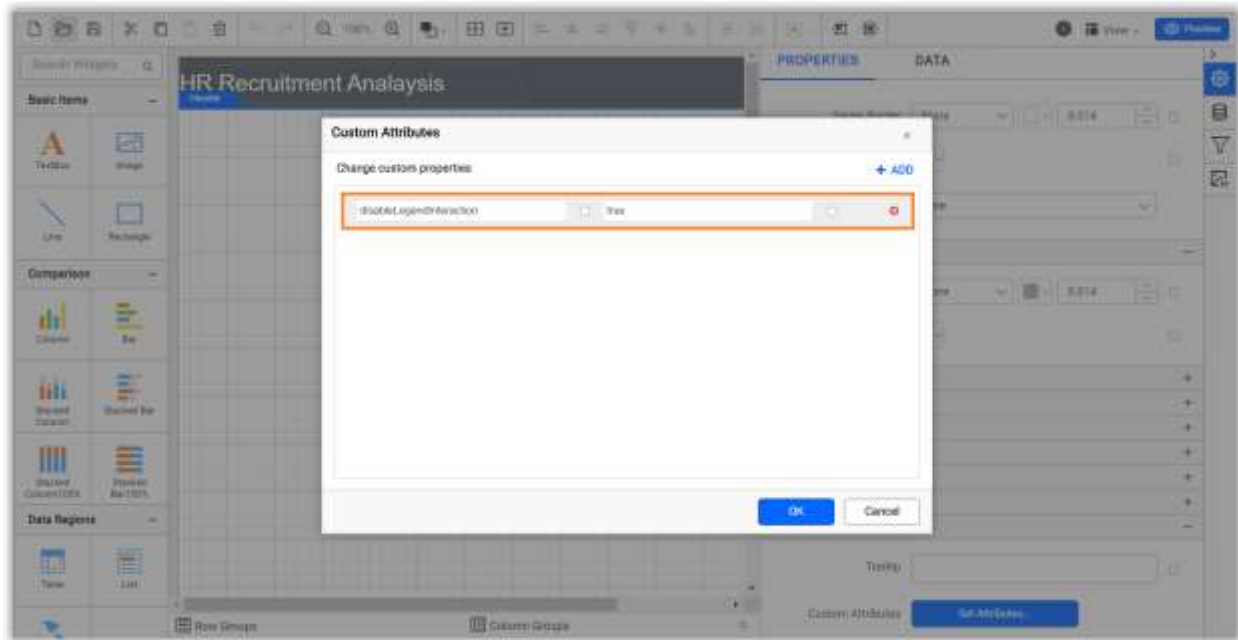
The screenshot shows a 'Custom Attributes' dialog box. At the top, there's a title bar with a close button. Below it, the 'Change custom properties' section is visible, featuring a '+ ADD' button. A list of properties is shown, with 'ANGLE' selected and its value '90' entered in a text field. The dialog box has 'OK' and 'Cancel' buttons at the bottom right.

Chart custom properties

This topic explains about the list of chart custom properties that are supported to render in the Angular Report Viewer.

Disable legend item interaction

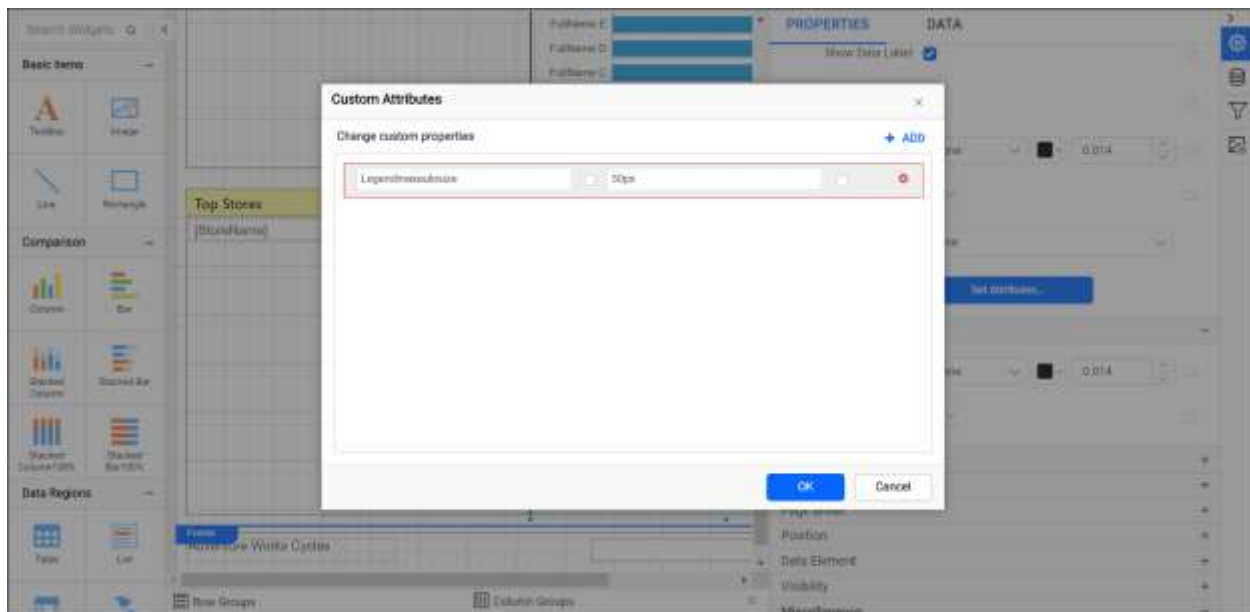
Set **DisableLegendInteraction** custom property value as **true** to stop the legend item interaction. The property value should be boolean. You can set the property value, as shown in the below.



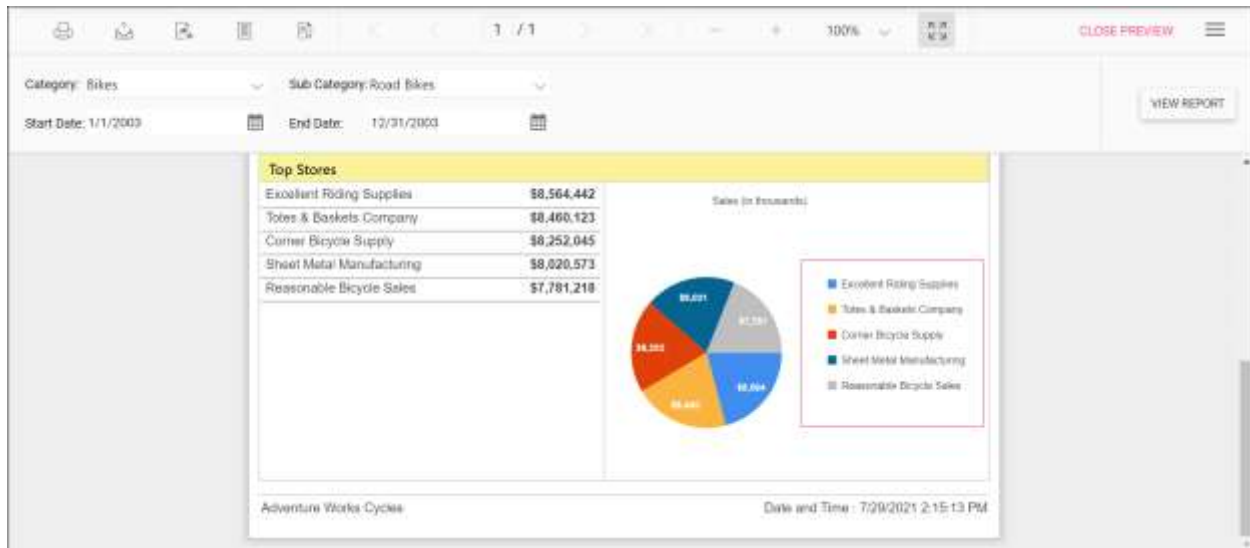
Set maximum size for chart legend

The `LegendMaxAutoSize` custom property specifies the maximum size of the legend container in the report.

You can set the `LegendMaxAutoSize` property value, as shown in the below.



Preview the report and the see legend container size in chart report.

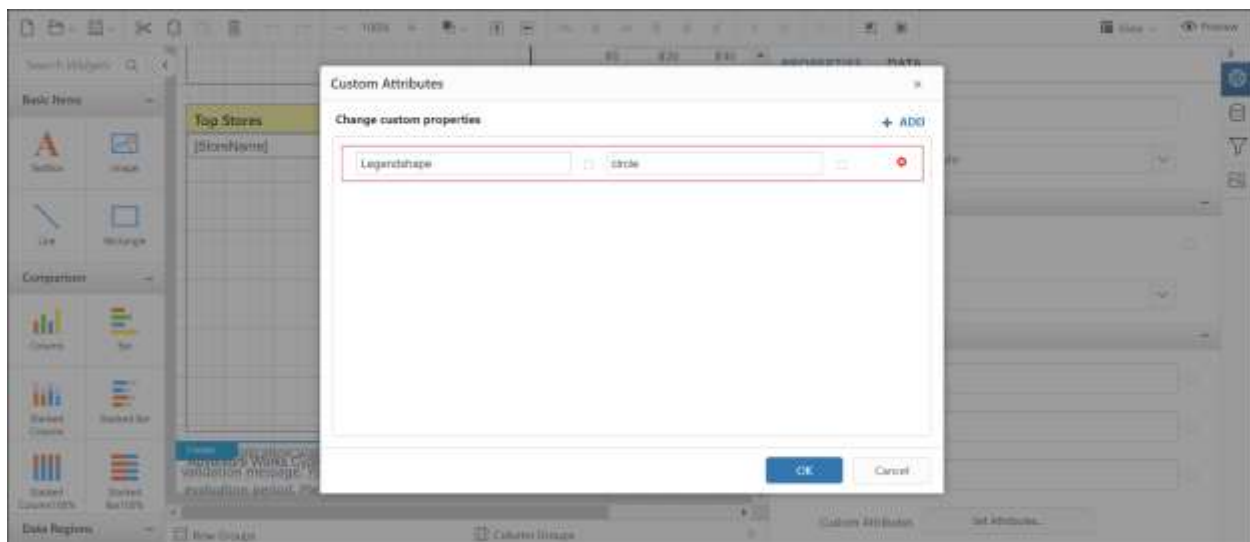


Change chart legend shape

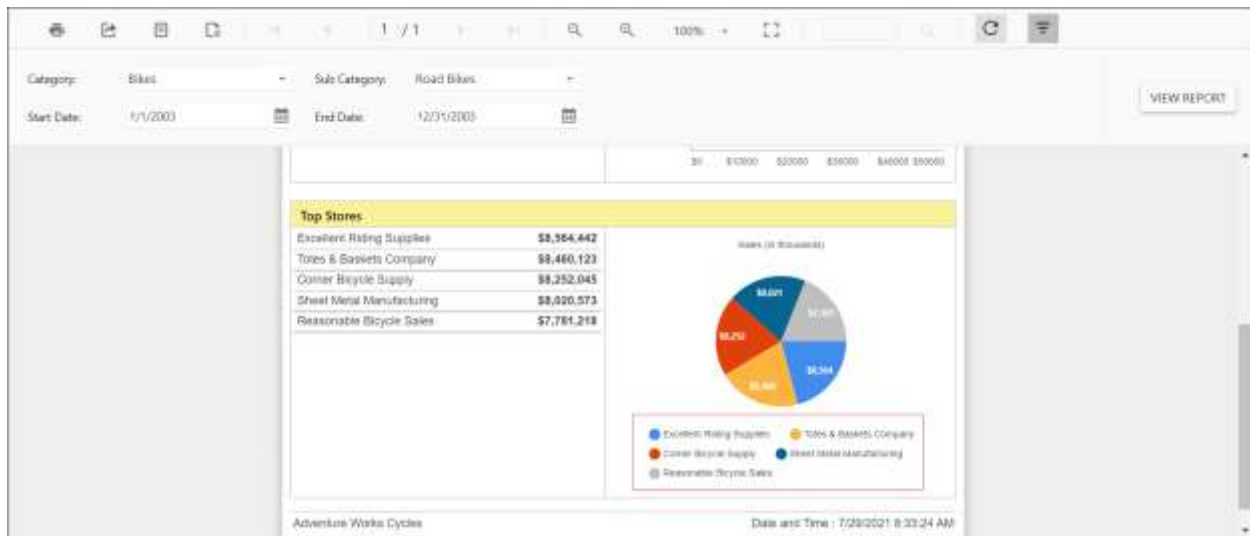
The **LegendShape** custom property allows changing the shape of the legend in the chart report item. By default, the **LegendShape** value is **rectangle**.

- Rectangle - legend shapes displayed in **rectangle**.
- Circle - legends are displayed in **circle**.
- Thumbnail - legends are displayed in **seriestype**.

You can set the **LegendShape** property value, as shown in the below.



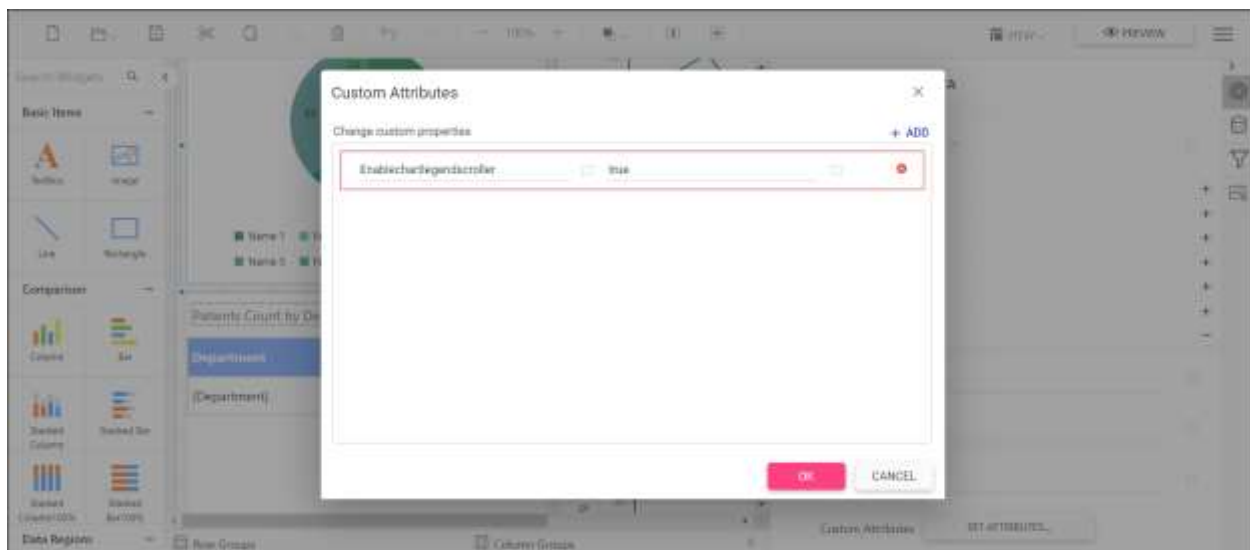
Preview the report and the see legend shape in chart report.



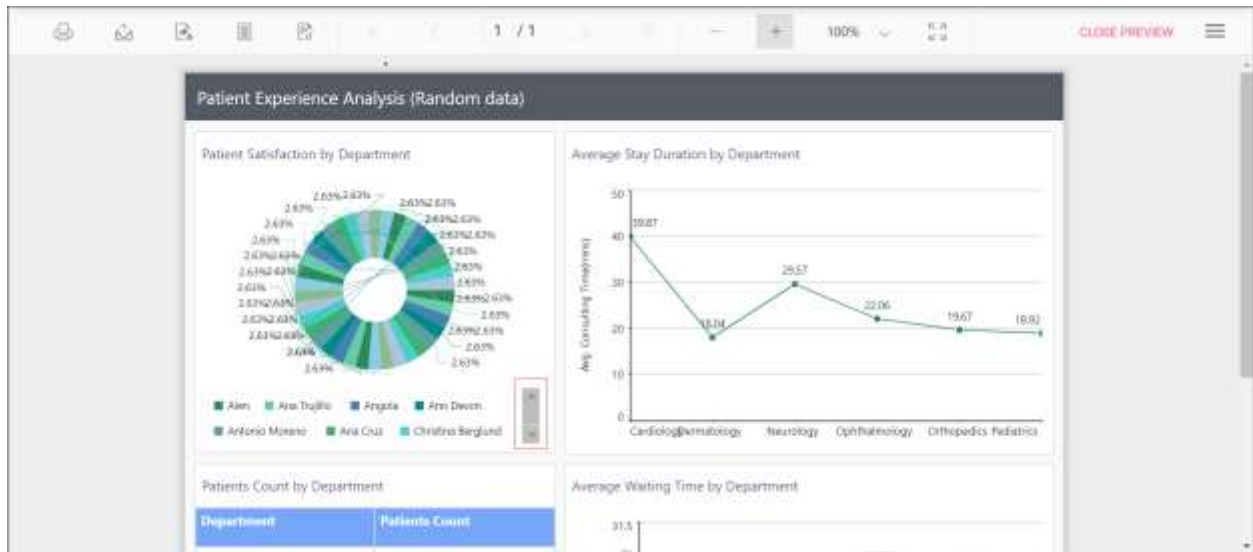
Show or hide chart legend scroller

The `EnableChartLegendScroller` custom property controls whether legend has to use scrollbar or not. The scrollbar appears depending upon size and position properties of legend. By default, the `EnableChartLegendScroller` value is `false`.

You can set the `EnableChartLegendScroller` property value, as shown in the below.



Preview the report and the see the legend scrollbar in chart report.



Set range padding for X-axis and Y-axis

Padding can be applied to the minimum and maximum extremes of the axis range by using the `XAxisRangePadding` and `YAxisRangePadding` property. The default value is `none`.

Numeric axis supports the following types of padding.

Name | Description

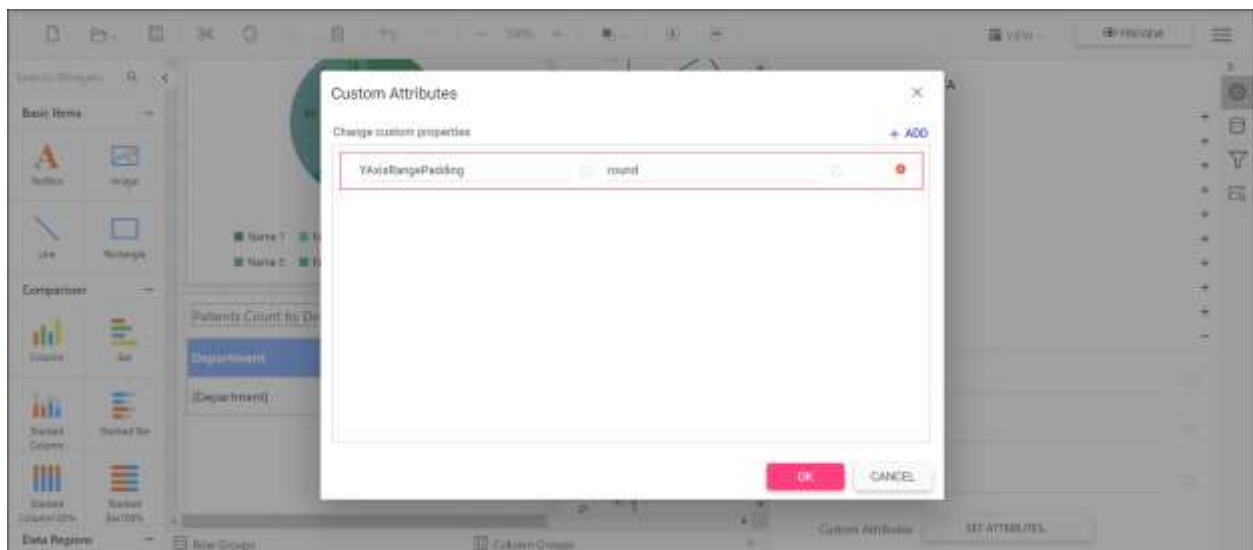
Additional | Interval of the axis is added as padding to the minimum and maximum values of the range

Normal | Padding is applied to the axis based on the range calculation

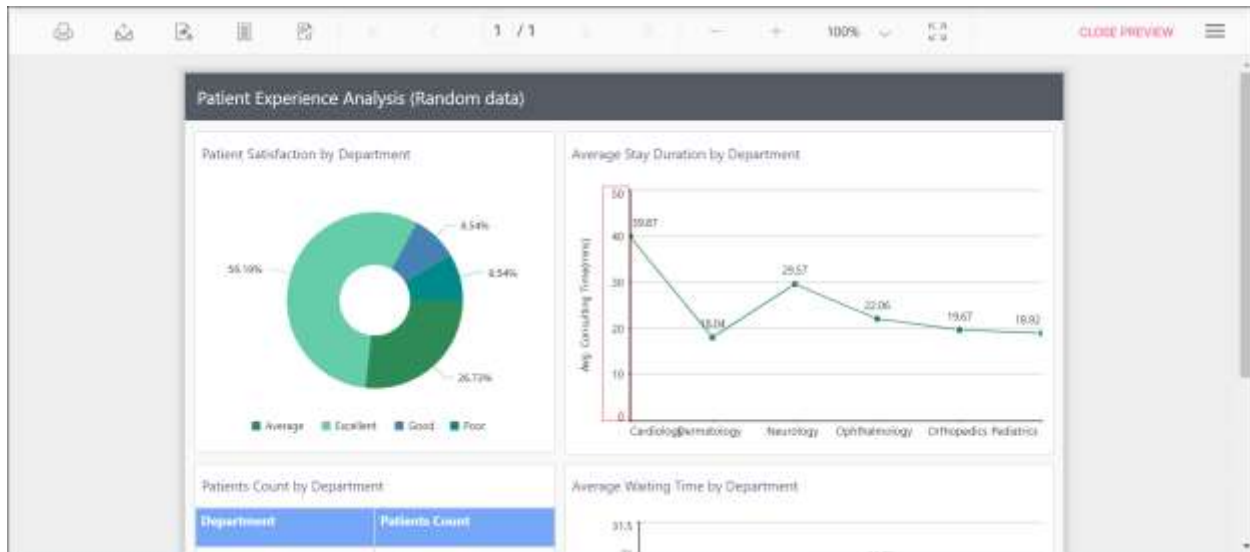
None | Padding cannot be applied to the axis

Round | Axis range is rounded to the nearest possible value divided by the interval

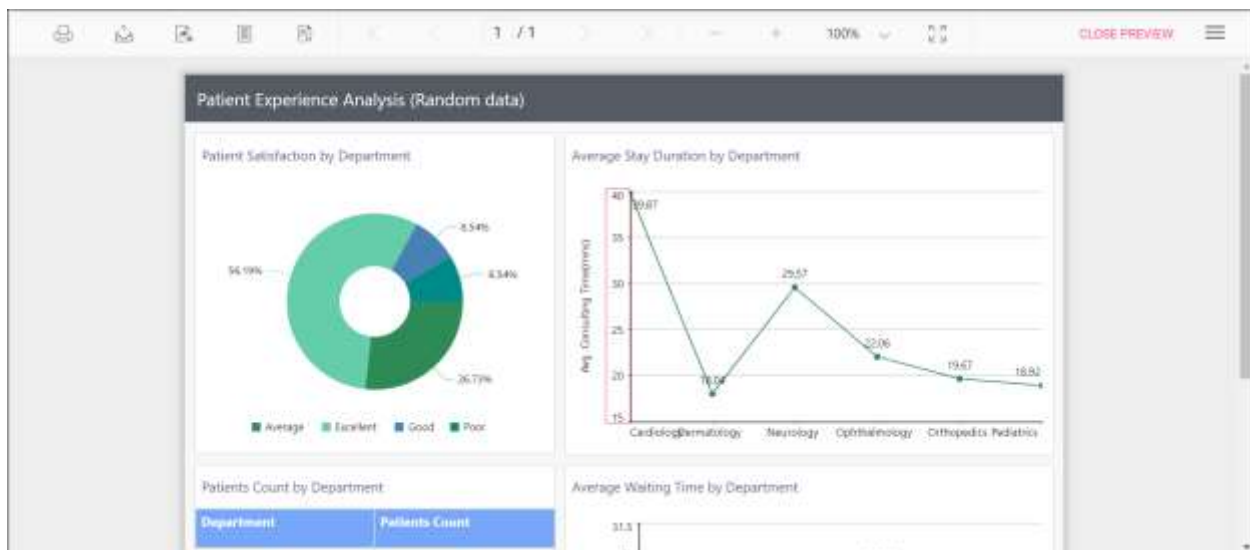
You can set the `XAxisRangePadding` and `YAxisRangePadding` property value, as shown in the below.



Before setting the range padding, the default value will be displayed as below.



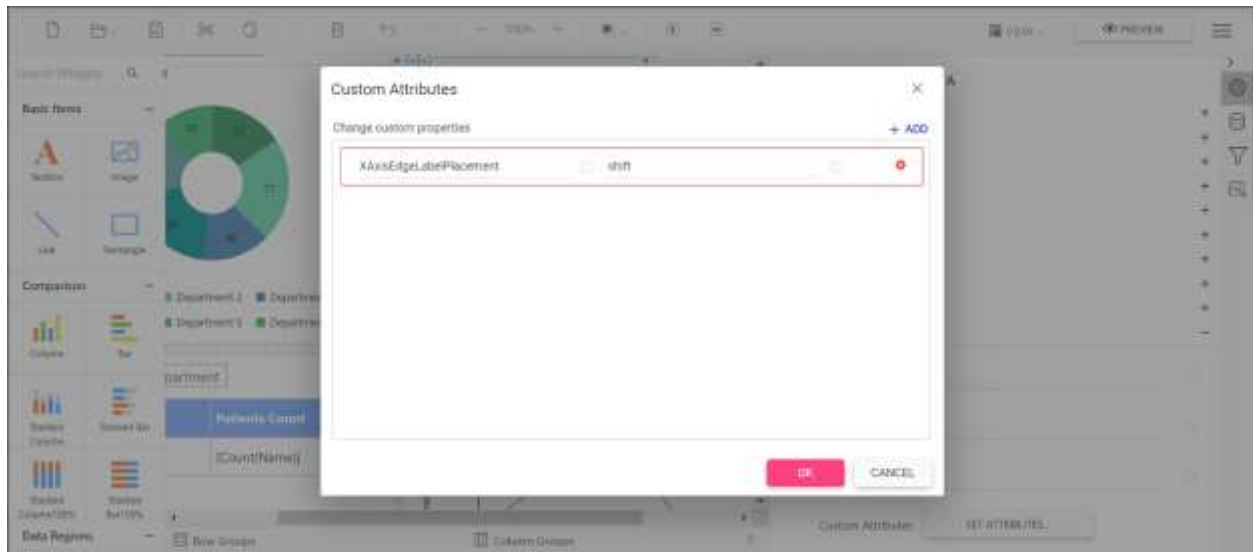
Set the padding range and see the changes in chart report as below.



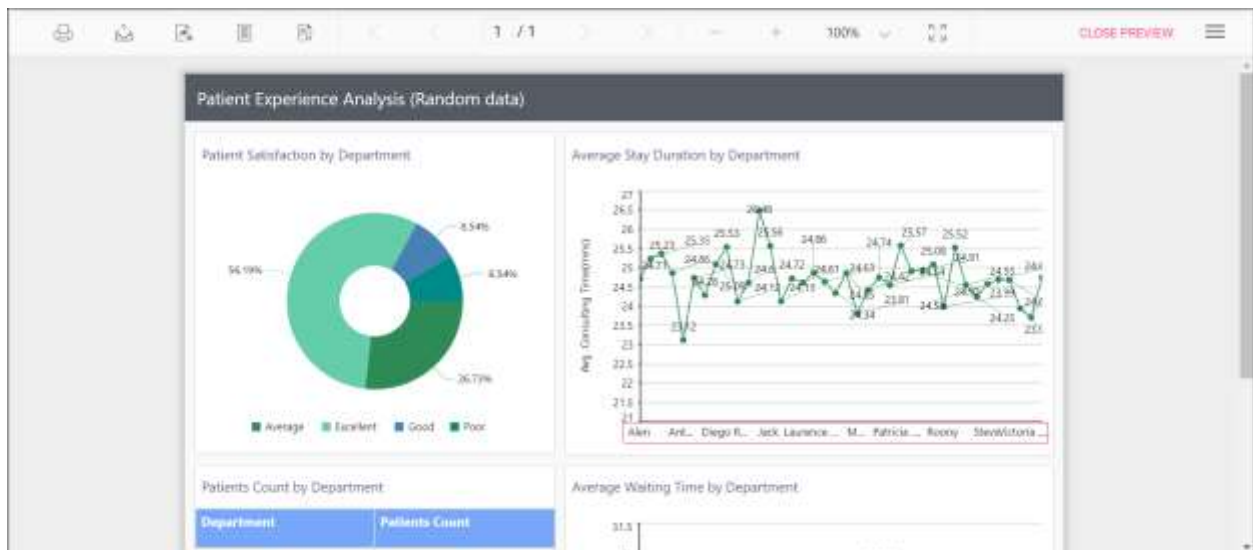
Control edge label placement in chart axis

Labels with long text at the edges of an axis may appear partially outside the chart. The `XAxisEdgeLabelPlacement` and `YAxisEdgeLabelPlacement` custom property can be used to avoid the partial appearance of the labels at the corners. The default value is `none`.

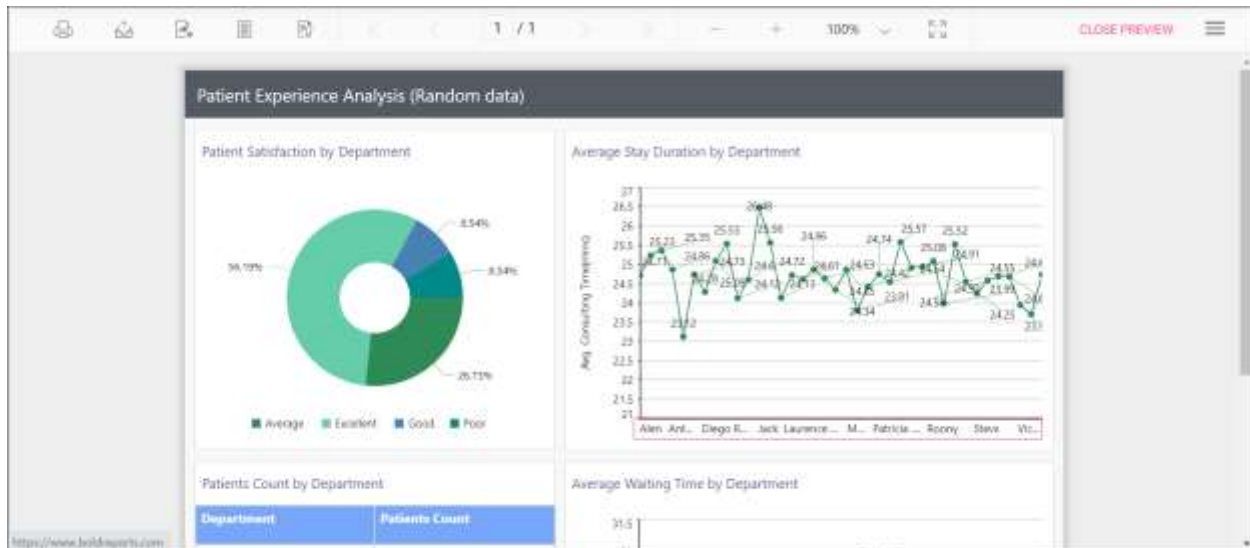
You can set the `XAxisEdgeLabelPlacement` property value, as shown in the below.



Before setting the edge label placement, the default value will be displayed as below.



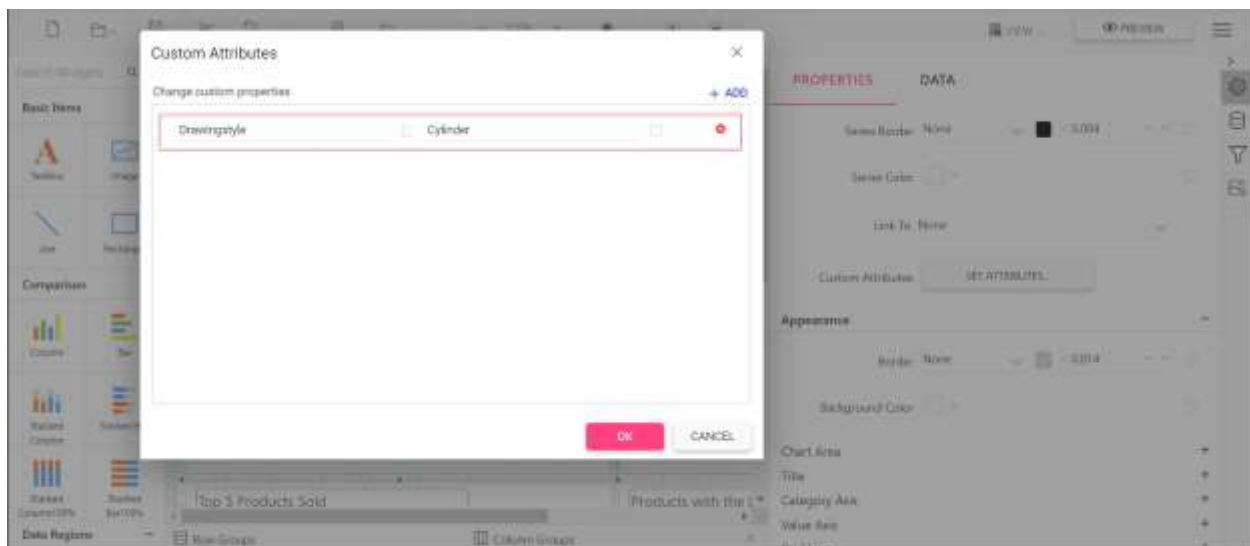
Set the edge label placement and see the changes in chart report as below.



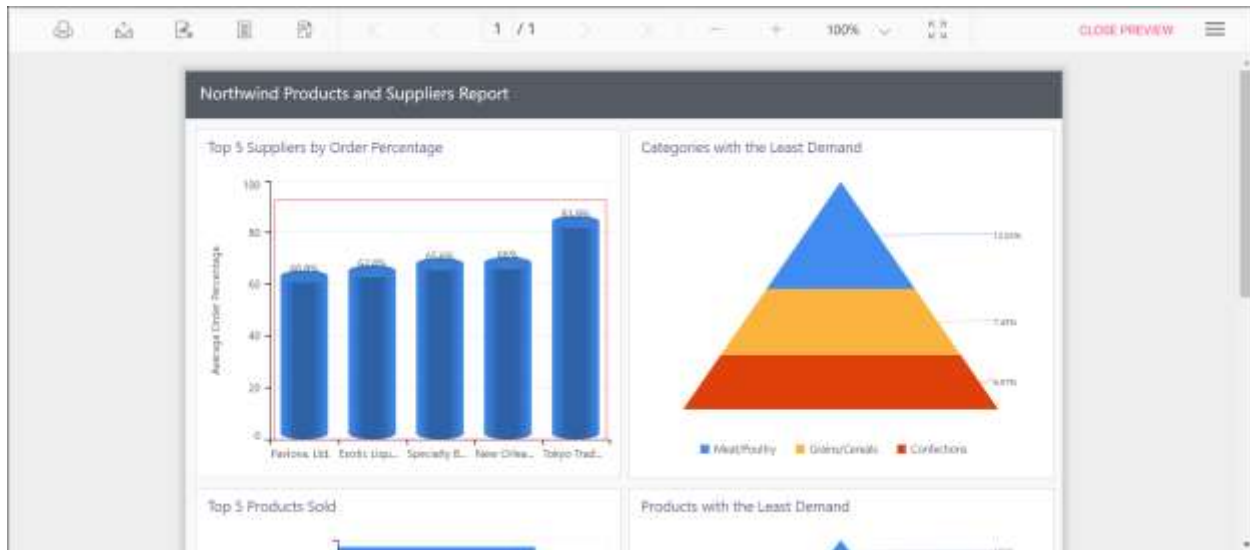
Change the drawing style of a chart column or bar series

The shape of the chart column or bar can be changed using this **Drawingstyle** custom property. By default, the **Drawingstyle** property value is **rectangle**.

You can set the **Drawingstyle** property value, as shown in the below.



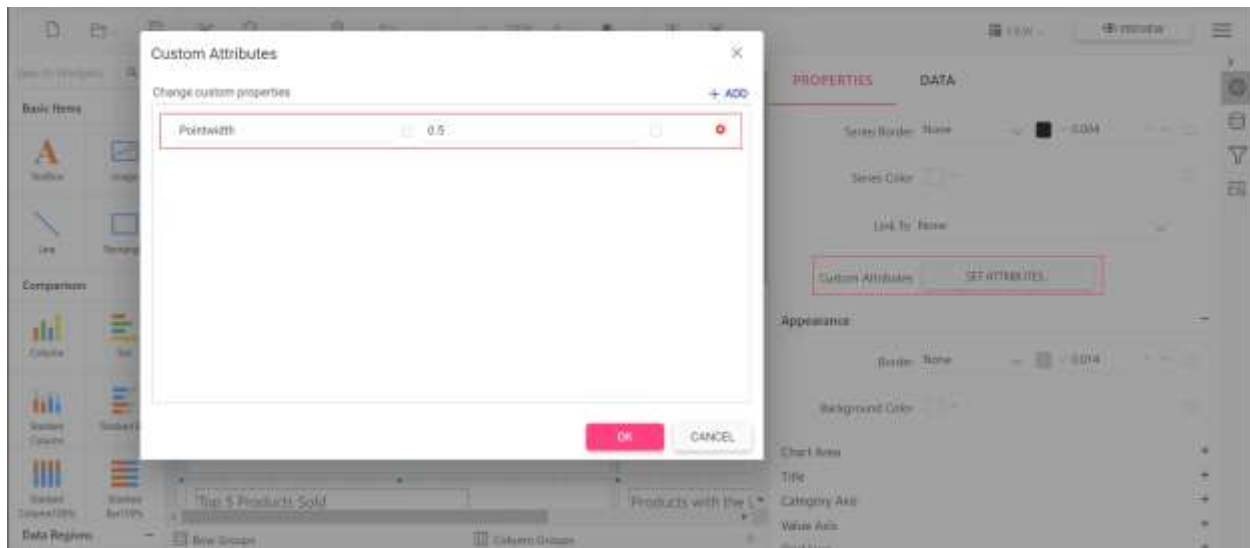
Preview the report and the see the column or bar shape in chart report.



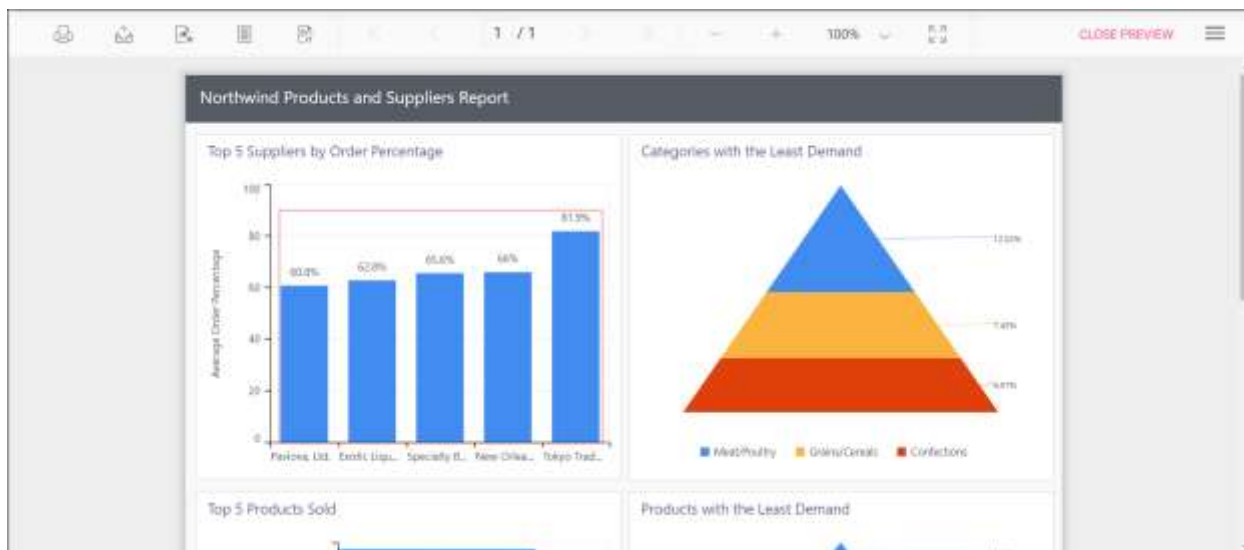
Change width of the column type series in chart

Width of the column type series can be customized by using the **Pointwidth** property. Default value of **Pointwidth** is 0.7. Value ranges from 0 to 1. Here 1 corresponds to 100% of available width and 0 corresponds to 0% of available width.

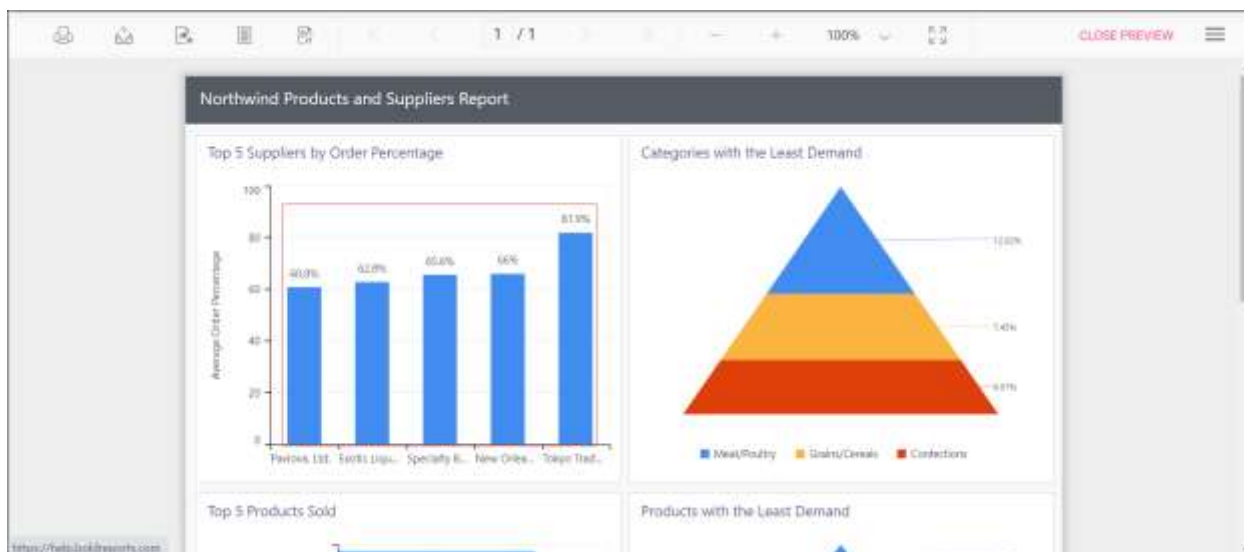
You can set the **Pointwidth** property value, as shown in the below.



Before setting the point width, the default value will be displayed as below.



Preview the report and the see the column width in chart report.



Report custom properties

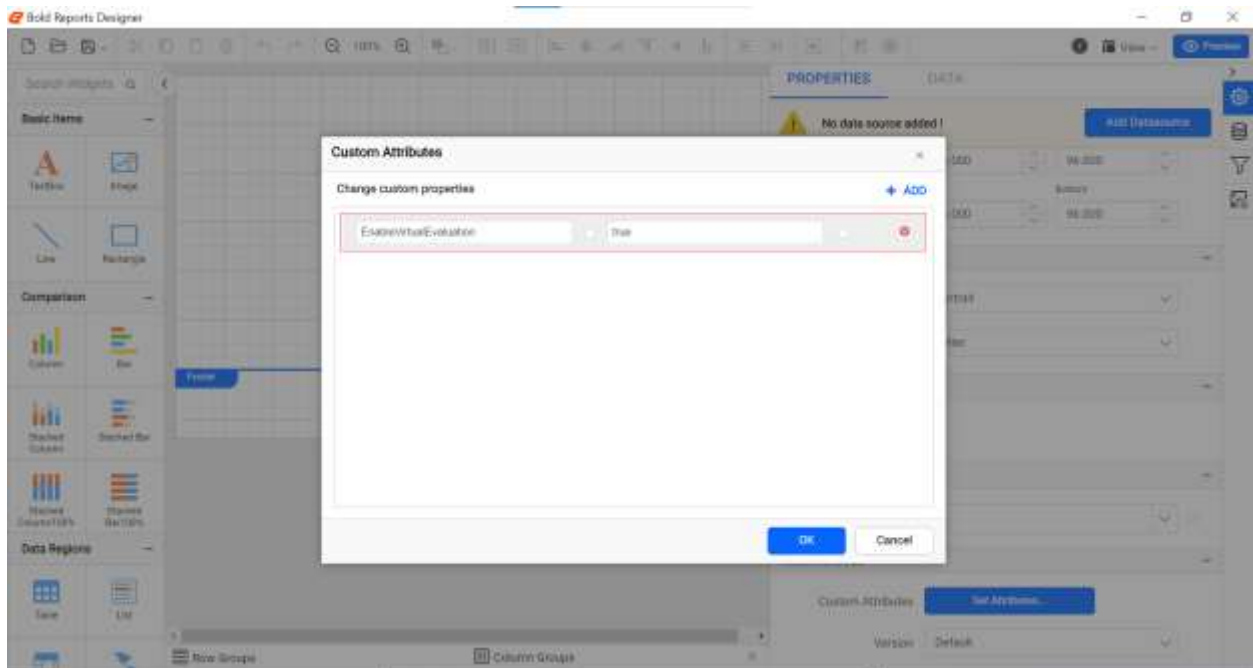
This topic explains about the list of report custom properties that are supported to render in Angular Report Viewer.

Improve performance and handle large amount of data

Set the `EnableVirtualEvaluation` and `DisablePageSplitting` custom properties in a report to improve performance and handle the larger amount of data with less memory footprint.

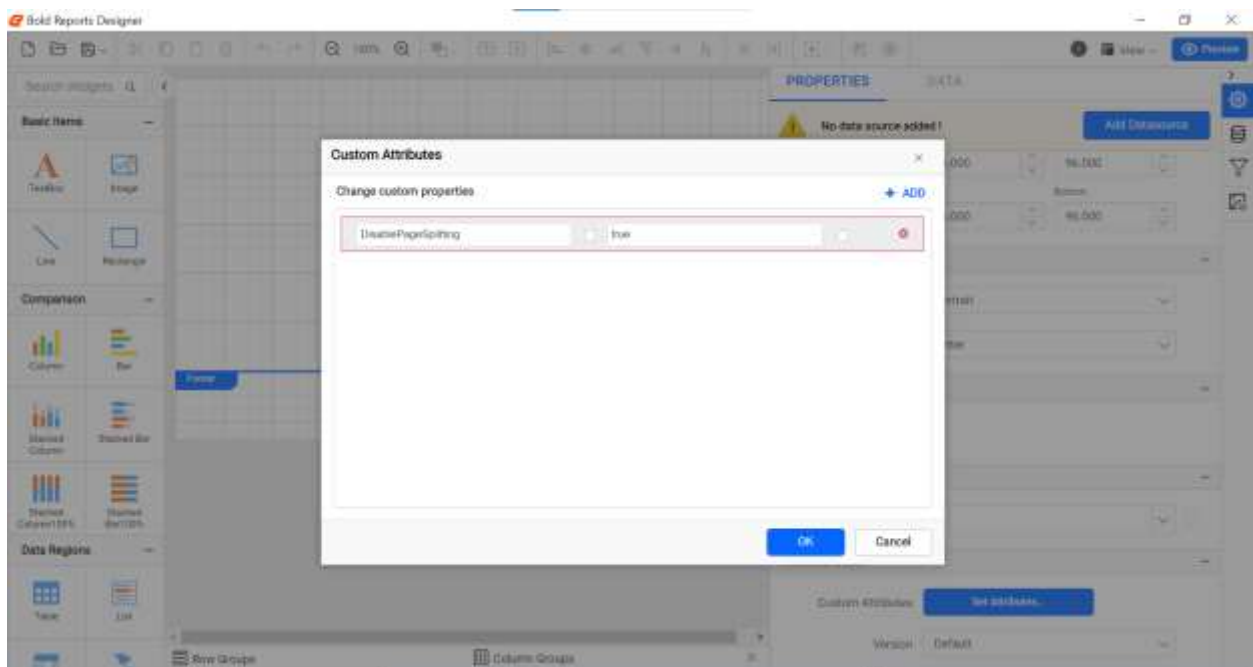
Render large data report faster

The `EnableVirtualEvaluation` custom property is used to render the large data report faster. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Reduce memory footprint for large data report

The `DisablePageSplitting` custom property is used to reduce the memory footprint for large data report. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Improve report items layout and avoid extra blank pages

When the `RoundLayoutMeasures` property is false, all non-integral values that are calculated during the report processing are rounded to whole pixel values. It provides following improvements,

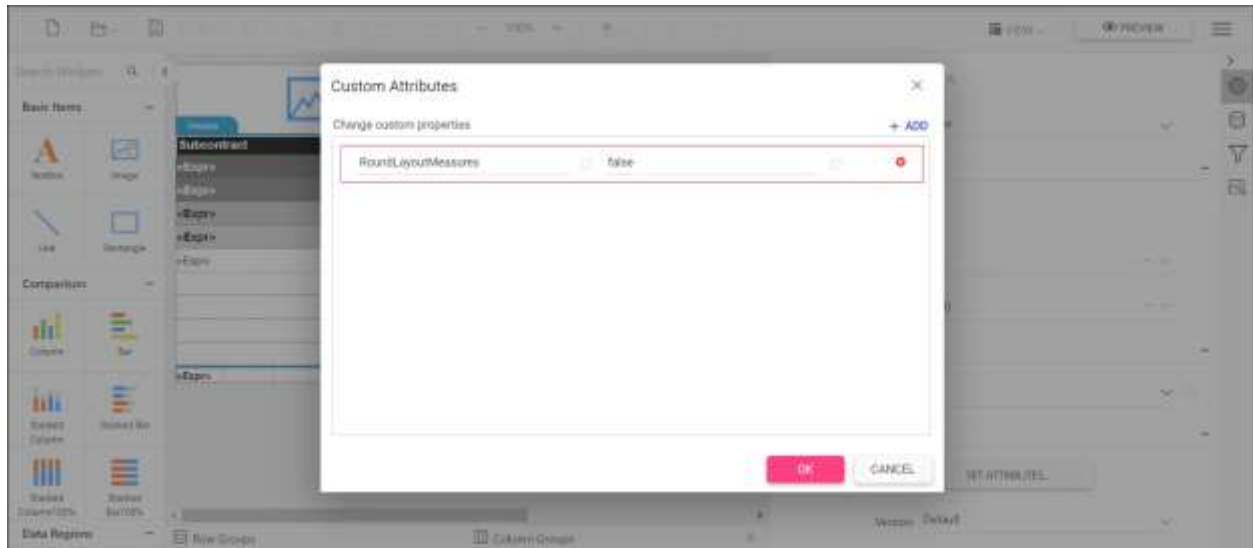
- Report text rendered without cuts.

Report custom properties
timeouts

Handling failure in long-running HTTP requests, report processes, and

- Eliminates extra blank pages.
- Removes item and text overlaps.
- Eliminates the blur semi-transparent edges that are produced by anti-aliasing.
- Produces identical look in report view and export output.

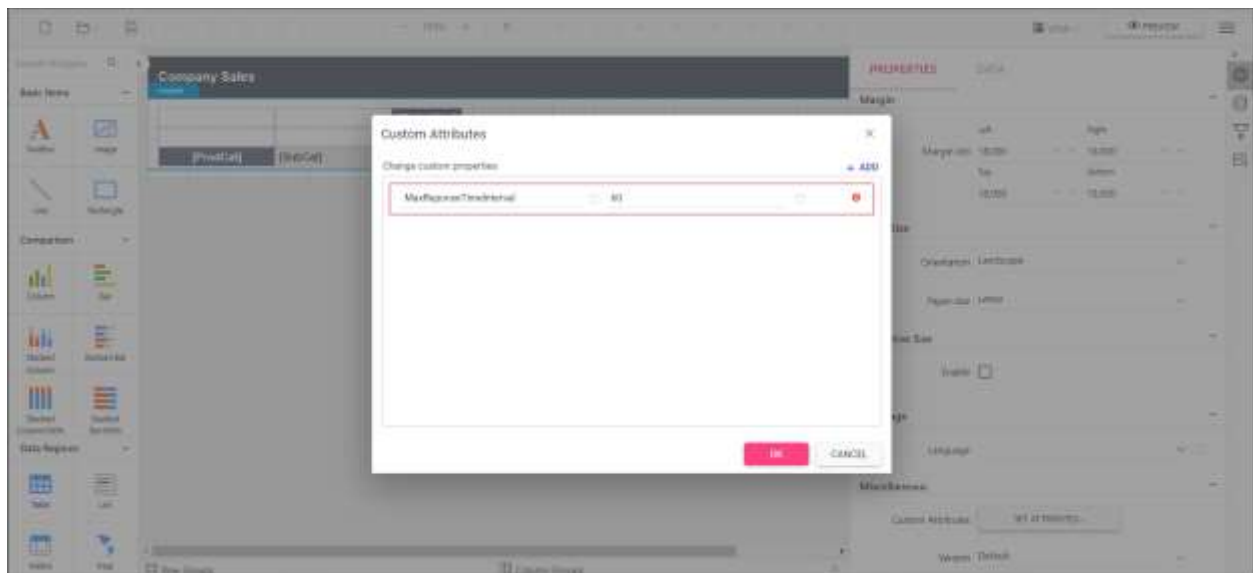
You can set the property value, as shown in the below.



Handling failure in long-running HTTP requests, report processes, and timeouts

When a report process for a long time due to huge records or a HTTP request takes too long to respond then it results in Gateway Timeout or report rendering errors. The `MaxResponseTimeInterval` allows to specify the seconds to process an HTTP request and respond back. It helps to keep the client and server connection live by avoiding the timeout.

You can set the property value as shown in the below.



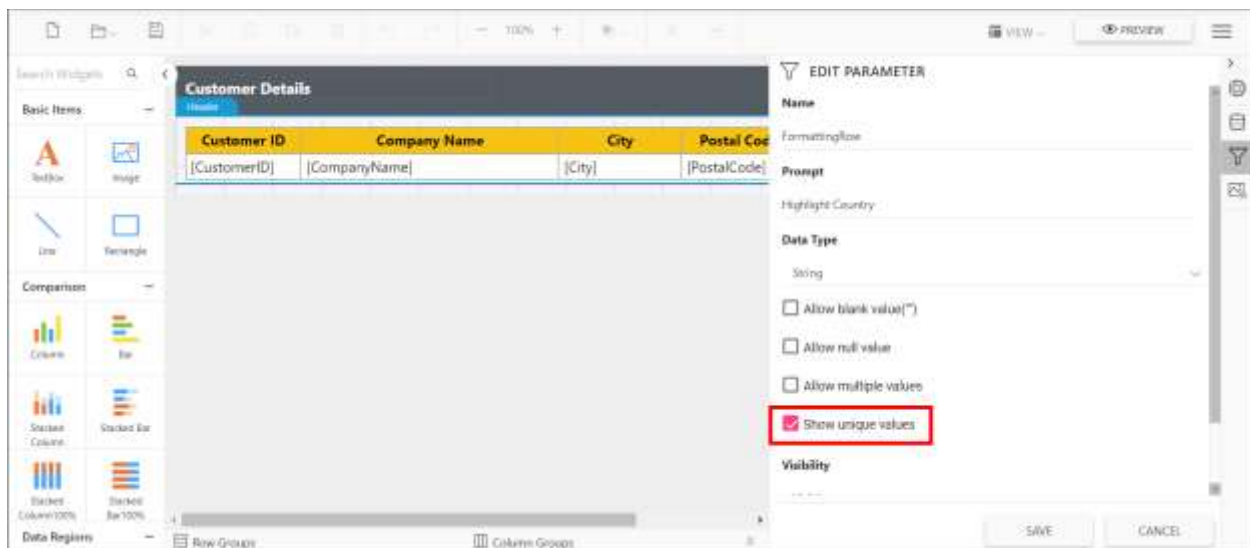
Parameter custom properties

This topic explains about the list of parameter custom properties that are supported to customize the reports preview in Report Viewer.

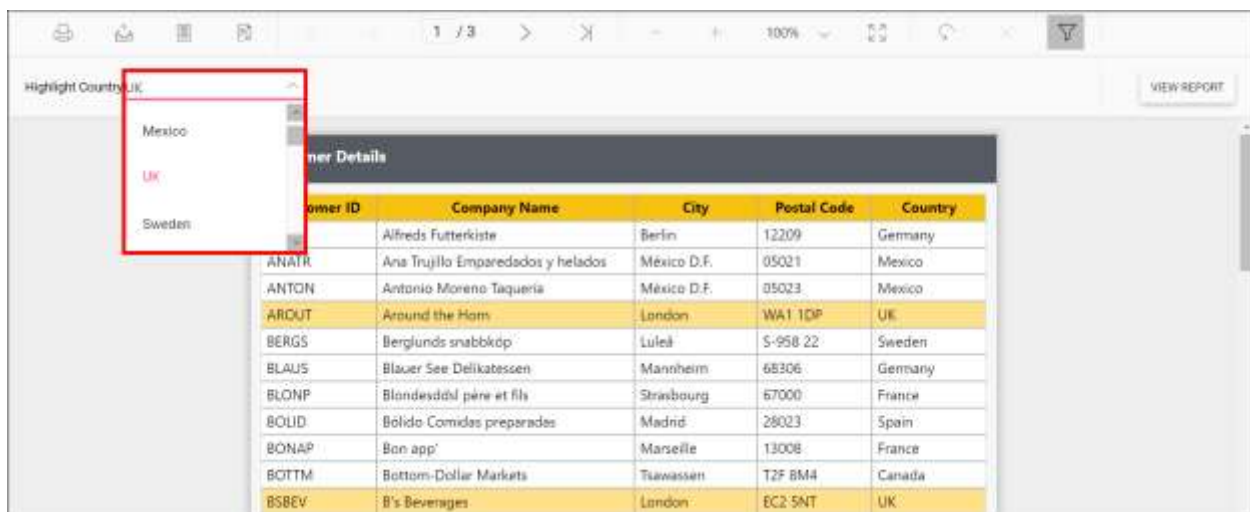
Show only distinct values in parameter

A parameter created with data set query values may contain duplicate values. The Report Viewer supports `UniqueValueParameters` custom property that helps show only unique values in the parameter drop-down while viewing the report, without creating new a data set. Refer to the below image for property setting option.

You can also provide the parameter names with comma (,) separator to set for multiple parameters.



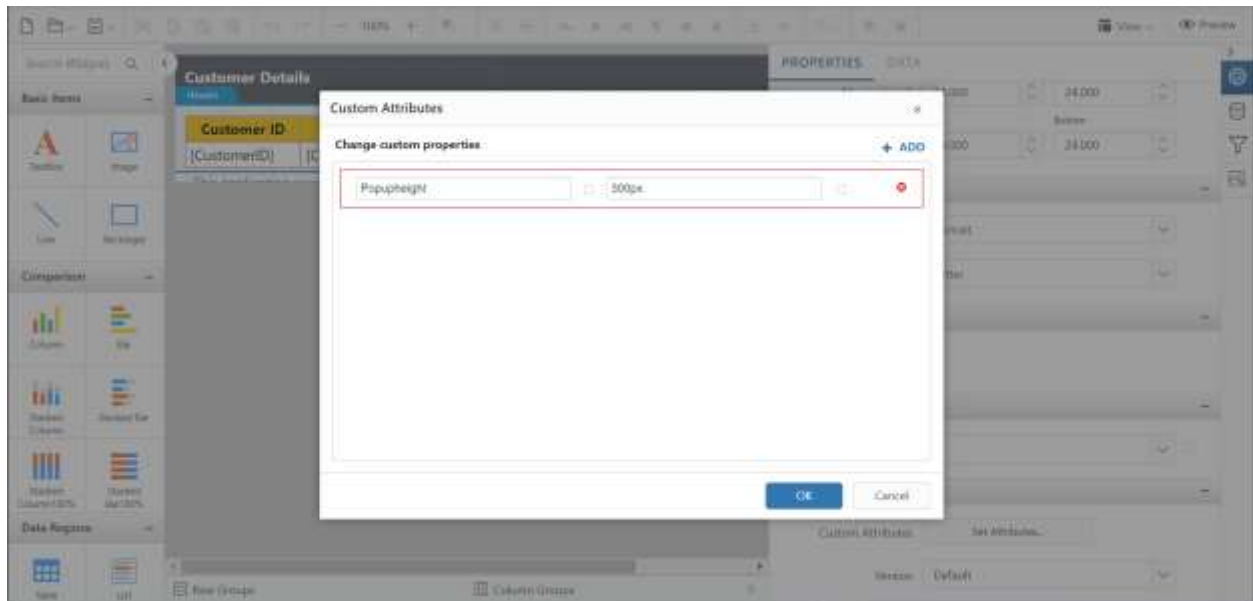
Preview the report and the unique values showed in the parameter drop down.



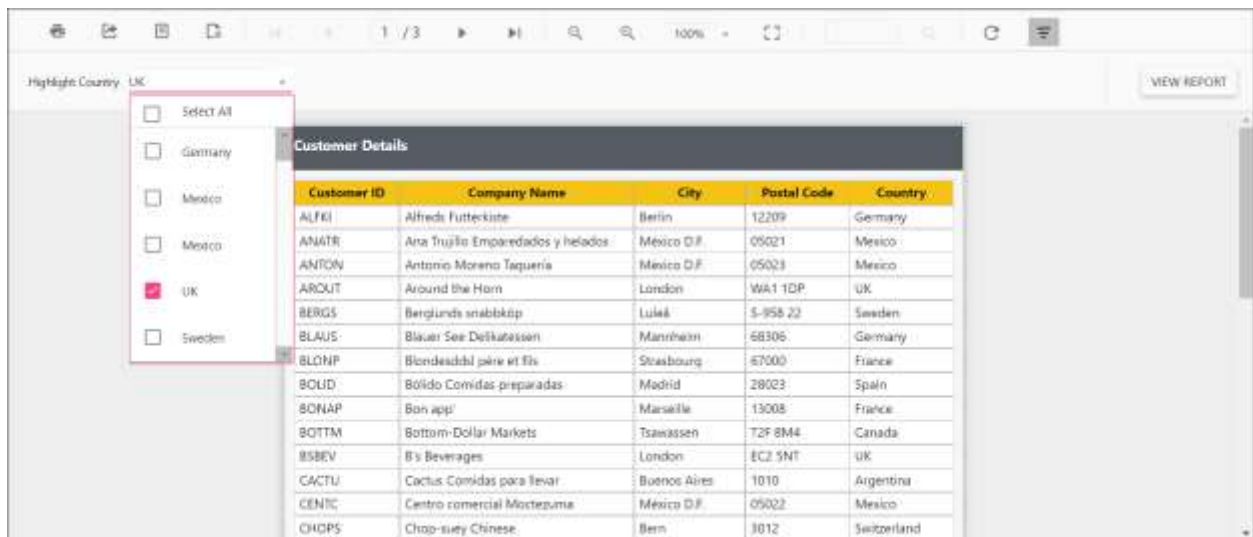
Change the dropdown parameter pop up height value

The `PopupHeight` custom property specifies the height of the parameter combobox popup list in the report. By default, the `PopupHeight` value is 152px.

You can set the `PopupHeight` property value, as shown in the below.



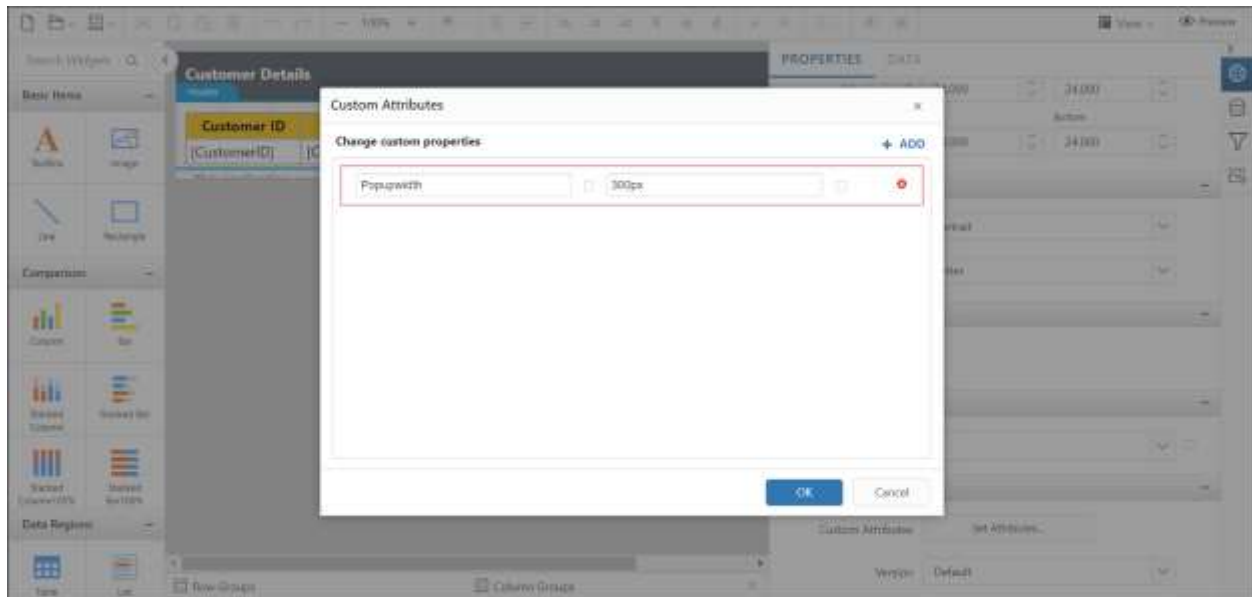
Preview the report and the pop up height showed in the parameter drop down.



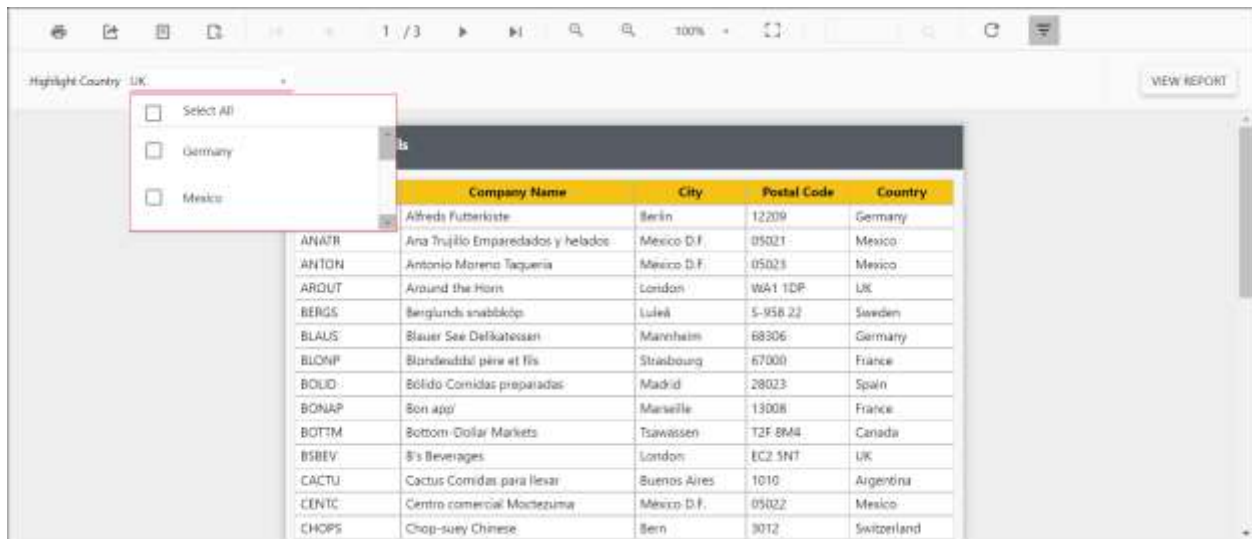
Change the dropdown parameter pop up width value

The **PopupWidth** custom property specifies the width of the parameter combobox popup list in the report. By default, the popup width sets based on the width of the component.

You can set the **PopupWidth** property value, as shown in the below.



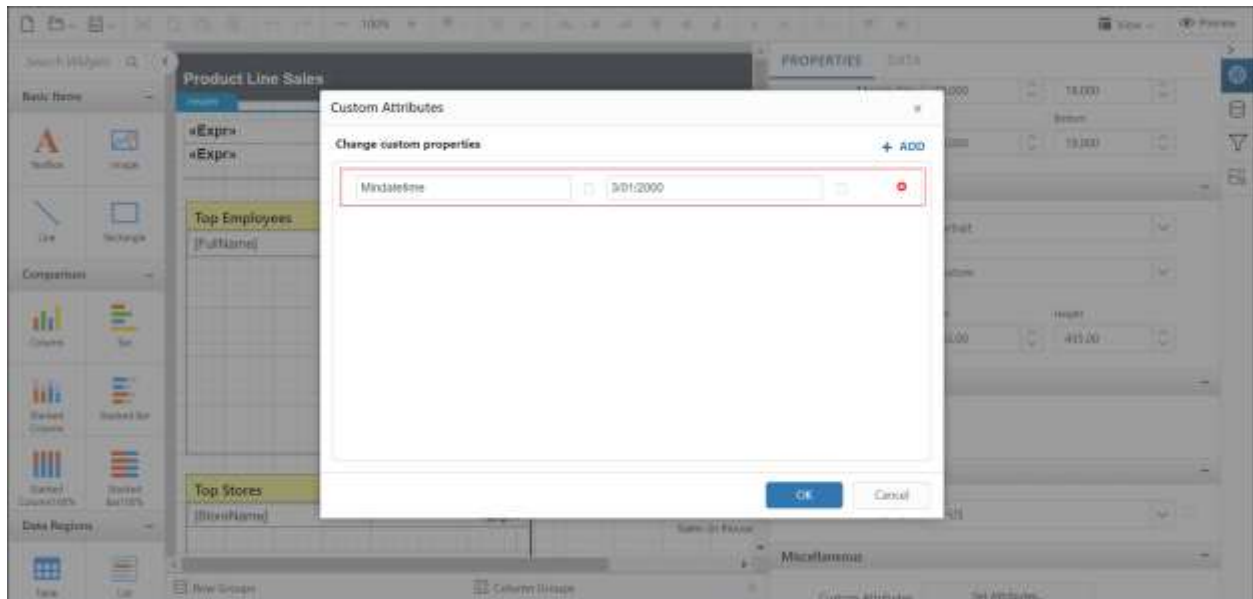
Preview the report and the pop up width showed in the parameter drop down.



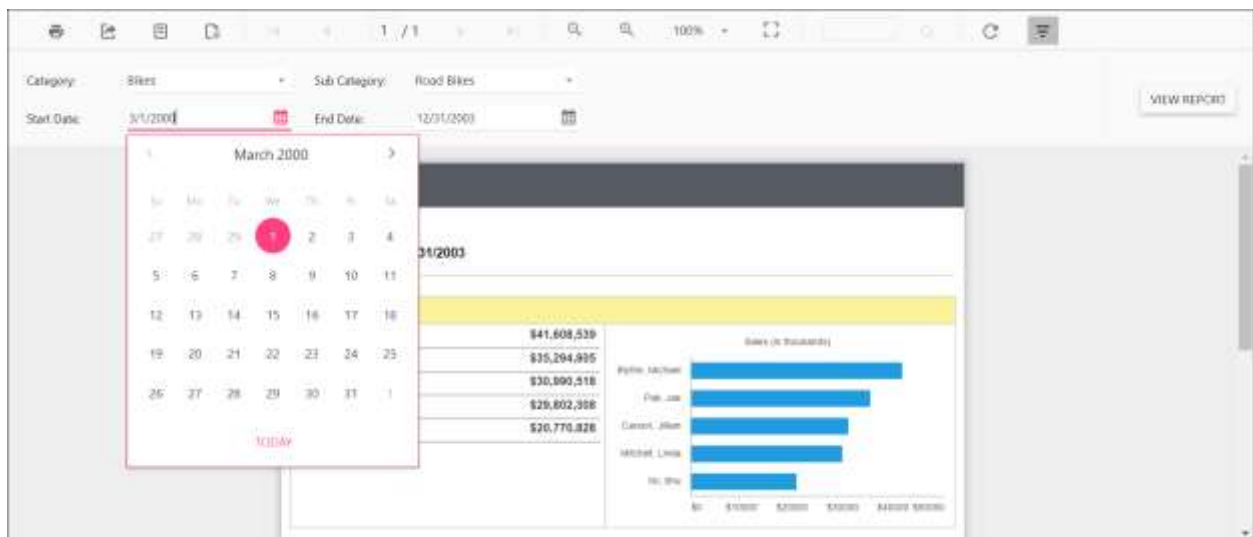
Set the minimum date range for the date report parameter

The `MinDateTime` custom property specifies the minimum date in the datetime parameter item. By default, the `MinDateTime` value is set as `null`.

You can set the `MinDateTime` property value, as shown in the below.



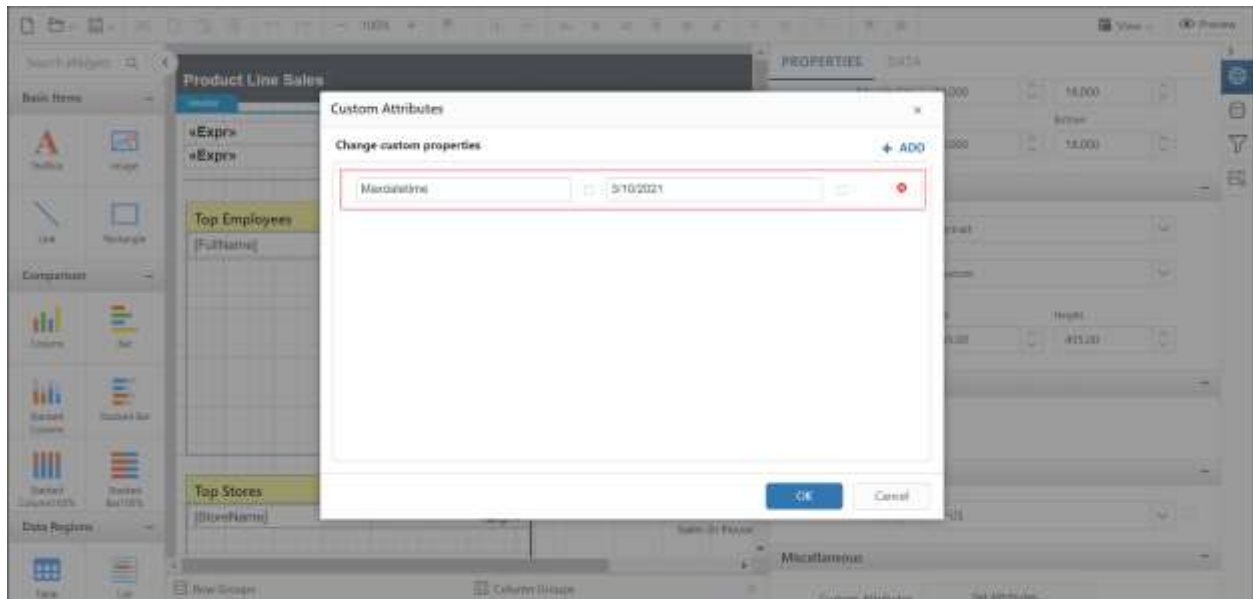
Preview the report and the minimum date showed in the datetime parameter drop down.



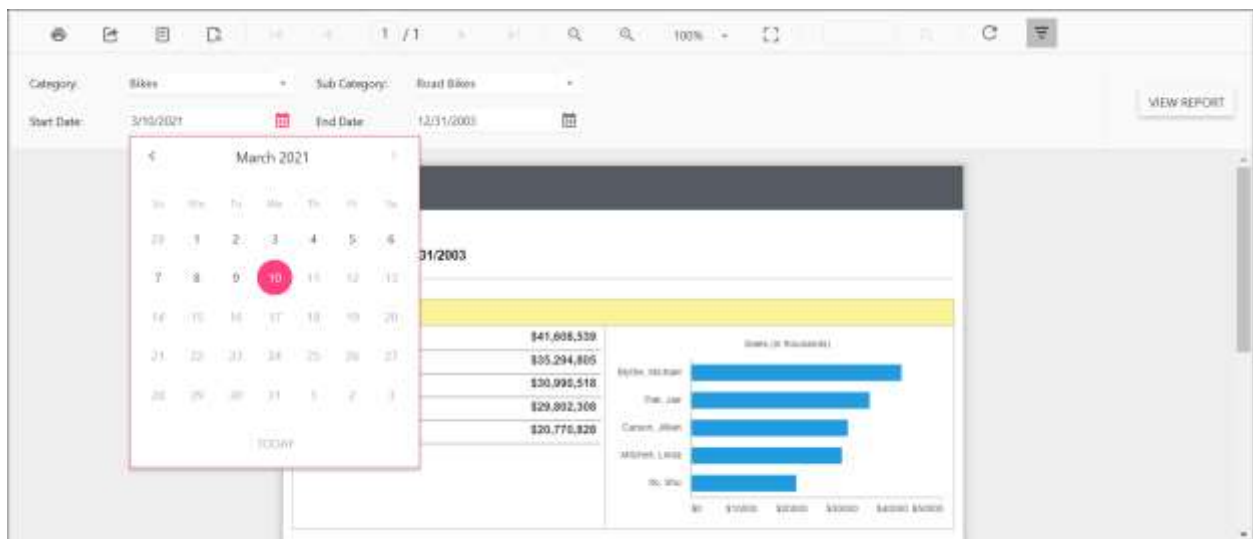
Set the maximum date range for date report parameter

The `MaxDateTime` custom property specifies the maximum date in the datetime parameter item. By default, the `MaxDateTime` value is set as `null`.

You can set the `MaxDateTime` property value, as shown in the below.



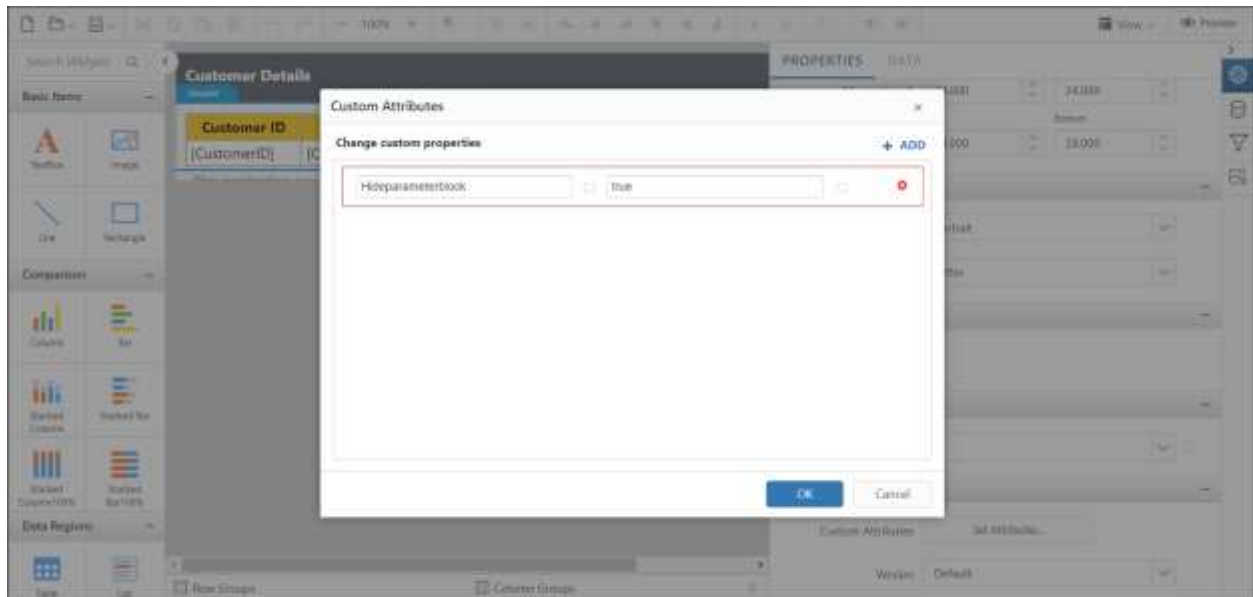
Preview the report and the maximum date showed in the datetime parameter drop down.



Hide the report parameter block

The `HideParameterBlock` custom property is used to hide the parameter block on report initial rendering. The property value should be boolean. By default, the `HideParameterBlock` value is `false`.

You can set the `HideParameterBlock` property value, as shown in the below.



Preview the report and see the parameter block is hidden.

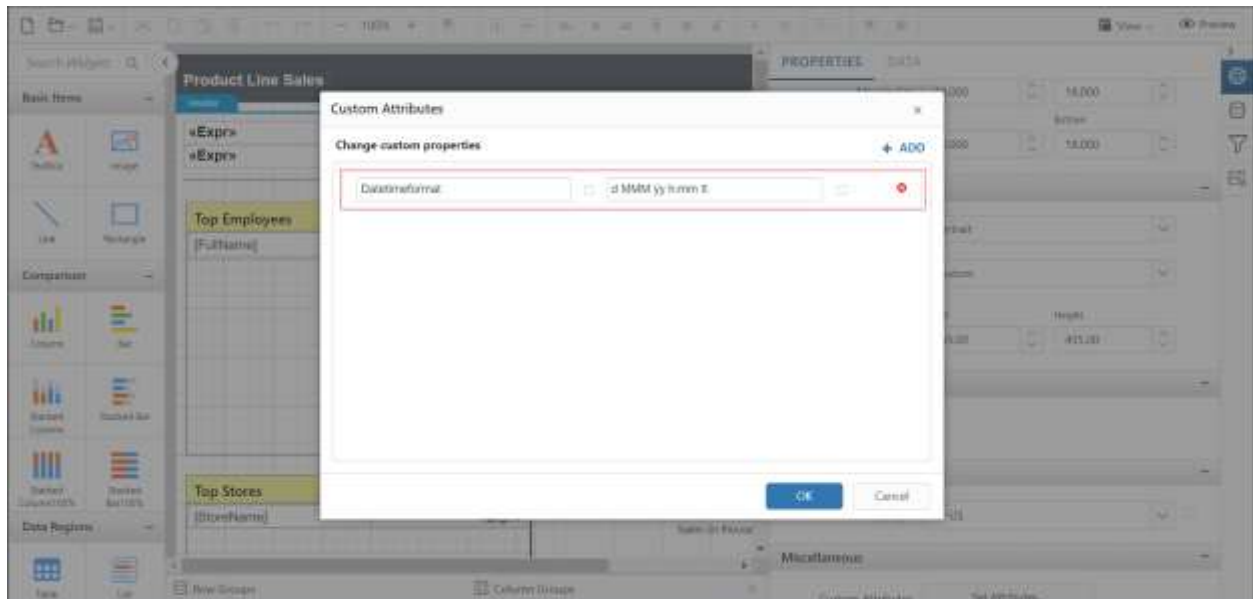
The screenshot shows a report preview titled 'Customer Details'. It displays a table with the following columns: Customer ID, Company Name, City, Postal Code, and Country. The data is filtered to show only customers from Germany, Mexico, and the UK.

Customer ID	Company Name	City	Postal Code	Country
ALFKI	Alfreds Futterkiste	Berlin	12209	Germany
ANATR	Ana Trujillo Emparedados y helados	México D.F.	05021	Mexico
ANTON	Antonio Moreno Taquería	México D.F.	05023	Mexico
AROUT	Around the Horn	London	WA1 1DP	UK
BERGS	Berglunds snabbköp	Luleå	S-958 22	Sweden
BLAUS	Blaauw Sea Delicatessen	Mannheim	68306	Germany
BONAP	Bonaparte's pizzeria	Strasbourg	67000	France
BOLID	Bólido Comidas preparadas	Madrid	28023	Spain
BONAP	Bon app'	Marseille	13008	France
BOITM	Bottom-Dollar Markets	Toronto	M5V 3M4	Canada
BSBEV	B's Beverages	London	EC2 5NT	UK
CACTU	Cactus Comidas para llevar	Buenos Aires	1010	Argentina
CENTC	Centro comercial Mochizuki	México D.F.	05022	Mexico
CHOPS	Chop-ney Chinese	Bern	3012	Switzerland
COMM1	Comércio Mineiro	São Paulo	05432-043	Brazil
CONSH	Consolidated Holdings	London	WX1 6LT	UK
DRACD	Drachendorff & Göttsche	Dresden	81066	Germany

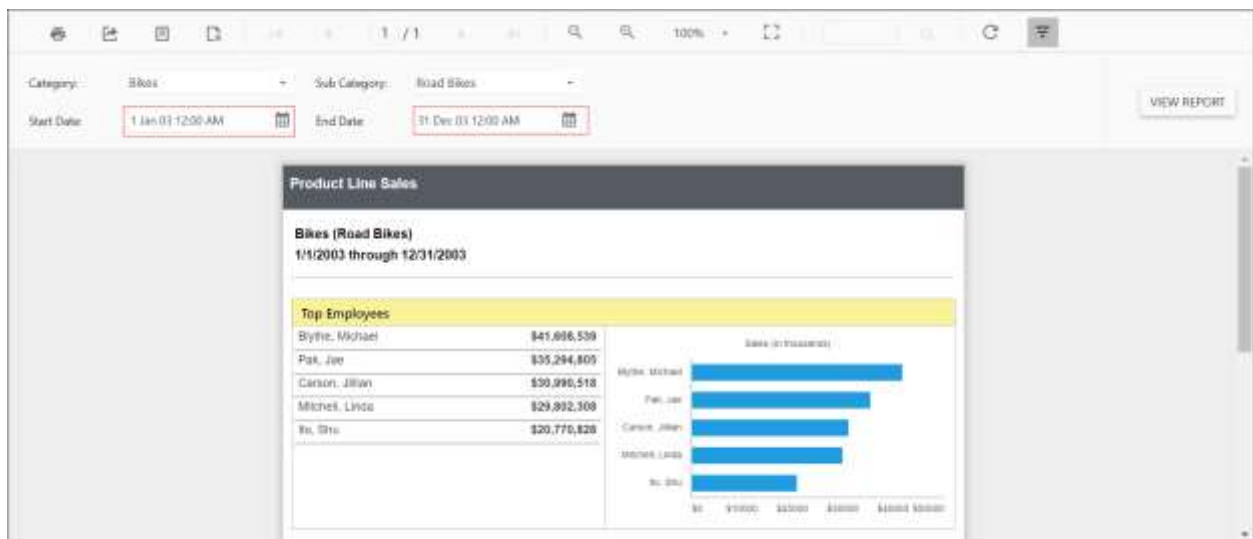
Set date and time format for parameter

The `DateTimeFormat` custom property defines the date time format to be displayed in the `DateTimePicker` popup. By default, the `DateTimeFormat` value is set as `empty`.

You can set the `DateTimeFormat` property value, as shown in the below.



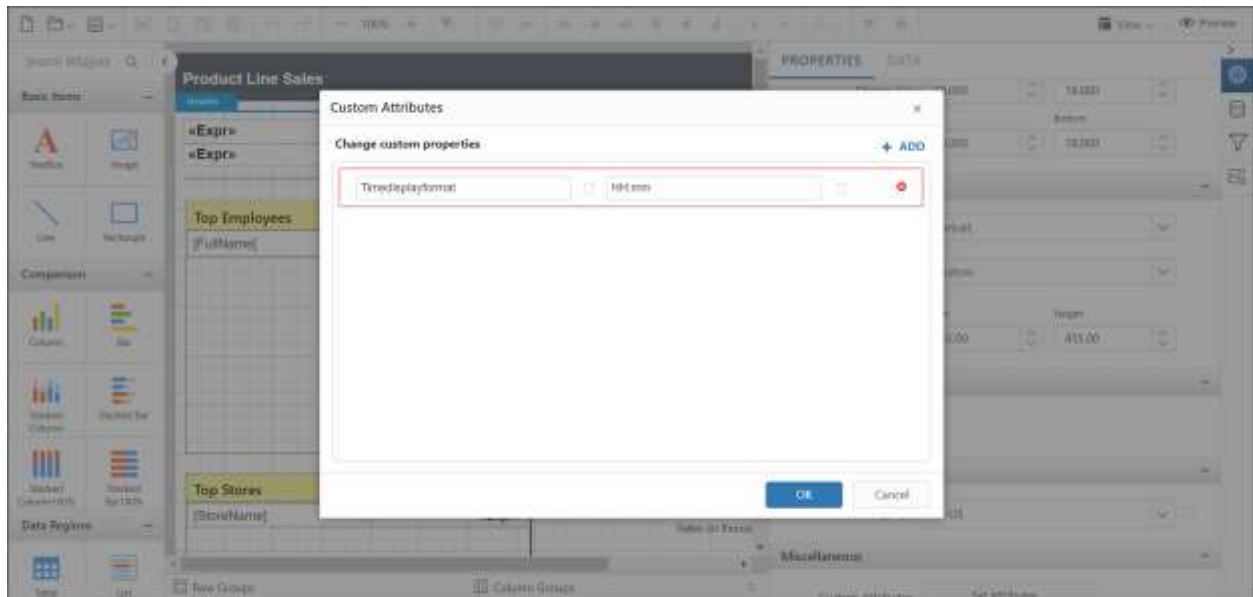
Preview the report and the see date time format in datetime parameter.



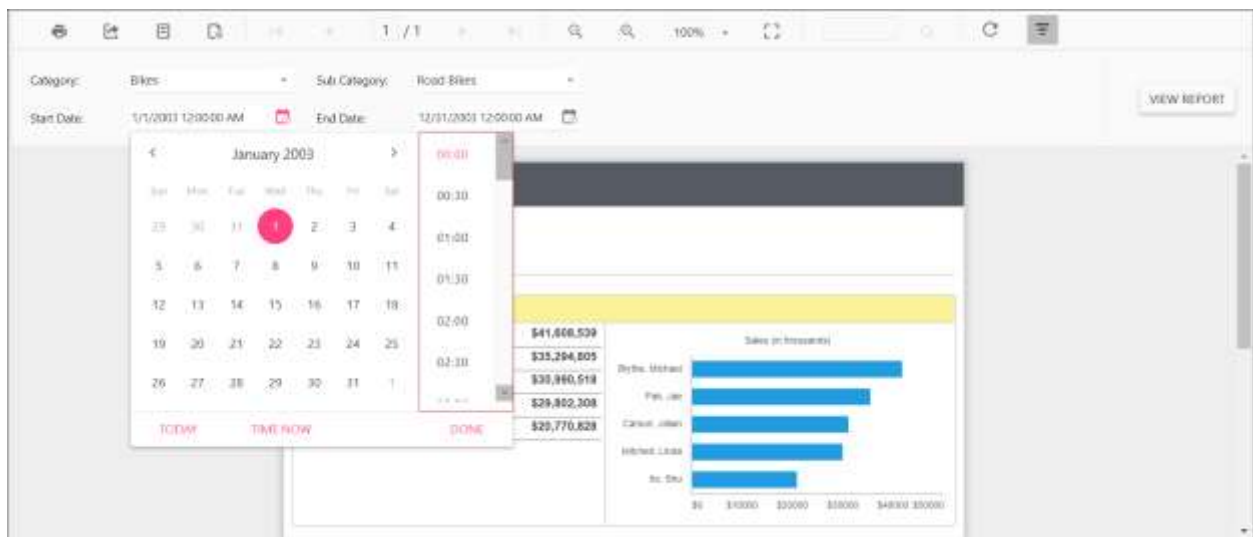
Change time display format for parameter

The `TimeDisplayFormat` custom property defines the time format to be displayed in the time dropdown inside the `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeDisplayFormat`. By default, the `TimeDisplayFormat` value is set as empty.

You can set the `TimeDisplayFormat` property value, as shown in the below.



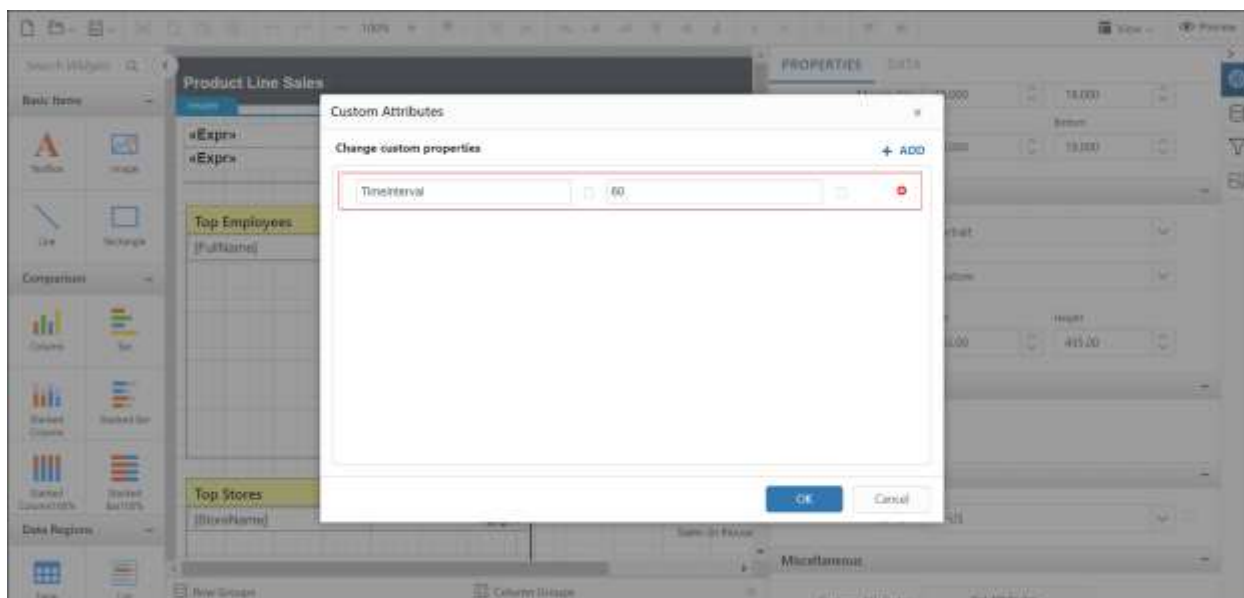
Preview the report and the see time format in datetime parameter.



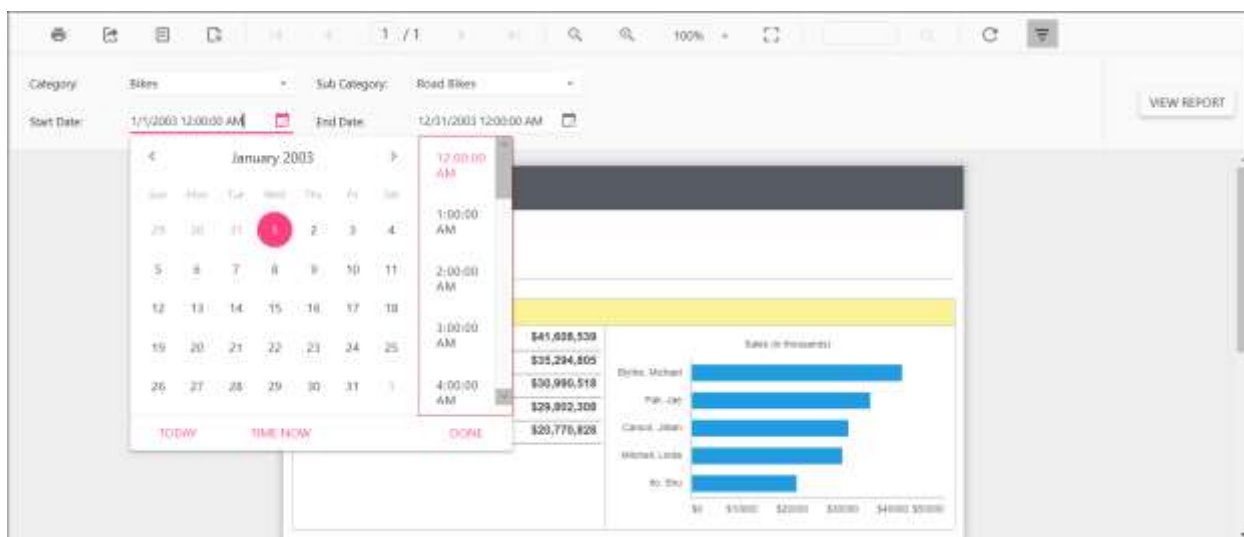
Set time interval in datetime parameter

The `TimeInterval` custom property is used to set the interval between the two adjacent time values in `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeInterval`. By default, the `TimeInterval` value is set as 30.

You can set the `TimeInterval` property value, as shown in the below.



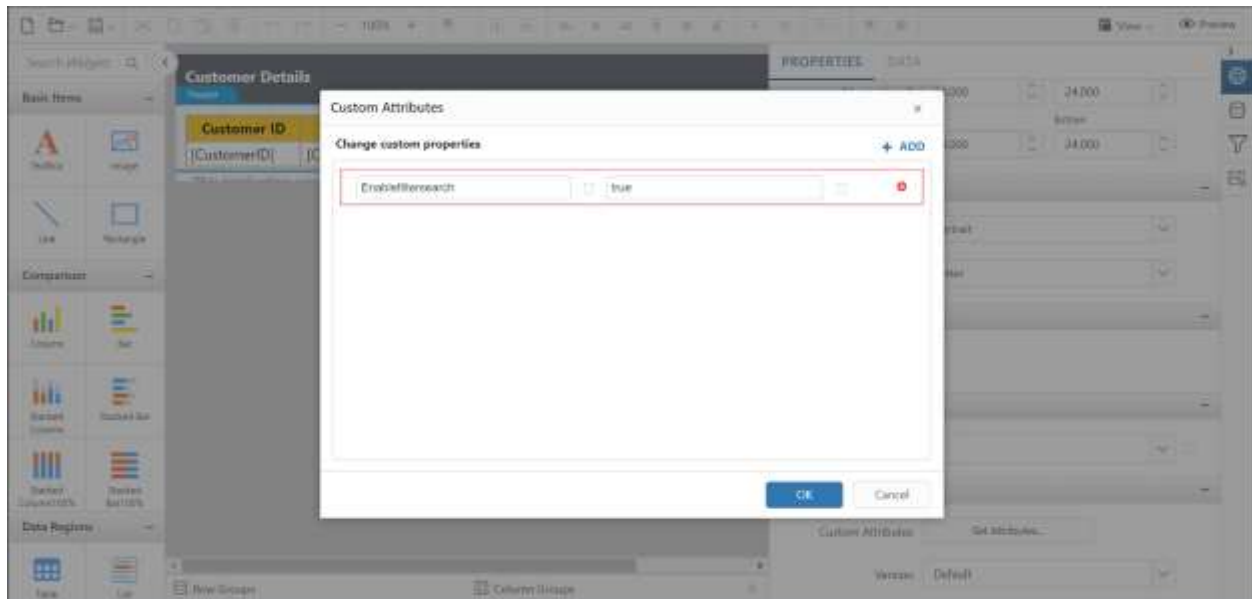
Preview the report and the see time interval in datetime parameter.



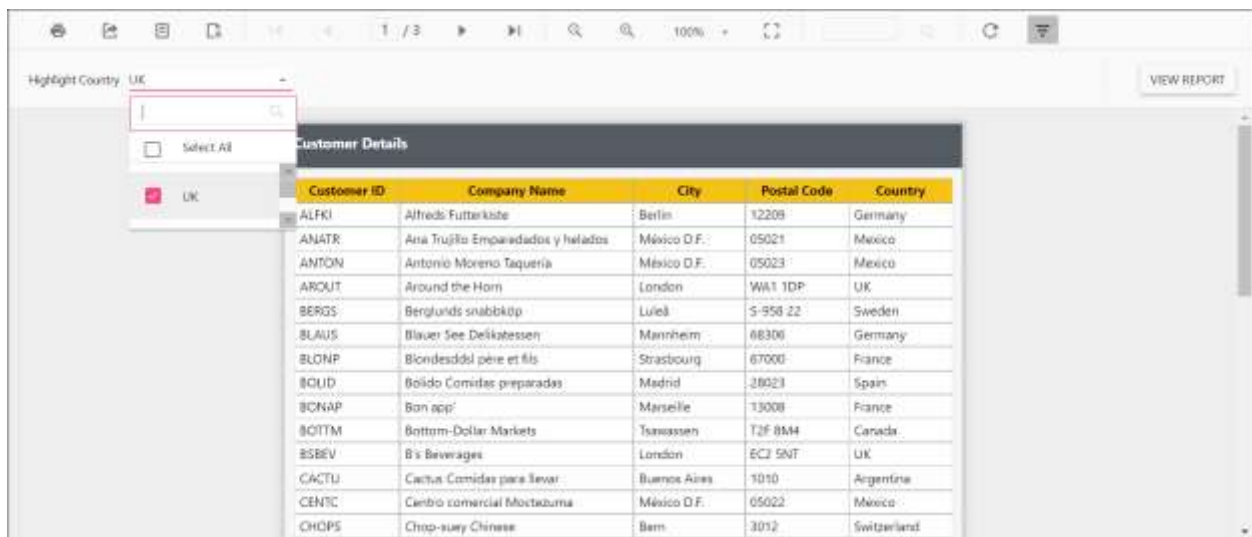
Enable filtering and searching in drop-down report parameter

Setting `EnableFilterSearch` custom property enables search and filtering option in dropdown parameter to easily find the values. The property value should be boolean. By default, the `EnableFilterSearch` value is `false`.

You can set the `EnableFilterSearch` property value, as shown in the below.



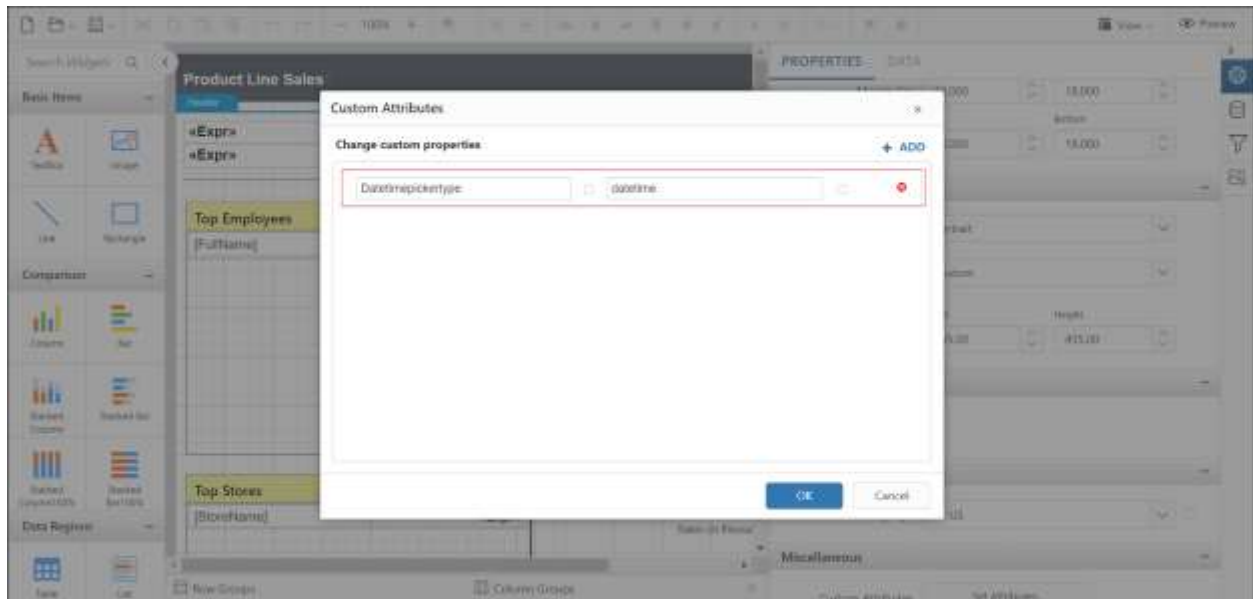
Preview the report and the see search filter in parameter.



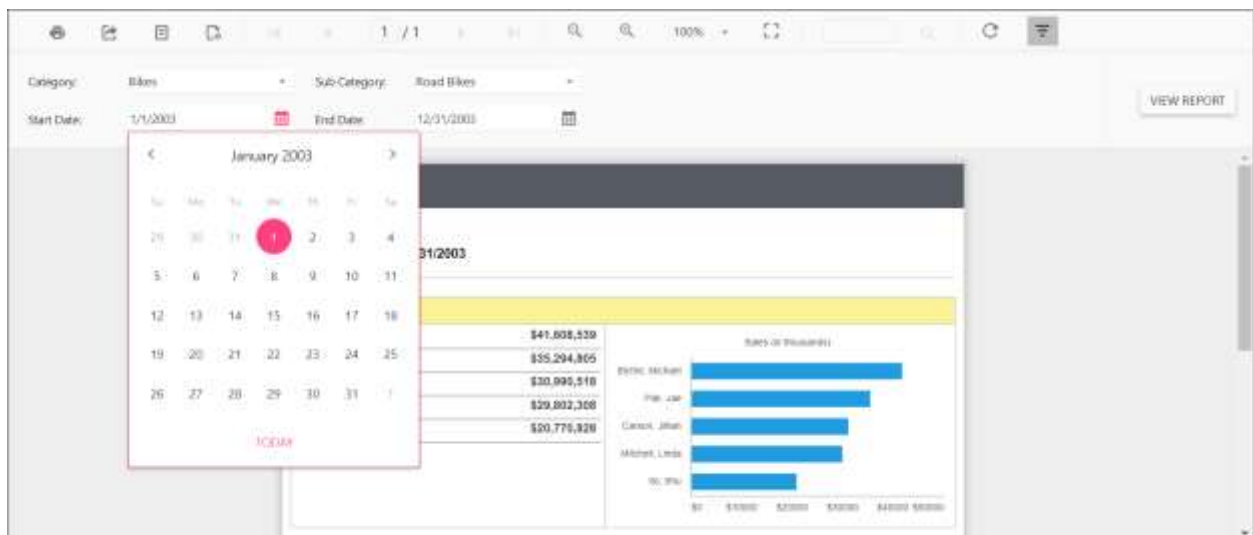
Change date report parameter to display date time picker

You can set `DateTimePickerType` custom property value as `DateTime` to change date parameter item to display `DateTimePicker` for value selection as shown the below.

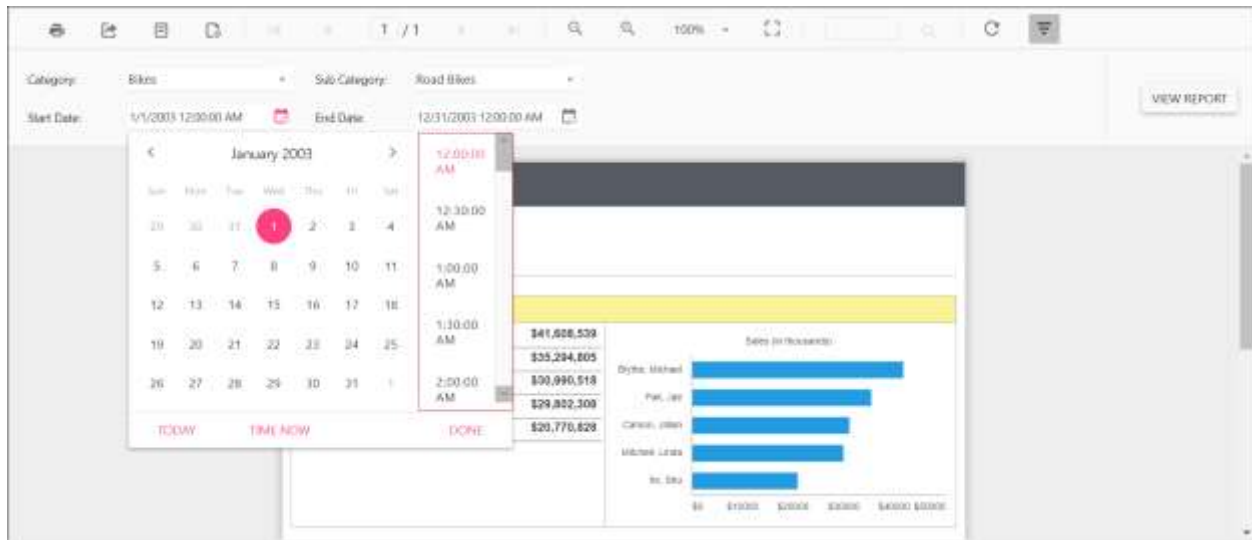
You can set the `DateTimePickerType` property value, as shown in the below.



Before enabling date time picker type, the default value will be displayed as below..



Enable date time picker type and see the time in `DateTimePicker` as in below output.



Export custom properties

This topic explains the list of custom properties that are supported at the report level to control the export behaviour in Angular Report Viewer.

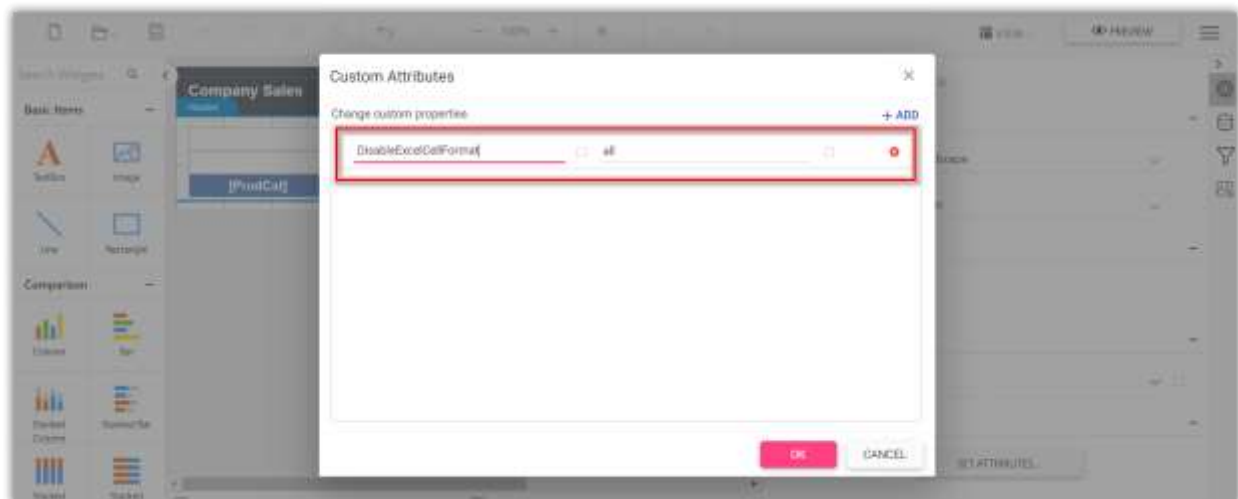
Improve excel export performance

The custom property `DisableExcelCellFormat` helps to improve the excel export performance by ignoring the rendering of report item styles. It allows property value from anyone of the following values.

- **Style** - Disables rendering of the cell styles like padding, background, color, and text style
- **Border** - Disables rendering of the cell border
- **All** - Disables rendering of the cell border, padding, background, color, and text style

Set the property value as **All** to improve maximum excel export performance.

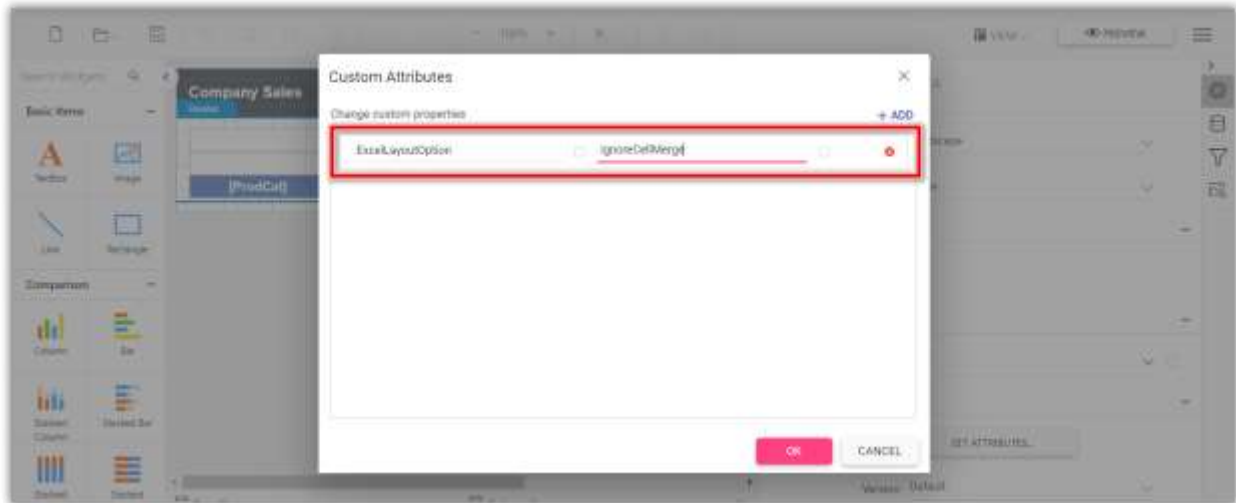
You can set the `DisableExcelCellFormat` custom property as shown below,



The `DisableExcelCellFormat` property must be added to report properties.

Improve excel export readability

Set `ExcelLayoutOption` custom property value as `IgnoreCellMerge` to improve the readability of the excel document by eliminating the tiny columns, rows, and merged cells. You can set the property value, as shown below,



The `ExcelLayoutOption` property must be added to report properties.

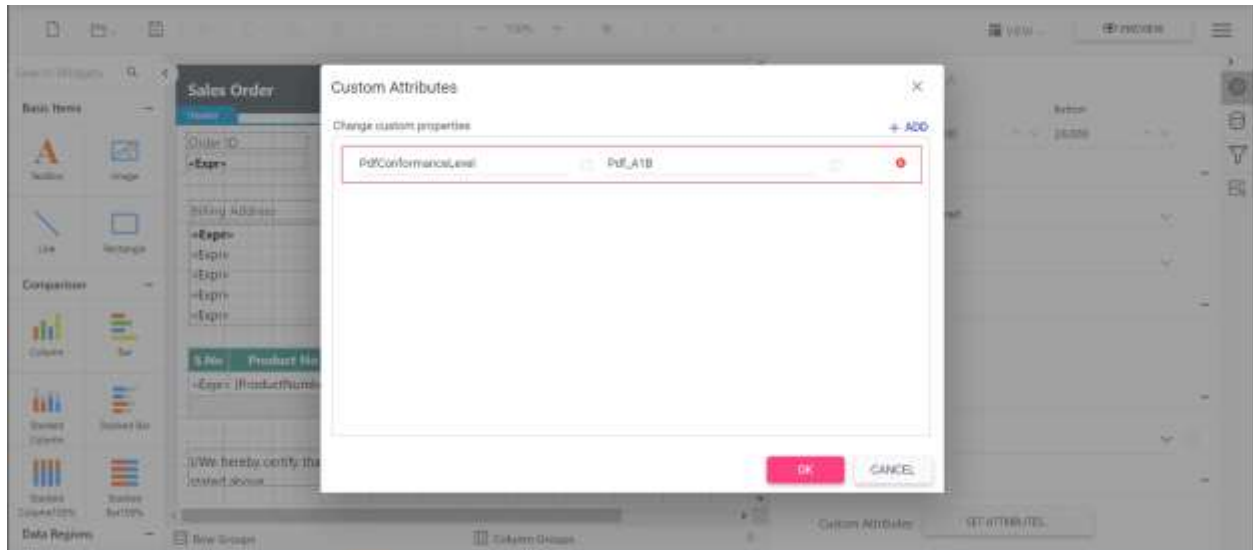
Set pdf conformance level

The `PdfConformanceLevel` allows you set PDF document versions.

You can set anyone of the following PDF conformance option.

- **PdfA1B** - You can create a PdfA1B document by specifying the conformance level as Pdf_A1B through PdfConformanceLevel Enum when creating the new PDF document.
- **PdfA2B** - You can create a PdfA2B document by specifying the conformance level as Pdf_A2B through PdfConformanceLevel Enum when creating the new PDF document
- **PdfA3B** - The PDF/A3B conformance supports the external files as attachment to the PDF document, so you can attach any document format such as Excel, Word, HTML, CAD, or XML files.
- **PdfA1A** - This conformance includes all PdfA1B requirements in addition to the features intended to improve a document's accessibility. PDF/A-1a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2A** - This conformance includes all PDF/A2B requirements in addition to the features intended to improve a document's accessibility. PDF/A-2a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2U** - This conformance includes all PdfA2U requirements, and additionally Unicode mapping for all text in the document.
- **PdfX1A2001** - You can create a PDF/X-1a document by specifying the conformance level as PdfX1A2001 through PdfConformanceLevel Enum when creating the new PDF document.

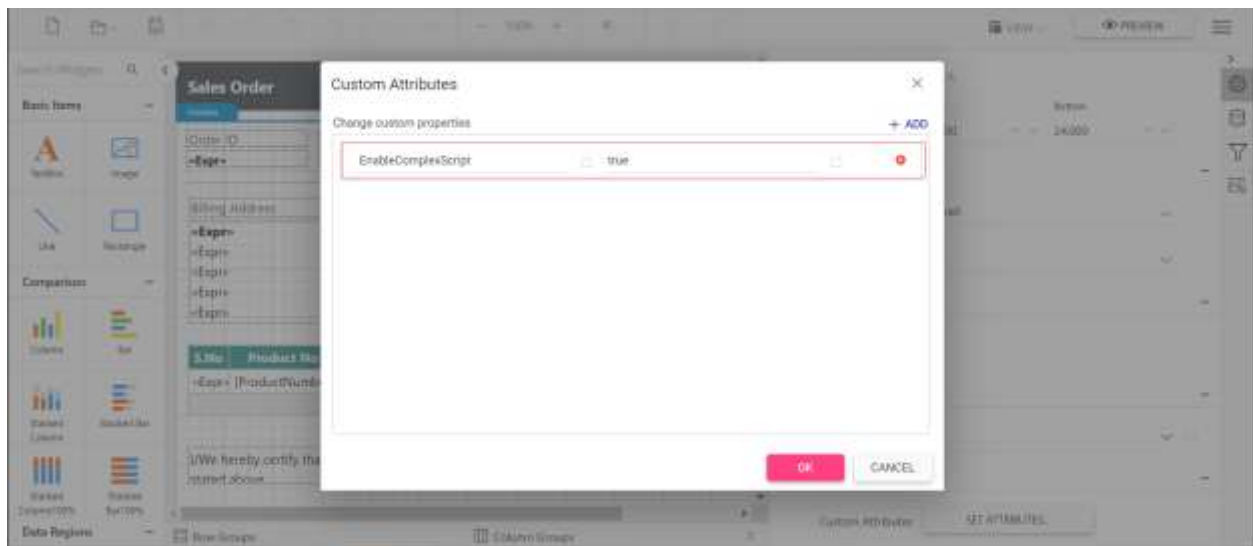
You can set the `PdfConformanceLevel` custom property as shown below,



Export pdf document with complex script

The `EnableComplexScript` custom property allows you to export pdf documents with complex script language texts.

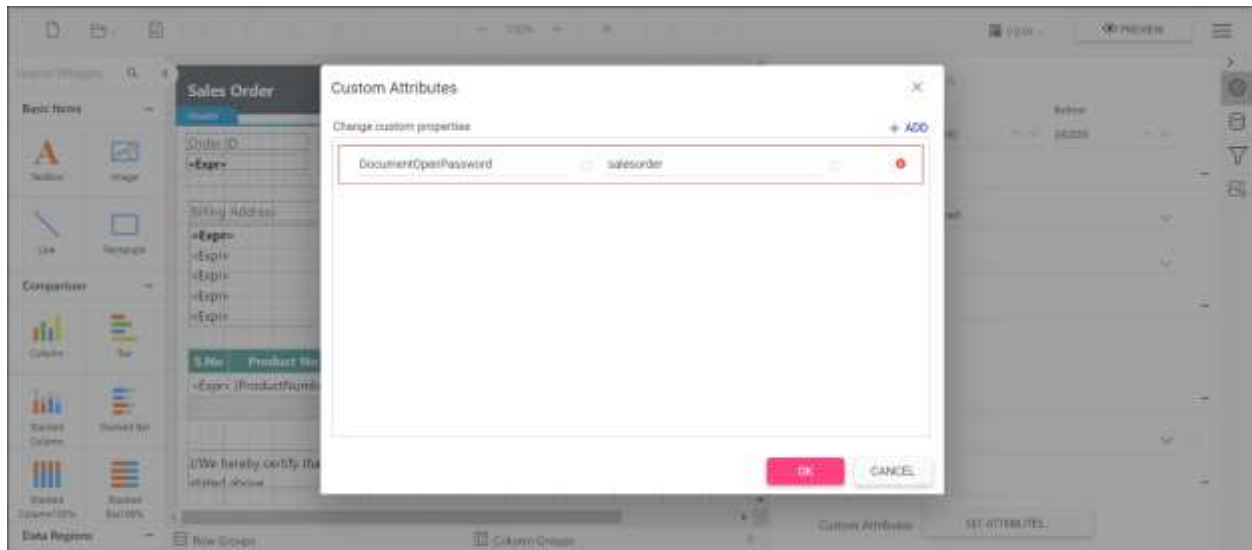
You can set the property value, as shown below,



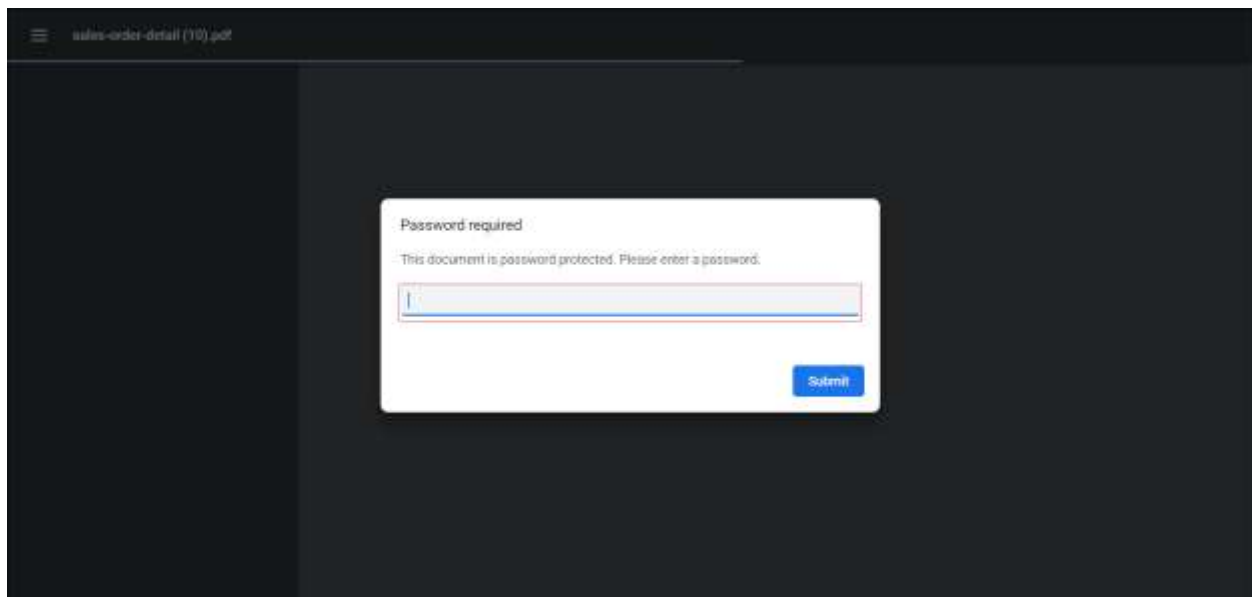
Encrypt and secure export documents

The custom property `DocumentOpenPassword` allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password.

You can set the property value, as shown below,



open the pdf document and see the below output.

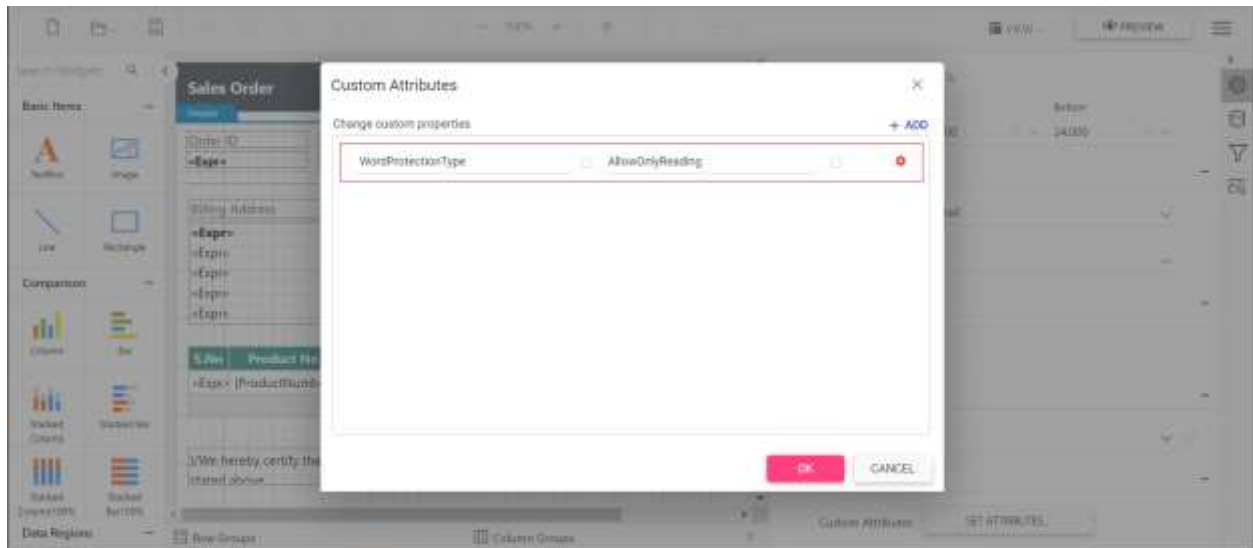


Protecting Word document from editing

The custom property `WordProtectionType` allows you to restrict a Word document from editing either by providing a password or without a password by using following types of protection.

- `AllowOnlyComments`- You can add or modify only the comments in the Word document.
- `AllowOnlyFormFields`- You can modify the form field values in the Word document.
- `AllowOnlyRevisions`- You can accept or reject the revisions in the Word document.
- `AllowOnlyReading`- You can only view the content in the Word document.
- `NoProtection`- You can access or edit the Word document contents as normally.

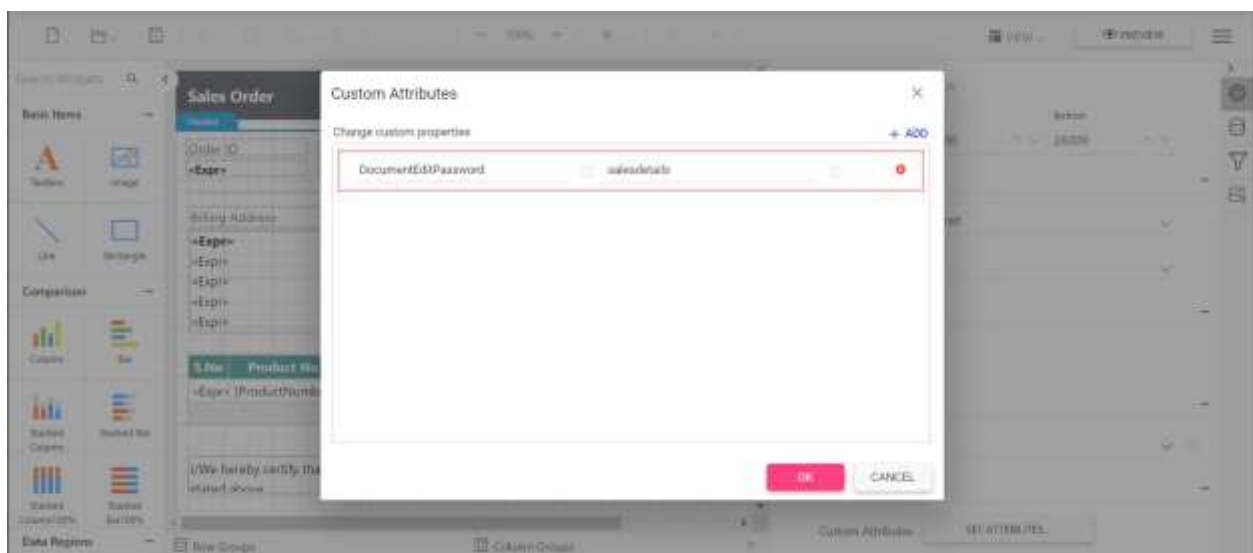
You can set the `WordProtectionType` custom property as shown below,



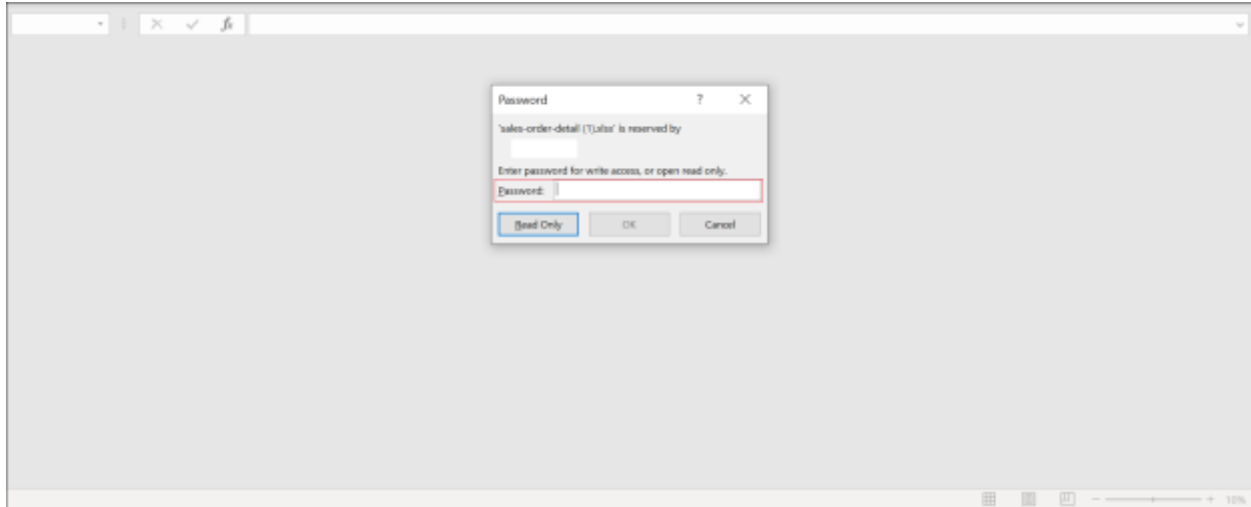
Set excel document edit password

The custom property `DocumentEditPassword` helps to allow specific users permission to modify the workbook data and save changes to the file in the excel document.

You can set the property value, as shown below,



open the excel document and see the below output.

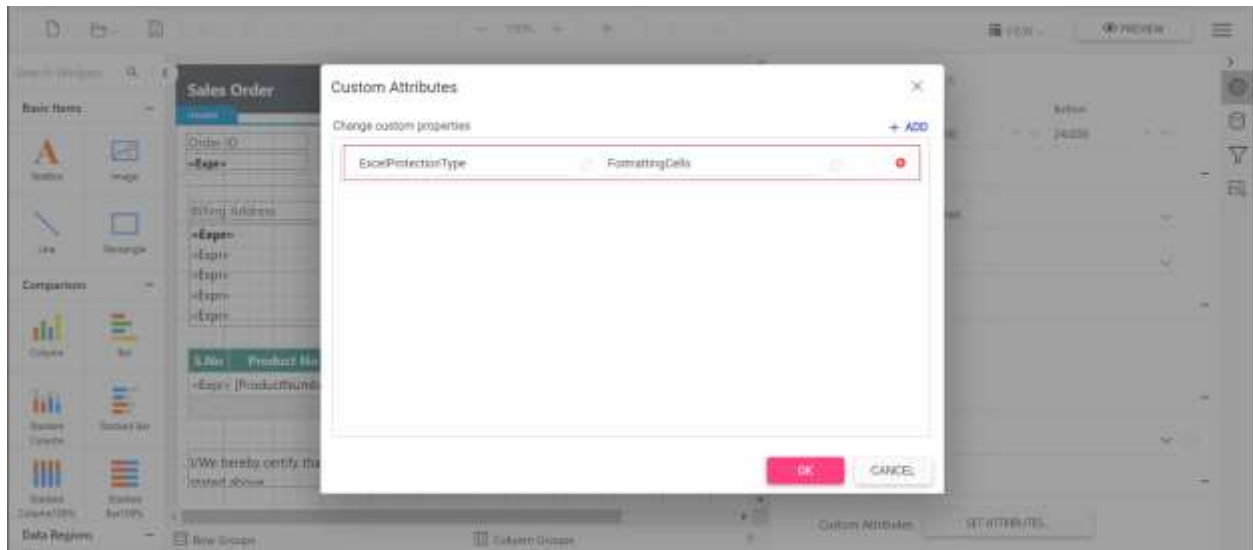


Set excel document protection

The custom property `ExcelProtectionType` allows you to protect the worksheet of excel document either by providing a password or without a password by using following types of protection.

- `None` - Represents no protection in excel sheet
- `Objects` - allows to protect shapes in excel sheet
- `Scenarios` - allows to protect scenarios.
- `FormattingCells` - allows the user to format any cell on a protected worksheet.
- `FormattingColumns` - allows the user to format any column on a protected worksheet.
- `FormattingRows` - allows the user to format any rows on a protected worksheet.
- `InsertingColumns` - allows the user to insert columns on the protected worksheet.
- `InsertingRows` - allows the user to insert rows on the protected worksheet.
- `InsertingHyperlinks` - allows the user to insert hyperlinks on the worksheet.
- `DeletingColumns` - allows the user to delete columns on the protected worksheet, where every cell in the column to be deleted is unlocked.
- `DeletingRows` - allows the user to delete rows on the protected worksheet, where every cell in the row to be deleted is unlocked.
- `LockedCells` - allows to protect locked cells.
- `Sorting` - allows the user to sort on the protected worksheet
- `Filtering` - allows the user to set filters on the protected worksheet. Users can change filter criteria but can not enable or disable an auto filter.
- `UsingPivotTables` - allows the user to use pivot table reports on the protected worksheet.
- `UnLockedCells` - allows to protect the user interface, but not macros.
- `Content` - allows to protect the contents in the excel sheet.
- `All` - allows to protect all type of protection.

You can set the `ExcelProtectionType` custom property as shown below,



Localization of Bold Reports Angular Report Viewer

Localization of Angular Report Viewer allows you to localize the static text such as tooltip, parameter block, and dialog text based on a specific culture. To render the static text with specific culture, refer to the following corresponding culture script files and set culture name to the `locale` property of the Report Viewer.

- `ej.localetexts.fr-FR.min.js`
- `ej.culture.fr-FR.min.js`

Refer this [CDN links for Localization and Culture](#) to get the Localization and Culture scripts for available Culture Code.

- Run the below command, to install the `@boldreports/global` package.

```
`typescript
```

```
npm install @boldreports/global --save
```

```
,
```

- Refer to the `ej.localetexts.fr-FR.min.js` and `ej.culture.fr-FR.min.js` script files from `node_modules` in the `app.module.ts` file .

```
`typescript
```

```
import { NgModule, enableProdMode, ErrorHandler } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RouterModule } from '@angular/router';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
```

```

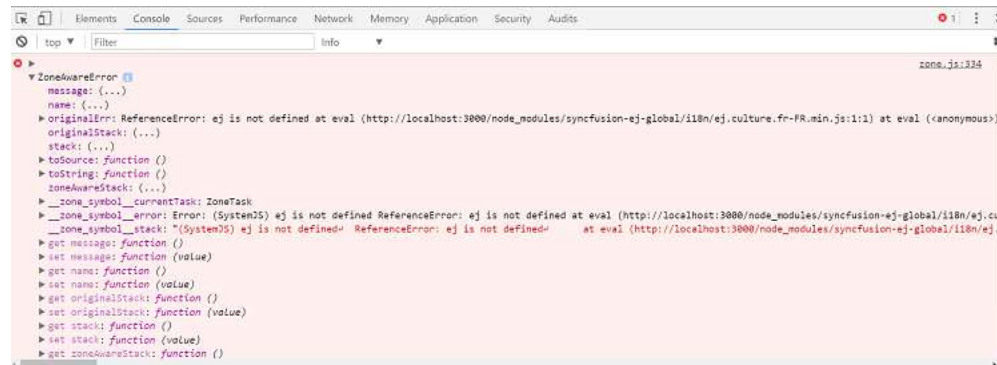
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components';
import '@boldreports/global/i18n/ej.localetexts.fr-FR.min.js';
import '@boldreports/global/i18n/ej.culture.fr-FR.min.js';
import { AppComponent } from './app.component';
import { TextboxComponent } from './textbox/textbox.component';

...

,

```

If you import the culture before the `BoldReportsAngularModule`, you will get the following error in the application. So, you should import the culture after the `@boldreports/angular-reporting-components` package.



Initialize the `locale` property in the `component.ts` and `component.html` pages.

```
`html
```

```
<bold-reportviewer id="reportViewer_Control"
```

```
[reportServiceUrl] = "serviceUrl"
```

```
[processingMode] = "Remote"
```

```
[reportServerUrl] = "serverUrl"
```

```
[reportPath] = "reportPath"
```

```
[locale]='locale'>
```

```
</bold-reportviewer>
```

```
,
```

```
`typescript
```

```
import { Component, ViewChild } from '@angular/core';
```

```
import { BoldReportsAngularModule } from '@boldreports/angular-reporting-components/src/core';
```

```
@Component({
```

```
selector: 'ej-app',
```

```

templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  public serviceUrl: string;
  public reportPath: string;
  public serverUrl: string;
  public locale: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportApi';
    this.reportPath = 'Sales Order Detail.rdl';
    this.locale = "fr-FR";
  }
}

```

Responsive layout rendering of Angular Report Viewer

Report Viewer will adaptively render itself with optimal user interfaces for phone, tablet, or desktop form factors. This helps your application to scale elegantly on all form factors with ease. You can enable responsive layout rendering in Report Viewer by setting `isResponsive` property to true.

```

`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'jsreport-sample';
  public serviceUrl: string;
  public reportPath: string;
  public isResponsive: boolean;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
  }
}

```

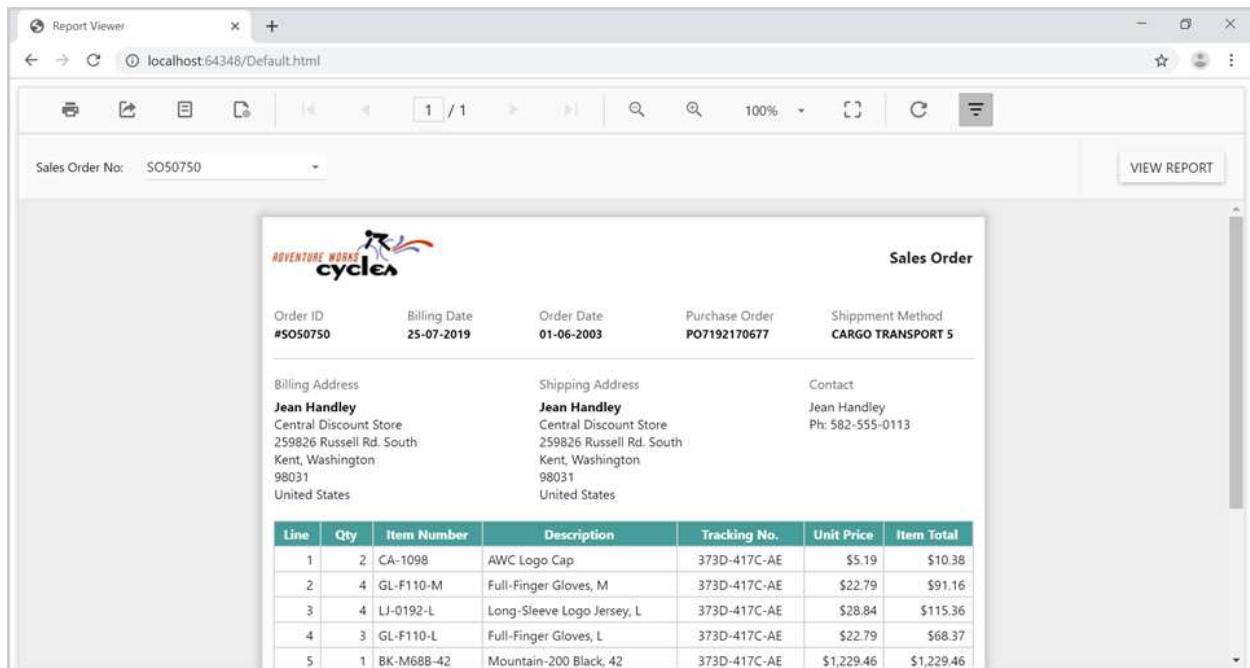
```

this.reportPath = '~/Resources/docs/sales-order-detail.rdl';
this.isResponsive = true;
}
}
`js
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [reportPath] =
"reportPath" [isResponsive] = "isResponsive" style="width: 100%;height: 950px">
</bold-reportviewer>

```

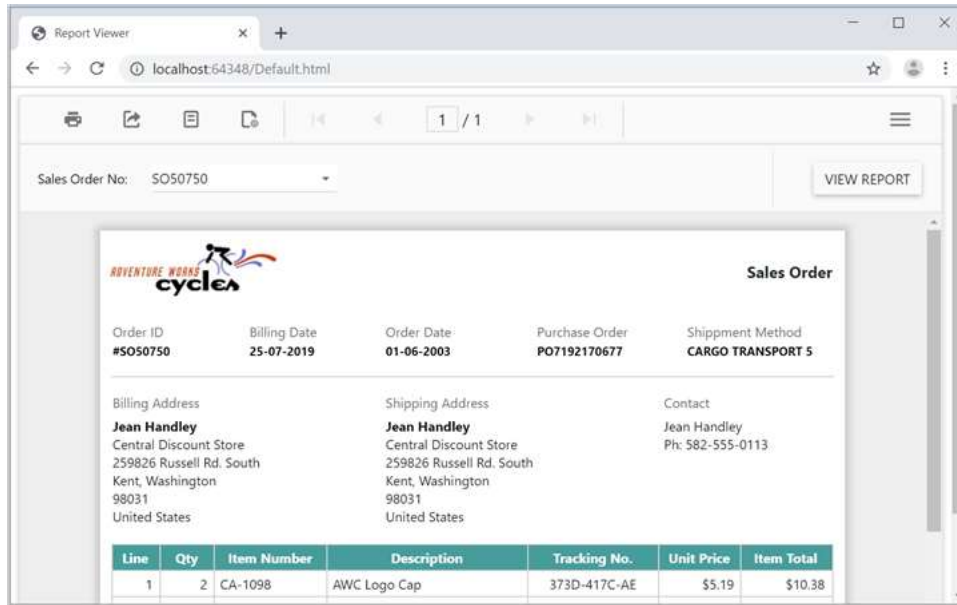
Normal layout

The following output shows the normal layout rendering of Report Viewer tool bar items.



Responsive layout

The following output shows the responsive layout rendering of Report Viewer tool bar items.



Limitations

RDL specification

The Report Viewer control does not support RDL Specification for SQL Server 2000 and SQL Server 2005.

Report layout

- In the Tablix cell split layout process, the entire cell moves to the next page to display the complete cell items, when the table cell width value exceeds the page width.

Expressions

The object function and VB function do not have complete support.

SSRS

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the server. If the report has any data source that uses credentials to connect with the database, then you should specify the data source credentials for each report data source to establish database connection.

Samples and demos

Browse and explore the ready-to-use RDL, RDLC reports, samples, online, and offline demos.

Locally installed reports

You can obtain sample rdl and rdcl files from Bold Reports installed location

```
%localappdata%\Bold Reports\Embedded Reporting\Samples\Common\Data\ReportTemplate.
```

Offline demos

The offline samples are provided in the Bold Reporting Tools setup. For more details, refer to the [Bold Reporting Tools sample deployment](#).

Online demos

You can view the Angular Report Viewer online demo samples from [here](#).

GitHub demo samples

Click [here](#) to view the GitHub Report Viewer demo samples.

Migrate Report Viewer application

In our Bold Reports new assemblies are introduced for both client and server-side to resolve the compatibility problem between Essential Studio Report Viewer versions. It has changes in both Web API service and client-side scripts.

This section provides step-by-step instructions for migrating Report Viewer from Syncfusion Essential Studio release version to Bold Reports version of Angular Report Viewer application:

Server-side migration

1. In the Solution Explorer, right-click the **References** and remove the **Syncfusion.EJ.ReportViewer** assembly reference.
2. Add the assembly from the Bold Reports NuGet package **BoldReports.Web**. To add from NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search for **BoldReports.Web** NuGet package, and then install in your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Web API Controller

1. The **IReportController** interface is moved to **BoldReports.Web.ReportViewer**. Open the Report Viewer Web API Controller file and remove the following using statement.

```
`csharp
using Syncfusion.EJ.ReportViewer;
`
```

2. Add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

Your application is successfully upgraded to the latest version of Report Viewer, and you can run the application with new assemblies.

Report export configuration

Now, the **BoldReports.Web** can export the reports with data visualization components only using web components. It is mandatory to configure the web scripts in Report Viewer Web API controller for

exporting data visualization components such as chart, gauge, and map that are used in report definition. To configure the scripts in Web API, refer to the following steps:

1. Open the Report Viewer Web API controller.
2. Configure the following scripts and styles in `OnInitReportOptions` on Web API controller:
 - `jquery-1.10.2.min.js`
 - `bold.reports.common.min.js`
 - `bold.reports.widgets.min.js`
 - `ej.chart.min.js` - Exports the chart item. Add this script only if your report contains the chart report item.
 - `ej2-base.min.js`, `ej2-data.min.js`, `ej2-pdf-export.min.js`, `ej2-svg-base.min.js`, `ej2-lineargauge.min.js` and `ej2-circulargauge.min.js` - Exports the gauge item. Add this script only if your report contains the gauge report item.
 - `ej2-maps.min.js` - Exports the map item. Add this script only if your report contains the map report item.
 - `bold.report-viewer.min.js`
3. Replace the following codes in Report Viewer Web API controller.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    var resourcesPath = System.Web.Hosting.HostingEnvironment.MapPath("~/Scripts");
    reportOption.ReportModel.ExportResources.Scripts = new List<string>
    {
        resourcesPath + @"\"bold-reports\\common\\bold.reports.common.min.js",
        resourcesPath + @"\"bold-reports\\common\\bold.reports.widgets.min.js",
        //Gauge component scripts
        resourcesPath + @"\"bold-reports\\common\\ej2-base.min.js",
        resourcesPath + @"\"bold-reports\\common\\ej2-data.min.js",
        resourcesPath + @"\"bold-reports\\common\\ej2-pdf-export.min.js",
        resourcesPath + @"\"bold-reports\\common\\ej2-svg-base.min.js",
        resourcesPath + @"\"bold-reports\\data-visualization\\ej2-lineargauge.min.js",
        resourcesPath + @"\"bold-reports\\data-visualization\\ej2-circulargauge.min.js",
        //Map component script
        resourcesPath + @"\"bold-reports\\data-visualization\\ej2-maps.min.js",
        //Report Viewer Script
        resourcesPath + @"\"bold-reports\\data-visualization\\ej.chart.min.js",
```

```
resourcesPath + @"\"bold-reports\"bold.report-viewer.min.js"
};
reportOption.ReportModel.ExportResources.DependentScripts = new List<string>
{
resourcesPath + @"\"jquery-1.7.1.min.js"
};
}
`
```

The data visualization components will not export without the above script configurations.

NuGet Packages for Angular

Refer to the following steps to configure Bold Reporting NuGet packages for Angular Web API application.

Configure NuGet feed URL

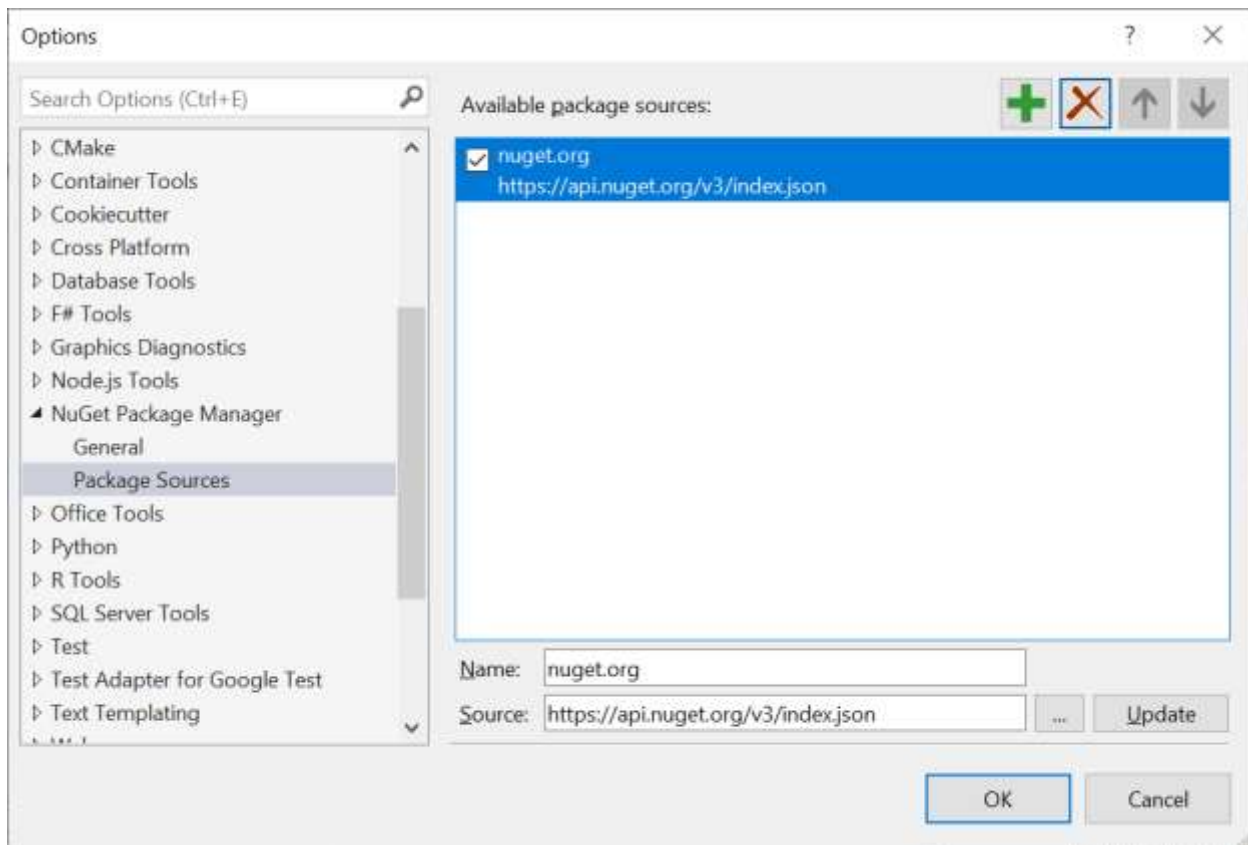
Online NuGet feed URL

The Bold Reporting NuGet packages are published in **Nuget.org**. To configure the online packages, use the following steps:

1. Open Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager** and select **Package Sources**.
4. Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: NuGet.org

Source: https://api.nuget.org/v3/index.json



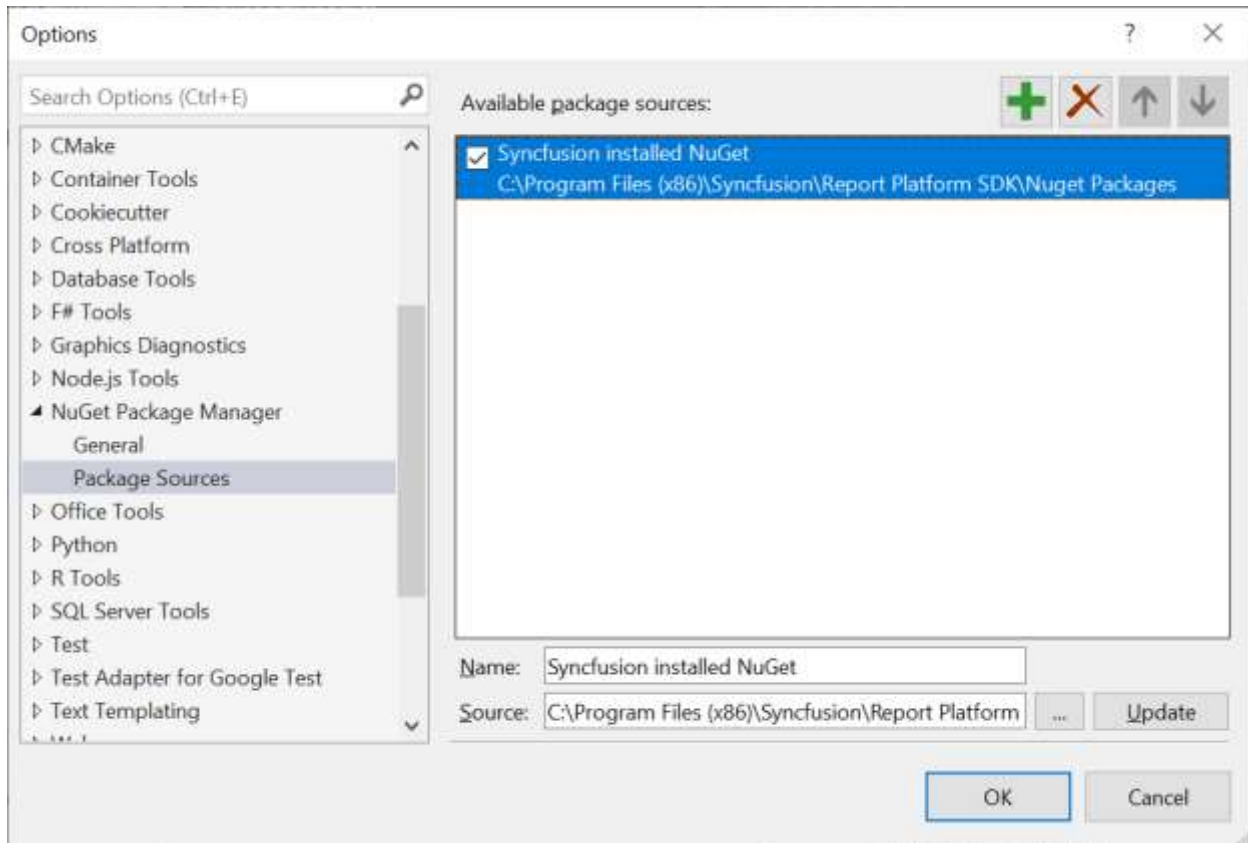
Offline NuGet feed URL

Bold Reports NuGet packages are shipped into our build. To configure the packages from Bold Reports installed location, use the following steps:

1. Open your Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager**, and then select **Package Sources**.
4. Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: Bold Reports installed NuGet

Source: {System Drive}:\Program Files (x86)\Bold Reports\Reporting Tools\Nuget Packages.



The system drive varies based on the installed location in your machine.

Installing NuGet packages

Install using NuGet Package Manager

The NuGet Package Manager can be used to search and install NuGet packages in Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution**. Alternatively, right-click the project/solution in Solution Explorer tab, and choose **Manage NuGet Packages**.
2. By default, the **NuGet.org** package is selected in the **Package source** drop-down. Select package source, search for the packages **BoldReports.Web**, and then click **Install** button.

Install using Package Manager Console

To install the Reporting component using the Package Manager Console as NuGet packages,

1. On the **Tools** menu, select **NuGet Package Manager**, and then click **Package Manager Console**.
2. Run the following NuGet installation commands:

```
`cmd
```

install specified package in default project

```
Install-Package <Package Name>
```

install specified package in default project with specified package source

Install-Package <Package Name> -Source <Source Location>

install specified package in specified project

Install-Package <Package Name> -ProjectName <Project Name>

`

For example:

`cmd

install specified package in default project

Install-Package BoldReports.Web

install specified package in default project with specified Package Source

Install-Package BoldReports.Web -Source "https://api.nuget.org/v3/index.json"

install specified package in specified project

Install-Package BoldReports.Web -ProjectName BoldReportsApplication

`

Upgrading NuGet packages

Upgrading using NuGet Package Manager

NuGet packages can be updated to their specific version or latest version available in the Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution....** Alternatively, right-click the project/solution in the Solution Explorer tab, and then choose **Manage NuGet Packages**.
2. Select the **Updates** tab to see the packages available for update from the desired package sources, select the required packages and specific version from the drop-down, and then click the **Update** button.

Upgrading using Package Manger Console

To update the installed Bold Reports NuGet packages using the Package Manager Console:

1. On the **Tools** menu, select **NuGet Package Manager**, and then select **Package Manager Console**.
2. Run the following NuGet installation commands:

`cmd

Update specific NuGet package in default project

Update-Package <Package Name>

Update all the packages in default project

Update-Package

Update specified package in default project with specified package source

Update-Package <Package Name> -Source <Source Location>

Update specified package in specified project

Update-Package <Package Name> -ProjectName <Project Name>

,

For example:

`cmd

Update specified Bold Reports NuGet package

Update-Package BoldReports.Web

Update specified package in default project with specified Package Source

Update-Package BoldReports.Web -Source "https://api.nuget.org/v3/index.json"

Update specified package in specified project

Update-Package BoldReports.Web -ProjectName BoldReportsApplication

,

Upgrading using NuGet CLI

Using the NuGet CLI, all the NuGet packages in the project can be updated to the available latest version:

1. Download the latest [NuGet CLI](#).

To update the existing `nuget.exe` to latest version use the following command:

`cmd

nuget update -self

,

2. Open the downloaded executable location in the command window. Run the following "update commands" to update the Bold Reports NuGet packages.

`cmd

update all NuGet packages from config file

nuget update <configPath> [options]

update all NuGet packages from specified Packages Source

nuget update -Source <Source Location> [optional]

,

`configPath` is optional. It identifies the `package.config` or solutions file lists the packages utilized in the project.

For example:

```
`cmd
```

Update all NuGet packages from config file

```
nuget update "C:\Users\BoldReportsApplication\package.config"
```

Update all NuGet packages from specified Packages Source

```
nuget update -Source "https://api.nuget.org/v3/index.json"
```

,

The Update command is not working as expected in Mono (Mac and Linux) and projects using PackageReference format.

Deployment

This topic explains about simplest production-ready deployment of your Angular Report Viewer application to a remote server by creating a production build.

To create a production build run the following command and copy the output directory to a web server.

```
`typescript
```

```
node_modules\.bin\ng build --prod
```

,

Please refer to the [angular deployment](#) documentation to know more about deploying your Angular application.

If you get error `FATAL ERROR: CALLANDRETRYLAST Allocation failed - JavaScript heap out of memory` add `--max-old-space-size=xx` space in `ngc.cmd` and `ng.cmd` file from `node_modules\.bin` folder.

Modify `ngc.cmd`

```
`js
```

```
@IF EXIST "%~dp0\node.exe" (
```

```
"%~dp0\node.exe" --max-old-space-size=8192 "%~dp0\..\@angular\compiler-cli\src\main.js" %*
```

```
) ELSE (
```

```
@SETLOCAL
```

```
@SET PATHEXT=%PATHEXT;:.JS;=;%
```

```
node --maxoldspace_size=8192 "%~dp0\..\@angular\compiler-cli\src\main.js" %*
```

```
)
```

```
,
```

Modify `ng.cmd`

```
`js
```

```
@IF EXIST "%~dp0\node.exe" (
```

```
"%~dp0\node.exe" --maxoldspace_size=8192 "%~dp0\..\@angular\cli\bin\ng" %*
```

```
) ELSE (
```

```
@SETLOCAL
```

```
@SET PATHEXT=%PATHEXT;:.JS;=;%
```

```
node --maxoldspace_size=8192 "%~dp0\..\@angular\cli\bin\ng" %*
```

```
)
```

```
,
```

Frequently asked questions

This section helps to get the answer for the frequently asked questions in Bold Reports Angular Report Viewer.

1. [How can improve the performance and handle the large amounts of data with Report Viewer?](#)
2. [Is Report Viewer compatible with latest version of JQuery library?](#)
3. [Is the back action from drillthrough report will load the report again with Report Viewer?](#)
4. [Is possible to change the culture of the date time parameter?](#)
5. [Is it possible to hide report parameters?](#)
6. [Is it possible to hide the export options in Report Viewer?](#)
7. [Is it possible to load reports providing parameter values at runtime?](#)

Can you use the Report Viewer component in Angular 9 application

Yes, you can use the Report Viewer component in Angular 9 application.

Is possible to change the culture of the date time parameter

Yes, we can change the culture of the date time parameter for Report Viewer by changing the locale. You can make use the following reference to change the locale of Report Viewer.

[ReportViewer Localization](#)

In Report Viewer, we could not change the culture of specific parameter or UI. So, we have to achieve the requirements only by changing the culture.

Is it possible to hide the parameters in Report Viewer

Yes, it is possible to hide the parameters in Report Viewer. On hiding the parameters, the users will not be able to have the parameter block in the Report Viewer. Refer to this [Hide parameter block and toolbar items](#) section.

Is it possible to hide the export options in Report Viewer

Yes, it is possible to hide the export options in Report Viewer. The Report Viewer provides the `exportOptions` property to show or hide the default export types available in the component. Refer to this [Decide or Hide the export options](#) section.

Is it possible to load reports providing parameter values at runtime

Yes, it is possible to load reports with application inputs as parameters in Report Viewer. You need to provide the input values to the Report Viewer from client side at runtime using the `parameters` property, which accepts JSON array values. Refer to this [Set parameter at client](#) section.

Angular Report Viewer Reporting Service

The Angular Report Viewer requires a Web API service to process the report files. The following topics explain how to create reporting Web API service to preview the reports.

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

IReportController

The `IReportController` interface has the declaration of action methods that is defined in the Web API Controller for processing the RDL, RDLC, and SSRS report and handling the request from the Report Viewer control. The `IReportController` has the following action methods declaration.

Methods | Description

`GetResource` | Returns the report resource to the requested key.

`ProcessReport` | Processes the report request and returns the result.

``csharp`

```
public class ReportsController: ApiController, IReportController
```

```
{
```

```
/// <summary>
```

```
/// Action (HttpGet) method for getting resource for report.
```

```
/// </summary>
```

```
/// <param name="key">The unique key to get the required resource.</param>
```

```
/// <param name="resourceType">The type of the requested resource.</param>
```

```
/// <param name="isPrinting">If set to <see langword="true"/>, then the resource is generated for printing.</param>
```

```
/// <returns>The object data.</returns>
```

```
public object GetResource(string key, string resourceType, bool isPrinting)
{
    //Returns the report resource for the requested key.
    return ReportHelper.GetResource(key, resourceType, isPrinting);
}

/// <summary>
/// Report Initialization method that is triggered when report begin processed.
/// </summary>
/// <param name="reportOptions">The ReportViewer options.</param>
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOptions)
{
    //You can update report options here
}

/// <summary>
/// Report loaded method that is triggered when report and sub report begins to be loaded.
/// </summary>
/// <param name="reportOptions">The ReportViewer options.</param>
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOptions)
{
    //You can update report options here
}

/// <summary>
/// Action (HttpPost) method for posting the request for report process.
/// </summary>
/// <param name="jsonData">The JSON data posted for processing report.</param>
/// <returns>The object data.</returns>
public object PostReportAction(Dictionary < string, object > jsonData)
{
    //Processes the report request and returns the result.
    return ReportHelper.ProcessReport(jsonData, this);
}
```



```
}  
,
```

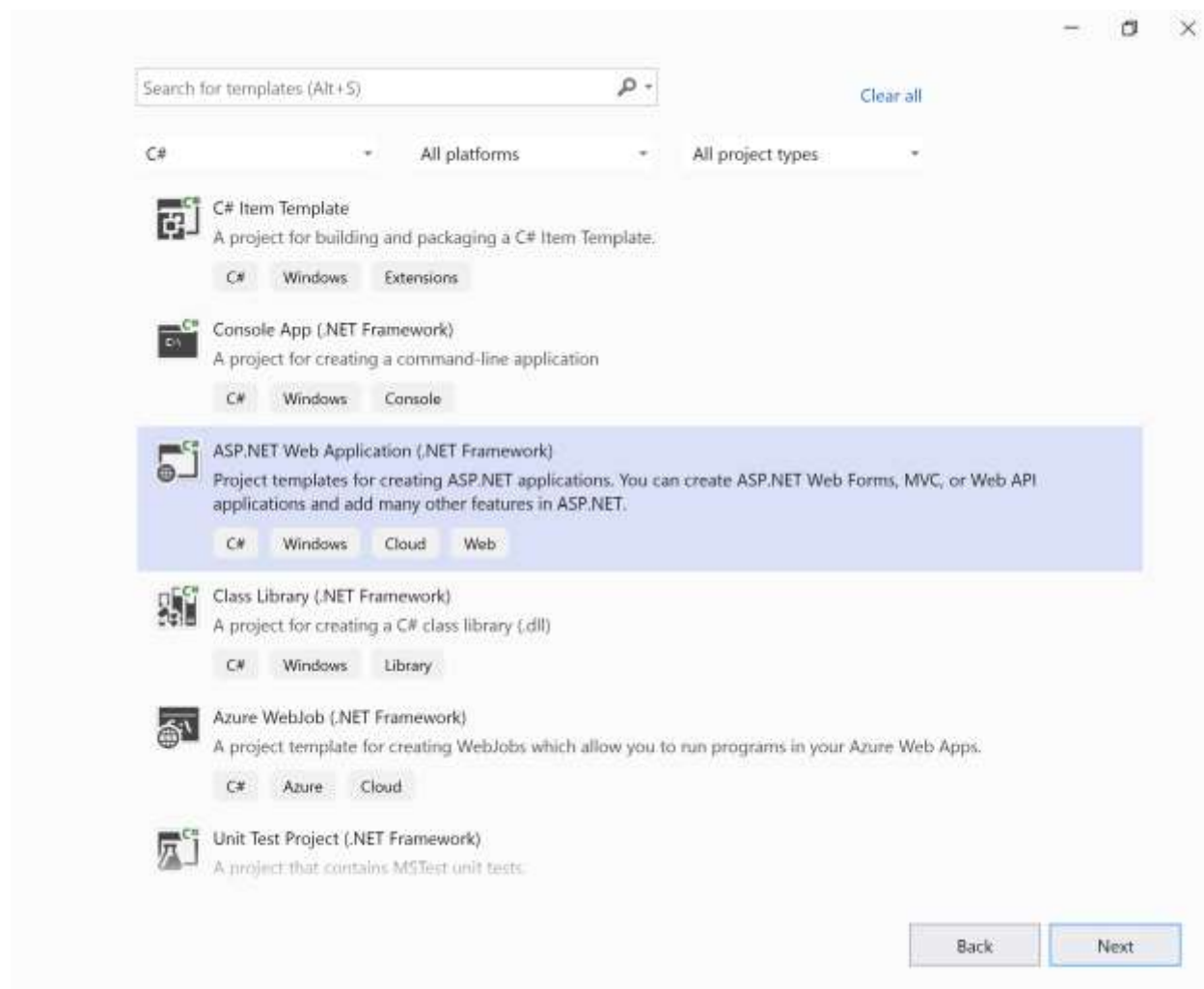
Create ASP.NET Web API service

In this section, you will learn how to create a Web API Service for Report Viewer using the new ASP.NET Empty Web Application template.

To get start quickly with Web API Service for Report Viewer, you can check on this video:

youtube: https://youtu.be/Qbho_8cnhJ8

1. Start Visual Studio 2022 and click **Create new project**.
2. Select **ASP.NET Web Application (.NET Framework)**.



3. Change the application name, and then select the required **.NET Framework** in the drop-down.

Configure your new project

ASP.NET Web Application (.NET Framework) **CR** Windows Cloud Web

Project name
ASP.NET Web API service

Location
C:\

Solution name
ASP.NET Web API service

☐ Place solution and project in the same directory

Framework
.NET Framework 4.5.1

Back Create

4. Choose **Empty, Web API** and then click **OK**. Now, the Web application project is created with default ASP.NET Web template.

Create a new ASP.NET Web Application

Empty
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Web Forms
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

MVC
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Web API
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Single Page Application
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
None

Add folders & core references

- ☐ Web Forms
- ☐ MVC
- ☒ Web API

Advanced

- ☒ Configure for HTTPS
- ☐ Docker support
(Requires Docker Desktop)
- ☐ Also create a project for unit tests

ASP.NET Web API service Tests

Back Create

Configure Report Viewer Web API

1. Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

2. Search for **BoldReports.Web** NuGet packages, and install them in your Web application.

Package | Purpose

PostReportAction | Action (HttpPost) method for posting the request in report process.

OnInitReportOptions | Report initialization method that occurs when the report is about to be processed.

OnReportLoaded | Report loaded method that occurs when the report and sub report start loading.

GetResource | Action (HttpGet) method to get resource for the report.

ReportHelper

The class **ReportHelper** contains helper methods that help to process a Post or Get request from the Report Viewer control and return the response to the Report Viewer control. It has the following methods:

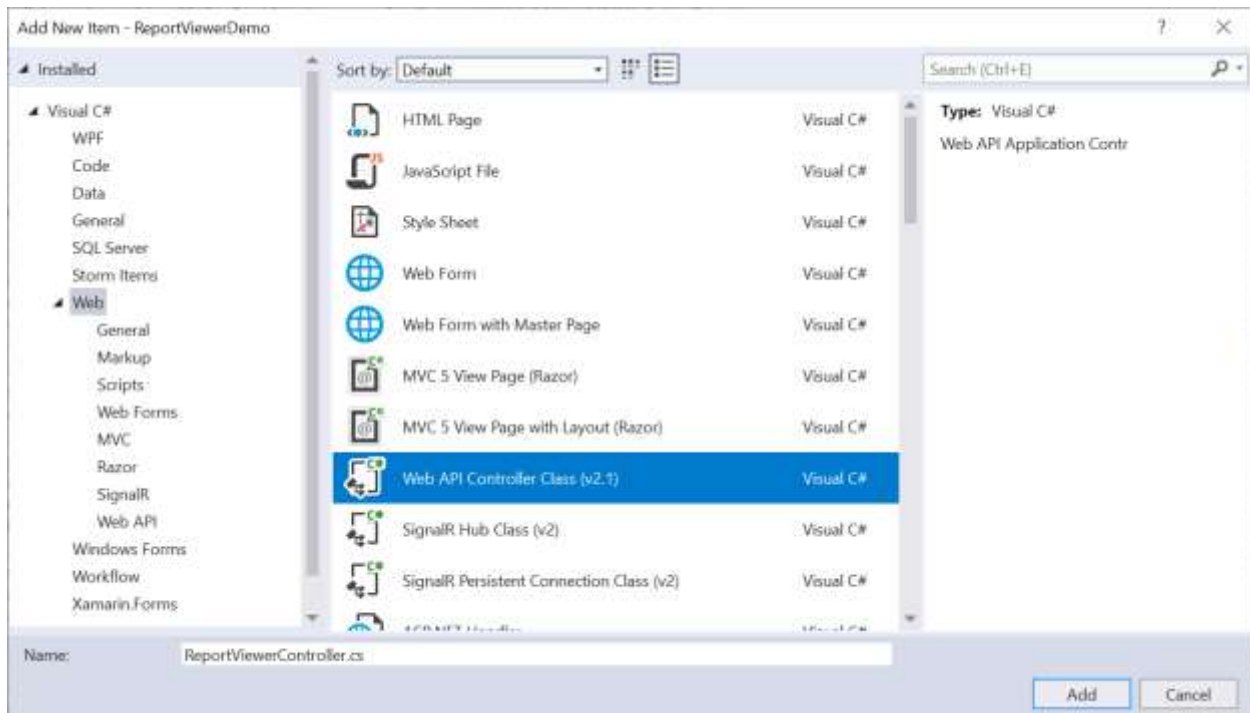
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click **Controller** folder in your project and select **Add > New Item** from the context menu.
2. Select Web API Controller Class from the listed templates and name it as **ReportViewerController.cs**



3. Click **Add**.

While adding Web API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and implement its methods (replace the following code in newly created Web API controller).

```
`csharp
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
}
```

```
[System.Web.Http.ActionName("GetResource")]
[AcceptVerbs("GET")]
public object GetResource(string key, string resourcetype, bool isPrint)
{
    return ReportHelper.GetResource(key, resourcetype, isPrint);
}
// Method that will be called when initialize the report options before start processing the report
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    // You can update report options here
}
// Method that will be called when reported is loaded
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    // You can update report options here
}
}
```

Add routing information

1. To configure routing to include an action name in the URI, open the **WebApiConfig.cs** file and change the **routeTemplate** in the **Register** method as follows,

```
`csharp
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services
        // Web API routes
        config.MapHttpAttributeRoutes();
        config.Routes.MapHttpRoute(
```

```

name: "DefaultApi",
routeTemplate: "api/{controller}/{action}/{id}",
defaults: new { id = RouteParameter.Optional }
);
}
}
,

```

2. Compile and run the Web API service application.

Enable Cross-Origin Requests

Browser security prevents Report Viewer from making requests to your Web API Service when both runs in a different domain. To allow access to your Web API service from a different domain, you must enable cross-origin requests.

1. Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, go to **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for `Microsoft.AspNet.WebApi.Cors` NuGet packages, and install them in your Web API application.
3. Call `EnableCors` in `WebApiConfig` to add CORS services to the **Register** method. Replace the following code to allow any origin requests.

```

`csharp
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Add Enable Cors
        config.EnableCors();

        // Web API configuration and services

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{action}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}

```

```

}
}
`

```

4. To specify the CORS policy for API controller, add the `[EnableCors]` attribute to the controller class. Specify the policy name.

```

`csharp
[EnableCors(origins: "", headers: "", methods: "*")]
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
    [System.Web.Http.ActionName("GetResource")]
    [AcceptVerbs("GET")]
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }
    // Method that will be called when initialize the report options before start processing the report
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
    // Method that will be called when reported is loaded
    [NonAction]
    public void OnReportLoaded(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
}

```

```
}  
,
```

Create ASP.NET Core Web API Service

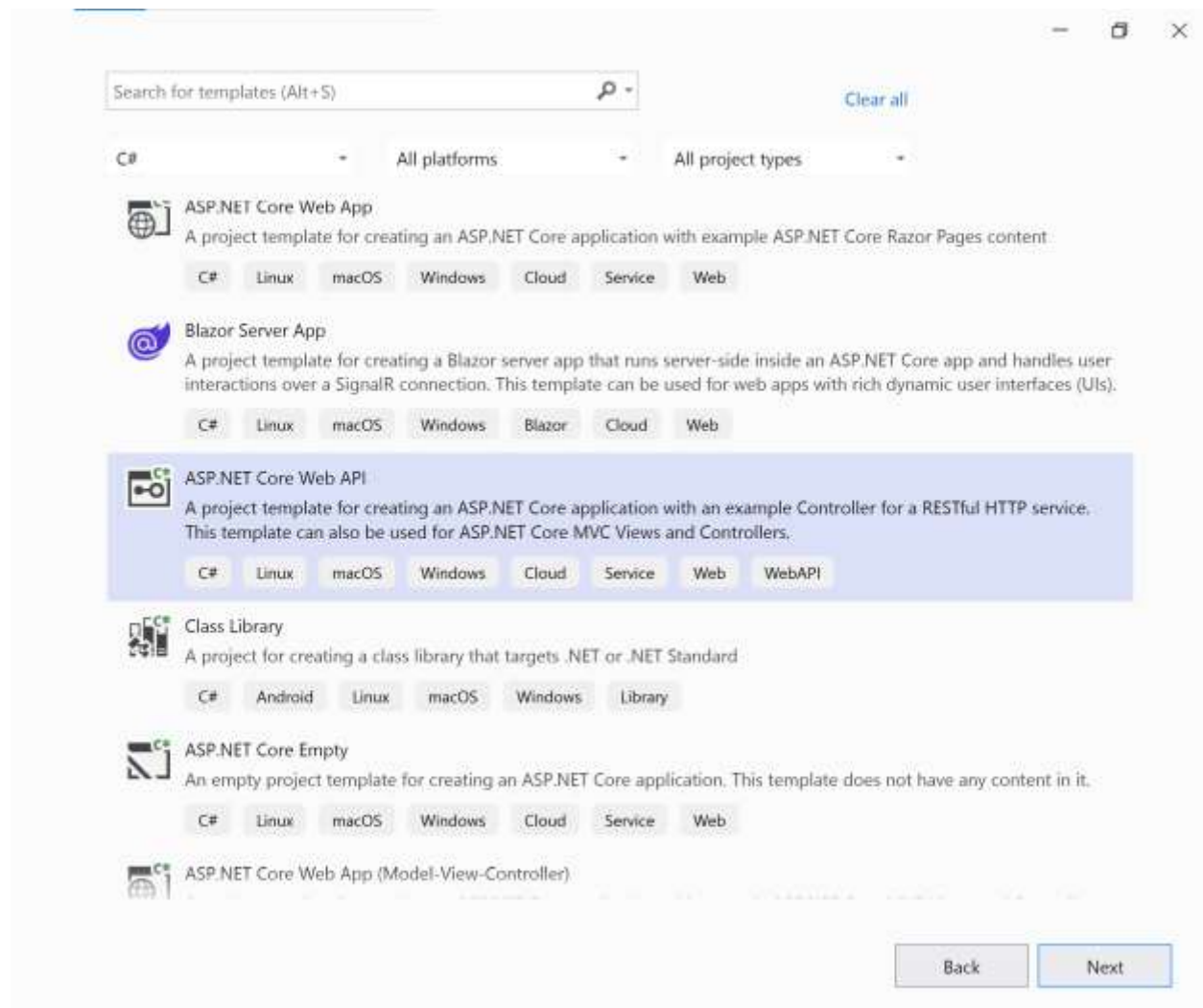
To create an ASP.NET Core Web API for Report Viewer using a new ASP.NET Core Web Application template, follow these steps:

Bold Reports ASP.NET Core supports from **.NET Core 2.1** only. So, choose the .NET Core version **ASP.NET Core 2.1** or higher versions for Viewer API creation.

To get start quickly with ASP.NET Core Web API for Report Viewer, you can check on this video:

youtube: <https://youtu.be/b1yZGAeWoHQ>

1. Start Visual Studio 2022 and click **Create new project**.
2. Choose **ASP.NET Core Web API**, and then click **Next**.



3. Change the project name, and then click **Next**.

4. In the dropdown for the **ASP.NET Core version**, choose **ASP.NET Core 6.0**, then click **Create**.

Additional information

ASP.NET Core Web API **CR** Linux macOS Windows Cloud Service Web WebAPI

Framework
 .NET 6.0 (Long Term Support)

Authentication type
 None

☒ Configure for HTTPS

☐ Enable Docker

Docker OS
 Linux

☒ Use controllers (unchecked to use minimal APIs)

☒ Enable OpenAPI support

☐ Do not use top-level statements

Back **Create**

If you need to use Bold Reports with ASP.NET Core on Linux or macOS, then refer to this [Can Bold Reports be used with ASP.NET Core on Linux and macOS](#) section.

List of dependency Libraries

The Web API service configuration requires the following reporting server-side packages. Right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages** and then search for **BoldReports.Net.Core** package, and install to the application. The following provides detail of the packages and its usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Supports for exporting the report to PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the packages **Syncfusion.Pdf.Net.Core** , **Syncfusion.DocIO.Net.Core** and **Syncfusion.XlsIO.Net.Core**.

Syncfusion.Pdf.Net.Core | Supports for exporting the report to a PDF.

Syncfusion.DocIO.Net.Core | Supports for exporting the report to a Word.

Syncfusion.XlsIO.Net.Core | Supports for exporting the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serialize and deserialize the data for report viewer. It is a mandatory package for the report viewer, and the package version should be higher of 10.0.1 for NET Core 2.0 and others should be higher of 9.0.1.

System.Data.SqlClient | This is an optional package for the report viewer. It should be referred in project when renders the RDL report and which contains the SQL Server and SQL Azure data source. Also, the package version should be higher of 4.1.0.

Configure Web API

The interface **IReportController** has declaration of action methods that are defined in the Web API Controller for processing the RDL, RDLC, and SSRS reports and for handling request from the Report Viewer control. The IReportController has the following action methods declaration:

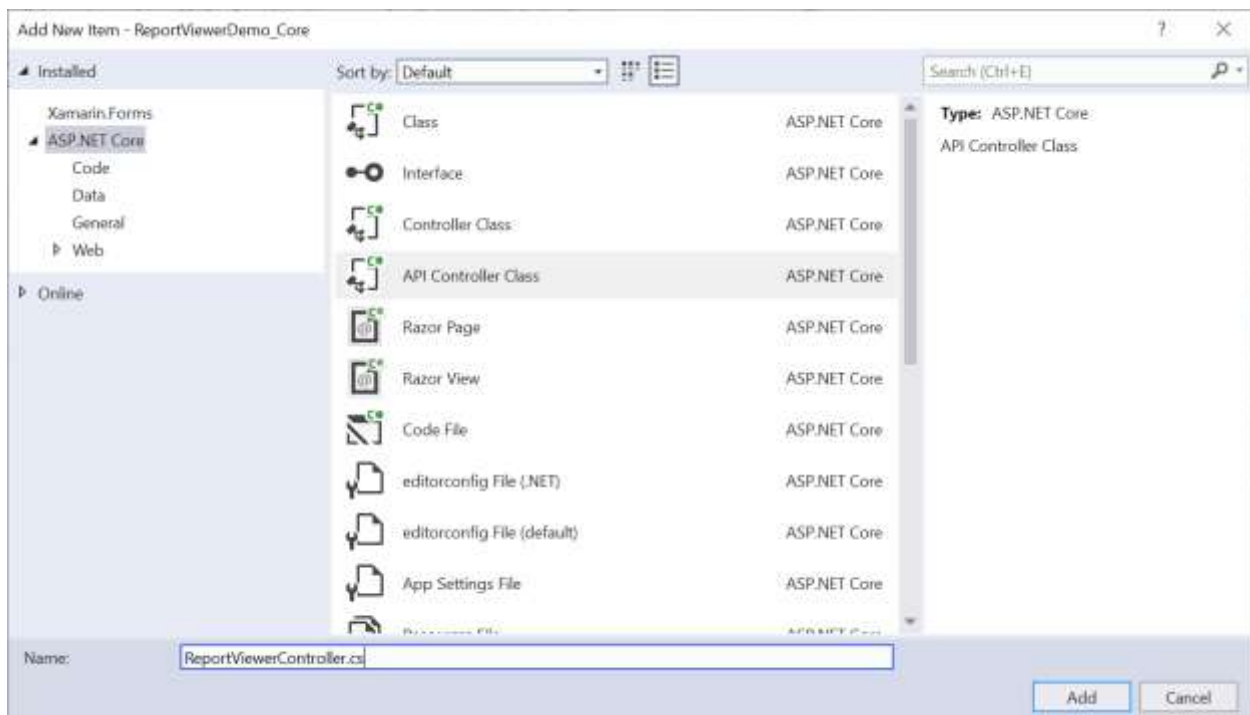
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller** class and name it as **ReportViewerController.cs**



3. Click **Add**.

While adding API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the `IReportController` interface, and then implement its methods.
6. Create local references for the interfaces given in following table.

Interface | Purpose

`IMemoryCache` | Report Viewer requires a memory cache to store the information of consecutive client request and have the rendered report viewer information in server.

`IWebHostEnvironment` | `IWebHostEnvironment` used to get the report stream from application `wwwroot\Resources` folder.

7. Next, add the `[EnableCors]` attribute to the `ReportViewerController` class and specify the policy name which given in `Program.cs`.

```
`csharp
[Route("api/[controller]/[action]")]
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
}
`
```

8. You cannot load the application report with path information in ASP.NET Core service. So, you should load the report as stream in `OnInitReportOptions`.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    System.IO.FileStream reportStream = new System.IO.FileStream(basePath + @"\Resources\sales-order-
    detail.rdl", System.IO.FileMode.Open, System.IO.FileAccess.Read);
    reportOption.ReportModel.Stream = reportStream;
}
`
```

9. You can replace the template code with the following code.

```
`csharp
```

```
[Route("api/[controller]/[action]")]
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered report viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
    public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
    {
        return ReportHelper.ProcessReport(jsonArray, this, this._cache);
    }
    // Method will be called to initialize the report information to load the report with ReportHelper for
    // processing.
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sales-order-detail.rdl report from application the folder
        // wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
        System.IO.FileStream reportStream = new System.IO.FileStream(basePath + @"\Resources\sales-order-
        detail.rdl", System.IO.FileMode.Open, System.IO.FileAccess.Read);
        reportOption.ReportModel.Stream = reportStream;
    }
}
```

```

}

// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}

//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
return ReportHelper.GetResource(resource, this, _cache);
}

[HttpPost]
public object PostFormReportAction()
{
return ReportHelper.ProcessReport(null, this, _cache);
}
}
`

```

The `sales-order-detail.rdl` report can be downloaded from [here](#). Also, you can add the report from Bold Reports installation location. For more information on installed sample location, see [Samples and demos](#).

10. Run the application and use the API URL in the Report Viewer `reportServiceUrl` property.

Enable Cross-Origin requests

Browser security prevents Report Viewer from making requests to your Web API Service when both runs in a different domain. To allow access to your Web API service from a different domain, you must enable cross-origin requests.

Call `AddCors` in `Program.cs` to add the CORS services to the app's service container. Replace the following code to allow any origin requests.

```

`csharp
builder.Services.AddCors(o => o.AddPolicy("AllowAllOrigins", builder =>

```

```
{
builder.AllowAnyOrigin()
.AllowAnyMethod()
.AllowAnyHeader();
});
.....
.....
app.UseHttpsRedirection();
app.UseCors();
app.UseAuthorization();
`
```

For more information about CORS, see [Configure CORS for API](#)

How to section for Angular Report Viewer

This section helps to get the answer for the frequently asked questions in Report Viewer.

- [Getting Started for earlier version](#)
- [How to create a RDL report](#)
- [How to create a RDLC report](#)
- [Change the exporting document file name based on the parameter](#)
- [Change the connection string datasource dynamically](#)
- [Disable the vertical scrollbar in parameter panel](#)
- [How to pass multiple values using custom data?](#)
- [How to clear cache when closing the Report Viewer?](#)

Display SSRS RDL report in Bold Reports Angular Report Viewer

This section explains you the steps required to create your first Angular reporting application in Angular CLI to display already created SSRS RDL report in Bold Reports Angular Report Viewer without using a Report Server, refer to the following steps.

If you are using higher version of [Bold Reports Angular](#) (>= v2.2.28), then refer [Getting Started for latest version](#).

Prerequisites

Before you begin, make sure your development environment includes the following:

- [Node JS](#) (version 8.x or 10.x)
- [NPM](#) (v3.x.x or higher)

Install the Angular CLI

Angular provides the easiest way to set Angular CLI projects using the [Angular CLI](#) tool. To install the CLI application globally to your machine, run the following command in the Command Prompt.

```
`typescript
npm install -g @angular/cli
`
```

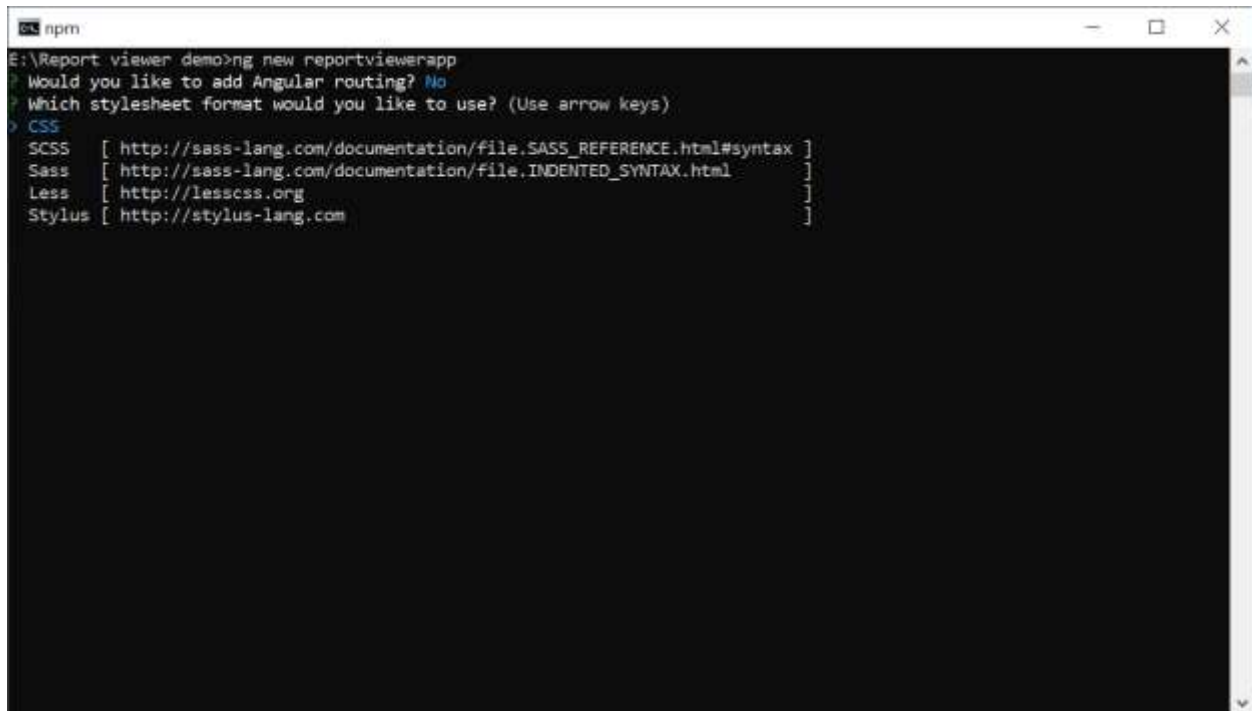
To learn more about `angular-cli` commands, click [here](#).

Create a new application

To create a new Angular application, run the following command in the Command Prompt.

```
`typescript
ng new project-name
E.g : ng new reportviewerapp
`
```

The `ng new` command prompts you for information about features to include in the initial app project. Accept the defaults by pressing the Enter or Return key.



```
npm
E:\Report viewer demo>ng new reportviewerapp
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS [ http://sass-lang.com/documentation/file.SASS_REFERENCE.html#syntax ]
  Sass [ http://sass-lang.com/documentation/file.INDENTED_SYNTAX.html ]
  Less [ http://lesscss.org ]
  Stylus [ http://stylus-lang.com ]
```

Configure Bold Report Viewer in Angular CLI

Bold reporting tools packages are distributed in NPM package as [@boldreports/angular-reporting-components](#).

1. To configure the Report Viewer component, change the directory to your application's root folder.

```
`typescript
cd project-name
```

Display SSRS RDL report in Bold Reports Angular Report Viewer **Configure** Bold Report Viewer in Angular CLI

E.g : cd reportviewerapp

,

2. Run the following commands to install the dependencies.

```
`typescript
npm install @boldreports/angular-reporting-components --save
```

,

3. Install the typings dependencies `jquery` and `boldreports/types`.

```
`typescript
npm install --save-dev @types/jquery
npm install --save-dev @boldreports/types
```

,

4. Add the typings `jquery`, and `reports.all` to the `src/tsconfig.app.json` file.

```
`js
{
...
...
"compilerOptions": {
...
...
"baseUrl": "",
"types": [
"jquery",
"reports.all"
]
},
...
...
}
```

,

5. Register the `@bold-reports/types` under the `typeRoots` node in the `tsconfig.json` file.


```

{} tsconfig.json x
1  {
2    "compileOnSave": false,
3    "compilerOptions": {
4      "baseUrl": "./",
5      "outDir": "./dist/out-tsc",
6      "sourceMap": true,
7      "declaration": false,
8      "downlevelIteration": true,
9      "experimentalDecorators": true,
10     "module": "esnext",
11     "moduleResolution": "node",
12     "importHelpers": true,
13     "target": "es2015",
14     "typeRoots": [
15       "node_modules/@types",
16       "node_modules/@boldreports/types"
17     ],

```

If you are using Angular project version 8.1.x or higher, You should set the `enableIvy` option to `false` in the project's `tsconfig.json` file to resolve the **Bold Reports** package compiling issue.

- Report Viewer requires `window.jQuery` object to render the component. Import jQuery in the `src/polyfills.ts` file as shown in the following code snippet.

```

`js
import * as jquery from 'jquery';
window['jQuery'] = jquery;
window['$'] = jquery;
`

```

Refer to the already configured import codes in the [webpack angular seed](#) application.

Adding CSS reference

Add Report Viewer component style (`bold.reports.all.min.css`) as given in the `angular.json` file within the `pojectname -> styles` section (Eg. `reportviewerapp -> styles`).

If you are using Angular 6 or lower version project, add the changes in the `angular-cli.json` file.

```

`js
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "project": {
    "name": "reportviewerapp"
  }
}

```

```

},
"reportviewerapp": [
{
"root": "src",
"outDir": "dist",
...
...
"styles": [
"styles.css",
"./node_modules/@boldreports/javascript-reporting-
controls/Content/material/bold.reports.all.min.css"
],
"scripts": [],
...
...
}
,

```

In the previous code, the `material` theme is used. You can modify the theme based on your application, refer the following syntax: `./node_modules/@boldreports/javascript-reporting-controls/Content/[theme-name]/bold.reports.all.min.css`

Adding Report Viewer component

To add the Report Viewer component, refer to the following steps:

1. Open the `app.module.ts` file.
2. You can replace the following code snippet in the `app.module.ts` file.

```

`typescript
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { BOLDREPORTVIEWERCOMPONENTS } from '@boldreports/angular-reporting-
components/src/reportviewer.component';
import { AppComponent } from './app.component';
// data-visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
import '@boldreports/javascript-reporting-controls/Scripts/common/ej2-base.min.js';

```

```

import '@boldreports/javascript-reporting-controls/Scripts/common/ej2-data.min.js';
import '@boldreports/javascript-reporting-controls/Scripts/common/ej2-pdf-export.min.js';
import '@boldreports/javascript-reporting-controls/Scripts/common/ej2-svg-base.min.js';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej2-lineargauge.min.js';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej2-circulargauge.min.js';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej2-maps.min.js';
@NgModule({
  declarations: [
    AppComponent,
    BOLDREPORTVIEWERCOMPONENTS
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

3. Open the `index.html` file and refer the following scripts in `<tag>`.

```

`html
<!-- Data-Visualization -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>

```

4. Open the `app.component.html` file and initialize the Report Viewer.
5. You can replace the following code snippet in the `app.component.html` file.

```
`javascript
<bold-reportviewer id="reportViewer_Control" style="width: 100%;height: 950px">
</bold-reportviewer>
`
```

6. Open the `app.component.ts` and replace the following code example.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'reportviewerapp';
  public serviceUrl: string;
  public reportPath: string;
  constructor() {
    // Initialize the Report Viewer properties here.
  }
}
`
```

If you have faced the issue `'ej' is not defined` after the above configuration in Angular CLI latest version 7, refer to the following code snippet in your application where you have rendered Syncfusion Components(model file) to resolve the issue.

```
`typescript
/// <reference types="reports.all" />
`
```

Create Web API service

The Report Viewer requires a Web API service to process the report files. You can skip this step and use the online [Web API services](#) to preview the already available reports or you should create any one of the following Web API services:

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

Adding already created report

If you have created a new service, you can add the reports from the Bold Reports installation location. For more information, refer to [samples and demos](#) section.

1. Create a folder `Resources` in your Web API application to store the RDL reports and add already created reports to it.
2. Add already created reports to the newly created folder.

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded in this [link](#).

Refer to the [create RDL report](#) section for creating new reports.

Set report path and Web API service

To set report path and Web API service, open the `app.component.ts` file and add the codes as shown in the `constructor`.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'jsreport-sample';
  public serviceUrl: string;
  public reportPath: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = '~/Resources/docs/sales-order-detail.rdl';
  }
}
```

In the above code, the `sales-order-detail.rdl` report and `reportServiceUrl` used from online URL.

Open the `app.component.html` to set `reportPath` and `reportServiceUrl` properties of Report Viewer as in the following.

```
`javascript
```

```
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [reportPath] =  
"reportPath" style="width: 100%;height: 950px">  
</bold-reportviewer>  
,
```

Serve the application

To serve the application, follow these steps:

1. Navigate to the root of the application and run the application using the following command.

```
`typescript  
ng serve  
,
```

2. Navigate to the appropriate port `http://localhost:4200` in the browser.

[See Also](#)

[Create RDLC report](#)

[Render RDLC reports](#)

[Preview report in print mode](#)

[Set data source credential for shared data sources](#)

[Change data source connection string](#)

[Production deployment](#)

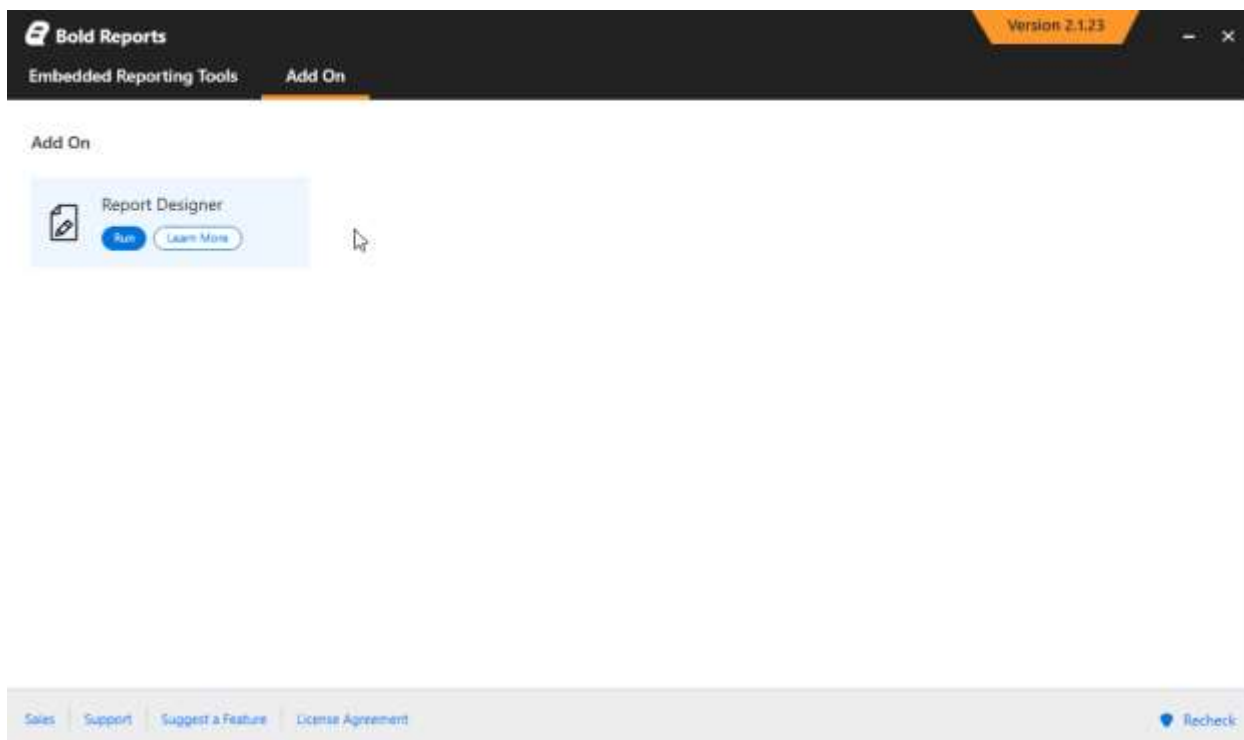
Create a SSRS RDL report

You can create an RDL report using any of the following reporting tools:

- Bold Reports Report Designer.
- Microsoft Report Builder.
- Visual Studio Report Server project template.

Bold Reports Report Designer

Bold Reports Report Designer provides the intuitive user interface to create and edit the RDL reports, which is available in Bold Embedded Reporting Tools Control Panel Add On.



Microsoft SQL Report Builder

You can create an RDL report using the Microsoft stand-alone Report Builder. For more details, refer to this [online documentation](#).

Visual Studio Report Server template

To create an RDL report in Visual Studio, a Report Server project is required where you can save your report definition (.rdl) file. For more details, refer to this [Visual Studio documentation](#).

If you do not have the Business Intelligence or Report Server Project options, you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create a RDLC report using business object data source

This section describes step by step procedure to create an RDLC report using Visual Studio Reporting project type.

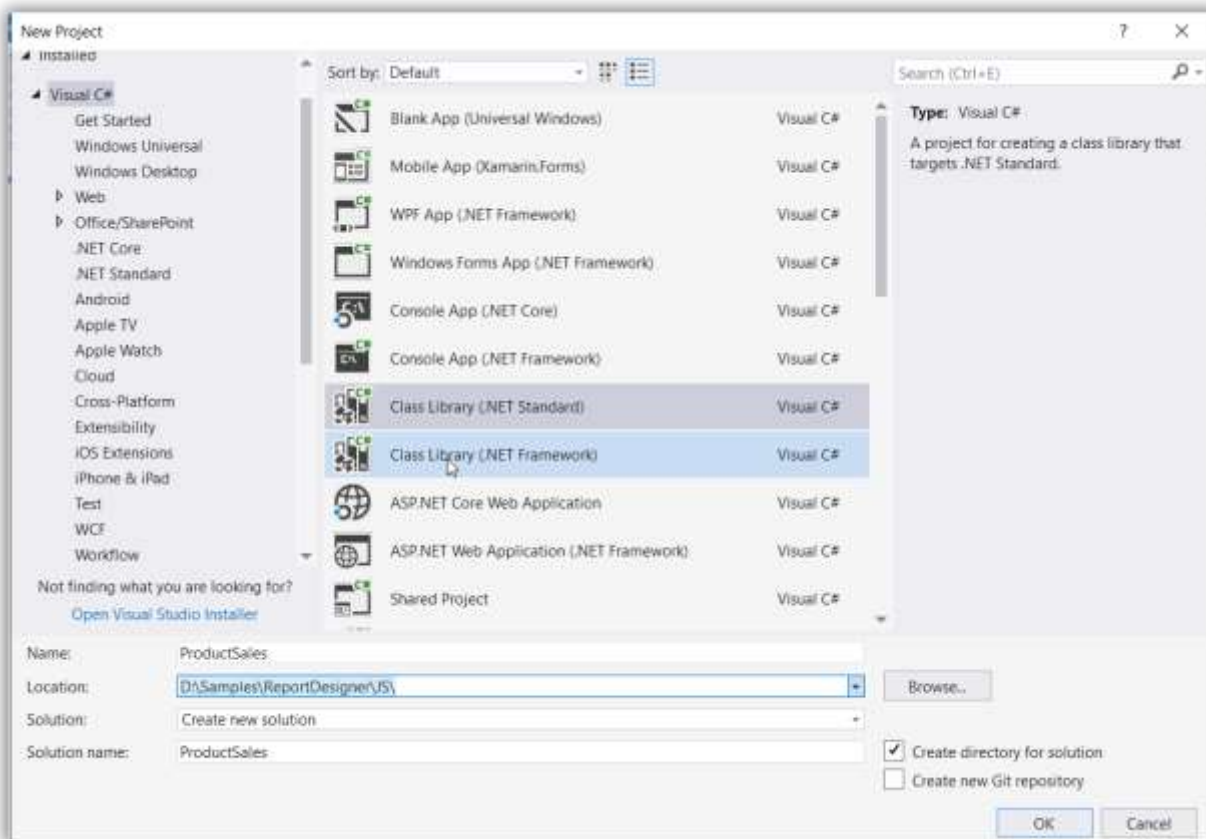
Prerequisites

- Microsoft Visual Studio 2017 or higher
- [Microsoft RDLC Report Designer](#)

If you are using Microsoft Visual Studio lower to 2017 version then you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create business object class

1. Open Visual Studio from the File menu and select **New Project**.
2. Create project with class library type from the project type list.



3. Create the class with necessary properties. You can find the reference below,

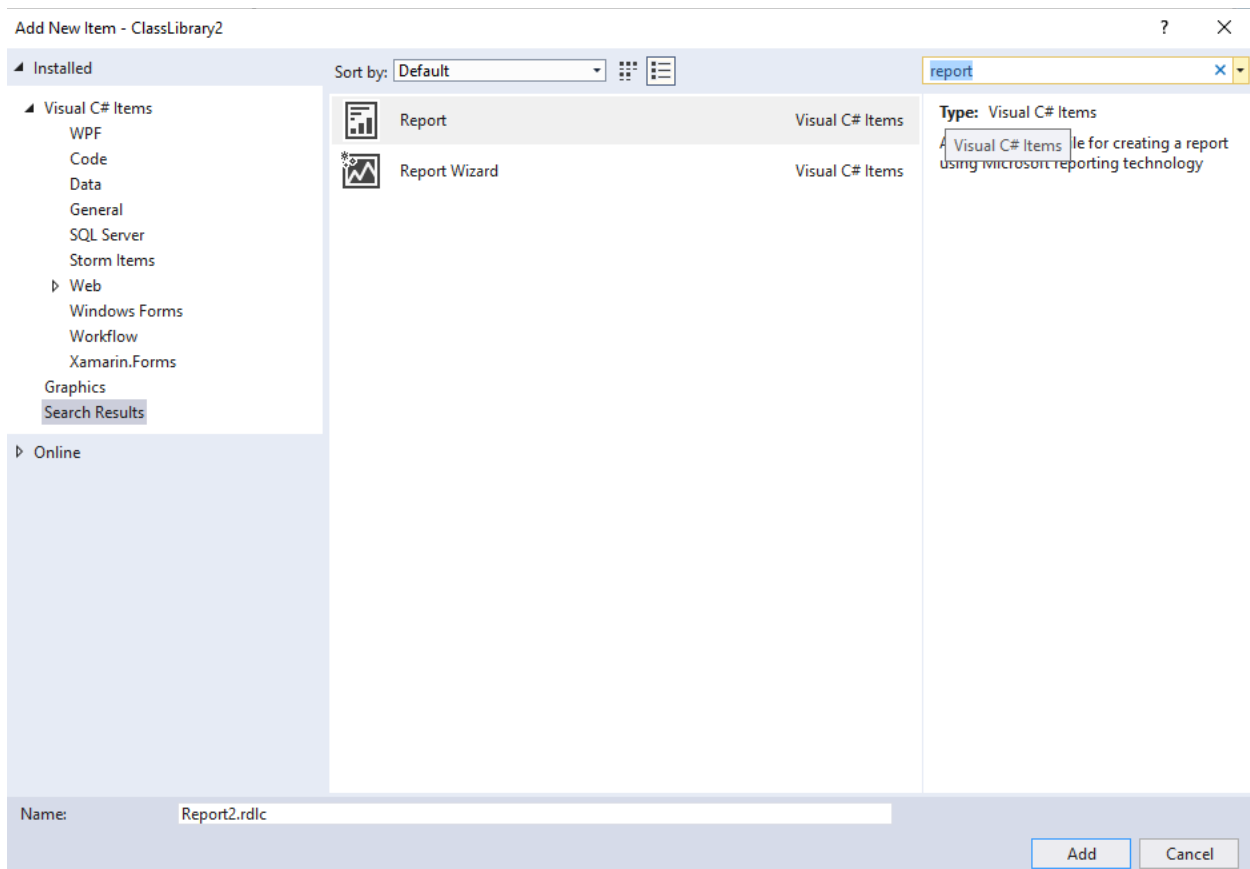
```
`csharp
public class ProductSales
{
    public string ProdCat { get; set; }
    public string SubCat { get; set; }
    public string OrderYear { get; set; }
    public string OrderQtr { get; set; }
    public double Sales { get; set; }
}
`
```

4. Clean and build the application.

Add an RDLC report

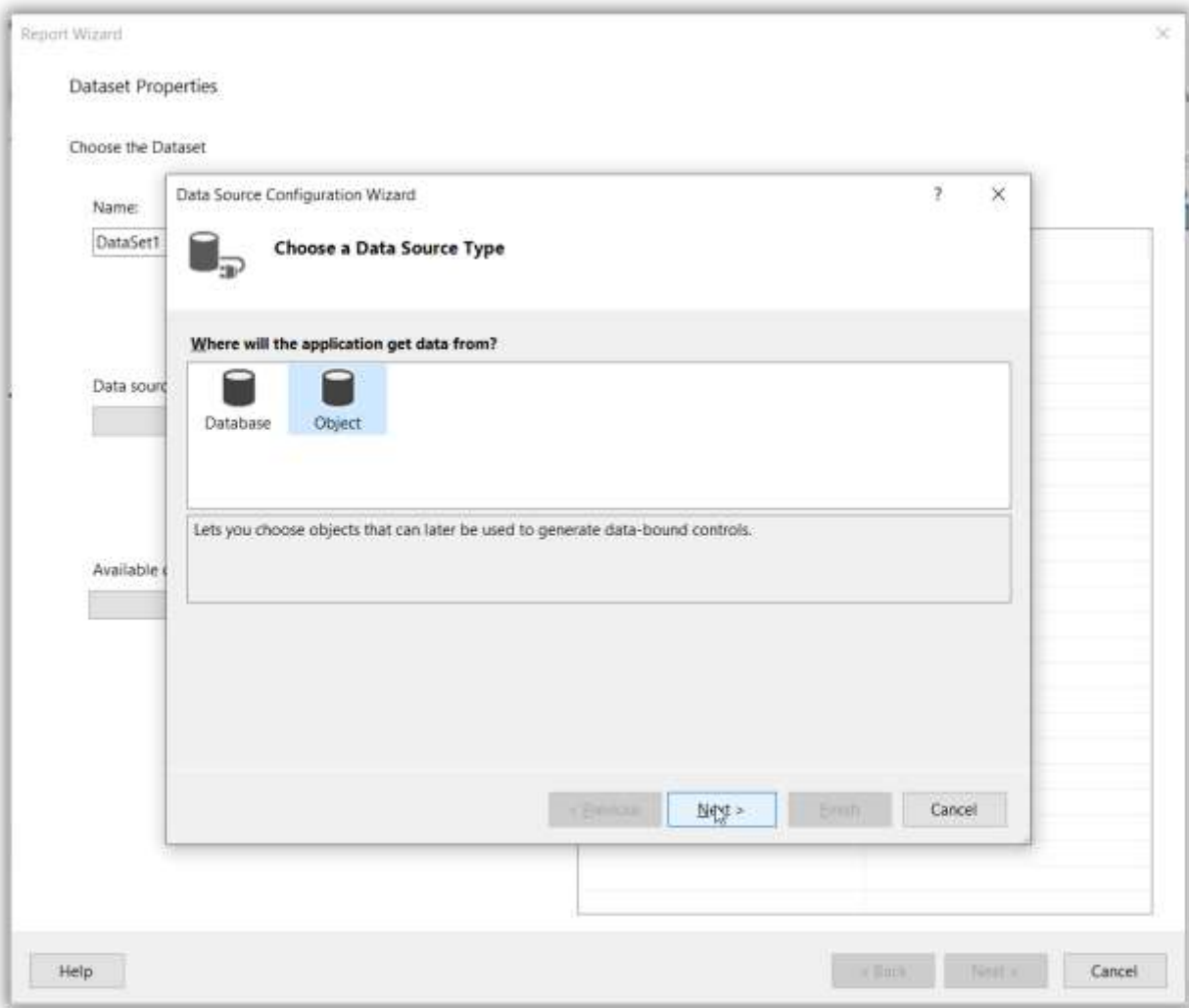
1. Right-click the project and click **Add > New Item**.

2. Search Report with new item and select **Report Wizard** to start the report creation with dataset selection.
3. Click **Add**.

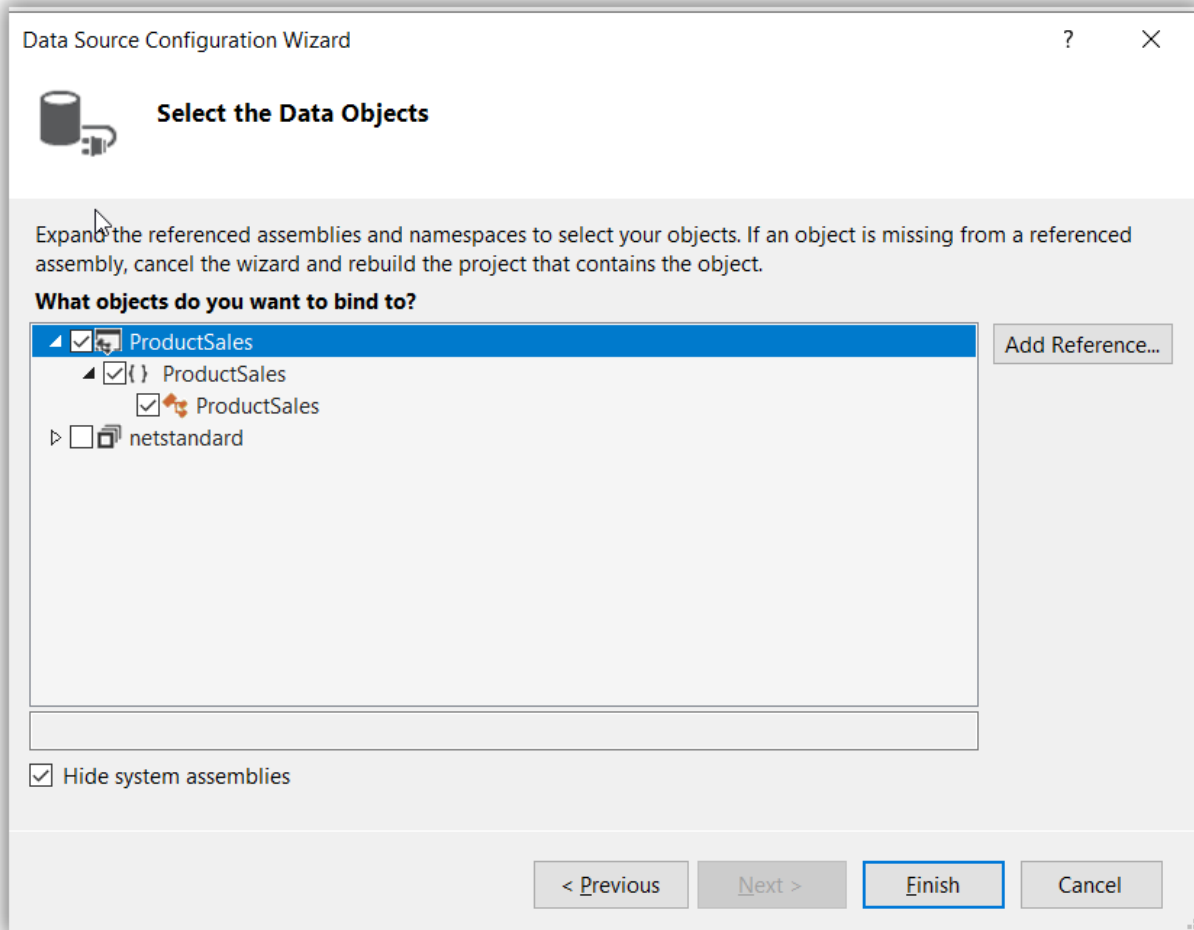


Data source and table configuration wizard

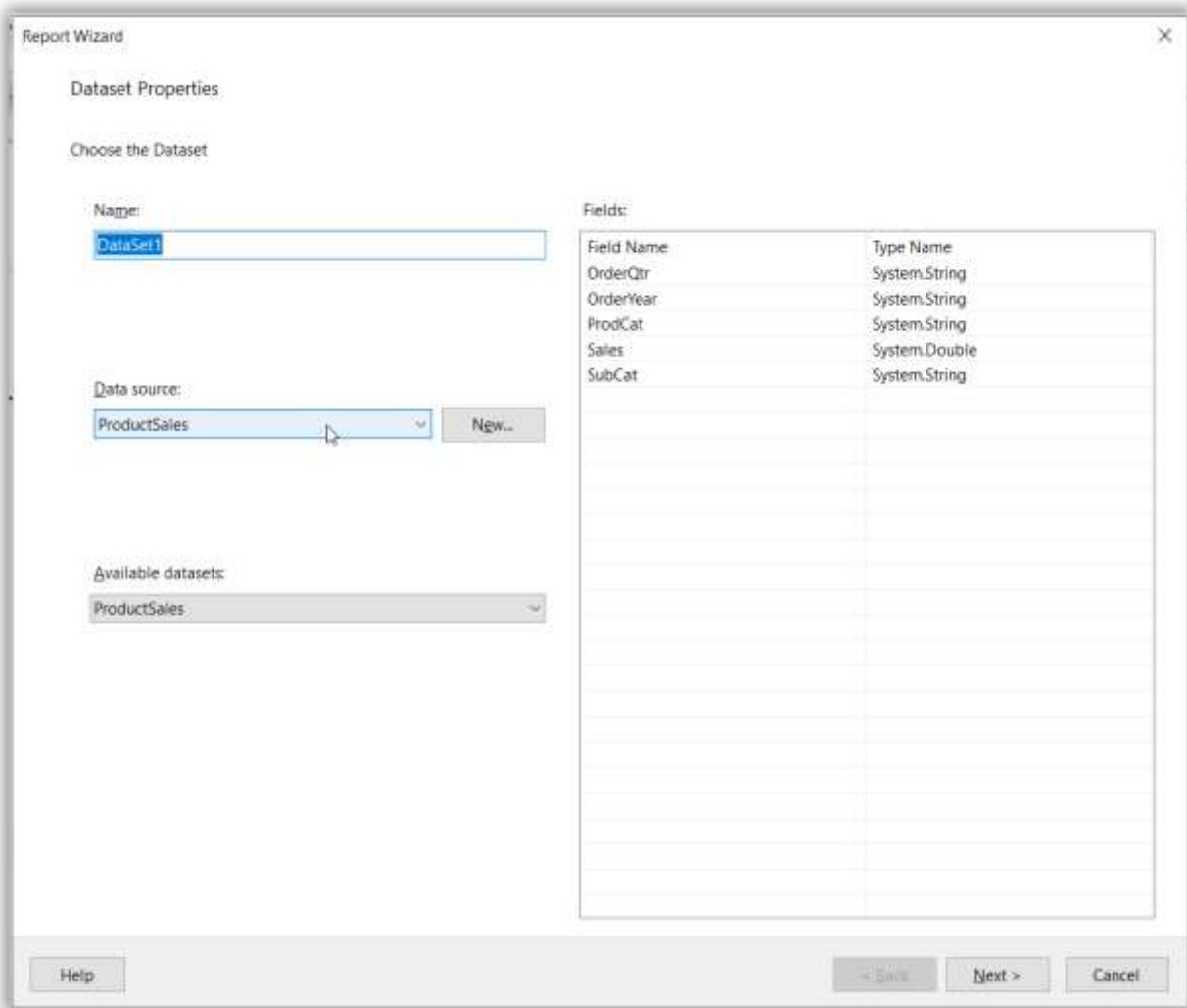
1. Choose object type from the Data Source Configuration wizard and click **Next**.



2. Expand the tree view and select **ProductSales**, and then click **Finish**.



3. In the DataSet Properties wizard, specify the dataset name as `SalesData`.



The image shows the 'Report Wizard' dialog box, specifically the 'Dataset Properties' tab. The 'Choose the Dataset' section is active. The 'Name' field is set to 'DataSet1'. The 'Data source' dropdown is set to 'ProductSales', with a 'New...' button next to it. The 'Available datasets' list also shows 'ProductSales'. On the right, the 'Fields' table lists the following fields and their types:

Field Name	Type Name
OrderQtr	System.String
OrderYear	System.String
ProdCat	System.String
Sales	System.Double
SubCat	System.String

At the bottom of the dialog, there are 'Help', '< Back', 'Next >', and 'Cancel' buttons.

4. Drag the fields into Values, Row, and Column groups, and then click **Next**.

The screenshot shows the 'Report Wizard' dialog box, specifically the 'Arrange fields' step. The window has a title bar with 'Report Wizard' and a close button. Below the title bar, the text 'Arrange fields' is displayed. A descriptive paragraph states: 'Arrange fields to group data in rows, columns, or both, and choose values to display. Data expands across the page in column groups and down the page in row groups. Use functions such as Sum, Avg, and Count on the fields in the Values box.' The main area is divided into four panes: 'Available fields' on the left, 'Column groups' at the top right, 'Row groups' at the bottom left, and 'Values' at the bottom right. The 'Available fields' pane lists 'OrderQtr', 'OrderYear', 'ProdCat', 'Sales', and 'SubCat', with 'Sales' selected. The 'Column groups' pane lists 'OrderYear' and 'OrderQtr'. The 'Row groups' pane lists 'ProdCat' and 'SubCat'. The 'Values' pane shows the function 'Sum(Sales)' with a plus icon to its right. At the bottom of the dialog are four buttons: 'Help', '< Back', 'Next >', and 'Cancel'. A mouse cursor is pointing at the 'Next >' button.

5. Choose the table layout and click **Next**.
6. Select table style and click **Finish**.

Report Wizard

Choose the layout

If you choose to show subtotals and grand totals, you can place them above or below the group. Stepped reports show hierarchical structure with indented groups in the same column.

Options:

- ☒ Show subtotals and grand totals
 - ☒ Blocked, subtotal below
 - ☐ Blocked, subtotal above
 - ☐ Stepped, subtotal above
- ☒ Expand/collapse groups

Preview

		[Order Year]		Total
Prod Cat	Sub Cat	[OrderQty]	Total	
[ProdCat]	[SubCat]	[Sum(Sales)]	[Sum(Sales)]	[Sum(Sales)]
	Total	[Sum(Sales)]	[Sum(Sales)]	[Sum(Sales)]
Total		[Sum(Sales)]	[Sum(Sales)]	[Sum(Sales)]

Help < Back Next > Cancel

Now, the RDLC report is displayed in the Visual Studio as follows.

Data source and table



Find the following steps to change the file based on parameter values in Report.

- ```
html

function onRenderingComplete(event) {
var parameters = event.reportParameters;
if(parameters){
for (var i = 0; i < parameters.length; i++) {
if(parameters[i].Name == "Department"){
this.exportFileName = "Sales for " + parameters[i].Value;
}
}
}
}
```

- ```
`html`  
function onExportItemClick(event) {
```

```
event.fileName = this.exportFileName ;
}
`
```

How to change the data source dynamically

You have to use the `reportOption.ReportModel.DataSourceCredentials` available with `OnInitReportOptions` method to change the datasource dynamically in the web API controller. The following code sample shows how to change the connection string of the `<database>` data source in the report.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    DataSourceCredentials dataSourceCredentials = new DataSourceCredentials();
    string connectionString = "Data Source =<instancename> ; Initial Catalog = <database>; User ID =
'<username>; Password = '<password>"";
    //You have to provide the shared data source name used with the report or the data source name
    available with the report.
    dataSourceCredentials.Name = "<database>";
    dataSourceCredentials.ConnectionString = connectionString;
    reportOption.ReportModel.DataSourceCredentials = new List<DataSourceCredentials> {
    dataSourceCredentials };
}
`
```

How to disable the vertical scrollbar in parameter panel

To disable the vertical scrollbar in parameter panel, set the `enableparameterblockscroller` property to false.

```
<span style="font-weight:bold">Example</span>
`javascript
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [reportPath] =
"reportPath" [enableParameterBlockScroller]= false >
</bold-reportviewer>
`
```

How to pass multiple values using the custom data

Pass the multiple data values as JSON in `ajaxBeforeLoad` event using the 'data' property, which needs to be serialized in the server side API using known class type and `JsonConvert.DeserializeObject`.

1. Map the [ajaxBeforeLoad](#) event with `onAjaxRequest` function in the client side to pass custom data to the server.

```
`html
<bold-reportviewer id="reportViewer_Control"
[reportServiceUrl] = "serviceUrl"
[processingMode] = "Remote"
[reportServerUrl] = "serverUrl"
[reportPath] = "reportPath"
-ajaxBeforeLoad) = "onAjaxRequest($event)">
</bold-reportviewer>
`js
function onAjaxRequest(args) {
//Passing custom data to server
}
```

2. You need to pass the multiple custom data values as JSON and use the `data` property to send the JSON string to the server in the Ajax request.

```
`js
function onAjaxRequest(args) {
//Passing custom data to server
var jsonData = {
customerID: "CI0021",
productID: "PO0022"
};
args.data = jsonData;
}
```

3. The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates deserialization of the JSON string and change the data source connection strings based on Customer ID and Product ID in the `OnInitReportOptions` method.

```
`csharp
public SampleData customDatas = new SampleData();
public class SampleData
{
    public string customerID { get; set; }
    public string productID { get; set; }
}
[HttpPost]
public object PostReportAction([FromBody]Dictionary<string, object> jsonResult)
{
    if (jsonResult.ContainsKey("customData"))
    {
        //Get client side JSON custom data, deserialize it and store in local variable.
        customDatas = JsonConvert.DeserializeObject<SampleData>(jsonResult["customData"].ToString());
    }
    return ReportHelper.ProcessReport(jsonResult, this, this._cache);
}
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (customDatas != null)
    {
        if (customDatas.customerID == "CI0021" && customDatas.productID == "PO0022") {
            //If you are changing the connection string based on customer id then could you please change the
            connection string as below.
            //reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));
            reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
            Catalog=<database>;"));
        }
    }
}
```

Use SASS stylesheet in Angular with Bold Reports Angular Report Viewer Component

This section explains you the steps required to create your Angular reporting application with SASS stylesheet in Angular CLI to display already created SSRS RDL report in Bold Reports Angular Report Viewer without using a Report Server, refer to the following steps

Prerequisites

Before you begin, make sure your development environment includes the following:

- [Node JS](#) (version 8.x or 10.x)
- [NPM](#) (v3.x.x or higher)

Install the Angular CLI

Angular provides the easiest way to set Angular CLI projects using the [Angular CLI](#) tool. To install the CLI application globally to your machine, run the following command in the Command Prompt.

```
`typescript
npm install -g @angular/cli@latest
`
```

To learn more about `angular-cli` commands, click [here](#).

Create a new application

To create a new Angular application, run the following command in the Command Prompt.

```
`typescript
ng new project-name
E.g : ng new reportviewerapp
`
```

The `ng new` command prompts you for information about features to include in the initial app project. Accept the defaults by pressing the Enter or Return key.

```
npm
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\Report viewer demo>ng new reportviewerapp
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use?
  CSS
  SCSS [ https://sass-lang.com/documentation/syntax#scss ]
> Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less [ http://lesscss.org ]
  Stylus [ http://stylus-lang.com ]
```

Configure Bold Report Viewer in Angular CLI

Bold reporting tools packages are distributed in NPM package as [@boldreports/angular-reporting-components](#).

1. To configure the Report Viewer component, change the directory to your application's root folder.

```
`typescript
cd project-name
E.g : cd reportviewerapp
`
```

2. Run the following commands to install the [Bold Reports Angular](#) library.

```
`typescript
npm install @boldreports/angular-reporting-components --save-dev
`
```

3. Also, Install [Bold Reports typings](#) by executing the below command.

```
`typescript
npm install --save-dev @boldreports/types
`
```

4. Register the `@bold-reports/types` under the `typeRoots` and add the typings `jquery` and `reports.all` to the `tsconfig.app.json` file.

```
`js
{
...
...
"compilerOptions": {
...
...
"typeRoots": [
"node_modules/@types",
"node_modules/@boldreports/types"
],
"types": [
"jquery",
"reports.all"
]
},
...
...
}
```

5. Report Viewer requires `window.jQuery` object to render the component. Import jQuery in the `src/polyfills.ts` file as shown in the following code snippet.

```
`js
import * as jquery from 'jquery';
window['jQuery'] = jquery;
window['$'] = jquery;
`
```

Adding Bold Reports styles reference

Add Report Viewer component style (`bold.reports.all.min.css`) as given in the `angular.json` file within the `pojectname -> styles` section (Eg. `reportviewerapp -> styles`).

If you are using Angular 6 or lower version project, add the changes in the `angular-cli.json` file.

```
`js
{
"$schema": "./node_modules/@angular/cli/lib/config/schema.json",
"project": {
"name": "reportviewerapp"
},
"reportviewerapp": [
{
"root": "src",
"outDir": "dist",
...
...
"styles": [
"src/styles.sass",
"./node_modules/@boldreports/javascript-reporting-
controls/Content/material/bold.reports.all.min.css"
],
"scripts": [],
...
...
}
`
```

In the previous code, the `material` theme is used. You can modify the theme based on your application, refer the following syntax: `./node_modules/@boldreports/javascript-reporting-controls/Content/[theme-name]/bold.reports.all.min.css`

Adding Report Viewer component

To add the Report Viewer component, refer to the following steps:

1. Open the `app.module.ts` file.
2. You can replace the following code snippet in the `app.module.ts` file.

```
`typescript
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { BoldReportViewerModule } from '@boldreports/angular-reporting-components';
```

```
import { AppComponent } from './app.component';
// Report viewer
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
// data-visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    BoldReportViewerModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3. Open the `index.html` file and refer the following scripts in `<script>` tag.

```
<html>
<!-- Data-Visualization -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
```

4. Open the `app.component.html` file and initialize the Report Viewer.
5. You can replace the following code snippet in the `app.component.html` file.

```
`javascript
<bold-reportviewer id="reportViewer_Control" style="width: 100%;height: 950px">
</bold-reportviewer>
`
```

6. Open the `app.component.ts` and replace the following code example.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.sass']
})
export class AppComponent {
  title = 'reportviewerapp';
  public serviceUrl: string;
  public reportPath: string;
  constructor() {
    // Initialize the Report Viewer properties here.
  }
}
`
```

If you have faced the issue `'ej' is not defined` after the above configuration in Angular CLI latest version 7, refer to the following code snippet in your application where you have rendered Syncfusion Components(model file) to resolve the issue.

```
`typescript
/// <reference types="reports.all" />
`
```

Create Web API service

The Report Viewer requires a Web API service to process the report files. You can skip this step and use the online [Web API services](#) to preview the already available reports or you should create any one of the following Web API services:

Use SASS stylesheet in Angular with Bold Reports Angular Report Viewer Component **Set** report path and Web API service

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

Adding already created report

If you have created a new service, you can add the reports from the Bold Reports installation location. For more information, refer to [samples and demos](#) section.

1. Create a folder **Resources** in your Web API application to store the RDL reports and add already created reports to it.
2. Add already created reports to the newly created folder.

In this tutorial, the **sales-order-detail.rdl** report is used, and it can be downloaded in this [link](#).

Refer to the [create RDL report](#) section for creating new reports.

Set report path and Web API service

To set report path and Web API service, open the **app.component.ts** file and add the codes as shown in the **constructor**.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.sass']
})
export class AppComponent {
  title = 'jsreport-sample';
  public serviceUrl: string;
  public reportPath: string;
  constructor() {
    this.serviceUrl = 'https://demos.boldreports.com/services/api/ReportViewer';
    this.reportPath = '~/Resources/docs/sales-order-detail.rdl';
  }
}
```

In the above code, the **sales-order-detail.rdl** report and **reportServiceUrl** used from online URL.

Open the **app.component.html** to set **reportPath** and **reportServiceUrl** properties of Report Viewer as in the following.

```
`javascript
```

Use SASS stylesheet in Angular with Bold Reports Angular Report Viewer Component application

Serve the

```
<bold-reportviewer id="reportViewer_Control" [reportServiceUrl] = "serviceUrl" [reportPath] = "reportPath" style="width: 100%;height: 950px">
```

```
</bold-reportviewer>
```

,

Serve the application

To serve the application, follow these steps:

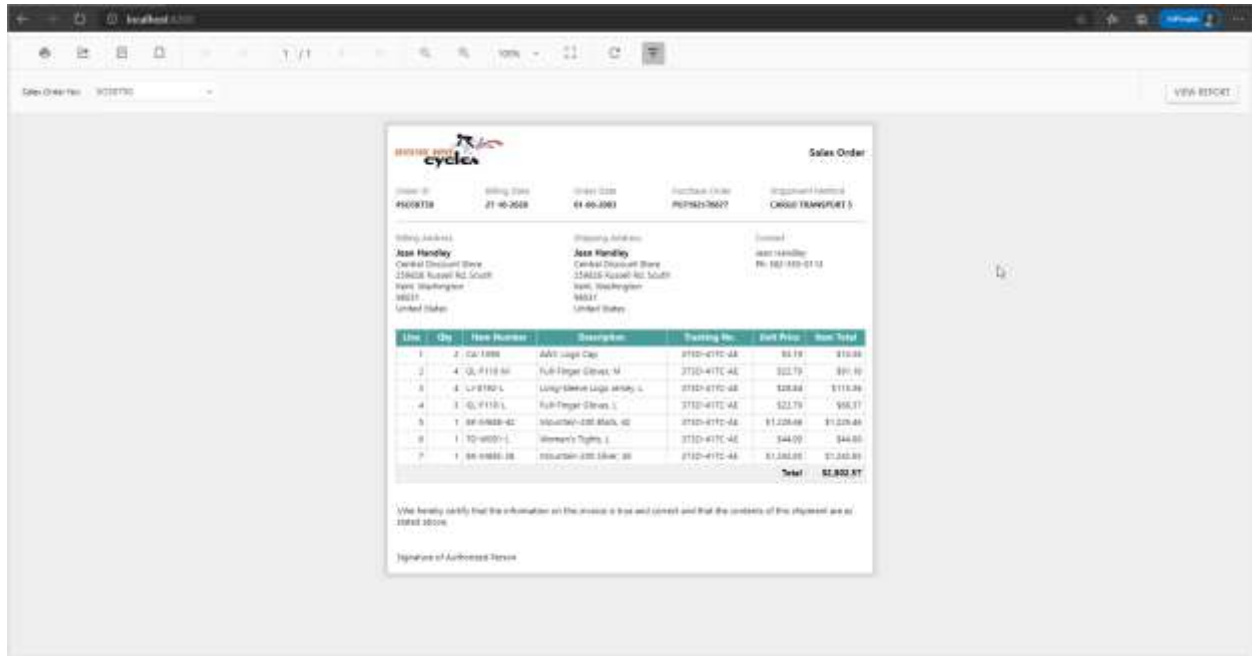
1. Navigate to the root of the application and run the application using the following command.

```
`typescript
```

```
ng serve
```

,

2. Navigate to the appropriate port `http://localhost:4200` in the browser.



See Also

[Create RDLC report](#)

[Render RDLC reports](#)

[Preview report in print mode](#)

[Set data source credential for shared data sources](#)

[Change data source connection string](#)

[Production deployment](#)

How to clear cache when closing the Report Viewer

By using `clearReportCache` and `destroy` method, you can clear the server side cache. You have to use this method when closing report viewer and switching to another report within your application.

TypeScript

Add the following code in `index.html` file.

```
`js
<div id="reportviewer"></div>
<script>
var isSubmit = true;
$(document.body).bind('submit', $.proxy(this.formSubmit, this));
function formSubmit(args)
{
isSubmit = false;
}
window.onbeforeunload = function () {
if (isSubmit) {
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.clearReportCache();
reportviewerObj.destroy();
}
isSubmit = true;
};
</script>
`
```

Web API

In the `PostReportAction` method, you have to collect the `GC` with `ClearCache` as shown in the following code sample.

```
`csharp
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
{
bool isclearcache = false;
if (jsonArray != null && jsonArray.ContainsKey("reportAction") && jsonArray["reportAction"].ToString()
== "ClearCache")
{
```

```

isclearcache = true;
}
var reportresult = ReportHelper.ProcessReport(jsonArray, this, this._cache);
if (isclearcache)
{
GC.Collect(); isclearcache = false;
}
return reportresult;
}
`

```

Migrate to Report Viewer v2.0 component

This section provides simple step-by-step instructions to update your existing Report Viewer application to our latest modern v2.0 scripts, styles and components.

Replacing CSS reference

1. Open the `angular.json` file.
2. Remove the Report Viewer component old style
 - o `bold.reports.all.min.css`
3. Added v2.0 style
 - o `\v2.0\tailwind-light\bold.report-viewer.min.css`
4. The final updated `angular.json` file.

If you are using an Angular 6 or lower version project, add the changes to the `angular-cli.json` file.

```

`js
{
"$schema": "./node_modules/@angular/cli/lib/config/schema.json",
"project": {
"name": "reportviewerapp"
},
"reportviewerapp": [
{
"root": "src",
"outDir": "dist",
...
...

```

```

"styles": [
  "styles.css",
  "./node_modules/@boldreports/javascript-reporting-controls/Content/v2.0/tailwind-light/bold.report-viewer.min.css"
],
"scripts": [],
...
...
}
`

```

Replacing Report Viewer component

1. Open the `app.module.ts` file.
2. Remove the Report Viewer component old scripts
 - o `bold.report-viewer.min`
 - o `ej.bulletgraph.min`
 - o `ej.chart.min`
3. Added v2.0 scripts
 - o `\v2.0\common\bold.reports.common.min`
 - o `\v2.0\common\bold.reports.widgets.min`
 - o `\v2.0\bold.report-viewer.min.css`
4. The final updated `app.module.ts` file.

```

`typescript
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { BoldReportViewerModule } from '@boldreports/angular-reporting-components';
import { AppComponent } from './app.component';
import '@boldreports/javascript-reporting-controls/Scripts/v2.0/common/bold.reports.common.min';
import '@boldreports/javascript-reporting-controls/Scripts/v2.0/common/bold.reports.widgets.min';
import '@boldreports/javascript-reporting-controls/Scripts/v2.0/bold.report-viewer.min';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,

```

```

BoldReportViewerModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
`

```

Breaking Changes

This section provides the detailed information on API and behaviour changes that break existing applications when upgrading them to latest version of Bold Reports.

Breaking Changes

To support Angular 9, we made the below, breaking changes with Bold Reports Angular components version 2.2.28.

Scenario	Existing Angular Package	Latest Angular Package
Importing components	// imports Report viewerimport { BOLDREPORTVIEWERCOMPONENTS } from '@boldreports/angular-reporting-components/src/reportviewer.component'â€™;import { BOLDREPORTDESIGNERCOMPONENTS } from '@boldreports/angular-reporting-components/src/reportdesigner.component'â€™;	// imports Report viewerimport { BoldReportViewerModule } from '@boldreports/angular-reporting-components'â€™;import { BoldReportDesignerModule } from '@boldreports/angular-reporting-components'â€™;
Importing modules	// imports both Report Viewer and Designerimport { BoldReportsAngularModule } from '@boldreports/angular-reporting-components'â€™;	// imports both Report Viewer and Designerimport { BoldReportsModule } from '@boldreports/angular-reporting-components'â€™;
Script reference	We only refer the data-visualization scripts.Example:import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min'â€™;import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej2-circulargauge.min;	We need to refer the Report Viewer and Designer scripts along with the existing data-visualization script references.Report Viewerimport '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min'â€™;Report Designerimport '@boldreports/javascript-reporting-controls/Scripts/bold.report-designer.min'â€™;Data visualizationExample:import

		<pre> @boldreports/javascript-reporting- controls/Scripts/data- visualization/ej.bulletgraph.min™;im port @boldreports/javascript- reporting-controls/Scripts/data- visualization/ej2-circulargauge.min; </pre>
--	--	---

How to section for Angular Bold Reporting Tools

This section helps to get the answer for the frequently asked how to queries in Bold Reporting Tools.

- [How to resolve ej undefined issue when using the Report Viewer properties?](#)

How to resolve ej undefined issue when using the Report Viewer properties

This issue occurs when the type references for the Bold Reports components are not properly referred in the Angular application. This issue can be resolved by registering the `@bold-reports/types` under the `typeRoots` and add the typings `jquery` and `reports.all` to the `tsconfig.app.json` or the `tsconfig.json` file.

```

`js
{
...
...
"compilerOptions": {
...
...
"typeRoots": [
"node_modules/@types",
"node_modules/@boldreports/types"
],
"types": [
"jquery",
"reports.all"
]
},
...
...

```

```

}
`

```

If you have still faced the issue of 'ej' is not defined after the above configuration in Angular CLI latest version 7, refer to the following code sample at the top of your `app.component.ts` file in your application where you have rendered Syncfusion Components(model file) to resolve the issue.

```

`typescript
/// <reference types="reports.all" />
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  ...
  ...
}
`

```

How to resolve @angular/core has no export member NgModuleDeclaration

The error message `@angular/core has no export member NgModuleDeclaration` occurs when you try to install Bold Reports 5.2.27 or greater in an Angular version lower than v12. This is because the Bold Reports package [@boldreports/angular-reporting-components](#) requires Angular v12 or higher to function properly.

If you are getting this error message, you can either upgrade your Angular version to 12 or higher, or you can downgrade the Bold Reports package to a version lower than 5.1.28.

[See Also](#)

[Report Viewer getting started](#)

Reporting tools for JavaScript

Enterprise-class reporting tools for JavaScript development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your JavaScript applications.

[How to best read this user guide](#)

The best way to get started would be to read the `Getting Started` section of the documentation for the control that you would like to start using first. The `Getting Started` guide gives just enough information

that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Another valuable resource is the API reference which provides detailed information on the object hierarchy as well as the settings available on every object.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

Reporting tools for JavaScript

Enterprise-class reporting tools for JavaScript development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your JavaScript applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Another valuable resource is the API reference which provides detailed information on the object hierarchy as well as the settings available on every object.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

System Requirements

This topic describes the software and hardware requirements for setting up the development environment of Bold Reports Javascript.

Supported Operating Systems

- Windows 7+, 8+
- Windows Server 2008 R2+

Hardware Environments

The following hardware environments are necessary to run the Bold Reports Javascript.

- 1 GHZ or faster, 32 bit or 64 bit processor.

- 1 GB RAM for 32 bit or 2 GB RAM for 64 bit.

Development Environments

The following development environments are necessary to run the Bold Reports Javascript.

- [Microsoft Visual Studio 2010](#) or [later](#)
- Internet Information Services (IIS) 7.0+

Browser Compatibility

- IE 9+
- Microsoft Edge
- Mozilla Firefox 22+
- Chrome 17+
- Opera 12+
- Safari 5+

See Also

- [Licensing procedure for deployment](#)

Overview

The Report Viewer is a visualization control used to display SSRS, RDL, RDLC, and Bold Report Server reports within web applications. It allows you to view RDL/RDLC reports with or without using SSRS or Bold Report Server. You can bind data sources, parameters, and render reports with all major capabilities of RDL reporting and export the report to PDF, Excel, CSV, Word, PowerPoint, and HTML formats. Some of the key features are,

- Renders interactive reports with drill down, drill through, hyperlinks, and interactive sorting.
- Easily customize each element of report viewer and provide events for report processing customization.
- Supports jQuery, Angular, React, Blazor, ASP.NET Core, ASP.NET MVC, ASP.NET WebForms, WPF, and UWP.

Introducing the newly redesigned Report Viewer! We have given it a refreshing new look that embraces sleek and modern design, aligning perfectly with the latest web design trends. It allows seamless integration into your application. For detailed guidance on the migration process, we recommend you to refer our [Report Viewer v2.0 migration document](#).

Display SSRS RDL report in Bold Reports HTML5 JavaScript Report Viewer

This section explains you the steps required to create your first JavaScript reporting application to display already created SSRS RDL report in the Bold Reports HTML5 JavaScript Report Viewer without using a Report Server.

To get start quickly with Report Viewer, you can check on this video:

youtube: <https://youtu.be/N0gaK2GJ0So>

HTML file creation

Create a basic HTML file as shown below and place it in a separate folder.

```
`html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Report Viewer first HTML page</title>
</head>
<body>
</body>
</html>
`
```

Refer scripts and CSS

Directly refer all the required scripts and style sheets from the [CDN](#) links that are mandatorily required to use the Report Viewer, in the following order.

- `bold.reports.all.min.css`
- `jquery.min.js`
- `ej2-base.min.js`, `ej2-data.min.js`, `ej2-pdf-export.min.js`, `ej2-svg-base.min.js`, `ej2-lineargauge.min.js`, and `ej2-circulargauge.min.js` - Renders the gauge item. Add these scripts only if your report contains the gauge report item.
- `ej2-maps.min.js` - Renders the map item. Add this script only if your report contains the map report item.
- `bold.reports.common.min.js`
- `bold.reports.widgets.min.js`
- `ej.chart.min.js` - Renders the chart item. Add this script only if your report contains the chart report item.
- `bold.report-viewer.min.js`

Refer to the [Bold Reports CDN](#) to learn more details about the Bold Reports CDN scripts and style sheet links.

You can replace the following code in the `<head>` tag of the Report Viewer HTML page.

Whether you want to get the scripts and style sheets as local, then install the `BoldReports.Javascript` NuGet package in your application.

```
`html
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
```

```

<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
<!-- Report Viewer component script-->
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
,

```

To learn more about rendering a report with data visualization report items, refer to the [how to render data visualization report items](#) section.

Adding Report Viewer Widget

Add the `<div>` element within the `<body>` section, which acts as a container for the `boldReportViewer` widget to render and then initialize the `boldReportViewer` widget within the script section as shown as follows.

```

`html
<div style="height: 600px; width: 950px;">
<!-- Creating a div tag which will act as a container for boldReportViewer widget.-->
<div style="height: 600px; width: 950px; min-height: 400px;" id="viewer"></div>
<!-- Setting property and initializing boldReportViewer widget.-->
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer();
});
</script>

```

Display SSRS RDL report in Bold Reports HTML5 JavaScript Report Viewer **Set** report path and Web API service

</div>

,

Create Web API service

The Report Viewer requires a Web API service to process the report files. You can skip this step and use the online [Web API services](#) to preview the already available reports or you should create any one of the following Web API services:

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

If you are looking to load the report directly from the SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#).

Adding already created report

If you have created a new service, add the reports from the Bold Reports installation location. For more information, refer to the [samples and demos](#) section.

1. Create a folder **Resources** in your Web API application to store RDL reports and add the already created reports to it.
2. Add already created reports to the newly created folder.

In this tutorial, the **sales-order-detail.rdl** report is used, and it can be downloaded at this [link](#).

To create a new report refer to the [create RDL report](#) section.

Set report path and Web API service

To render the reports available in your application, set the [reportPath](#) and [reportServiceUrl](#) properties of the Report Viewer.

`html

```
<div style="height: 600px; width: 950px;">
```

```
<!-- Creating a div tag which will act as a container for boldReportViewer widget.-->
```

```
<div style="height: 600px; width: 950px; min-height: 400px;" id="viewer"></div>
```

```
<!-- Setting property and initializing boldReportViewer widget.-->
```

```
<script type="text/javascript">
```

```
$(function () {
```

```
$("#viewer").boldReportViewer({
```

```
reportServiceUrl: "https://demos.boldreports.com/services/api/ReportViewer",
```

```
reportPath: '~/Resources/docs/sales-order-detail.rdl'
```

```
});
```

```
});
```

```
</script>
```

```
</div>
```

In the above code, the `reportServiceUrl` used from online URL. You can host the Bold Reports service at any Azure, AWS, or own domain URL and use it in the Report Viewer. You can view the already created Web API service from the [Reporting Service](#) git hub location.

Preview the report

Open your HTML page in the web browser and the Report Viewer will display the report as shown below.

```
{% tab demoPath="javascript-reporting/report-viewer/getting-started/initialize-report-viewer"
files="index.html,index.js,ReportViewerController.cs" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>

`

{% endtabItem %}

{% tabItem header="ReportViewerController.cs" %}

`csharp
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
    [System.Web.Http.ActionName("GetResource")]
    [AcceptVerbs("GET")]
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }
}
```

// Method that will be called when initialize the report options before start processing the report

[NonAction]

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
// You can update report options here
```

```
}
```

// Method that will be called when reported is loaded

[NonAction]

```
public void OnReportLoaded(ReportViewerOptions reportOption)
```

```
{
```

```
// You can update report options here
```

```
}
```

```
}
```

```
,
```

```
{% endtabItem %}
```

```
{% endtab %}
```

Note: You can refer to our feature tour page for the [JavaScript Report Viewer](#) to see its innovative features. Additionally, you can view our [JavaScript Report Viewer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

[See Also](#)

[Migrate to Report Viewer v2.0](#)

[Render report with data visualization report items](#)

[Create RDLC report](#)

[Render RDLC reports](#)

[Preview report in print mode](#)

[Set data source credential for shared data sources](#)

[Change data source connection string](#)

[List of SSRS server versions are supported in Bold Reports](#)

Load SSRS Report Server reports

Report Viewer has support to load RDL reports from SSRS Report Server. To render SSRS Reports set the [reportServerUrl](#), [reportPath](#), and [reportServiceUrl](#) properties as in the following code snippet.

1. To create your first Javascript reporting application, refer to the [Getting-started](#) section.

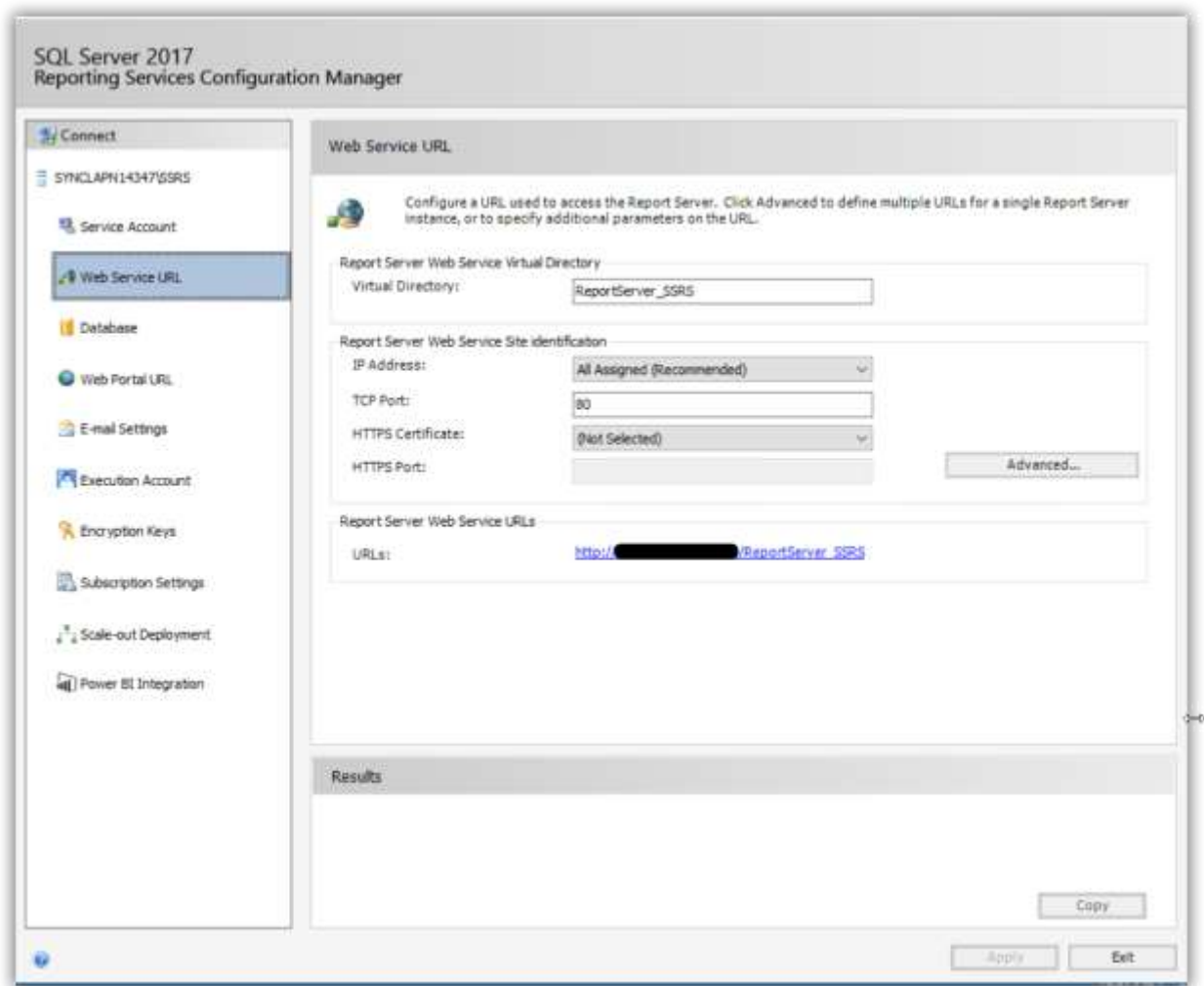
If you need to know about the difference between `reportServiceUrl` and `reportServerUrl`, then refer to [Difference between Report Service URL and Report Server URL](#).

2. Set the `reportServerUrl` API on Bold Report Viewer with `WebServiceURL`. Open the `App.js` or `app.component.ts` and replace the following code example.

```
`js
<div style="height: 100%; width: 100%;">
<div style="height: 600px; width: 950px; min-height: 400px; id="viewer"></div>
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/SSRSReports",
reportServerUrl: "http://<servername>/Reports_SSRS"
});
});
</script>
</div>
`
```

The Web Service URL should be set as `reportServerUrl` in the report viewer configuration. The Web Service URL can be found from the Reporting Services Configuration manager under the `Web Service`

URL section, as shown in the following image.



3. Set the report path for loading the reports from the SSRS Report Server. The report path should be in the format of `/folder name/report name`. Open the `index.html` and replace the following code example.

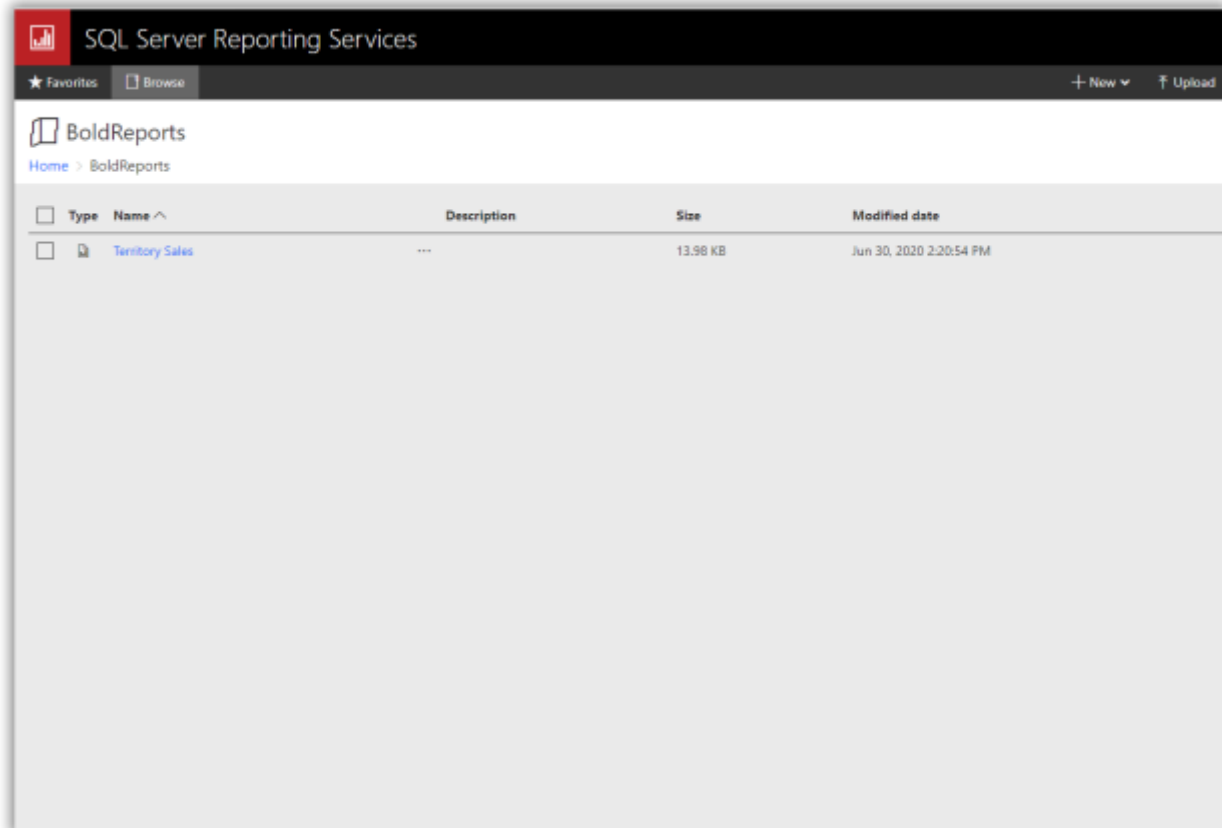
```
`js
<div style="height: 100%; width: 100%;">
<div style="height: 600px; width: 950px; min-height: 400px;" id="viewer"></div>
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: "/BoldReports/Territory Sales",
reportServerUrl: "http://<servername>/Reports_SSRS"
```

```

});
});
</script>
</div>
`

```

The report path can be found from the SSRS Report Server by navigating to the path of the report to be loaded, as shown in the following image.



Network credentials for SSRS

The network credentials are required to connect with the specified SSRS Report Server using Report Viewer. Specify the `ReportServerCredential` property in the Web API Controller `OnInitReportOptions` method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server credential
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
    "RDLReport1");
}

```

```
}
,
```

If you are facing problem to access the SSRS Report server reports, you can refer [How to provide the permission for user to access the SSRS Report Server reports](#).

Set data source credential for shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS server. If the report has any data source that uses credentials to connect with the database, then you must specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server and data source credentials
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
    "RDLReport1");
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
}
,
```

Data source credentials must be added for shared data sources that do not have credentials in the connection strings.

Build and run the application. Preview and edit the result using the following.

```
`js
$(function () {
    $("#viewer").boldReportViewer({
        reportServiceUrl: "https://demos.boldreports.com/services/api/SSRSDataSourceCredentials",
        reportPath: "/SSRSSamples/Territory Sales",
        reportServerUrl: "http://<servername>/reportserver$instanceName"
    });
});
,
```

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
//Add SSRS Report Server and data source credentials
reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
"RDLReport1");
reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "<username>", "<password>"));
}
`
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

Change data source connection string

You can change the connection string of a report data source before it is loaded in the Report Viewer. The `DataSourceCredentials` class provides the option to set and update the modified connection string as in the following code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "<username>", "<password>", "Data
Source=<instancename>;Initial Catalog=<database>"));
}
`
```

The previous code shows an option to change the connection string only, but the class provides multiple options to change data source information. To learn more about this, refer to the `DataSourceCredentials` class.

[See also](#)

[Does Bold Report Viewer use SSRS Report processing?](#)

Load SharePoint Server reports

To render SharePoint server reports set the [reportServerUrl](#), [reportPath](#), and `reportServiceUrl` properties as in the following code snippet.

```
`js
<div style="height: 100%; width: 100%;">
<div style="height: 600px; width: 950px; min-height: 400px; id="viewer"></div>
<script type="text/javascript">
$(function () {
```

```

$("#viewer").boldReportViewer({
  reportServiceUrl: "/api/SharePointReports",
  reportPath: "http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl",
  reportServerUrl: "http://<servername>/reportserver$instanceName"
});
});
</script>
</div>
`

```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

In SharePoint integrated mode, the `reportServerUrl` will be same as your site URL. The `reportPath` is relative to the Report Server URL with the file extension.

Forms credential for SharePoint server

The Forms credentials are required to connect with the specified SharePoint integrated SSRS Report Server using the Report Viewer. Specify the `ReportServerFormsCredential` property in the Web API Controller `OnInitReportOptions` method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
  //Add ReportServerFormsCredential for server
  reportOption.ReportModel.ReportServerFormsCredential = new
  BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");
}
`

```

Set data source credential for shared data sources

The shared data source credentials can be added to the `DataSourceCredentials` property to connect with the database.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
  //Add ReportServerFormsCredential and data source credentials

```

```
reportOption.ReportModel.ReportServerFormsCredential = new
BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));

}
```

Data source credentials must be added for shared data sources that do not have credentials in the connection strings.

Build and run the application. Preview and edit the result using the following.

```
`js
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "https://demos.boldreports.com/services/api/SharePointReports",
reportPath: "http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl",
reportServerUrl: "http://<servername>/reportserver$instanceName"
});
});
```

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
//Add ReportServerFormsCredential and data source credentials
reportOption.ReportModel.ReportServerFormsCredential = new
BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
}
```

Render RDLC report

The data binding support, allows you to view RDLC reports that exist on the local file system with JSON array and custom business object data collection. The following steps demonstrates how to render a RDLC report with JSON array and custom business object data collection.

Add the RDLC report `product-list.rdlc` from Bold Reports installation location to your application `Resources` folder. For more information, see [Samples and demos](#).

Bind data source at client side

1. Set the RDLC report path to the `reportPath` property.
2. Assign the `processingMode` property to `ProcessingMode.Local`.
3. Bind the JSON array collection to the `dataSources` property as shown in following code.

```
`js
<div style="height: 100%; width: 100%;">
<div style="height: 600px; width: 950px; min-height: 400px; id="viewer"></div>
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
processingMode: ej.ReportViewer.ProcessingMode.Local,
reportPath: '~/Resources/docs/product-list.rdlc',
dataSources: [{
value: [
{
ProductName: "Baked Chicken and Cheese", OrderId: "323B60", Price: 55, Category: "Non-Veg",
Ingredients: "Grilled chicken, Corn and Olives.", ProductImage: ""
},
{
ProductName: "Chicken Delite", OrderId: "323B61", Price: 100, Category: "Non-Veg", Ingredients:
"Cheese, Chicken chunks, Onions & Pineapple chunks.", ProductImage: ""
},
{
ProductName: "Chicken Tikka", OrderId: "323B62", Price: 64, Category: "Non-Veg", Ingredients: "Onions,
Grilled chicken, Chicken salami & Tomatoes.", ProductImage: ""
}},
name: "list"
]]
});
});
</script>
</div>
`
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

4. Build and run the application to view the output result.

```
{% tab demoPath="javascript-reporting/report-viewer/rdlc-report/bind-data-at-client"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`

{% endtabItem %}

{% endtab %}
```

Bind data source in Web API controller

The following steps help you to configure the Web API to render the RDLC report with business object data collection.

- Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = null;
        data = new ProductList()
```



```
{
    ProductName = "Baked Chicken and Cheese",
    OrderId = "323B60",
    Price = 55,
    Category = "Non-Veg",
    Ingredients = "grilled chicken, corn and olives.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Delite",
    OrderId = "323B61",
    Price = 100,
    Category = "Non-Veg",
    Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
```

- Set the `ProcessingMode` to `ProcessingMode.Local` and `ReportPath` to the RDLC report location.
- Bind the business object data values collection by adding new item to the `DataSources` as in the following code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ProcessingMode = ProcessingMode.Local;
    reportOption.ReportModel.ReportPath =
        System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/docs/product-list.rdlc");
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
        Value = ProductList.GetData() });
}
`
```

Here the `Name` is case sensitive and it should be same as in the data source name in the report definition.

The `Value` accepts `IList`, `DataSet`, and `DataTable` inputs.

Load report as stream

To load report as a stream, create a report stream using the `FileStream` class and assign the report stream to the `Stream` property.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string filePath = System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/docs/product-list.rdlc"); ;
    // Opens the report from application Resources folder using FileStream
    FileStream reportStream = new FileStream(filePath, FileMode.Open, FileAccess.Read);
    reportOption.ReportModel.Stream = reportStream;
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
        Value = ProductList.GetData() });
}
`
```

In the above code, `product-list.rdlc` report is loaded from the `Resources` folder location.

View Report Click

You can get the user selected parameter details when the user clicks on the **ViewReport** button in the parameter block. The **viewReportClick** event lets you to handle the **ViewReport** button click at client-side as in the following code.

```
`csharp
<script type="text/javascript">
$(function () {
$("#container").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/sales-order-detail.rdl',
viewReportClick:"onViewReportClick"
});
});
function onViewReportClick(args) {
args[0] = ({ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'], nullable: false });
}
</script>
`
```

The model property in the event argument has the details of current processing report model.

Render subreport

You can display another report inside the body of a main report using the Report Viewer. The following steps helps you to customize the subreport properties such as data source, report path, and parameters.

- Add the sub report and main reports to your application **Resources** folder. In this tutorial, using the already created reports. Refer to the [Create RDL Report section](#) or [Create RDLC Report section](#) section for creating new reports.

Download the **side-by-side-main-report.rdl**, **side-by-side-sub-report.rdl** reports from [here](#). Also, you can add the report from Syncfusion installation location. For more information, see [Samples and demos](#). The reports used from installed location, requires **NorthwindIO_Reports.sdf** database to run, so add it to your application.

- Set the **reportPath** and **reportServiceUrl** properties of the Report Viewer as in following code snippet.

```
`js
<script type="text/javascript">
$(function () {
```

```

$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/side-by-side-main-report.rdl'
});
});
</script>
`

```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

- Build and run the application. Preview and edit the result using the following.

```

{% tab demoPath="javascript-reporting/report-viewer/subreport/preview-subreport"
files="index.html,index.js" %}
{% tabItem header="index.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
{% endtabItem %}
{% endtab %}

```

Change subreport path

To change the subreport file path, set the **ReportPath** property of **SubReportModel** in the **OnInitReportOptions** method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
if (reportOption.SubReportModel != null)
{
reportOption.SubReportModel.ReportPath =
System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/docs/sub-report-detail.rdl");
}
}
}

```

,

Set subreport parameter

You can change the parameter default values of a subreport in the Web API Controller

`OnReportLoaded`, method as given in the following code snippet.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.Parameters = new BoldReports.Web.ReportParameterInfoCollection();
        reportOption.SubReportModel.Parameters.Add(new BoldReports.Web.ReportParameterInfo()
        {
            Name = "SalesPersonID",
            Values = new List<string>() { "2" }
        });
    }
}
```

,

Modify subreport data source connection string

You can change the credential and connection information of the data sources used in the subreport using the `SubReportModel` in `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSourceCredentials = new
        List<BoldReports.Web.DataSourceCredentials>();
        reportOption.SubReportModel.DataSourceCredentials.Add(new
        BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
        Catalog=<database>;user id=<username>;password=<password>"));
    }
}
```

Set subreport data source

You can bind local business object data source collection only for RDLC reports. To specify data source of a RDLC subreport, set the `ReportDataSource` property in `OnReportLoaded` method.

The RDL report has the connection information in report definition itself, so no need to bind data source.

1. Add the RDLC sub report and main reports to your application `Resources` folder. You can downloaded it from [here](#).
2. Set the `reportPath` and `reportServiceUrl` properties of the Report Viewer as in following code snippet.

```
`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/SubreportDataSources",
processingMode: ej.ReportViewer.ProcessingMode.Local,
reportPath: '~/Resources/docs/product-list-main.rdlc'
});
});
</script>
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

3. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
public class ProductList
{
public string ProductName { get; set; }
public string OrderId { get; set; }
public double Price { get; set; }
public string Category { get; set; }
public string Ingredients { get; set; }
public string ProductImage { get; set; }
```

```
public static IList GetData()
{
    List<ProductList> datas = new List<ProductList>();
    ProductList data = null;
    data = new ProductList()
    {
        ProductName = "Baked Chicken and Cheese",
        OrderId = "323B60",
        Price = 55,
        Category = "Non-Veg",
        Ingredients = "grilled chicken, corn and olives.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Delite",
        OrderId = "323B61",
        Price = 100,
        Category = "Non-Veg",
        Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Tikka",
        OrderId = "323B62",
        Price = 64,
        Category = "Non-Veg",
        Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
        ProductImage = ""
    };
};
```

```

datas.Add(data);
return datas;
}
}
`

```

4. Bind the business object data values collection to subreport by adding new item to the **DataSources** as in the following code snippet.

```

`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'product-list.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
"list", Value = ProductList.GetData() });
    }
}
`

```

The data source name is case sensitive, it should be same as in the report definition.

Load subreport stream

To load subreport as stream, set the **Stream** property in **OnInitReportOptions** method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        // Opens the report from application Resources folder using FileStream and loads the sub report stream.
        FileStream reportStream = new
        FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/docs/product-list.rdlc"),
        FileMode.Open, FileAccess.Read);
        reportOption.SubReportModel.Stream = reportStream;
    }
}
`

```



```

}
}
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'Product List.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
        "list", Value = ProductList.GetData() });
    }
}

```

Report parameters

Provides property options to pass or set report parameters default values at run-time using the [parameters](#) property. You can set the report parameters while creating the Report Viewer control in a script or in the Web API Controller.

In this tutorial, the `sales-order-dtail.rdl` report is used, and it can be downloaded from [here](#).

Set parameter at client

The [parameters](#) property takes the JSON array value input with parameter details.

1. Set the default value data to the [values](#) property and name of the report parameter to the [name](#) property.

The parameter name is case sensitive, it should be same as available in the report definition.

2. The following code example illustrates how to set report parameter in the script.

```

`js
<script type="text/javascript">
$(function () {
    $("#viewer").boldReportViewer({
        reportServiceUrl: "/api/ReportViewer",
        reportPath: '~/Resources/docs/sales-order-detail.rdl',
        parameters: [{ name: 'SalesOrderNumber', labels: ['SO50751'], values: ['SO50751'] }]
    })

```

```
});
});
</script>
`
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

- Build and run the application. Preview and edit the result using the following.

```
{% tab demoPath="javascript-reporting/report-viewer/report-parameters/set-parameter-at-client"
files="index.html,index.js" %}
{% tabItem header="index.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
{% endtabItem %}
{% endtab %}
```

Set parameters in Web API Controller

To set parameter default value in Web API Controller, use the following code in the `OnReportLoaded` method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    List<BoldReports.Web.ReportParameter> userParameters = new
    List<BoldReports.Web.ReportParameter>();
    userParameters.Add(new BoldReports.Web.ReportParameter()
    {
        Name = "SalesOrderNumber",
        Values = new List<string>() { "SO50756" }
    });
    reportOption.ReportModel.Parameters = userParameters;
}
```

The Report Parameters name should be case sensitive

Get report parameter

The **ReportHelper** class provides methods that help you to get the report parameters used in the report. The following helper methods used to get parameter with or without values.

Methods | Description

minDateTime | Specify minimum range value of a date parameter

maxDateTime | Specify maximum range value of a date parameter

Refer to the following code sample to set data range using the Report Viewer component API's.

```
{% tab demoPath="javascript-reporting/report-viewer/report-parameters/set-date-range"
files="index.js,index.html" %}
{% tabItem header="index.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
{% endtabItem %}
{% endtab %}
```

The above code sets date range for all the date parameters used in the report.

To set different date range for each date parameter used in the report, register the event **beforeParameterAdd** and specify range based on parameter name as in below code sample.

```
{% tab demoPath="javascript-reporting/report-viewer/report-parameters/set-date-range"
files="beforeparameteradd.js,beforeparameteradd.html" %}
{% tabItem header="beforeparameteradd.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="beforeparameteradd.js"></script>
</body>
`
{% endtabItem %}
{% endtab %}
```

Set date time display format for date parameter

The properties [dateTimeFormat](#) and [timeDisplayFormat](#) in the `parameterSettings` are used to set date and time format to be displayed in the DateTimePicker control in a report.

Format | Display in DateTimePicker

Short Date and Time - d/M/yy h:mm tt | 9/12/2014 2:04 PM

Medium Date - d MMM yy h:mm tt | 12 Sep 14 2:04: PM

Full Date and short time - dddd, MMMM dd, yyyy HH:mm tt | Friday, September 12,2014 2:04 PM

Full Date and Long Time - dddd, MMMM dd, yyyy HH:mm:ss tt | Friday, September 12,2014 2:04:00 PM

UTC - yyyy-MM-dThh:mm:ssz | 2014-09-12T2:04:00+5

Refer to the following code sample to set date and time value to be display, using the Report Viewer component API's.

```
{% tab demoPath="javascript-reporting/report-viewer/report-parameters/set-datetimepicker-display-format" files="index.js,index.html" %}
```

```
{% tabItem header="index.html" %}
```

```
`html
```

```
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
```

```
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
```

```
<script src="index.js"></script>
```

```
</body>
```

```
,
```

```
{% endtabItem %}
```

```
{% endtab %}
```

The above code sets date and time value to be display for all the date parameters used in the report.

To set different date and time value to be display for each date parameter used in the report, register the event `beforeParameterAdd` and specify date and time value based on parameter name as in below code sample.

```
{% tab demoPath="javascript-reporting/report-viewer/report-parameters/set-datetimepicker-display-format" files="beforeparameteradd.js,beforeparameteradd.html" %}
```

```
{% tabItem header="beforeparameteradd.html" %}
```

```
`html
```

```
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
```

```
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
```

```
<script src="beforeparameteradd.js"></script>
```

```
</body>
```

```
,
```

```
{% endtabItem %}
```

```
{% endtab %}
```

Change the Parameter drop-down height and width

The `parameterSettings` helps you to change the height and width of the parameter available in parameter panel.

```
`html
```

```
<div id="viewer"></div>
```

```
<script>
```

```
$("#viewer").boldReportViewer({
```

```
parameterSettings: {
```

```
popupHeight: "200px",
```

```
popupWidth: "150px",
```

```
}
```

```
});
```

```
</script>
```

```
</html>
```

```
,
```

Hide a Parameter scroller

The `enableparameterblockscroller` helps you to hide the scrollbar in parameter panel.

```
`html
```

```
<div id="viewer"></div>
```

```
<script>
```

```
$("#viewer").boldReportViewer(
```

```
{
```

```
enableParameterBlockScroller: false
```

```
});
```

```
</script>
```

```
</html>
```

```
,
```

Hide a Parameter Pane on load

The `parameterSettings` helps you to hide and show the parameter block.

```
`html
```

```

<div id="viewer"></div>
<script>
$("#viewer").boldReportViewer(
{
parameterSettings: { hideParameterBlock: true}
});
</script>
</html>
`

```

Change the Parameter Item Width and Label Width

The `parameterSettings` helps you to change the parameter Item width and label width.

```

`html
<div id="viewer"></div>
<script>
$("#viewer").boldReportViewer(
{
parameterSettings: {
itemWidth: '250px',
labelWidth: 'auto'
}
});
</script>
</html>
`

```

Access the hidden or internal parameter information

The `accessInternalValue` property in the `parameterSettings` helps you to expose the `hidden` or `internal` report parameter information used in report to the user.

```

`html
<div id="viewer"></div>
<script>
$("#viewer").boldReportViewer(
{
parameterSettings:{ accessInternalValue: true }
});

```

```
</script>
```

```
</html>
```

```
,
```

[Set the report parameter visibility in Web API Controller](#)

The `Hidden` property of `ReportParameter` allows you to show or hide the parameter at the top of the report viewer panel. The following code example shows hiding a report parameter in the Web API controller's `OnReportLoaded` method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    var reportParameters = ReportHelper.GetParameters(jsonArray, this);
    List<BoldReports.Web.ReportParameter> modifiedParameters = new
    List<BoldReports.Web.ReportParameter>();
    if (reportParameters != null)
    {
        foreach (var rptParameter in reportParameters)
        {
            modifiedParameters.Add(new BoldReports.Web.ReportParameter()
            {
                Name = rptParameter.Name,
                Hidden = true
            });
        }
        reportOption.ReportModel.Parameters = modifiedParameters;
    }
}
,
```

[See Also](#)

[Parameter Custom Properties](#)

Report interaction events

You can handle the report interaction events with reports using the following events.

- `ReportLoaded`
- `ReportError`

- ShowError
- Drill through
- Hyperlink

Report loaded

The [reportLoaded](#) event fires once the report loading is completed and ready to start the processing. You can handle the event and specify data source, parameters at client-side. The following sample code loads a report by assigning report data source input in the [reportLoaded](#) event.

In this tutorial, [product-list.rdlc](#) report is used, you can add the report from Bold Reports installation location. For more information, see [Samples and demos](#).

```
{% tab demoPath="javascript-reporting/report-viewer/report-interaction-events/report-loaded"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`

{% endtabItem %}

{% endtab %}
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

Report error

When an error occurs in the report processing, it raises the [reportError](#) event. You can handle the event and show the details in your custom dialog instead of component default error detail interface.

```
{% tab demoPath="javascript-reporting/report-viewer/report-interaction-events/report-error"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`

{% endtabItem %}
```



```
{% endtab %}
```

To suppress the default error dialog, set the cancel argument to true.

Show error

The [showError](#) event invoked whenever the user clicks a report item that contains an error in processing. It provides detailed information about the cause of the error. You can hide the default dialog and show your customized dialog.

```
{% tab demoPath="javascript-reporting/report-viewer/report-interaction-events/show-error"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
```

```
{% endtabItem %}
```

```
{% endtab %}
```

Drill through

When a drill through item is selected in a report, it invokes the [drillThrough](#) event. You can change the drill through arguments such as report parameter and data sources. The following sample code can be used to change the drill through report name and set the parameter value before the drill through report is rendered.

```
{% tab demoPath="javascript-reporting/report-viewer/report-interaction-events/drill-through"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
```

```
{% endtabItem %}
```

```
{% endtab %}
```

Hyperlink

The [hyperlink](#) event occurs when the user clicks a hyperlink in a report, before the hyperlink is followed. The following sample code redirects to a new custom URL and cancels the component default action.

```
{% tab demoPath="javascript-reporting/report-viewer/report-interaction-events/hyperlink"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">

<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>

<script src="index.js"></script>

</body>

`

{% endtabItem %}

{% endtab %}
```

Handle post actions

Report processing actions are sent in an Ajax request to exchange data with the Web API service. You can handle post actions event to customize the Ajax requests.

- `AjaxBeforeLoad`
- `AjaxSuccess`
- `AjaxError`

AjaxBeforeLoad

This event can be triggered before an Ajax request is sent to the Report Viewer Web API service. It allows you to set additional headers, custom data in the Ajax request. The following code sample demonstrates adding custom authorization header and passing default parameter values to service.

Add custom header in Ajax request

Initialize the [ajaxBeforeLoad](#) event in the script and add the authorization token to the `headers` property

```
`js

<script type="text/javascript">

$(function () {

$("#viewer").boldReportViewer({

reportServiceUrl: "/api/ReportViewer",

reportPath: '~/Resources/docs/sales-order-detail.rdl',

ajaxBeforeLoad: onAjaxRequest

});

});

function onAjaxRequest(args) {
```

```
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
}
</script>
`
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded from [here](#).

Get the custom header value from the `HttpContext` header collection using the key name specified at client-side.

```
`csharp
string authenticationHeader;
public object PostReportAction(Dictionary<string, object> jsonResult)
{
    if (jsonResult != null)
    {
        //Get client side custom ajax header and store in local variable
        authenticationHeader = HttpContext.Current.Request.Headers["Authorization"];
        //Perform your custom validation here
        if (authenticationHeader == "")
        {
            return new Exception("Authentication failed!!!");
        }
        else
        {
            return ReportHelper.ProcessReport(jsonResult, this);
        }
    }
    return null;
}
`
```

Perform your own action to validate the header values.

Pass custom data in Ajax request

Use the `data` property to set custom data to the server in the Ajax request. In the following code sample, parameter values are passed to the server-side.

```
`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/sales-order-detail.rdl',
ajaxBeforeLoad: onAjaxRequest
});
});
function onAjaxRequest(args) {
//Passing custom data to server
var customerID = "CI0021";
args.data = customerID;
}
</script>
`
```

The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates to change the datasource connection strings based on Customer ID in the `OnInitReportOptions` method.

```
`csharp
string customerID = null;
public object PostReportAction(Dictionary<string, object> jsonResult)
{
if (jsonResult != null)
{
if (jsonResult.ContainsKey("customData"))
{
//Get client side custom data and store in local variable.
customerID = jsonResult["customData"].ToString();
}
}
return ReportHelper.ProcessReport(jsonResult, this);
}
[NonAction]
```

```

public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (customerID != null)
    {
        if(customerID.Contains("CI0021"))
        {
            //If you are changing the connection string based on customer id then could you please change the
            connection string as below.

            //reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

            reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
            Catalog=<database>;"));
        }
        else if(customerID.Contains("CI0022"))
        {
            //If you are changing the connection string based on customer id then could you please change the
            connection string as below.

            //reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

            reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
            Catalog=<database>;"));
        }
    }
}

```

AjaxSuccess

To perform custom action or show user defined message, use the [ajaxSuccess](#) event on the successful Ajax request.

```

`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/sales-order-detail.rdl',

```

```

ajaxBeforeLoad: onAjaxRequest,
ajaxSuccess: onAjaxSuccess
});
});
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
//Passing custom parameter data to server
args.data = [{ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'] }];
}
function onAjaxSuccess(args) {
//Perform your custom success message here
alert("Ajax request success!!!");
}
</script>
`

```

AjaxError

The [ajaxError](#) event is called, if an error occurred with the request, you can display the customized error detail in the event method.

```

`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/sales-order-detail.rdl',
ajaxBeforeLoad: onAjaxRequest,
ajaxError: onAjaxFailure
});
});
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
//Passing custom parameter data to server
args.data = [{ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'] }];
}
function onAjaxFailure(args) {

```

```

alert("Status: " + args.status + "\n" +
"Error: " + args.responseText);
}
</script>
`

```

You can never have both an error and a success callback with a request.

[Change Report Viewer default WEB API action with custom endpoint](#)

The `actionName` event argument allows to change the methods of `IReportController` to custom WEB API action endpoints.

[Change report processing endpoint](#)

Create a new Web Api action method in Report Viewer API controller as in the following code snippet,

```

`csharp
public object PostReportCustomAction(Dictionary<string, object> jsonResult)
{
    return ReportHelper.ProcessReport(jsonResult, this);
}
`

```

The custom method must have the `Dictionary<TKey, TValue>` argument and add code `ReportHelper.ProcessReport` for processing the report.

Register the event `ajaxBeforeLoad` in your html page and set the newly created name to `actionName` property as in the below code snippet.

```

`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/sales-order-detail.rdl',
ajaxBeforeLoad: onAjaxRequest
});
});
function onAjaxRequest(args) {
args.actionName = "PostReportCustomAction";
}
</script>

```

Change export action endpoint

Create a new Web Api action method in Report Viewer API controller as in the following code snippet,

```
`csharp
public object ExportReportCustomAction()
{
    return ReportHelper.ProcessReport(null, this);
}
```

The custom method must have the code `ReportHelper.ProcessReport` for exporting the report.

Register the event `onExportProgressChanged` in your html page and set the newly created name to `actionName` property as in the below code snippet.

```
`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
    reportServiceUrl: "/api/ReportViewer",
    reportPath: '~/Resources/docs/sales-order-detail.rdl',
    exportProgressChanged: "onExportProgressChanged"
});
});
function onExportProgressChanged(args) {
if(args.stage === "exportStarted"){
    args.actionName = "ExportReportCustomAction";
}
}
}</script>
```

Error logging in JavaScript Report Viewer

If an error occurred in report processing, JavaScript Report Viewer displays short messages about the error in view. You need to configure the `ApiController` to save all logs, stack trace, and error information.

This section explains how to log the detailed error information to your JavaScript application.

This section requires a JavaScript Report Viewer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

1. In Solution Explorer, open the Report Viewer Controller file.
2. Inherit the `IReportLogger` interface and implement the interface methods.

```
`csharp
public class ReportViewerController : ApiController, IReportController, IReportLogger
{
    public void LogError(string message, Exception exception, MethodBase methodType, ErrorType
    errorType)
    {
        throw new NotImplementedException();
    }

    public void LogError(string errorCode, string message, Exception exception, string errorDetail, string
    methodName, string className)
    {
        throw new NotImplementedException();
    }
}
```

3. Create a method in `ReportViewerController` to write the error text into application folder.

```
`csharp
internal void WriteLogs(string errorMessage)
{
    string filePath = HttpContext.Current.Server.MapPath("/App_Data/ErrorDetails.txt");
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
        writer.WriteLine(errorMessage);
    }
}
```

4. Invoke the newly created function in `LogError` as follows.

```
`csharp
public void LogError(string message, Exception exception, MethodBase methodType, ErrorType
errorType)
{
    WriteLogs(string.Format("Error Message: {0} \n Stack Trace: {1}", message, exception.StackTrace));
}

public void LogError(string errorCode, string message, Exception exception, string errorDetail, string
methodName, string className)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2} \n Stack Trace:
{3}", className, methodName, errorDetail, exception.StackTrace));
}
`
```

In cases of any issues faced in the report rendering, share the log file to our technical support team to get assistance on that.

5. The final controller is given as follows, you can replace it in your application.

```
`csharp
public class ReportViewerController : ApiController, IReportController, IReportLogger
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }

    // Get action for getting resources from the report
    [System.Web.Http.ActionName("GetResource")]
    [AcceptVerbs("GET")]
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }

    // Method that will be called when initialize the report options before start processing the report
    [NonAction]
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    // You can update report options here
}

// Method that will be called when reported is loaded
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    // You can update report options here
}

public void LogError(string message, Exception exception, MethodBase methodType, ErrorType
errorType)
{
    WriteLogs(string.Format("Error Message: {0} \n Stack Trace: {1}", message, exception.StackTrace));
}

public void LogError(string errorCode, string message, Exception exception, string errorDetail, string
methodName, string className)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2} \n Stack Trace:
{3}", className, methodName, message, exception.StackTrace));
}

internal void WriteLogs(string errorMessage)
{
    // Error details text file path location
    string filePath = HttpContext.Current.Server.MapPath("/App_Data/ErrorDetails.txt");
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
        writer.WriteLine(errorMessage);
    }
}
,
```

Print report

The Report Viewer provides print report option in the toolbar to print a copy of the report. The page setup dialog allows you to set the paper size or other page setup properties. To see print margins, click [Print Layout](#) on the toolbar.

The values allow you to set in the Page Setup dialog box for current session only. When you close the report and reopen it, it will have the default values again. The default values for the page setup dialog come from the report properties, which are set in the design view.

View report in print mode

Print margins are displayed in the Print Layout only, to view report in print mode by default, set the [printMode](#) property to true.

```
{% tab demoPath="javascript-reporting/report-viewer/printing/view-report-in-print-mode"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`

{% endtabItem %}

{% endtab %}
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

By default, the Report Viewer renders report in normal layout in which the print margins are not displayed.

Print in new page

To open the print in a new tab of the current browser, set the property [printOption](#) to **NewTab**. By default, it shows the print dialog in the same page.

```
{% tab demoPath="javascript-reporting/report-viewer/printing/print-in-new-page"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
```

```
{% endtabItem %}
```

```
{% endtab %}
```

The pop-up blocker must be enabled for the page to open the print view in new tab.

Set page orientation and paper size

You can specify the print page paper size, orientation at client-side to change page setup properties by setting the [pageSettings](#) property.

```
{% tab demoPath="javascript-reporting/report-viewer/printing/set-page-orientation-and-paper-size"
files="index.html,index.js" %}
```

```
{% tabItem header="index.html" %}
```

```
`html
```

```
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
```

```
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
```

```
<script src="index.js"></script>
```

```
</body>
```

```
{% endtabItem %}
```

```
{% endtab %}
```

Set report margin

To set margin values to the report page setup, use the property [margins](#) and specify the value to top, right, bottom, and left.

```
{% tab demoPath="javascript-reporting/report-viewer/printing/set-report-margin"
files="index.html,index.js" %}
```

```
{% tabItem header="index.html" %}
```

```
`html
```

```
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
```

```
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
```

```
<script src="index.js"></script>
```

```
</body>
```

```
{% endtabItem %}
```

```
{% endtab %}
```

The values set in the margin property is considered as inches input.

Set page height and width

To set height and width values to the report page setup, use the [height](#), and [width](#) properties.

```
{% tab demoPath="javascript-reporting/report-viewer/printing/set-page-height-and-width"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`

{% endtabItem %}

{% endtab %}
```

The values set in the height and width property is considered as inches input.

Print report with images

When the report has more images, the browser will send the report stream to the print dialog before the images are completely loaded. To load the print report stream with complete images, you should set the `EmbedImageData` property to true in `OnInitReportOptions` as shown in the following code.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.EmbedImageData = true;
}
`
```

Replace the following code sample in client side HTML file.

```
`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer(
{
    reportServiceUrl: "/api/PrintWithImages",
    reportPath: '~/Resources/docs/product-details.rdl'
});
});
</script>
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

In this tutorial, the `product-details.rdl` report is used, and it can be downloaded from [here](#).

External styles in report printing

While printing report, the external styles are used in the application overrides printable page style and prints output with incorrect alignments. To avoid the external script overriding, set the `isStyleLoad` property to false, which will print the page using only the Report Viewer styles.

```
`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer(
{
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/product-details.rdl',
reportPrint: "onReportPrint"
});
});
function onReportPrint(args) {
args.isStyleLoad = false;
}
</script>
```

Show print progress

Report Viewer provides events that help you to show the progress information when the printing takes a long time to complete.

1. Set the [printProgressChanged](#) in Report Viewer initialization.
2. Implement the function and add code samples to show a custom message based on the print progress status as shown in the following code snippet.

```
{% tab demoPath="javascript-reporting/report-viewer/printing/show-print-progress"
files="index.html,index.js" %}
{% tabItem header="index.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
```

```
<script src="index.js"></script>
```

```
</body>
```

```
,
```

```
{% endtabItem %}
```

```
{% endtab %}
```

Remove empty spaces in printing

The extra blank page is created when the Body of your report is too wide for your page. If you want the report to appear on a single page, all the content within the report body must fit on the physical page and the body width should be lesser or equal to the following formula:

Body Width <= Page Width - (Left Margin + Right Margin)

For more details on designing a report to remove the empty pages in the report, refer to the knowledge base article of [report page sizing](#).

Export report

The Report Viewer provides events and properties to control and customize the report export functionality.

Export event handling

You can show the progress information, when the exporting takes long time to complete using the [exportProgressChanged](#) event.

1. Set the [exportProgressChanged](#) in Report Viewer initialization.
2. Implement the function and replace the following code samples to show a custom message based on the progress stage.

```
{% tab demoPath="javascript-reporting/report-viewer/export/export-event-handling"
```

```
files="index.html,index.js" %}
```

```
{% tabItem header="index.html" %}
```

```
`html
```

```
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
```

```
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
```

```
<script src="index.js"></script>
```

```
</body>
```

```
,
```

```
{% endtabItem %}
```

```
{% endtab %}
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

Change Excel and Word export format

This allows you to change the default file format to any other file format using the [excelFormat](#) and [wordFormat](#) properties. The following code sample changes the default versions.

```
{% tab demoPath="javascript-reporting/report-viewer/export/change-excel-and-word-export-format"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">

<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>

<script src="index.js"></script>

</body>

`

{% endtabItem %}

{% endtab %}
```

Hide specific export type for report

Show or hide the default export types available in the component using the [exportOptions](#) property. The following code hides the HTML export type from the default export options.

```
{% tab demoPath="javascript-reporting/report-viewer/export/hide-specific-export-type-for-report"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">

<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>

<script src="index.js"></script>

</body>

`

{% endtabItem %}

{% endtab %}
```

PDF export options

The `PDFOptions` provides properties to manage PDF export behaviors. You have to set the properties in the `OnInitReportOptions` method of the Web API service.

Export with complex scripts

To export reports with the complex scripts, set the `ComplexScript` property of `PDFOptions` instance to true.

```
`csharp

[NonAction]
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
    {
        EnableComplexScript = true
    };
}
```

PDF Conformance

You can export the report as **PDF/A-1b** document by specifying the conformance level **PdfConformanceLevel.Pdf_A1B** in the **PdfConformanceLevel** property.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
    {
        PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B
    };
}
```

Add custom PDF fonts

This allows you to have custom fonts in the PDF exported document by adding the font streams to **Fonts** collection in **PDFOptions** instance.

1. Add the font **.ttf** files into your application **Resources** folder.
2. In the Solution Explorer, open the properties of the font file and set the property Copy to Output Directory as Copy always.
3. Initialize the **Font** collection and add the font stream to it.

The key value provided in the font collection should be same as in the report item font property.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
```

```
{
//Load Missing font stream
Fonts = new Dictionary<string, System.IO.Stream>
{
{ "Segoe UI",
System.IO.File.OpenRead(System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/docs/font_symbols.ttf")) }
}
};
}
```

Any fonts used in the report definition that is not installed or available in the local system, then you must load the font stream. In the above code, loaded `font_symbols` font stream to export `Sales Order Detail.rdl` report as symbols.

Word export options

The `WordOptions` provides properties to manage Word document export behaviors.

Word document type

You can save the report to the required document version by setting the `FormatType` property.

```
`csharp
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
FormatType = BoldReports.Writer.WordFormatType.Docx
};
`
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` as `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
LayoutOption = BoldReports.Writer.WordLayoutOptions.TopLevel,
ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
Bottom = 0.5f,
```

```
Top = 0.5f
```

```
}
```

```
};
```

```
,
```

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior.

The table in word document is not a stand-alone object, if you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, added an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without a password. The following are the types of protection.

1. **AllowOnlyComments**: You can add or modify only the comments in the Word document.
2. **AllowOnlyFormFields**: You can modify the form field values in the Word document.
3. **AllowOnlyRevisions**: You can accept or reject the revisions in the Word document.
4. **AllowOnlyReading**: You can only view the content in the Word document.
5. **NoProtection**: You can access or edit the Word document contents as normally.

```
`csharp
```

```
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
```

```
{
```

```
ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading
```

```
};
```

```
,
```

Excel export options

The **ExcelOptions** provides properties to manage Excel document export behaviors.

Excel document type

You can save the report to the required excel version by setting the **ExcelSaveType** property.

```
`csharp
```

```
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
```

```
{
```

```
ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013
```

```
};
```

```
,
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` as `IgnoreCellMerge`.

```
`csharp
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
};
`
```

Protecting Excel document from editing

You can restrict the Excel document from editing either by providing the `ExcelSheetProtection` or enabling the `ReadOnlyRecommended` properties.

```
`csharp
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    ReadOnlyRecommended = true,
    ExcelSheetProtection = ExcelSheetProtection.DeletingColumns
};
`
```

PowerPoint export options

You can save the report to the required PowerPoint version by setting the `FormatType` property.

```
`csharp
reportOption.ReportModel.PPTOptions = new BoldReports.Writer.PPTOptions()
{
    FormatType = BoldReports.Writer.PPTSaveType.PowerPoint2013
};
`
```

CSV export options

The `CsvOptions` allows you to change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document.

```
`csharp
reportOption.ReportModel.CsvOptions = new BoldReports.Writer.CsvOptions()
{
    Encoding = System.Text.Encoding.Default,
    FieldDelimiter = ",",

```

```
UseFormattedValues = false,  
Qualifier = "#",  
RecordDelimiter = "@",  
SuppressLineBreaks = true,  
FileExtension = ".txt"  
};  
`
```

HTML export options

You can hide the separator added at the end of each page by setting the `HidePageSeparator` property to true.

```
`csharp  
reportOption.ReportModel.HTMLOptions = new BoldReports.Writer.HTMLOptions()  
{  
    HidePageSeparator = true  
};  
`
```

Password protect exported document

This allows you to protect the exported document such as PDF, Word, Excel, and PowerPoint from unauthorized users by encrypting the document using encryption password. The following code snippet illustrates how to encrypt the exported document with the user defined password.

```
`csharp  
[NonAction]  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
    //PDF encryption  
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions();  
    reportOption.ReportModel.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity()  
    {  
        UserPassword = "password"  
    };  
    //Word encryption  
    reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()  
    {  
        EncryptionPassword = "password"  
    }  
};  
`
```

```
};
//Excel encryption
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    PasswordToModify = "password",
    PasswordToOpen = "password"
};
//PPT encryption
reportOption.ReportModel.PPTOptions = new BoldReports.Writer.PPTOptions()
{
    EncryptionPassword = "password"
};
}
```

Password protection is not supported for HTML export format.

Toolbar customization

You can hide the component toolbar to show customized user interface or to customize the toolbar icons and elements appearances using the templates and Report Viewer toolbar customization properties.

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded from [here](#). You can add the reports from Syncfusion installation location. For more information, see [Samples and demos](#).

Hide toolbar items

To hide toolbar items, set the [toolbarSettings](#) property. The following code can be used to remove the parameter option from the toolbar and hide the parameter block.

Similarly, you can show or hide all other toolbar options with the help of [toolbarSettings.items](#) enum.

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/hide-parameter-block"
files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
```

```
{% endtabItem %}
```

```
{% endtab %}
```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

The following code sample hides the print options from the toolbar items.

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/hide-toolbar-items"
files="index.html,index.js" %}
```

```
{% tabItem header="index.html" %}
```

```
`html
```

```
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
```

```
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
```

```
<script src="index.js"></script>
```

```
</body>
```

```
,
```

```
{% endtabItem %}
```

```
{% endtab %}
```

Enable stop option in toolbar

To enable stop option in toolbar, set the [toolbarSettings.items](#) property to `ej.ReportViewer.ToolbarItems.All`. The following code can be used to enable stop option in toolbar.

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/enable-stop-option-in-
toolbar" files="index.html,index.js" %}
```

```
{% tabItem header="index.html" %}
```

```
`html
```

```
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
```

```
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
```

```
<script src="index.js"></script>
```

```
</body>
```

```
,
```

```
{% endtabItem %}
```

```
{% endtab %}
```

Enable export setup option in toolbar

To enable export setup option in toolbar, set the [toolbarSettings.items](#) property to `ej.ReportViewer.ToolbarItems.All`. The following code can be used to enable export setup option in toolbar.

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/enable-stop-option-in-
toolbar" files="index.html,index.js" %}
```



```
{% tabItem header="index.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
{% endtabItem %}
{% endtab %}
```

Enable search text option in toolbar

To enable search text option in toolbar, set the [toolbarSettings.items](#) property to `ej.ReportViewer.ToolbarItems.All`. The following code can be used to enable search text option in toolbar.

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/enable-stop-option-in-
toolbar" files="index.html,index.js" %}
{% tabItem header="index.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
{% endtabItem %}
{% endtab %}
```

Hide toolbar

To hide the Report Viewer toolbar set the [showToolbar](#) property to false.

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/hide-toolbar"
files="index.html,index.js" %}
{% tabItem header="index.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
`
```

```
{% endtabItem %}
```

```
{% endtab %}
```

Decide or hide the export option

The Report Viewer provides the [exportOptions](#) property to show or hide the default export types available in the component. The following code hides the HTML export type from the default export options.

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/decide-or-hide-the-
export-option" files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>

<script src="index.js"></script>

</body>
`

{% endtabItem %}

{% endtab %}
```

Add custom items to the export drop-down

To add custom items to the export drop-down available in the Report Viewer toolbar, use the property [customItems](#) and provide the JSON array of collection input with the `index`, `cssClass` name, and `value` properties. Register the `exportItemClick` event to handle the custom item actions as given in following code snippet..

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/add-custom-items-to-the-
export-drop-down" files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>

<script src="index.js"></script>

</body>
`

{% endtabItem %}

{% endtab %}
```

Add custom toolbar item

You can add custom items to Report Viewer toolbar using the [toolbarSettings](#) property. You must register the `toolbarItemClick` event to handle the newly added custom items action.

Add custom item to exiting toolbar group

To add a custom item to existing toolbar group use the property [customItems](#) in `toolbarSettings` and provide the JSON array of collection input with the `groupIndex`, `index`, `itemType`, `cssClass` name, and `tooltip` properties as given in following code snippet.

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/add-custom-toolbar-item/add-custom-item-to-exiting-toolbar-group" files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>

<script src="index.js"></script>

</body>
`

{% endtabItem %}

{% endtab %}
```

Add new toolbar group

To add new toolbar group and custom items to it, use the property [customGroups](#) in `toolbarSettings` and provide the JSON array of collection input with the `groupIndex`, `items` properties. The `items` must have the properties `itemType`, `cssClass` and `tooltip` as given in following code snippet.

```
{% tab demoPath="javascript-reporting/report-viewer/toolbar-customization/add-custom-toolbar-item/add-new-toolbar-group" files="index.html,index.js" %}

{% tabItem header="index.html" %}

`html

<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>

<script src="index.js"></script>

</body>
`

{% endtabItem %}

{% endtab %}
```

Custom Actions

This section explains you the steps required to add user defined buttons in Report Viewer toolbar and invoke custom actions.

Send a report as an email attachment

This topic describes steps required to create custom email option and share a report as an email attachment to other users.

Add email button in Report Viewer

1. Create email button option in the toolbar using the [customItems](#) property with the values such as `groupIndex`, `index`, `itemType`, `cssClass`, `tooltip`, `toolBarItemClick` event to fire when you click the button.
2. Access the Report Viewer model and create a JSON array for sending requests to the Web API server. You can use the following codes for creating the event with custom action.

```
`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/sales-order-detail.rdl',
toolbarSettings: {
showToolbar: true,
items: ej.ReportViewer.ToolbarItems.All & ~ej.ReportViewer.ToolbarItems.Print,
customItems: [{
groupIndex: 1,
index: 1,
type: 'Default',
cssClass: "e-icon e-mail e-reportviewer-icon",
id: 'E-Mail',
tooltip: {
header: 'E-Mail',
content: 'Send rendered report as mail attachment'
}
}]
},
toolBarItemClick: 'ontoolBarItemClick'
});
});
//Toolbar click event handler
function ontoolBarItemClick(args) {
if (args.value == "E-Mail") {
```

```

var proxy = $('#viewer').data('boldReportViewer');
var Report = proxy.model.reportPath;
var lastIndex = Report.lastIndexOf("/");
var reportName = Report.substring(lastIndex + 1);
var requrl = proxy.model.reportServiceUrl + '/SendEmail';
var _json = {
  exportType: "PDF", reportViewerToken: proxy._reportViewerToken, ReportName: reportName
};
$.ajax({
  type: "POST",
  contentType: "application/json; charset=utf-8",
  url: requrl,
  data: JSON.stringify(_json),
  dataType: "json",
  crossDomain: true
})
}
}
</script>
`

```

You can view the Web API service used in the above code from the [Reporting Service](#) git hub location. For more information, see [Samples and demos](#).

Create custom email action

- Create a new action method `SendEmail` in the Web API service.
- Export the report to the required type using `ReportHelper.GetReport` to send report stream as an attachment.
- The following code sample exports the report to stream and send it as an attachment to a specified mail address. In the code, `SmtpClient` is used to send the report as an email attachment.

```

`csharp
public object SendEmail(Dictionary<string, object> jsonResult)
{
  string _token = jsonResult["reportViewerToken"].ToString();
  var stream = ReportHelper.GetReport(_token, jsonResult["exportType"].ToString());

```

```
stream.Position = 0;
if (!ComposeEmail(stream, jsonResult["reportName"].ToString()))
{
    return "Mail not sent !!!";
}
return "Mail Sent !!!";
}

public bool ComposeEmail(Stream stream, string reportName)
{
    try
    {
        MailMessage mail = new MailMessage();
        SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
        mail.IsBodyHtml = true;
        mail.From = new MailAddress("xx@gmail.com");
        mail.To.Add("xx@gmail.com");
        mail.Subject = "Report Name : " + reportName;
        stream.Position = 0;
        if (stream != null)
        {
            ContentType ct = new ContentType();
            ct.Name = reportName + DateTime.Now.ToString() + ".pdf";
            System.Net.Mail.Attachment attachment = new System.Net.Mail.Attachment(stream, ct);
            mail.Attachments.Add(attachment);
        }
        SmtpServer.Port = 587;
        SmtpServer.Credentials = new System.Net.NetworkCredential("xx@gmail.com", "xx");
        SmtpServer.EnableSsl = true;
        SmtpServer.Send(mail);
        return true;
    }
    catch (Exception ex)
    {
    }
```

```

return ex.ToString();
}
return false;
}
`

```

In the above code sample, the report is exported to PDF and sent to users using `SmtpClient`.

- Build and run the application. Preview and edit the result using the following.

```

{% tab demoPath="javascript-reporting/report-viewer/custom-actions/add-email-button"
files="index.html,index.js" %}
{% tabItem header="index.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
{% endtabItem %}
{% endtab %}

```

Cancel report processing in Report Viewer

This topic describes steps required to create custom cancel option to cancel the report processing in Report Viewer.

Add cancel report button

1. Create a cancel button in the toolbar using the `customItems` property with the values such as `cssClass`, `tooltip`.
2. Register the event `toolbarItemClick`, `ajaxSuccess`, `ajaxBeforeLoad`, `renderingComplete` to handle the cancel button click action.
3. You can use the following client side codes to create the cancel button.

```

`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "https://demos.boldreports.com/services/api/ReportViewer",
reportPath: '~/Resources/demos/Report/product-line-sales.rdl',

```

```
toolbarSettings: {
  showToolbar: true,
  customItems: [{
    type: 'Default',
    cssClass: "e-icon e-close e-reportviewer-icon",
    id: 'Stop',
    tooltip: {
      header: 'Stop',
      content: 'Stop processing the report'
    }
  ]
},
toolbarItemClick: 'onToolbarItemClick',
ajaxSuccess: onAjaxSuccess,
ajaxBeforeLoad: onAjaxRequest,
renderingComplete: onRenderingComplete
});

//Toolbar click event handler
function onToolbarItemClick(args) {
  if (args.value === "Stop") {
    var proxy = $('#viewer').data('boldReportViewer');
    proxy.cancelRendering();
    $('#viewertoolbarStop').addClass('e-disable');
  }
}

//Ajax request event handler to enable cancel button
function onAjaxRequest() {
  $('#viewertoolbarStop').removeClass('e-disable');
}

//Ajax success event handler to disable cancel button
function onAjaxSuccess() {
  $('#viewertoolbarStop').addClass('e-disable');
```



```

}
//Rendering complete event handler to disable cancel button
function onRenderingComplete() {
$('#viewertoolbarStop').addClass('e-disable');
}
</script>
`

```

4. Build and run the application. Preview and edit the result using the following.

```

{% tab demoPath="javascript-reporting/report-viewer/custom-actions/add-cancel-report-button"
files="index.html,index.js" %}
{% tabItem header="index.html" %}
`html
<body style="overflow: hidden; position: static; margin: 0; padding: 0; height: 100%; width: 100%;">
<div id="viewer" style="position: absolute; height: 100%; width: 100%;"></div>
<script src="index.js"></script>
</body>
`
{% endtabItem %}
{% endtab %}

```

Custom properties

The custom properties helps you to include additional features that are not natively supported in the RDL reporting. This topic explains the list of custom properties supported in JavaScript Report Viewer.

See Also

- [Textbox Custom Properties](#)
- [Table Custom Properties](#)
- [Image Custom Properties](#)
- [Chart Custom Properties](#)
- [Report Custom Properties](#)
- [Parameter Custom Properties](#)
- [Export Custom Properties](#)

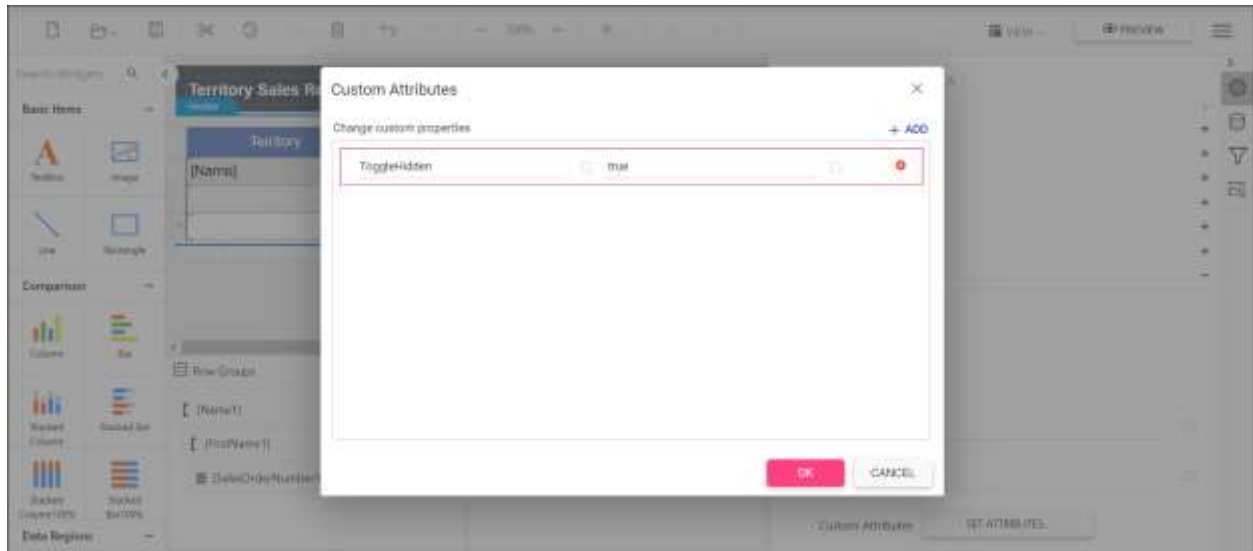
Textbox custom properties

This topic explains about the list of textbox report item custom properties that are supported to render in JavaScript Report Viewer.

Show or hide toggle icon in text box report item

The `ToggleHidden` custom property is used to show or hide the toggle icon in the textbox.

You can set the `ToggleHidden` property value, as shown in the below.



Before setting the toggle hidden property, the default value will be displayed as below.

The screenshot shows the preview of the 'Territory Sales Report'. The report is displayed in a table format with the following data:

Territory	Sales Person	Order Number	Total Sales
Australia	Lynn Tsoflias		\$1,943,016
Canada	Garrett Vargas		\$12,808,458
	José Sernaiva		\$4,840,689
			\$7,967,769
Central	Jillian Carson		\$13,434,510
France	Ranjit Varkey		\$6,083,691
	Chudakatil		\$6,083,691
Germany	Rachel Valdez		\$2,476,530
			\$2,476,530
Northeast	Michael Blythe		\$12,433,503
			\$12,433,503
Northwest			\$6,305,407

Preview the report and the see the toggle icon is hidden in the report.



Territory	Sales Person	Order Number	Total Sales
Australia	Lynn Tsiftas		\$1,943,016
Canada	Garrett Vargas		\$12,808,458
	José Saravia		\$4,840,689
Central	Jillian Carson		\$7,967,769
France	Ranjit Varkey Chudakatti		\$13,434,510
Germany	Rachel Valdez		\$13,434,510
Northeast	Michael Blythe		\$6,083,691
Northwest			\$6,083,691
			\$2,476,530
			\$2,476,530
			\$12,433,503
			\$12,433,503
			\$6,306,407

Table custom properties

This topic explains about the list of table report item custom properties that are supported to render in JavaScript Report Viewer.

Limit number of table records on each page

The `RowsPerPage` custom property is used to specify the number of table records to display on each page. It supports integer data value greater than zero.

This property is ignored when table rows heights higher than current page size. Increase the report page height or reduce `RowsPerPage` count that fits within the page.

You can set the `RowsPerPage` property value, as shown in the below.

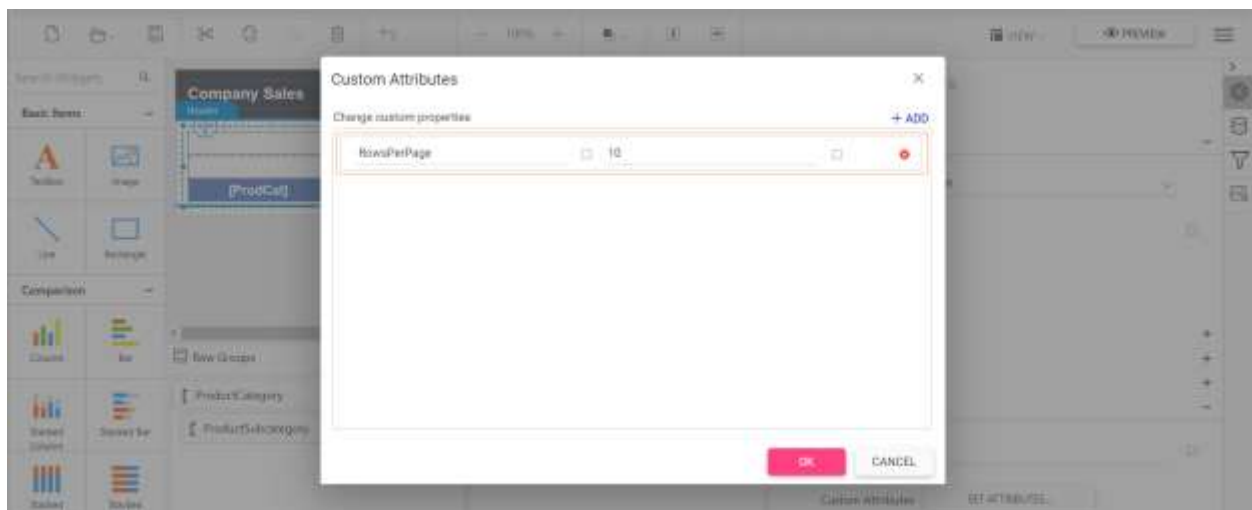
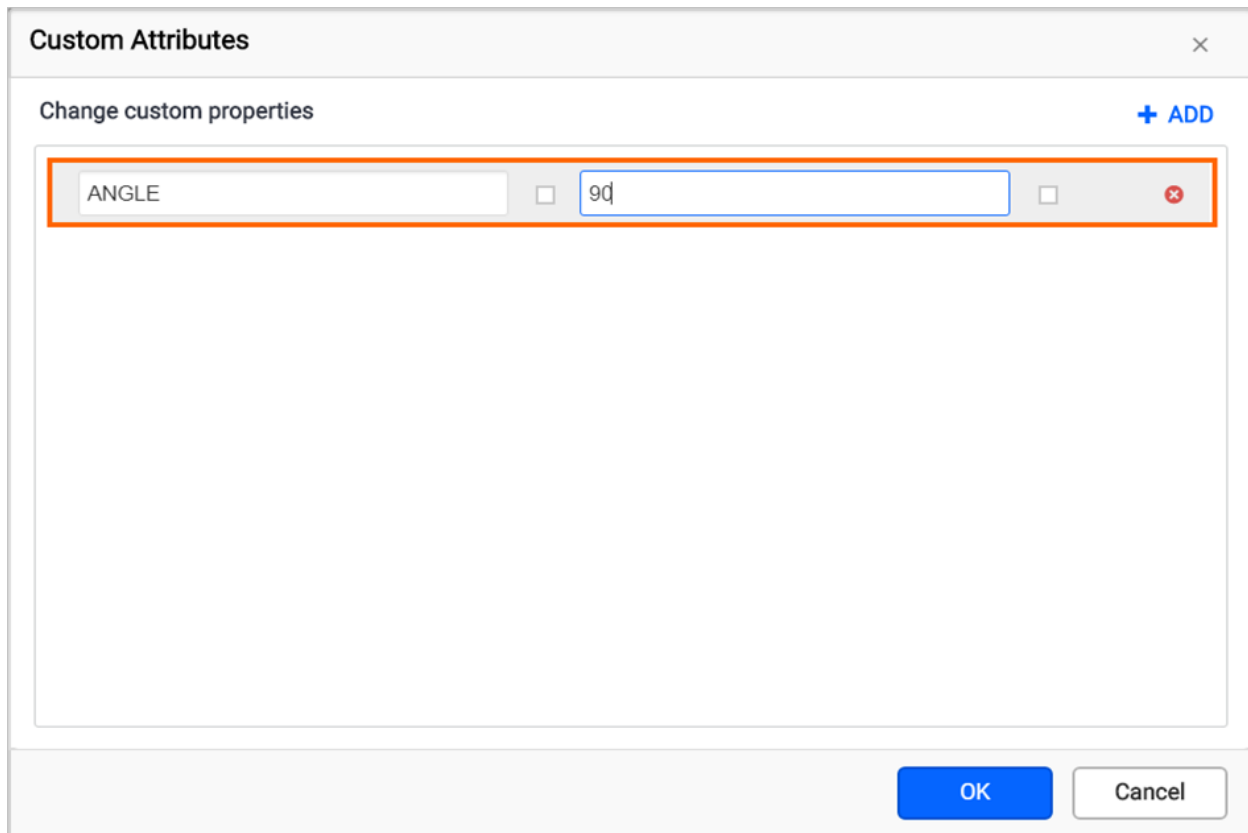


Image custom properties

This topic explains about the list of image custom properties that are supported to render in the Report Viewer.

Angle

Set **Angle** custom property to image report item to rotate the image on a specified angle. It supports the angle values 0, 90, 180, and 270. You can set the property value, as shown in the below.



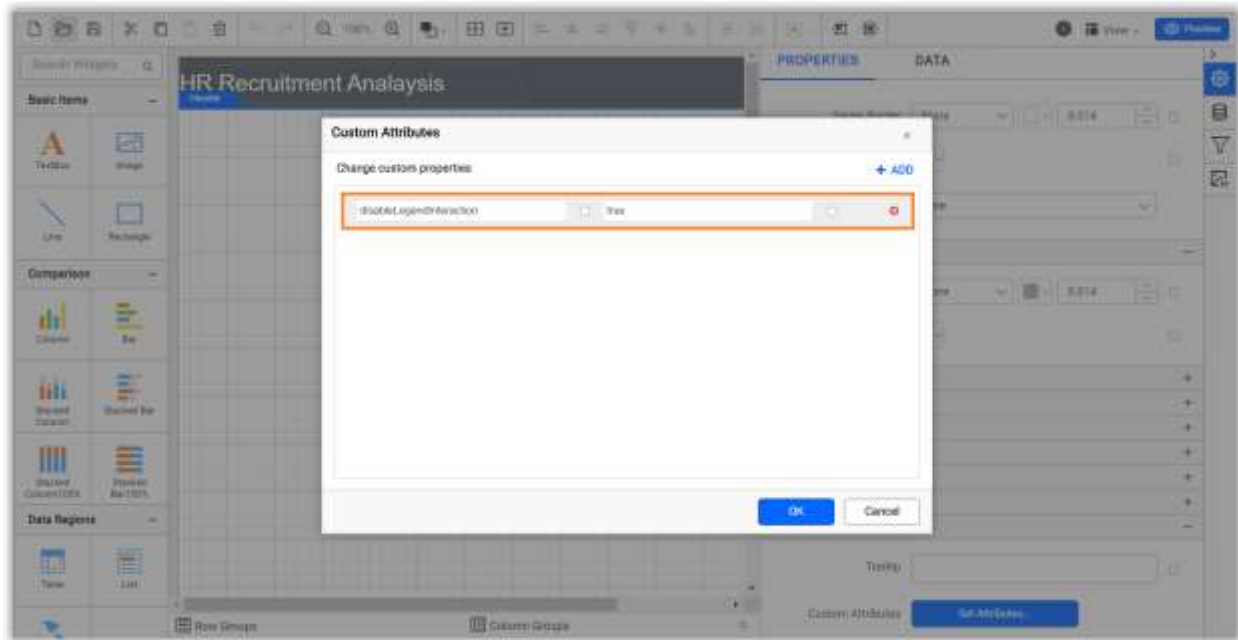
The screenshot shows a dialog box titled "Custom Attributes" with a close button (X) in the top right corner. Inside the dialog, there is a section labeled "Change custom properties" with a blue "+ ADD" button to its right. Below this section, there is a list of custom properties. The first property is "ANGLE", which is highlighted with an orange border. To the right of the property name is a small square checkbox, followed by a text input field containing the value "90". To the right of the input field is another small square checkbox and a red "X" button. At the bottom right of the dialog, there are two buttons: "OK" (blue) and "Cancel" (white with a gray border).

Chart custom properties

This topic explains about the list of chart custom properties that are supported to render in the JavaScript Report Viewer.

Disable legend item interaction

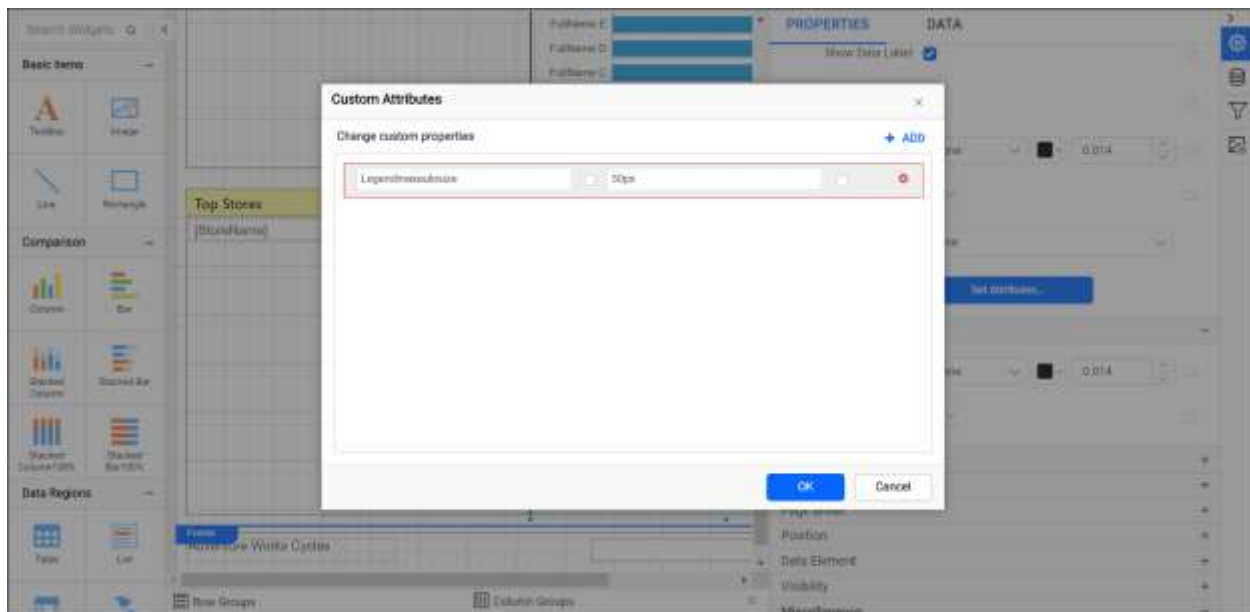
Set **DisableLegendInteraction** custom property value as **true** to stop the legend item interaction. The property value should be boolean. You can set the property value, as shown in the below.



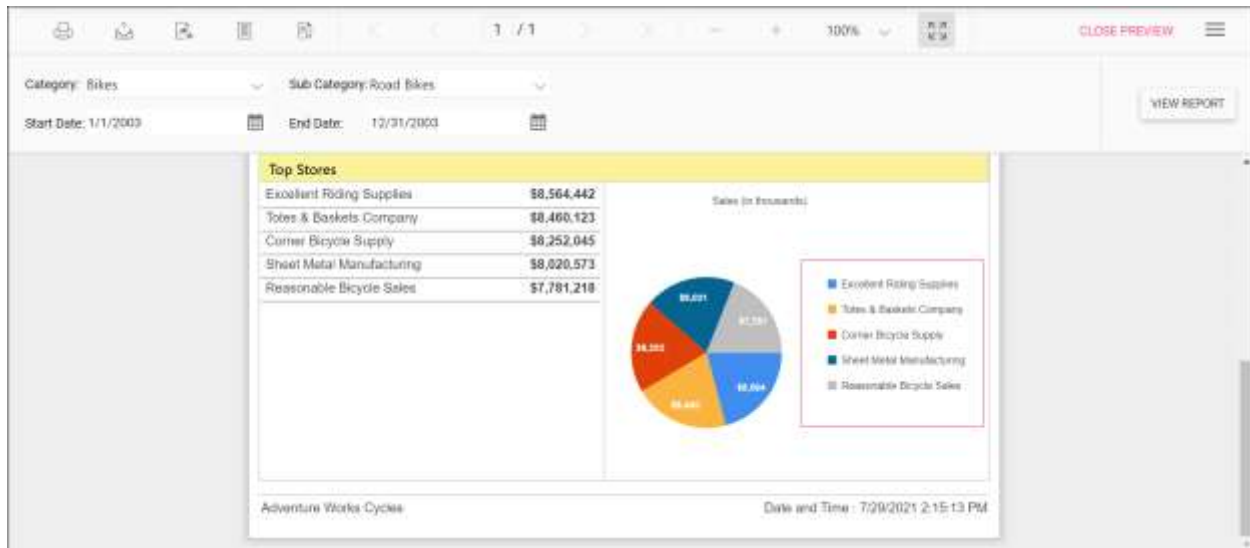
Set maximum size for chart legend

The `LegendMaxAutoSize` custom property specifies the maximum size of the legend container in the report.

You can set the `LegendMaxAutoSize` property value, as shown in the below.



Preview the report and the see legend container size in chart report.

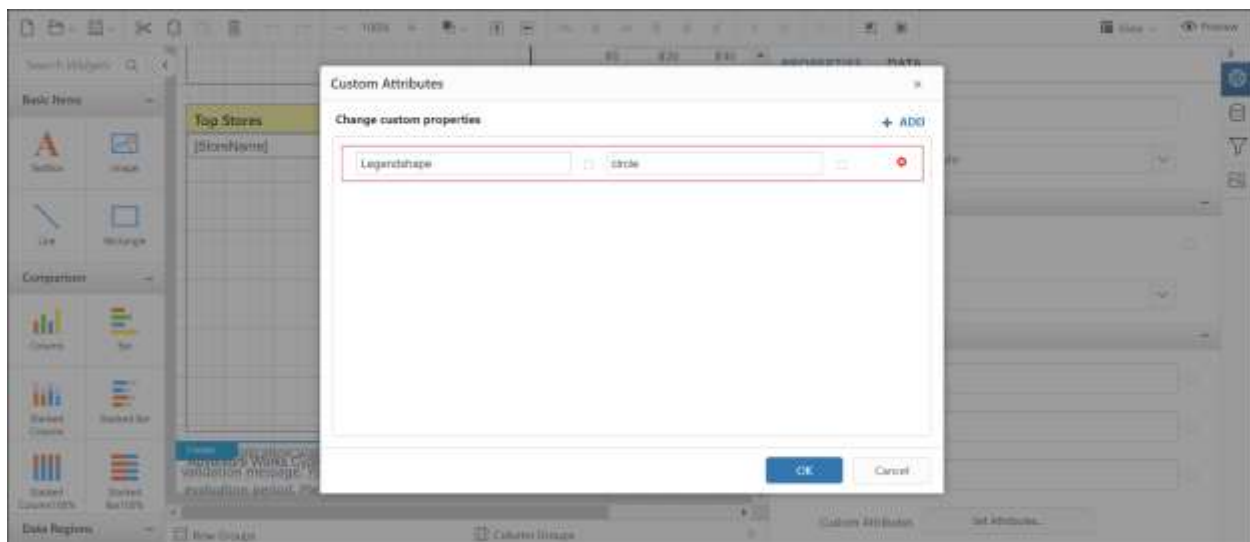


Change chart legend shape

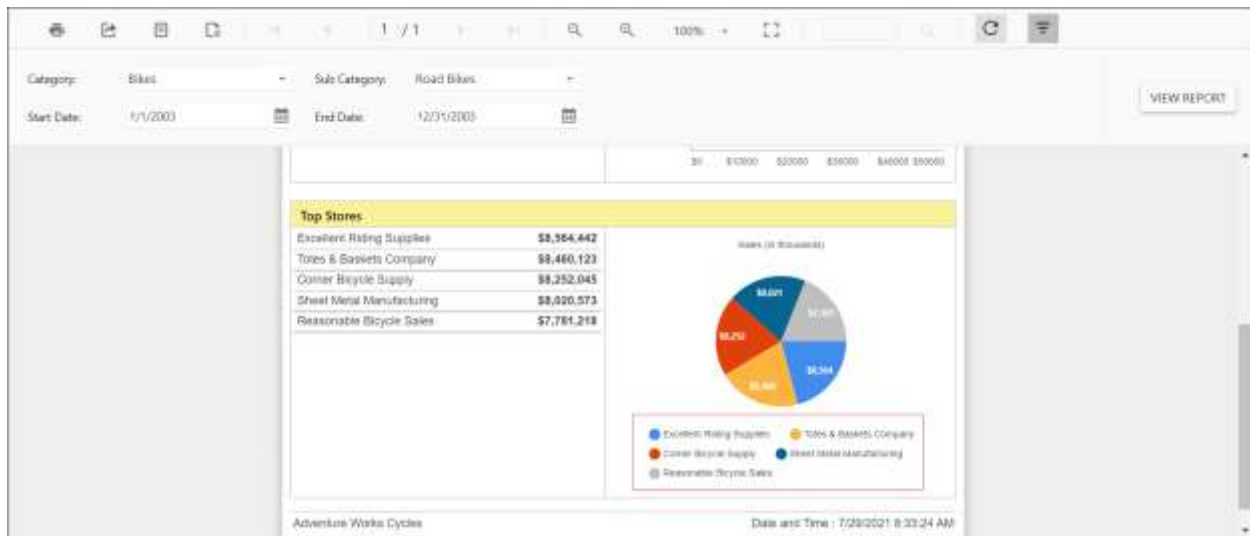
The **LegendShape** custom property allows changing the shape of the legend in the chart report item. By default, the **LegendShape** value is **rectangle**.

- Rectangle - legend shapes displayed in **rectangle**.
- Circle - legends are displayed in **circle**.
- Thumbnail - legends are displayed in **seriestype**.

You can set the **LegendShape** property value, as shown in the below.



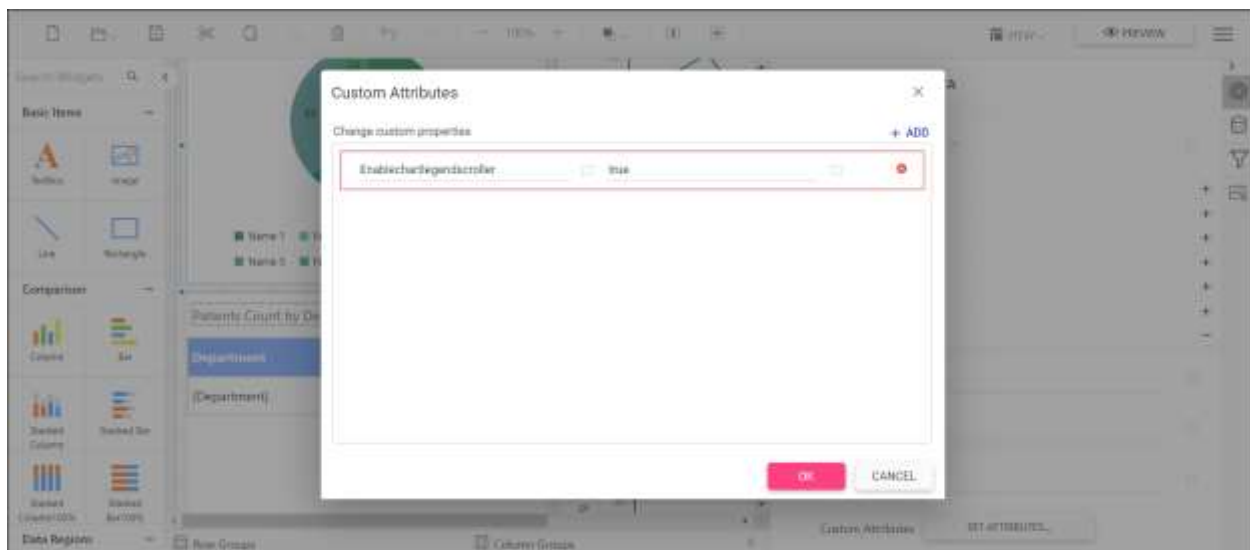
Preview the report and then see the legend shape in the chart report.



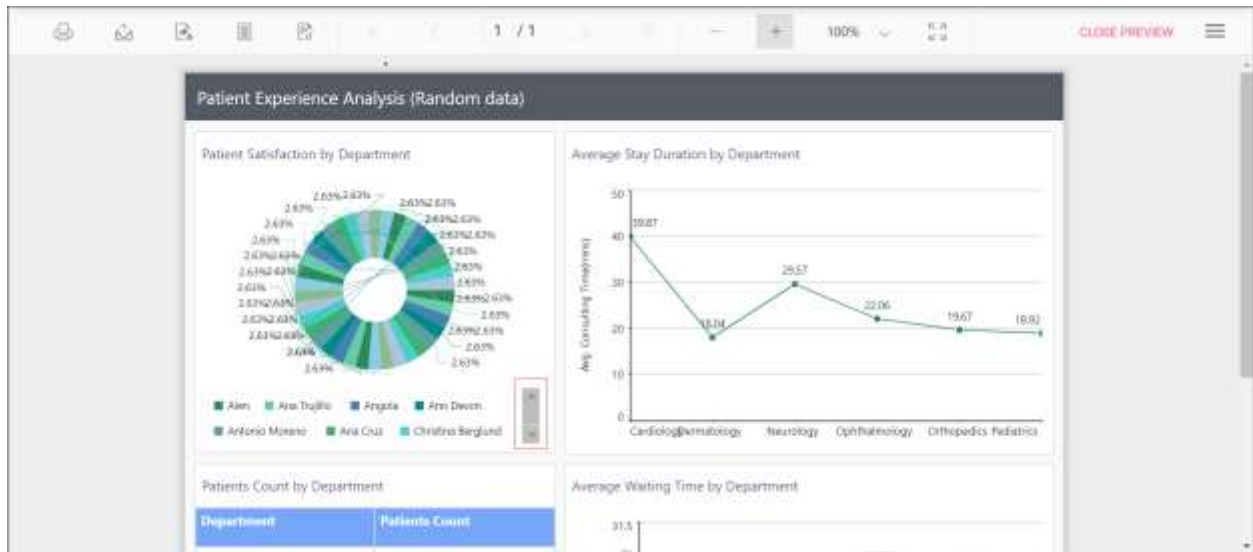
Show or hide chart legend scroller

The `EnableChartLegendScroller` custom property controls whether legend has to use scrollbar or not. The scrollbar appears depending upon size and position properties of legend. By default, the `EnableChartLegendScroller` value is `false`.

You can set the `EnableChartLegendScroller` property value, as shown in the below.



Preview the report and the see the legend scrollbar in chart report.



Set range padding for X-axis and Y-axis

Padding can be applied to the minimum and maximum extremes of the axis range by using the `XAxisRangePadding` and `YAxisRangePadding` property. The default value is `none`.

Numeric axis supports the following types of padding.

Name | Description

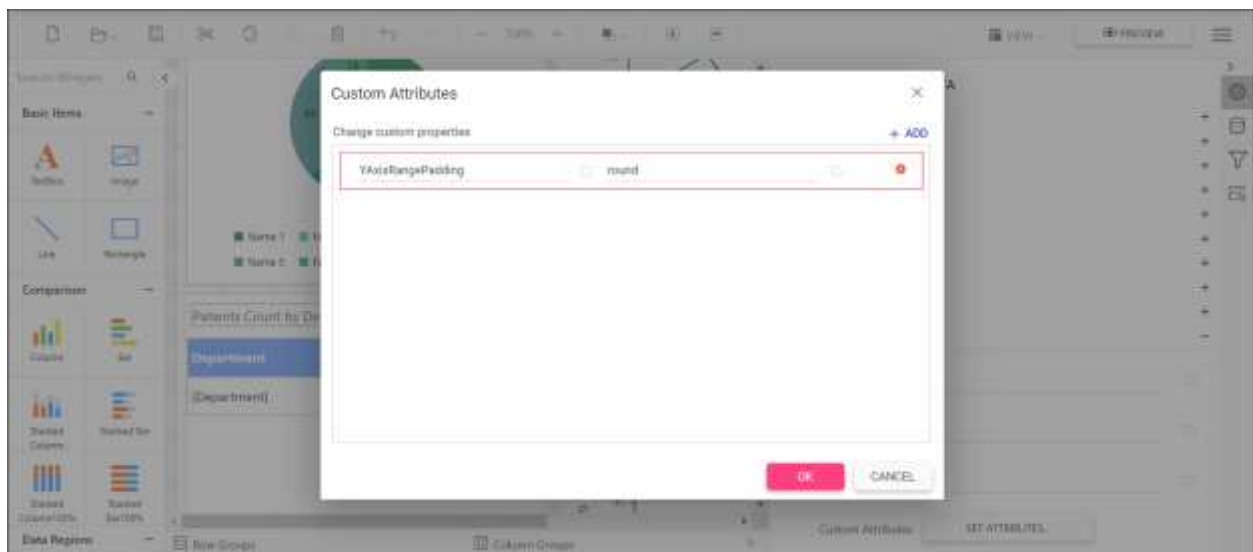
Additional | Interval of the axis is added as padding to the minimum and maximum values of the range

Normal | Padding is applied to the axis based on the range calculation

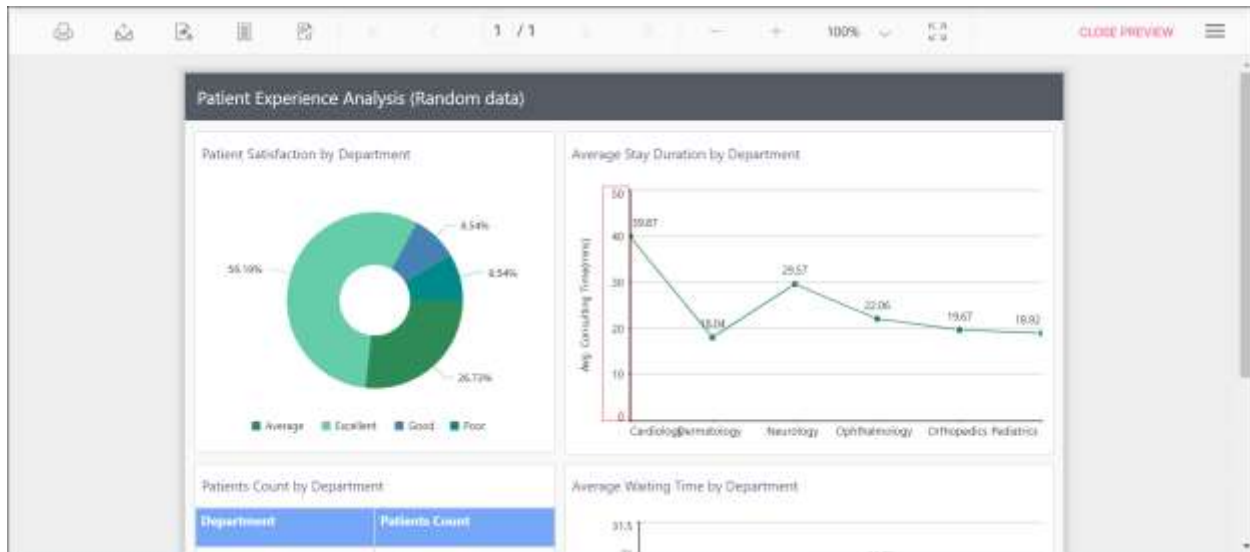
None | Padding cannot be applied to the axis

Round | Axis range is rounded to the nearest possible value divided by the interval

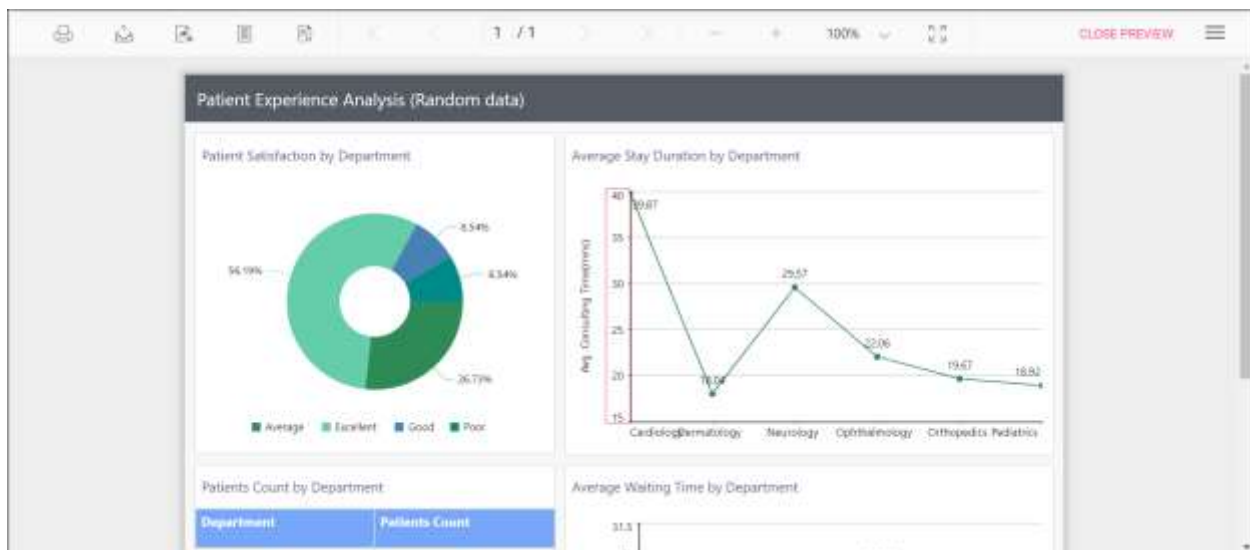
You can set the `XAxisRangePadding` and `YAxisRangePadding` property value, as shown in the below.



Before setting the range padding, the default value will be displayed as below.



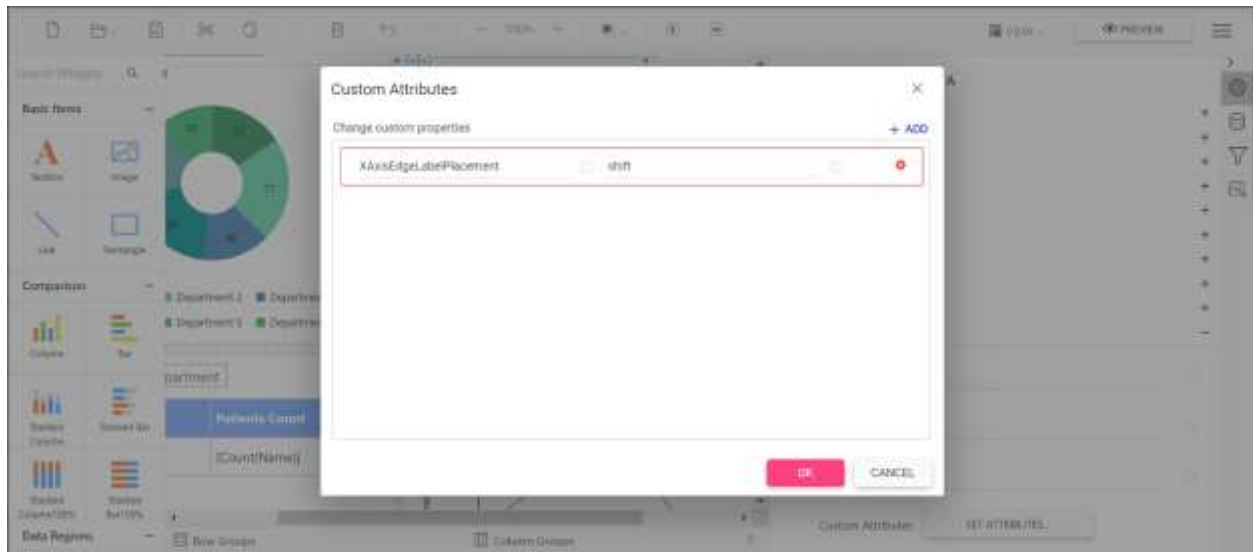
Set the padding range and see the changes in chart report as below.



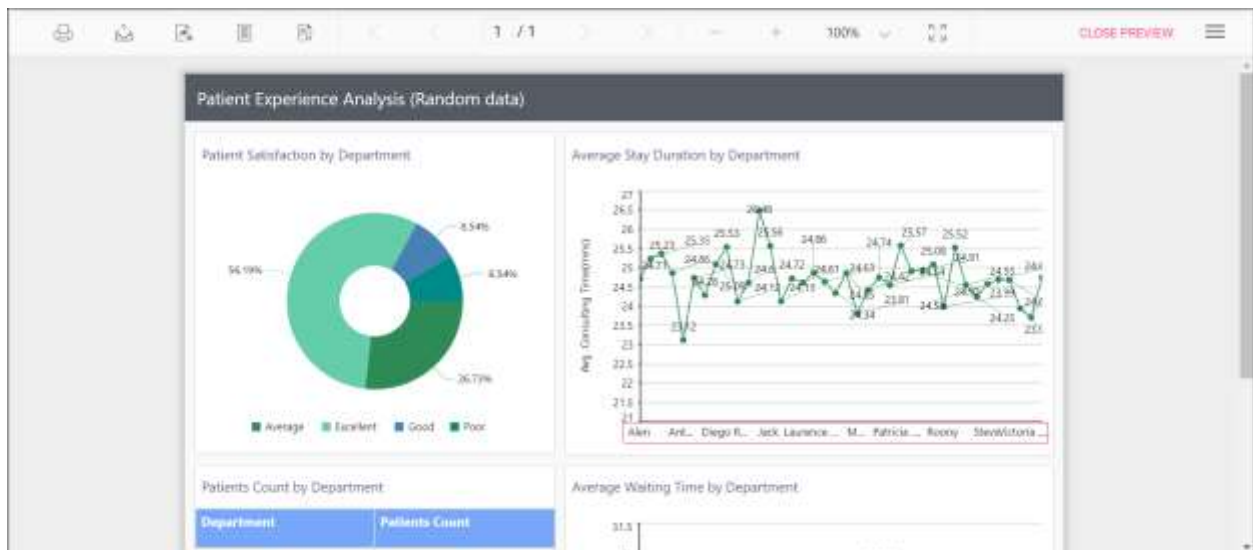
Control edge label placement in chart axis

Labels with long text at the edges of an axis may appear partially outside the chart. The `XAxisEdgeLabelPlacement` and `YAxisEdgeLabelPlacement` custom property can be used to avoid the partial appearance of the labels at the corners. The default value is `none`.

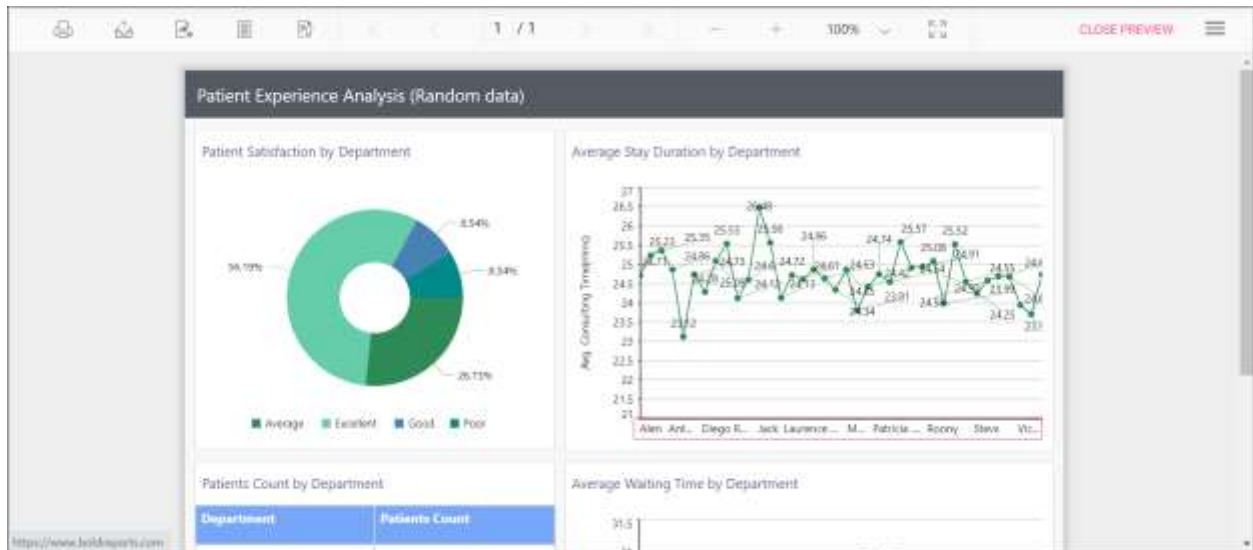
You can set the `XAxisEdgeLabelPlacement` property value, as shown in the below.



Before setting the edge label placement, the default value will be displayed as below.



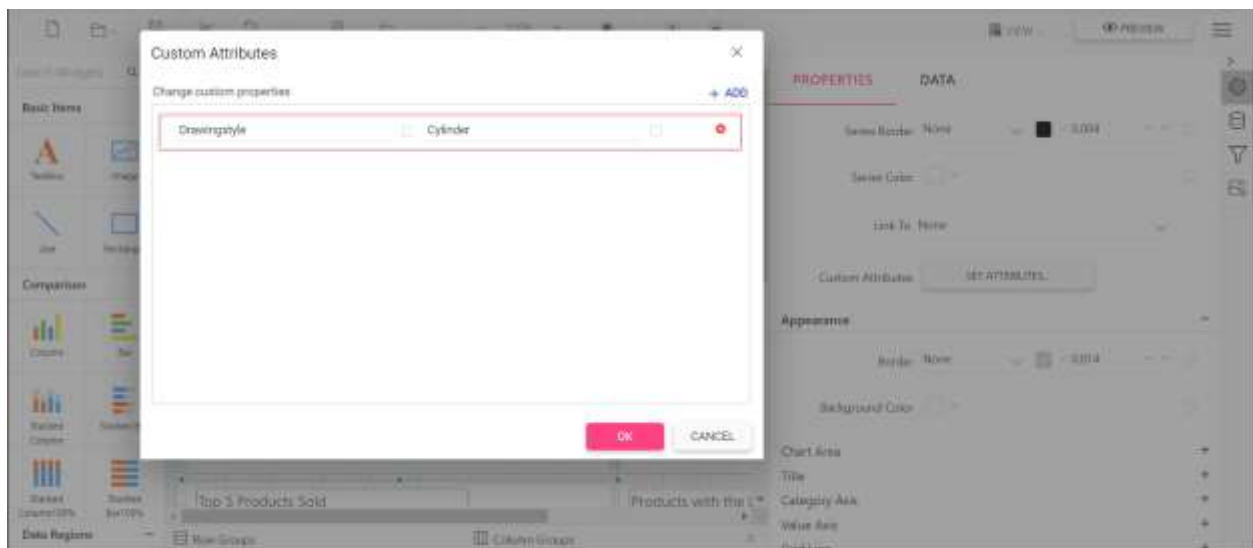
Set the edge label placement and see the changes in chart report as below.



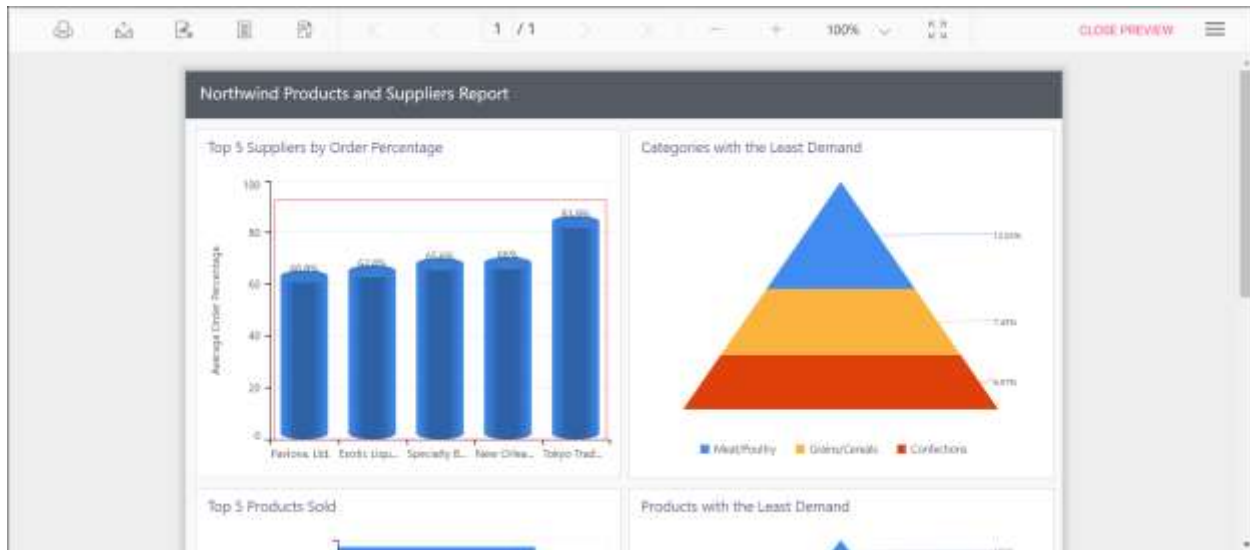
Change the drawing style of a chart column or bar series

The shape of the chart column or bar can be changed using this **Drawingstyle** custom property. By default, the **Drawingstyle** property value is **rectangle**.

You can set the **Drawingstyle** property value, as shown in the below.



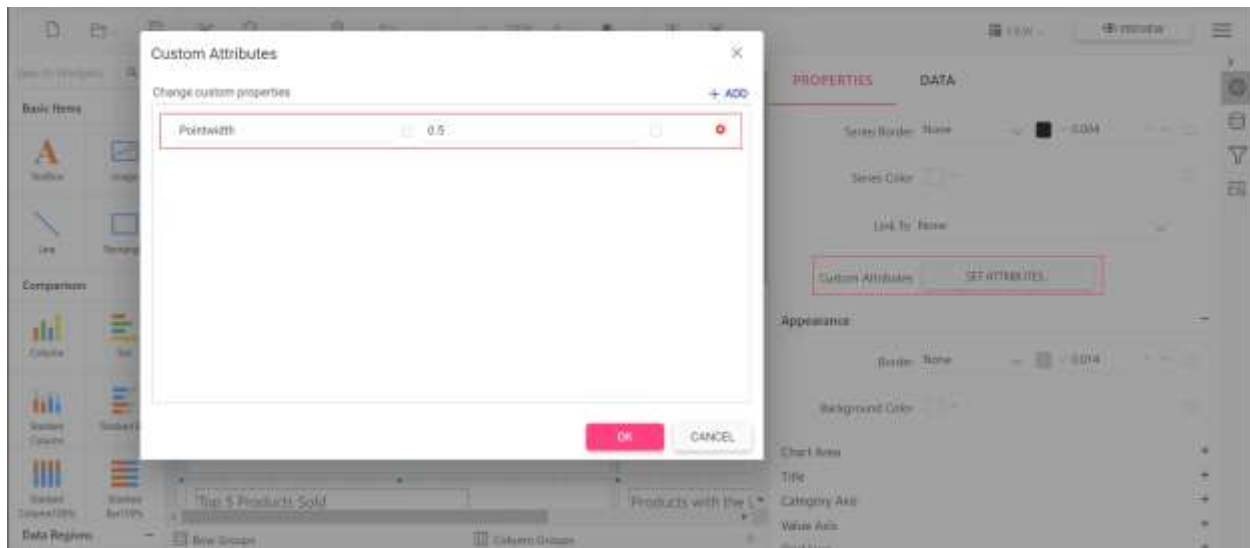
Preview the report and the see the column or bar shape in chart report.



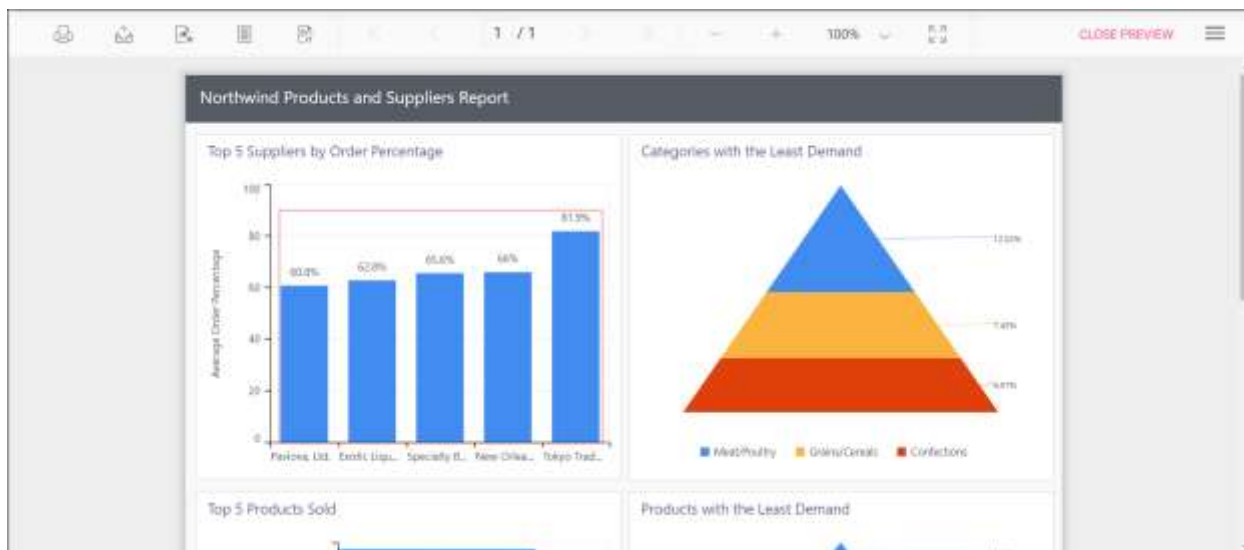
Change width of the column type series in chart

Width of the column type series can be customized by using the **Pointwidth** property. Default value of **Pointwidth** is 0.7. Value ranges from 0 to 1. Here 1 corresponds to 100% of available width and 0 corresponds to 0% of available width.

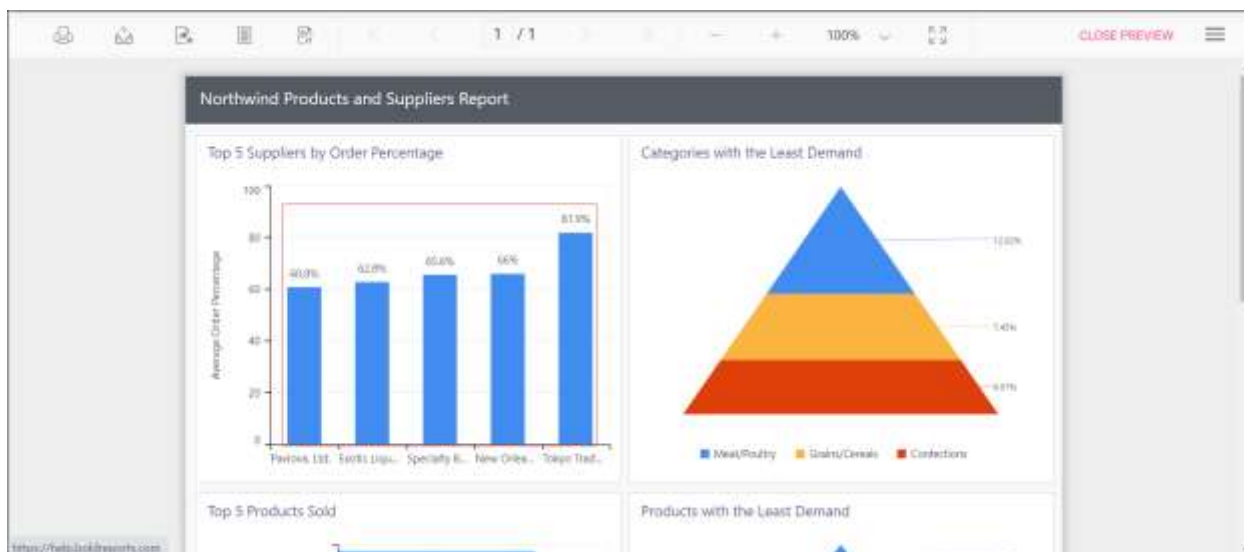
You can set the **Pointwidth** property value, as shown in the below.



Before setting the point width, the default value will be displayed as below.



Preview the report and the see the column width in chart report.



Report custom properties

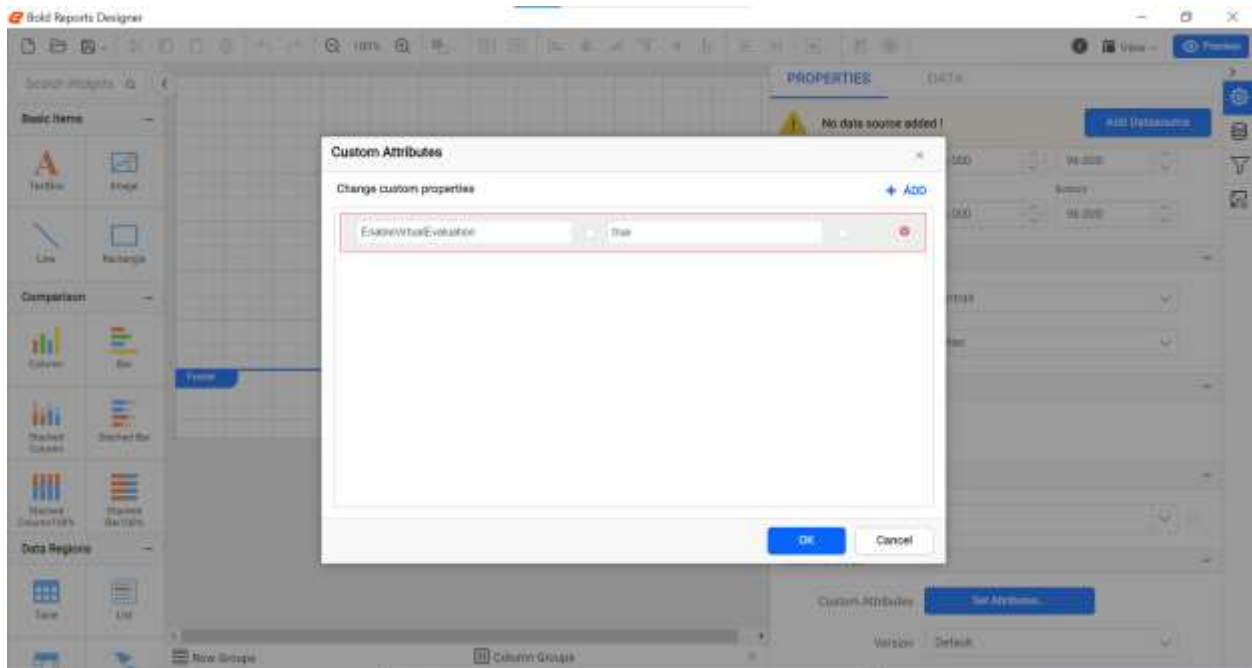
This topic explains about the list of report custom properties that are supported to render in Report Viewer.

Improve performance and handle large amount of data

Set the `EnableVirtualEvaluation` and `DisablePageSplitting` custom properties in a report to improve performance and handle the larger amount of data with less memory footprint.

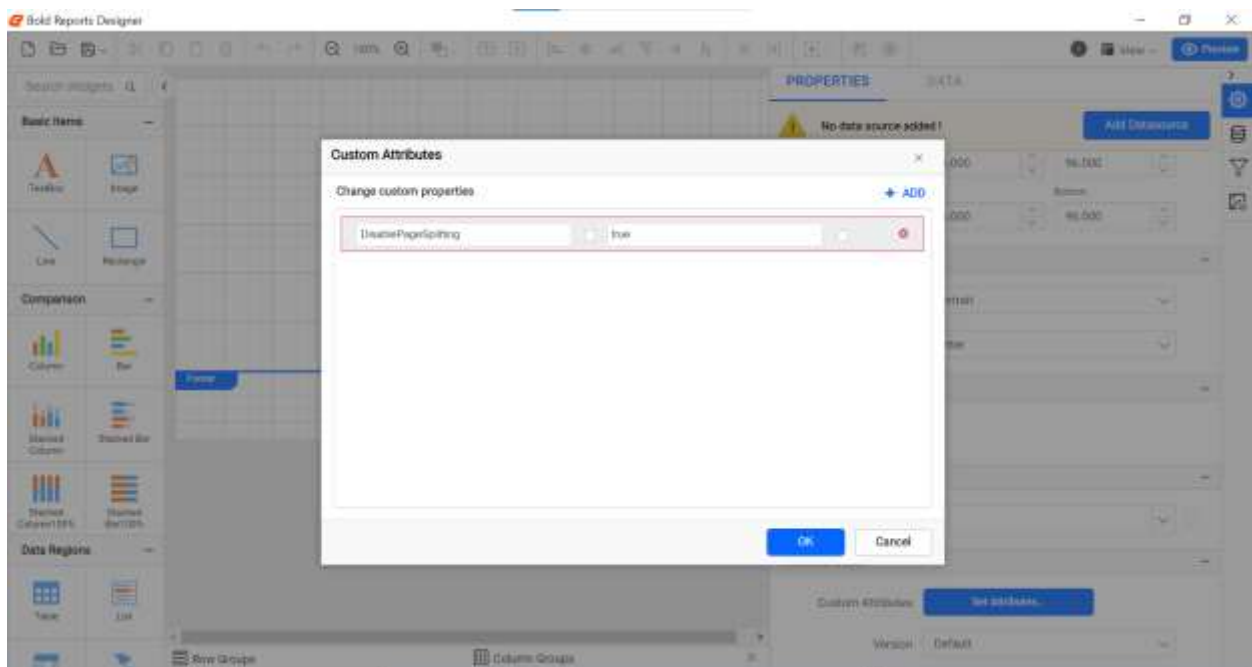
Render large data report faster

The `EnableVirtualEvaluation` custom property is used to render the large data report faster. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Reduce memory footprint for large data report

The `DisablePageSplitting` custom property is used to reduce the memory footprint for large data report. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Improve report items layout and avoid extra blank pages

When the `RoundLayoutMeasures` property is false, all non-integral values that are calculated during the report processing are rounded to whole pixel values. It provides following improvements,

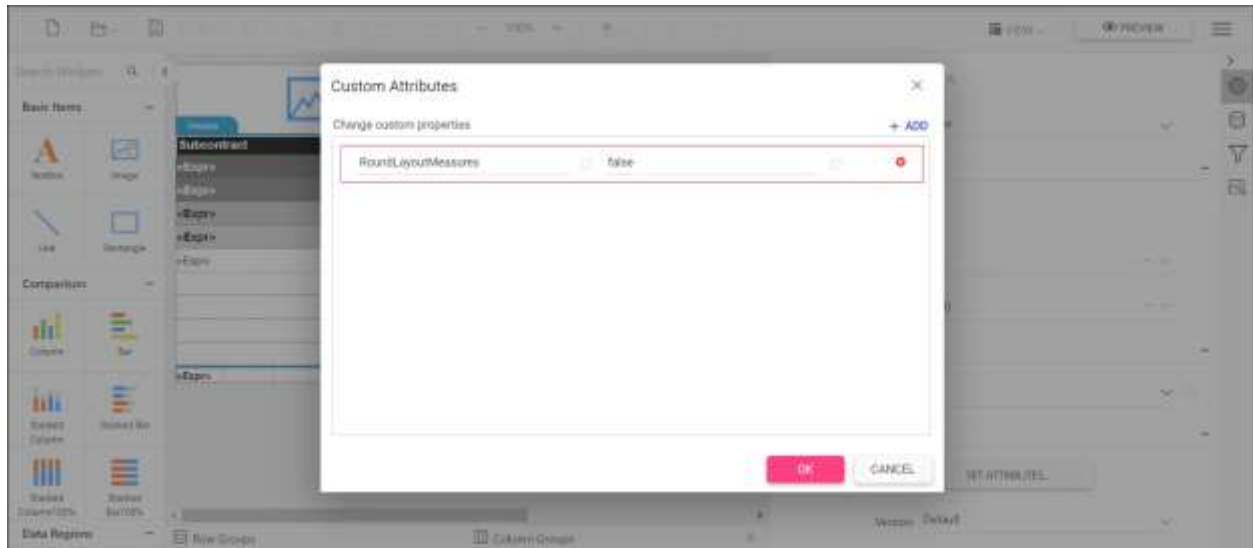
- Report text rendered without cuts.

Report custom properties
timeouts

Handling failure in long-running HTTP requests, report processes, and

- Eliminates extra blank pages.
- Removes item and text overlaps.
- Eliminates the blur semi-transparent edges that are produced by anti-aliasing.
- Produces identical look in report view and export output.

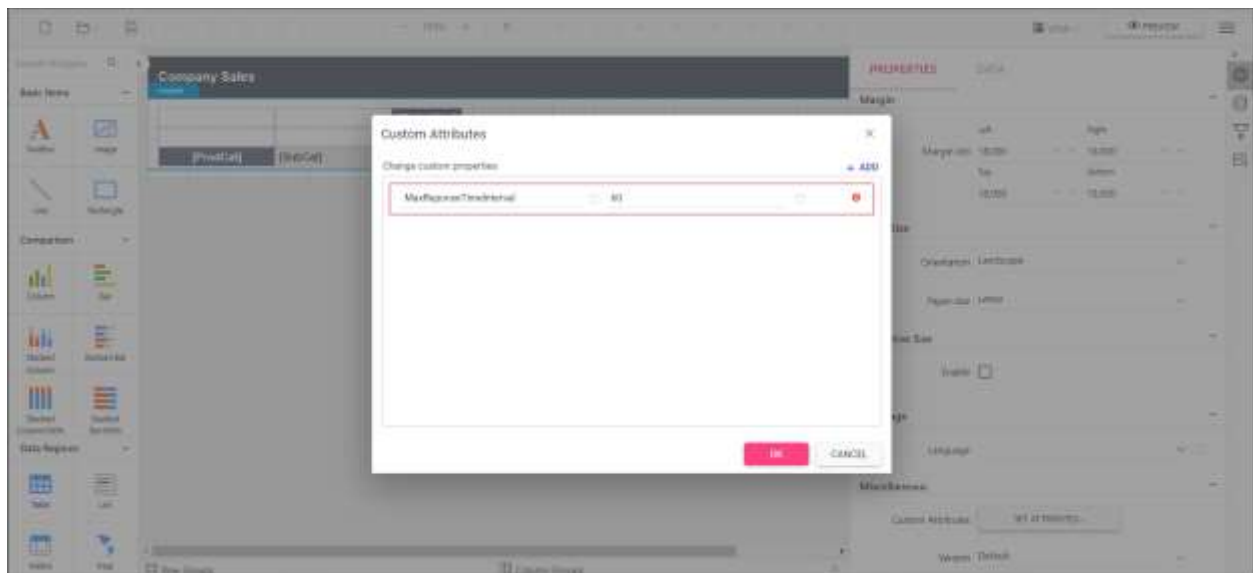
You can set the property value, as shown in the below.



Handling failure in long-running HTTP requests, report processes, and timeouts

When a report process for a long time due to huge records or a HTTP request takes too long to respond then it results in Gateway Timeout or report rendering errors. The `MaxResponseTimeInterval` allows to specify the seconds to process an HTTP request and respond back. It helps to keep the client and server connection live by avoiding the timeout.

You can set the property value as shown in the below.



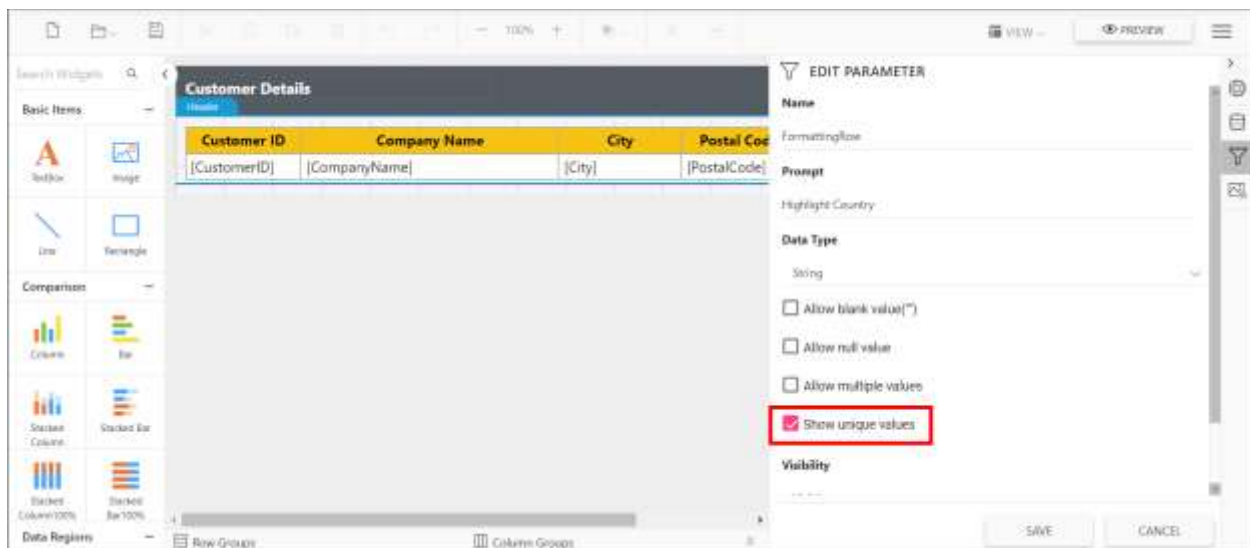
Parameter custom properties

This topic explains about the list of parameter custom properties that are supported to customize the reports preview in Report Viewer.

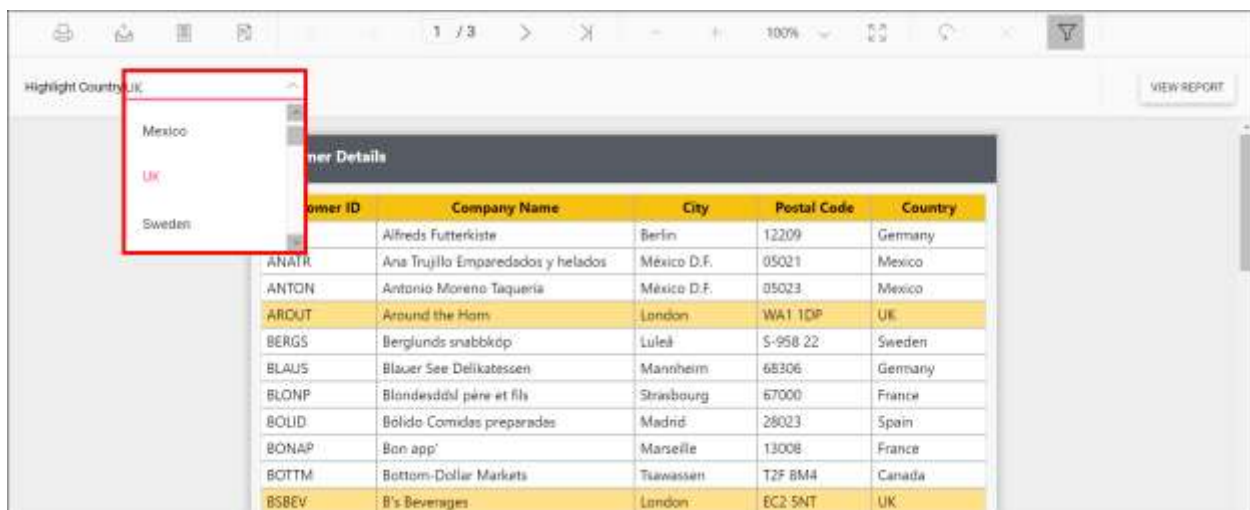
Show only distinct values in parameter

A parameter created with data set query values may contain duplicate values. The Report Viewer supports `UniqueValueParameters` custom property that helps show only unique values in the parameter drop-down while viewing the report, without creating new a data set. Refer to the below image for property setting option.

You can also provide the parameter names with comma (,) separator to set for multiple parameters.



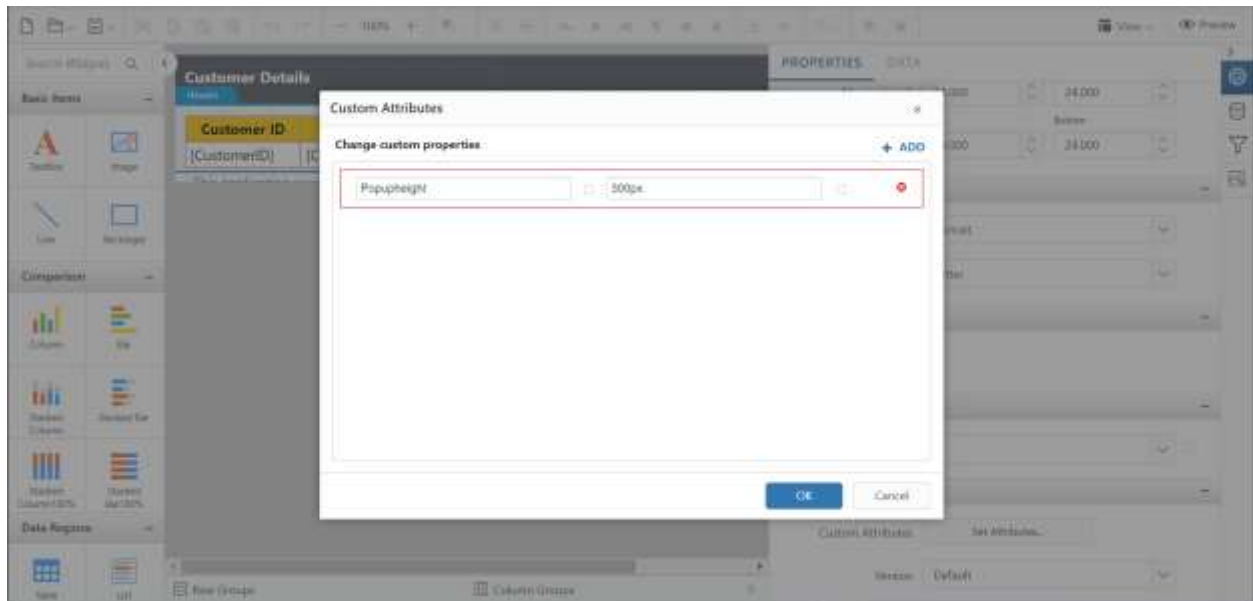
Preview the report and the unique values showed in the parameter drop down.



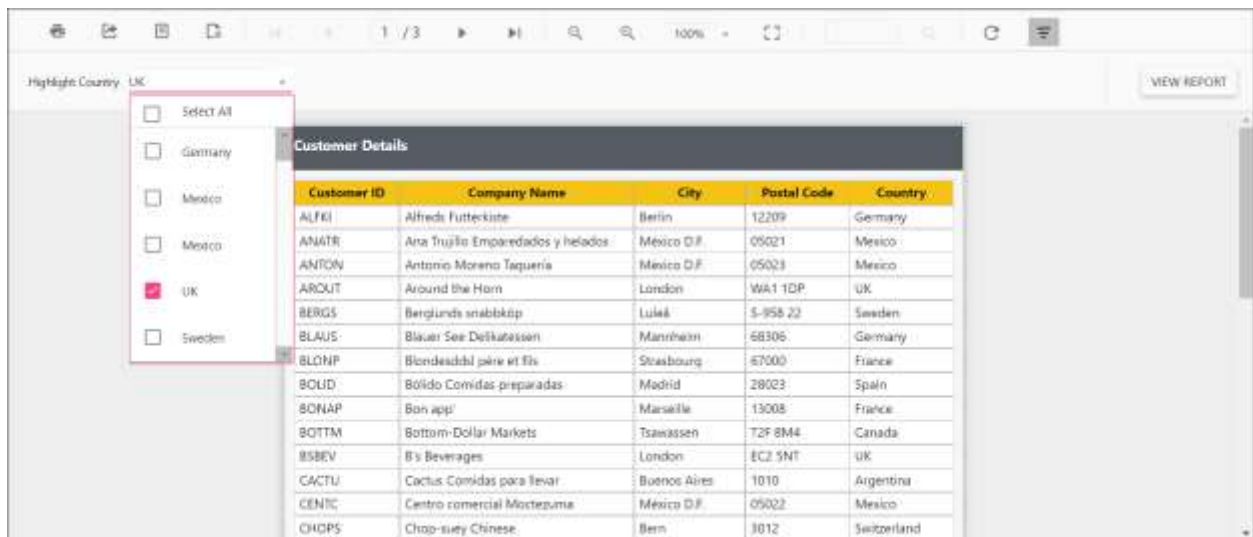
Change the dropdown parameter pop up height value

The `PopupHeight` custom property specifies the height of the parameter combobox popup list in the report. By default, the `PopupHeight` value is 152px.

You can set the `PopupHeight` property value, as shown in the below.



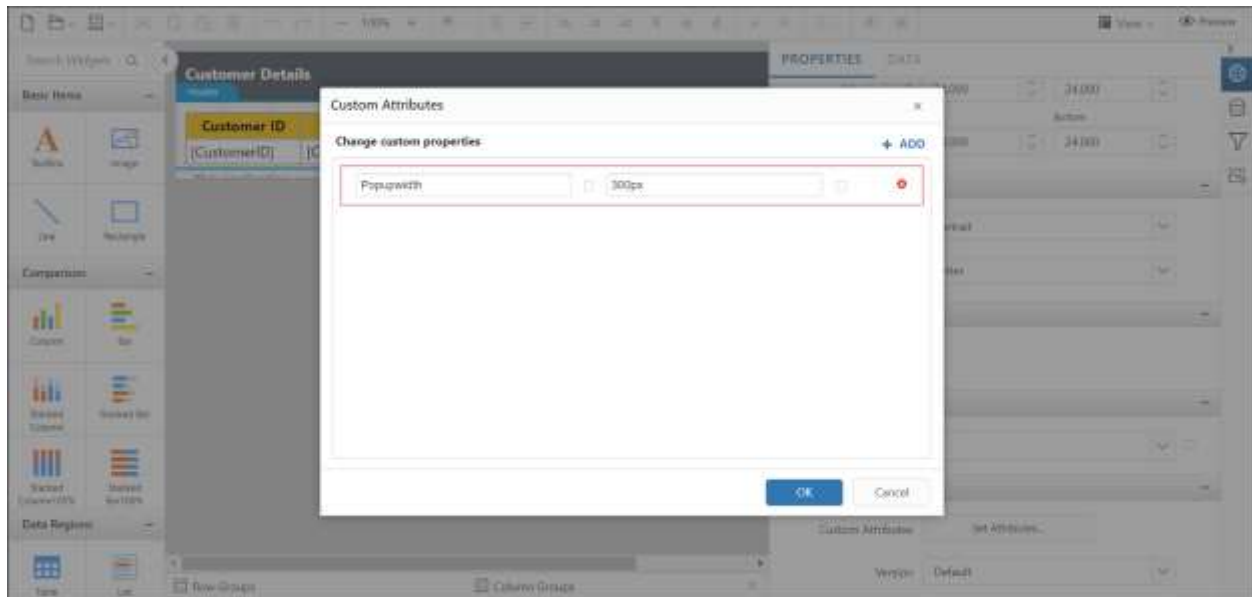
Preview the report and the pop up height showed in the parameter drop down.



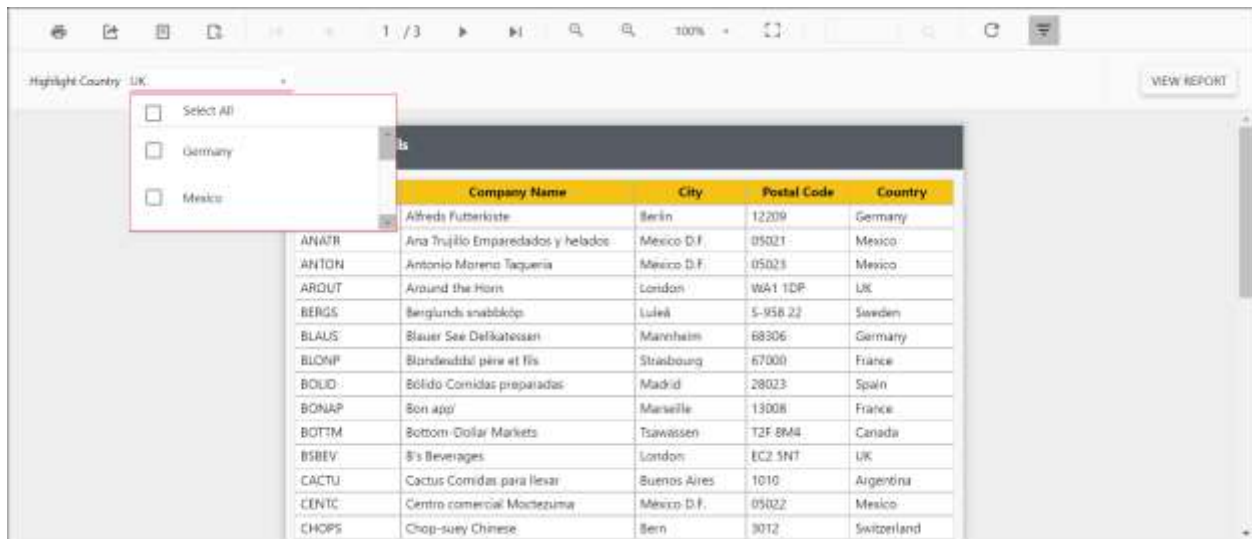
Change the dropdown parameter pop up width value

The **PopupWidth** custom property specifies the width of the parameter combobox popup list in the report. By default, the popup width sets based on the width of the component.

You can set the **PopupWidth** property value, as shown in the below.



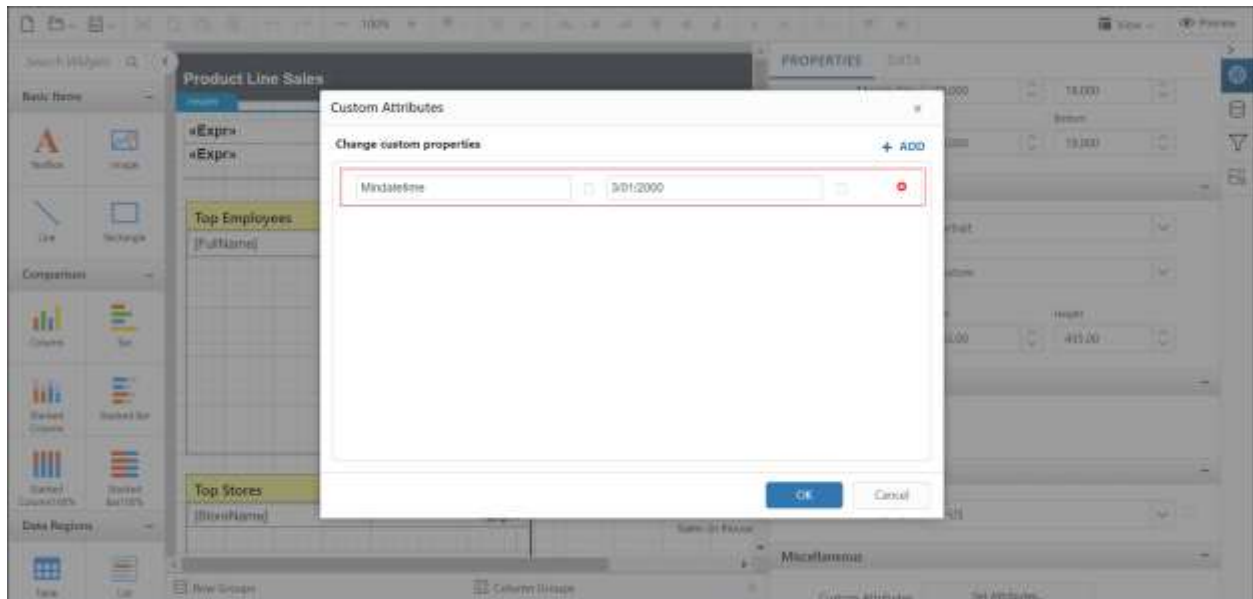
Preview the report and the pop up width showed in the parameter drop down.



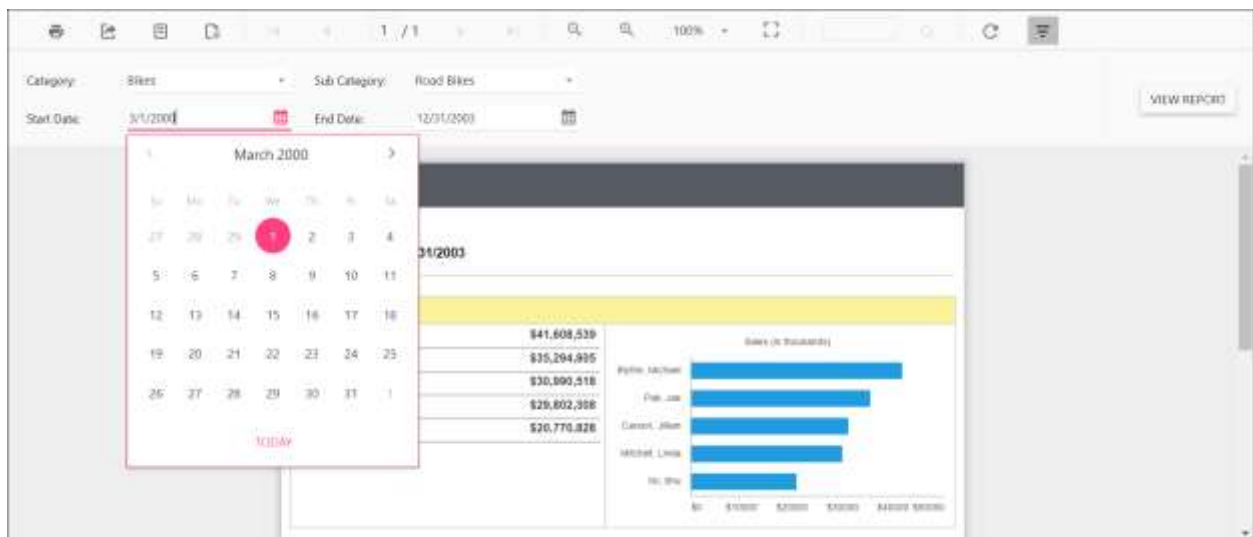
Set the minimum date range for the date report parameter

The `MinDateTime` custom property specifies the minimum date in the datetime parameter item. By default, the `MinDateTime` value is set as `null`.

You can set the `MinDateTime` property value, as shown in the below.



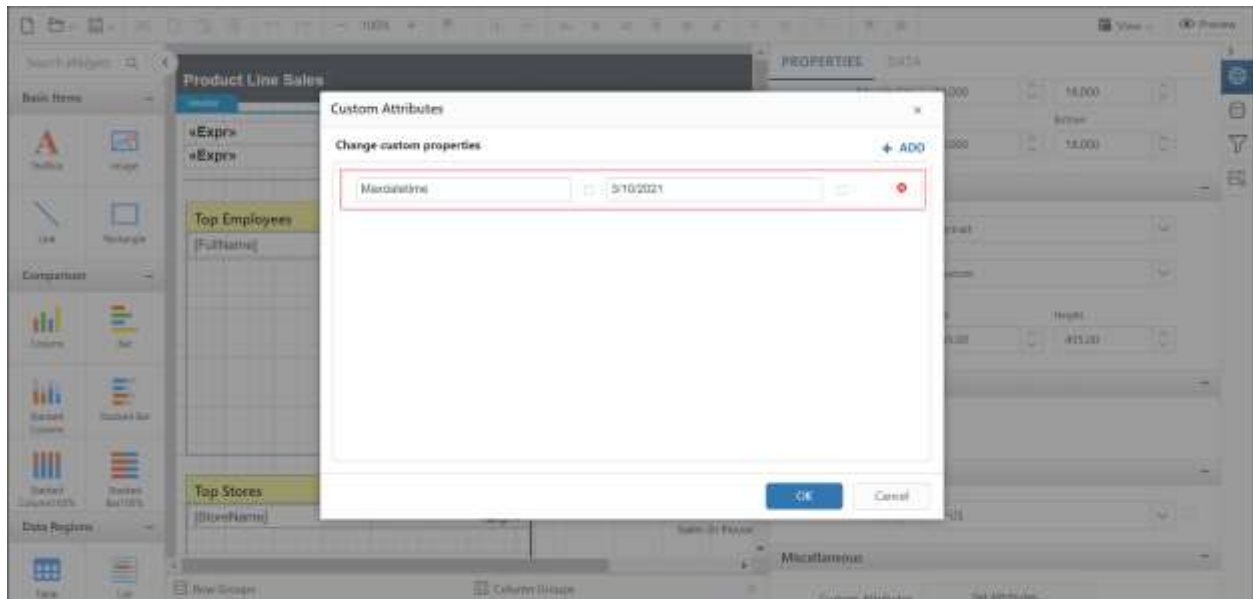
Preview the report and the minimum date showed in the datetime parameter drop down.



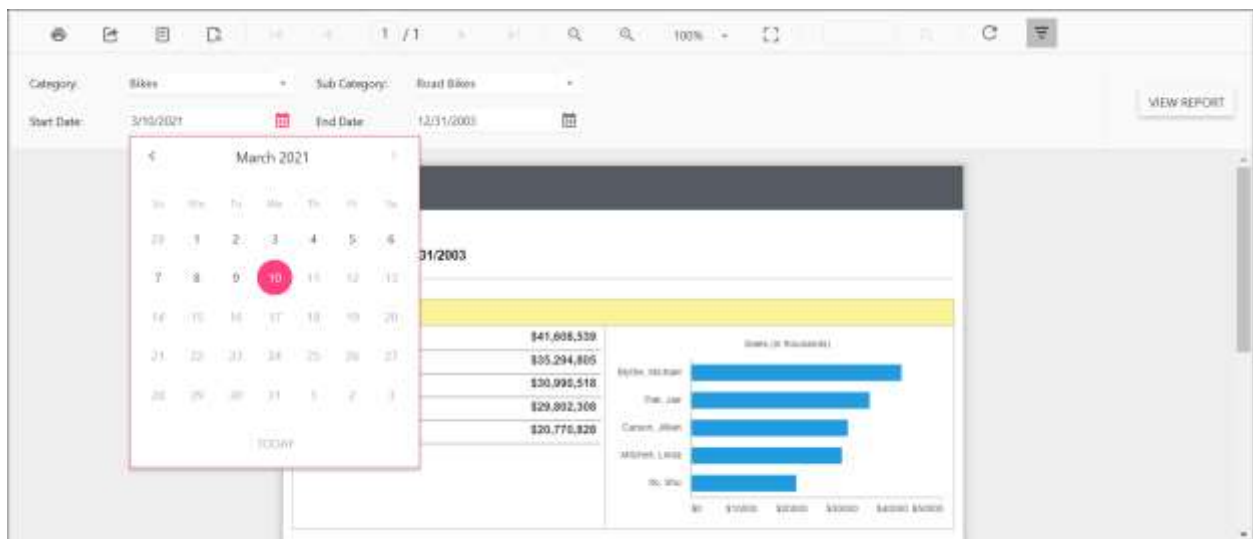
Set the maximum date range for date report parameter

The `MaxDateTime` custom property specifies the maximum date in the datetime parameter item. By default, the `MaxDateTime` value is set as `null`.

You can set the `MaxDateTime` property value, as shown in the below.



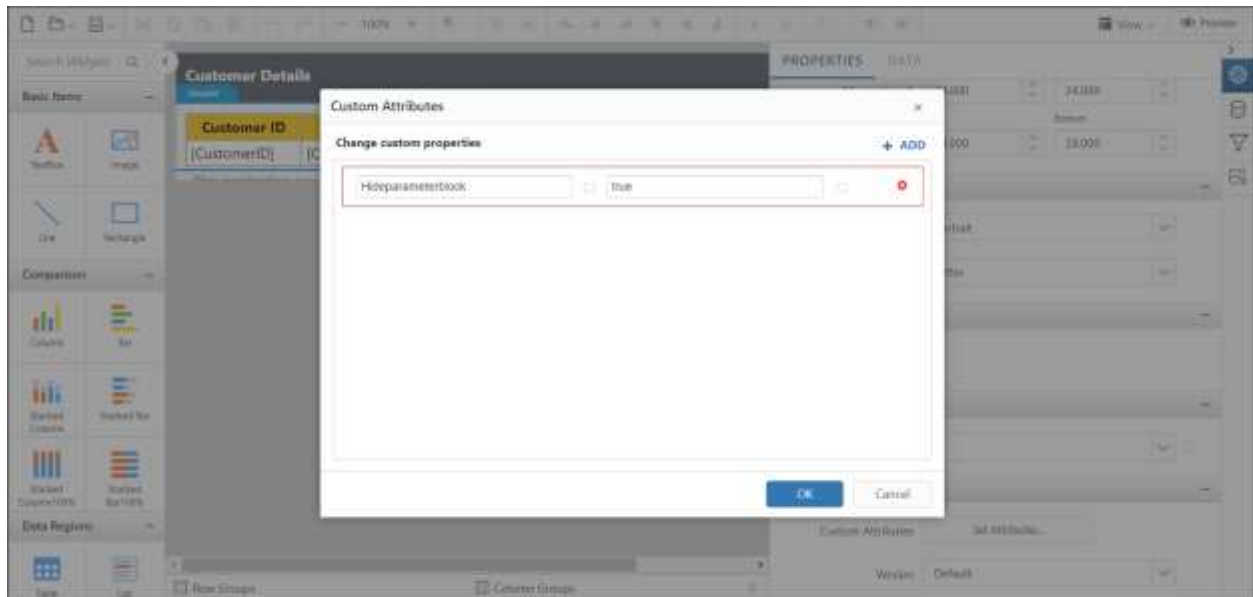
Preview the report and the maximum date showed in the datetime parameter drop down.



Hide the report parameter block

The `HideParameterBlock` custom property is used to hide the parameter block on report initial rendering. The property value should be boolean. By default, the `HideParameterBlock` value is `false`.

You can set the `HideParameterBlock` property value, as shown in the below.



Preview the report and see the parameter block is hidden.

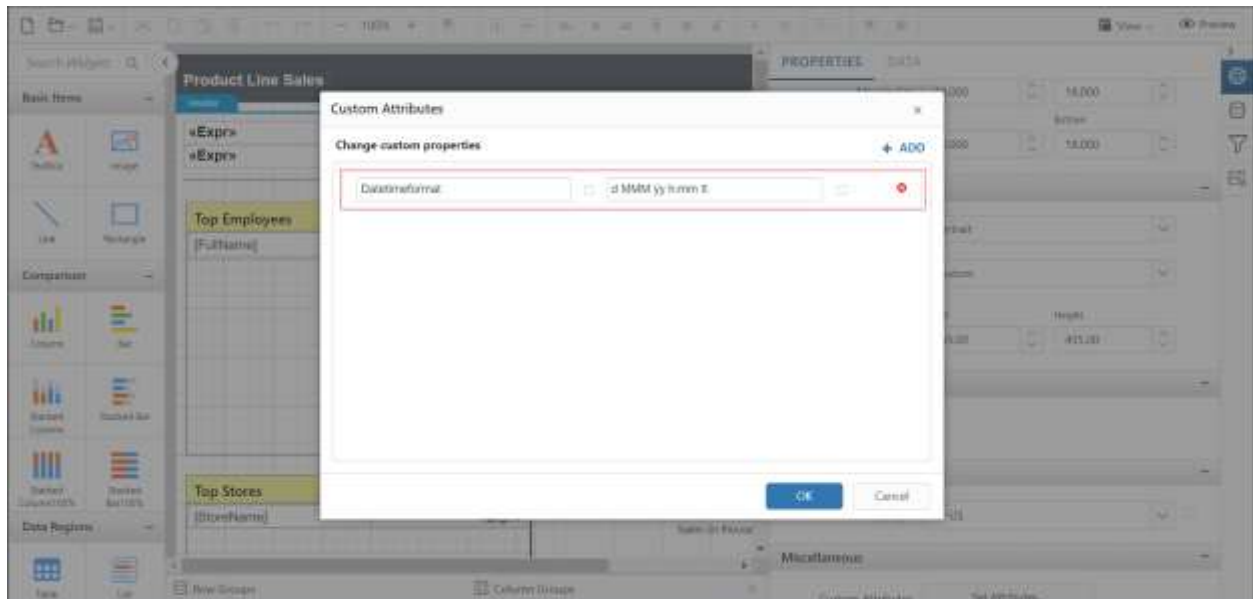
The screenshot shows a report preview titled 'Customer Details'. It displays a table with the following columns: Customer ID, Company Name, City, Postal Code, and Country. The data is filtered to show only customers from Germany, Mexico, and the UK.

Customer ID	Company Name	City	Postal Code	Country
ALFKI	Alfreds Futterkiste	Berlin	12209	Germany
ANATR	Ana Trujillo Emparedados y helados	México D.F.	05021	Mexico
ANTON	Antonio Moreno Taquería	México D.F.	05023	Mexico
AROUT	Around the Horn	London	WA1 1DP	UK
BERGS	Berglunds snabbköp	Luleå	S-958 22	Sweden
BLAUS	Blaauw Sea Delicatessen	Mannheim	68306	Germany
BONAP	Bonaparte's pizzeria	Strasbourg	67000	France
BOLID	Bólido Comidas preparadas	Madrid	28023	Spain
BONAP	Bon app'	Marseille	13008	France
BOITM	Bottom-Dollar Markets	Toronto	M5V 3M4	Canada
BSBEV	B's Beverages	London	EC2 5NT	UK
CACTU	Cactus Comidas para llevar	Buenos Aires	1010	Argentina
CENTC	Centro comercial Mochizuki	México D.F.	05022	Mexico
CHOPS	Chop-uey Chinese	Bern	3012	Switzerland
COMM1	Comércio Mineiro	São Paulo	05432-043	Brazil
CONSH	Consolidated Holdings	London	WX1 6LT	UK
DRACD	Drachendorff & Göttsche	Dresden	81062	Germany

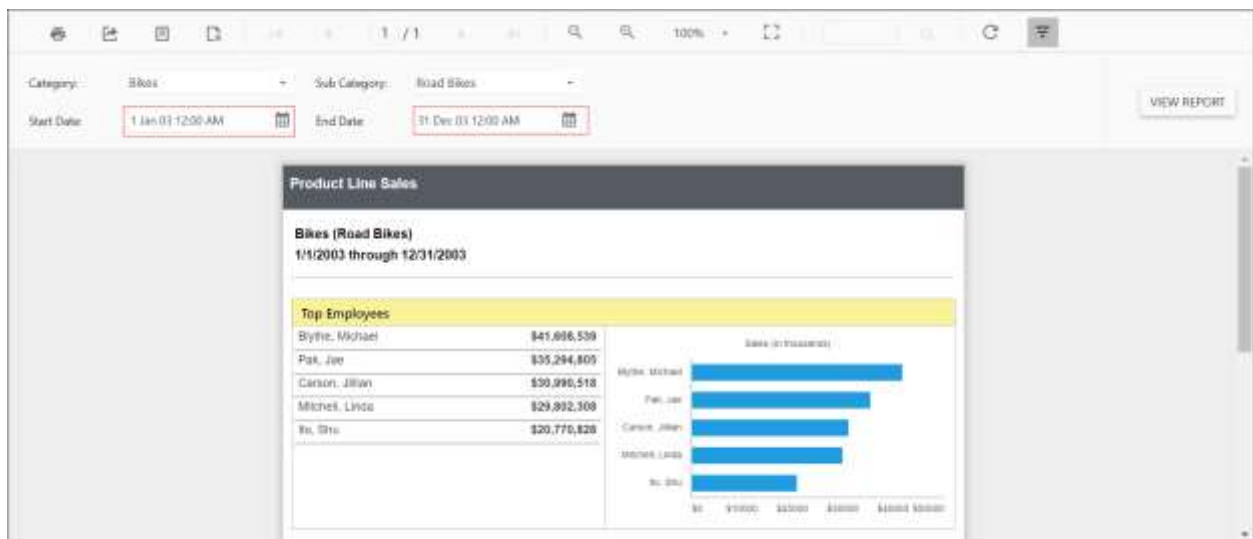
Set date and time format for parameter

The `DateTimeFormat` custom property defines the date time format to be displayed in the `DateTimePicker` popup. By default, the `DateTimeFormat` value is set as `empty`.

You can set the `DateTimeFormat` property value, as shown in the below.



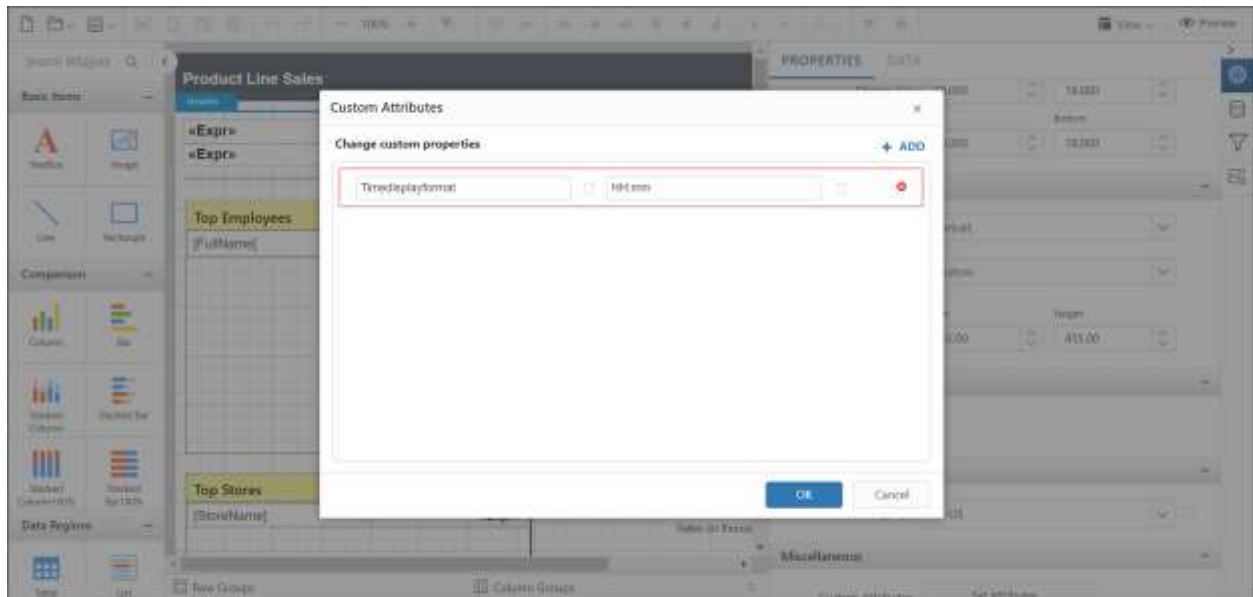
Preview the report and the see date time format in datetime parameter.



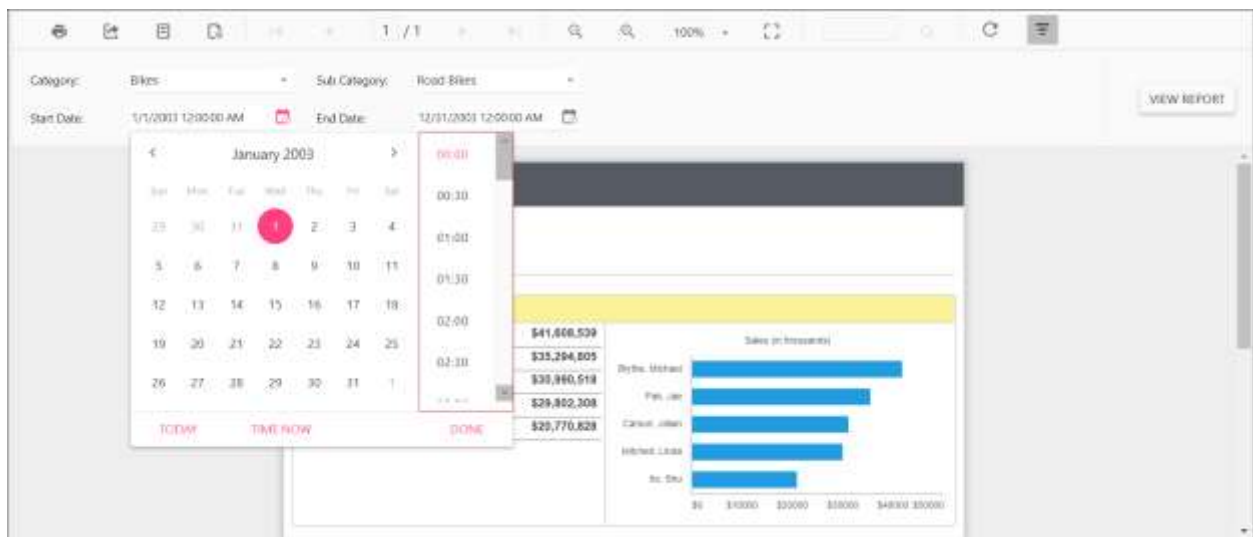
Change time display format for parameter

The `TimeDisplayFormat` custom property defines the time format to be displayed in the time dropdown inside the `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeDisplayFormat`. By default, the `TimeDisplayFormat` value is set as empty.

You can set the `TimeDisplayFormat` property value, as shown in the below.



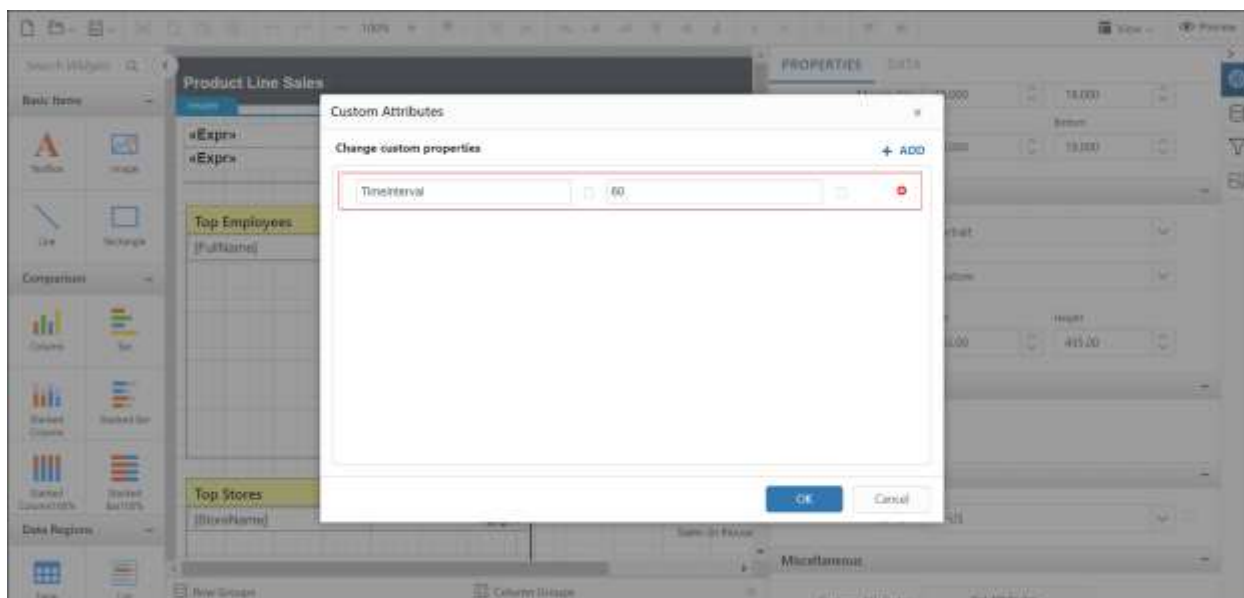
Preview the report and the see time format in datetime parameter.



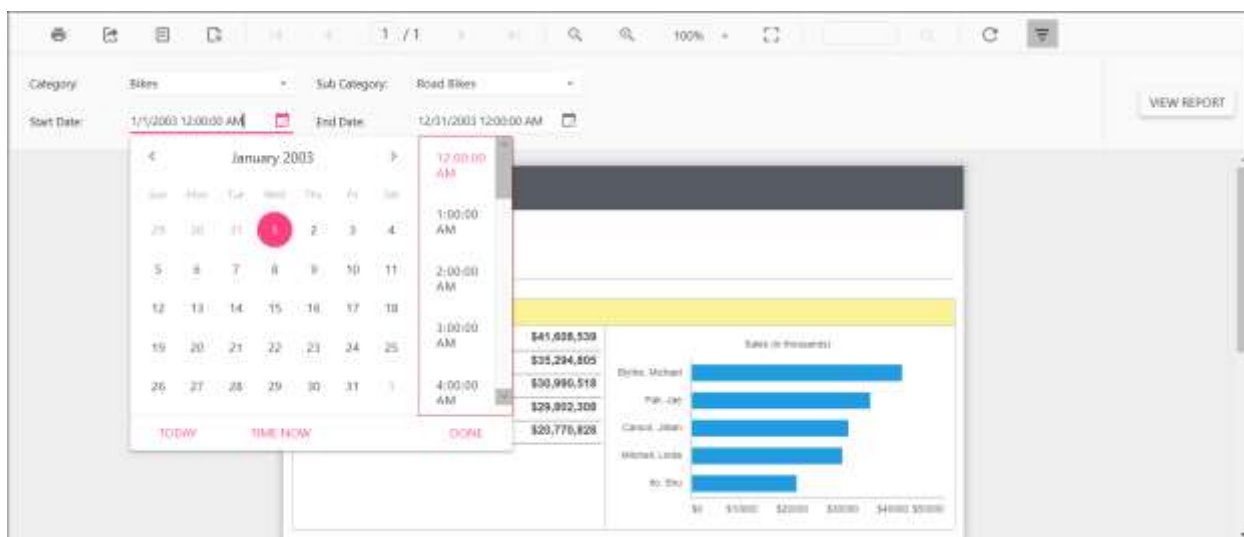
Set time interval in datetime parameter

The `TimeInterval` custom property is used to set the interval between the two adjacent time values in `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeInterval`. By default, the `TimeInterval` value is set as 30.

You can set the `TimeInterval` property value, as shown in the below.



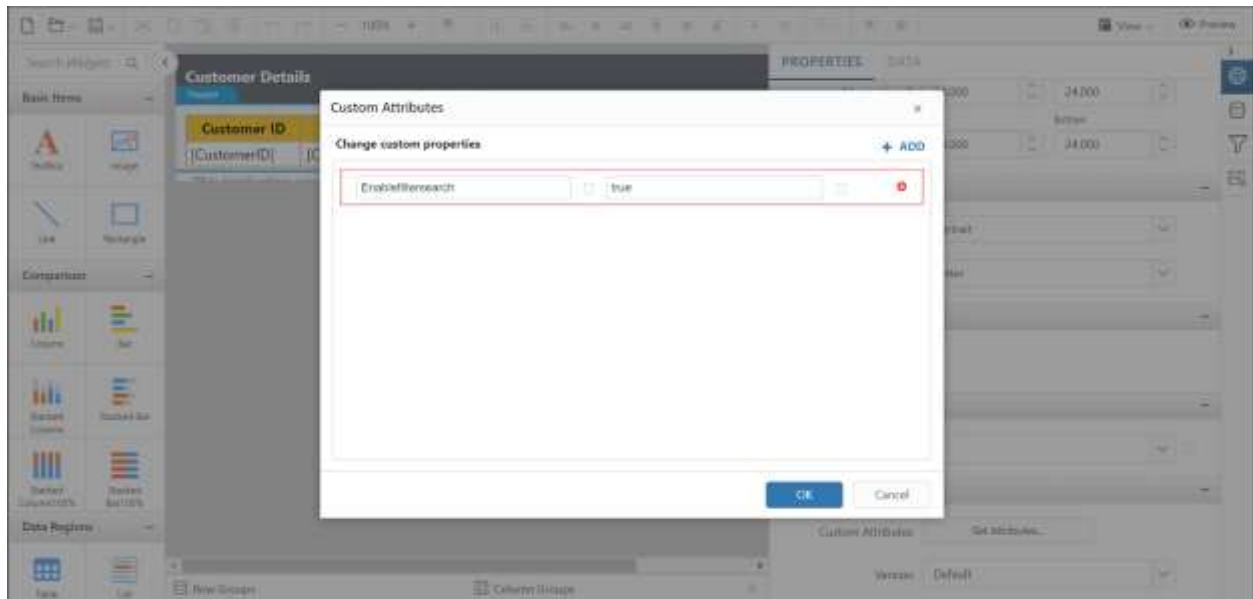
Preview the report and the see time interval in datetime parameter.



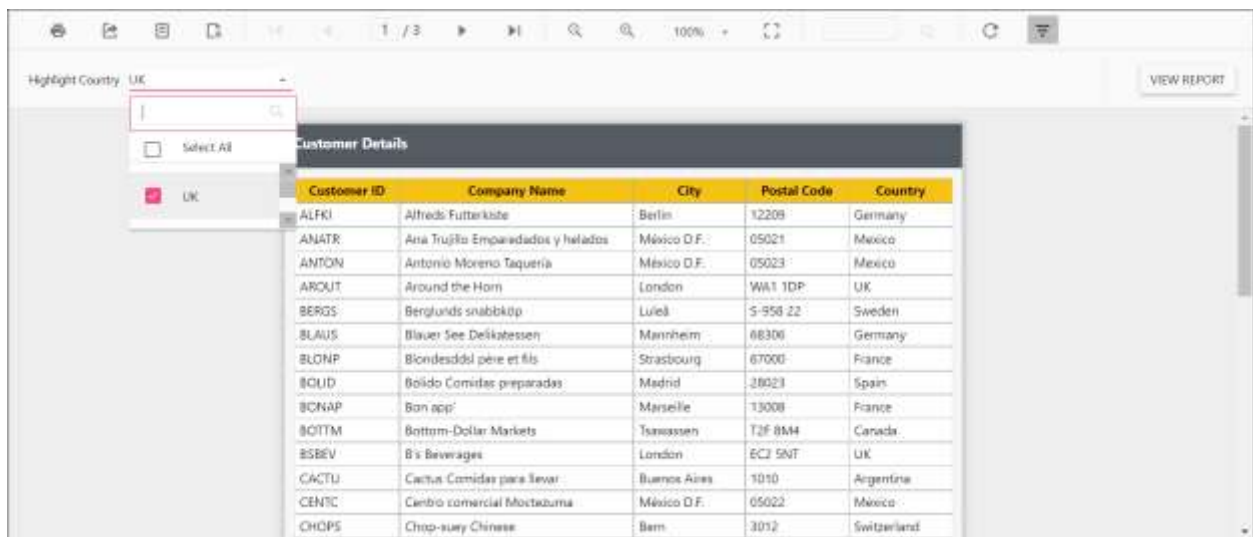
Enable filtering and searching in drop-down report parameter

Setting `EnableFilterSearch` custom property enables search and filtering option in dropdown parameter to easily find the values. The property value should be boolean. By default, the `EnableFilterSearch` value is `false`.

You can set the `EnableFilterSearch` property value, as shown in the below.



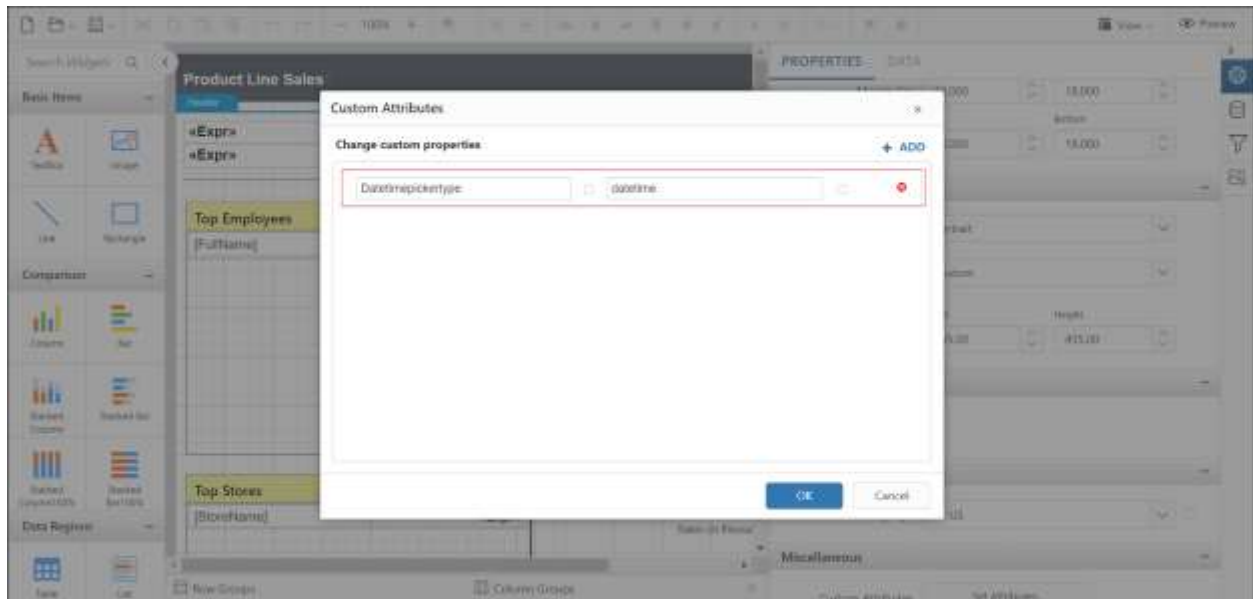
Preview the report and the see search filter in parameter.



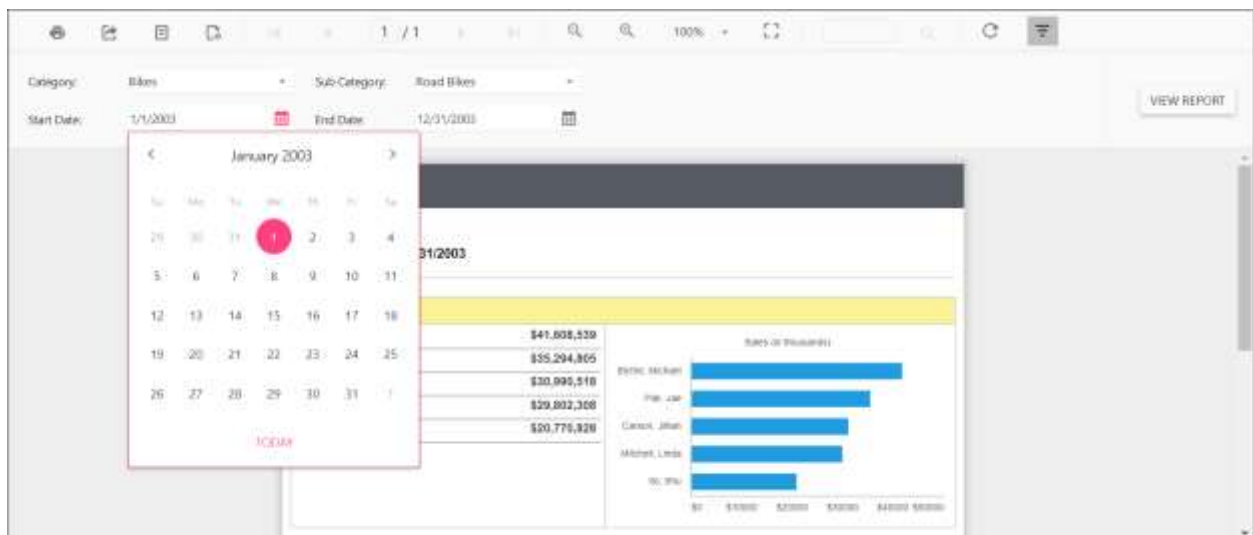
Change date report parameter to display date time picker

You can set `DateTimePickerType` custom property value as `DateTime` to change date parameter item to display `DateTimePicker` for value selection as shown the below.

You can set the `DateTimePickerType` property value, as shown in the below.

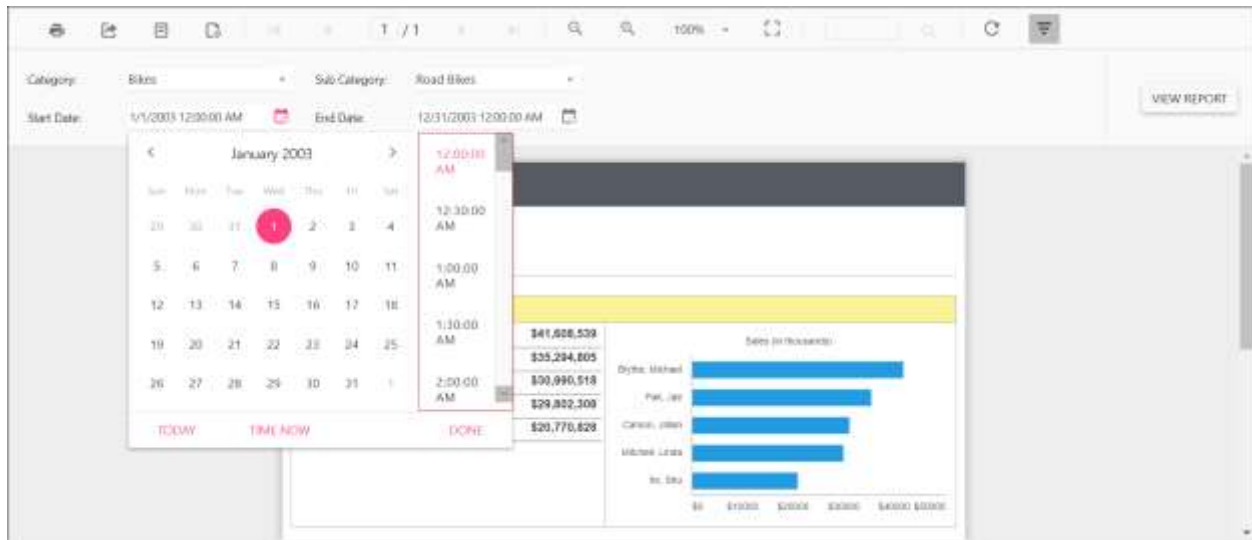


Before enabling date time picker type, the default value will be displayed as below..



Enable date time picker type and see the time in `DateTimePicker` as in below output.

Parameter custom properties **Show** the parameter values in sorted order like Ascending or Descending



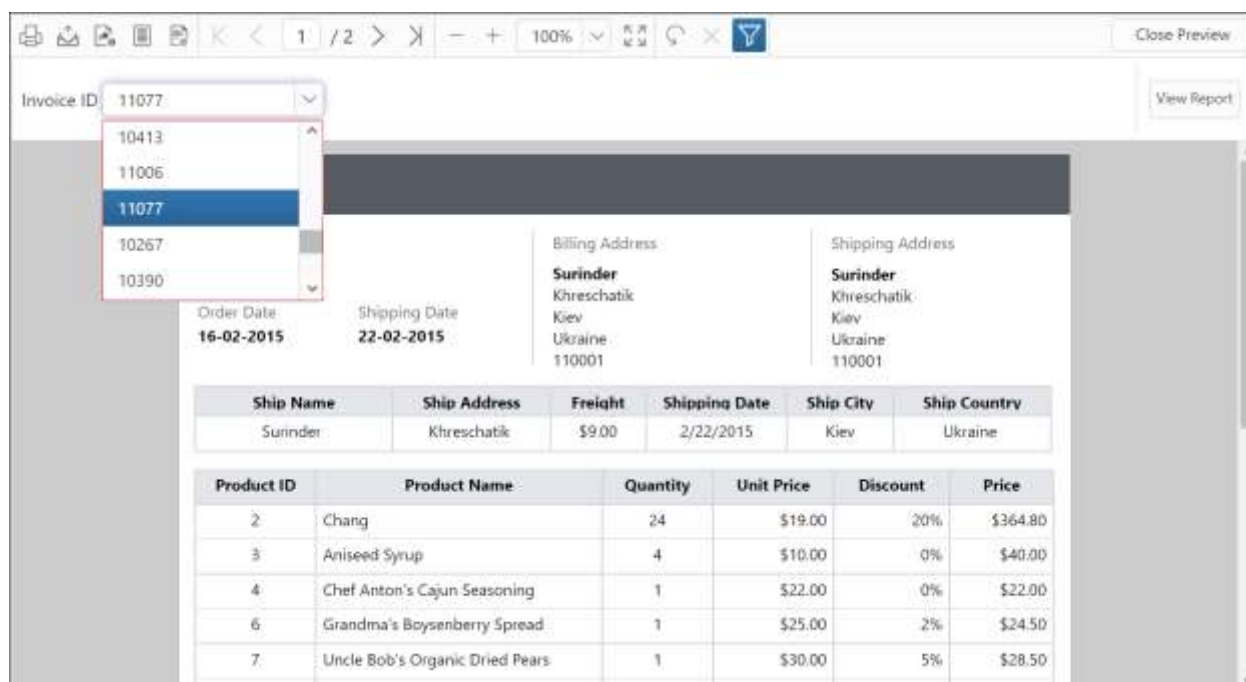
Show the parameter values in sorted order like Ascending or Descending

A drop-down type report parameter created with multiple parameter values does not show in any specific sorting order like ascending or descending. To sort any parameter values, you need to set the **Enable Sorting** option in the parameter configuration panel as shown in the below image.

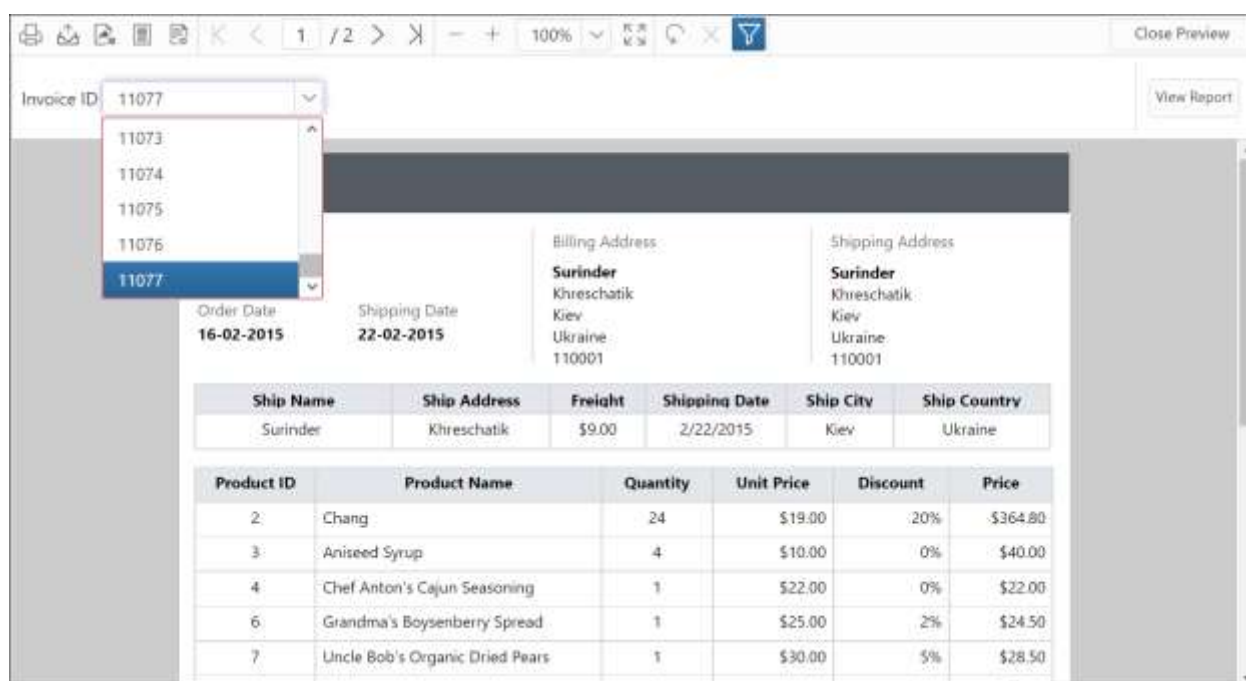
The screenshot shows the 'EDIT PARAMETER' dialog box for a parameter named 'InvoiceID'. The 'Prompt' is 'Invoice ID'. The 'Data Type' is 'Integer'. There are checkboxes for 'Allow blank value()', 'Allow null value', 'Allow multiple values', and 'Show unique values'. The 'Enable Sorting' checkbox is checked. Below it, a dropdown menu is open, showing 'Ascending' and 'Descending' options. The 'Ascending' option is selected. The dialog box also has 'Save' and 'Cancel' buttons at the bottom right.

Preview of report parameter drop down without sorting enabled.

Export custom properties **Show** the parameter values in sorted order like Ascending or Descending



Preview of report parameter drop down with sorting enabled.



Export custom properties

This topic explains the list of custom properties that are supported at the report level to control the export behaviour in JavaScript Report Viewer.

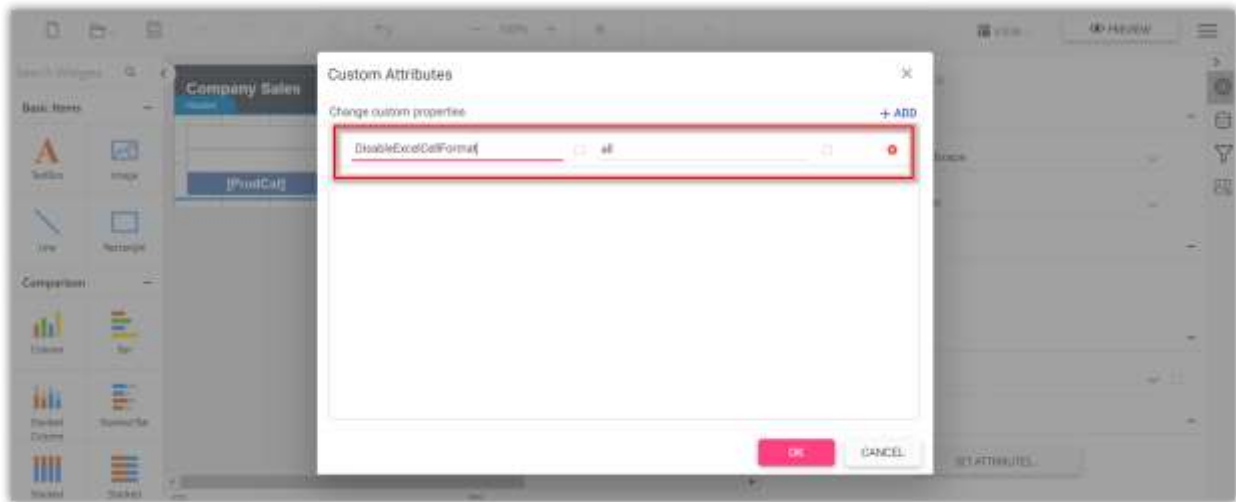
Improve excel export performance

The custom property `DisableExcelCellFormat` helps to improve the excel export performance by ignoring the rendering of report item styles. It allows property value from anyone of the following values.

- **Style** - Disables rendering of the cell styles like padding, background, color, and text style
- **Border** - Disables rendering of the cell border
- **All** - Disables rendering of the cell border, padding, background, color, and text style

Set the property value as **All** to improve maximum excel export performance.

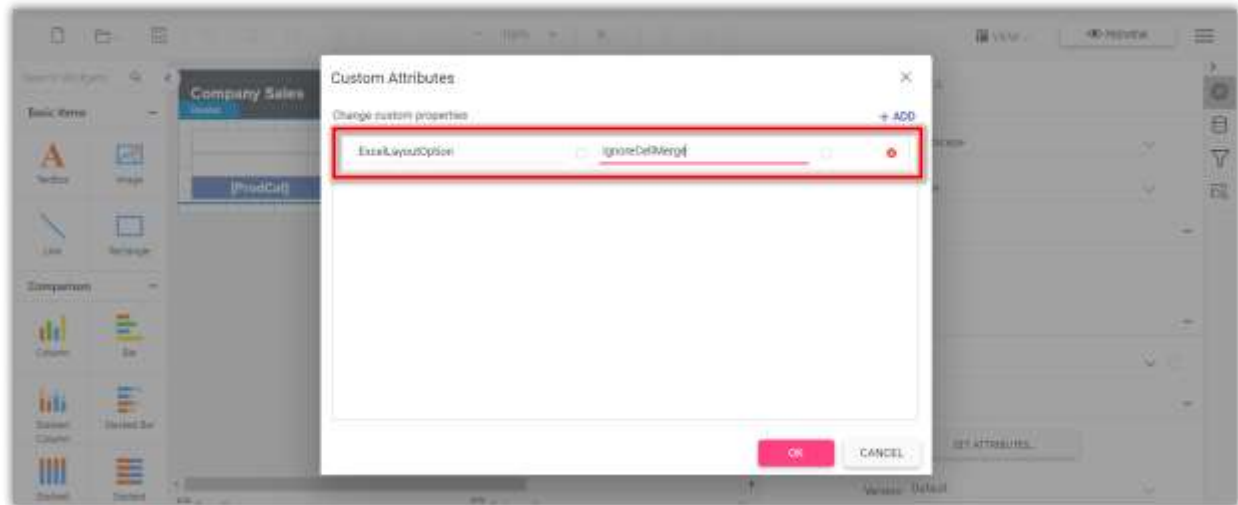
You can set the `DisableExcelCellFormat` custom property as shown below,



The `DisableExcelCellFormat` property must be added to report properties.

Improve excel export readability

Set `ExcelLayoutOption` custom property value as `IgnoreCellMerge` to improve the readability of the excel document by eliminating the tiny columns, rows, and merged cells. You can set the property value, as shown below,



The `ExcelLayoutOption` property must be added to report properties.

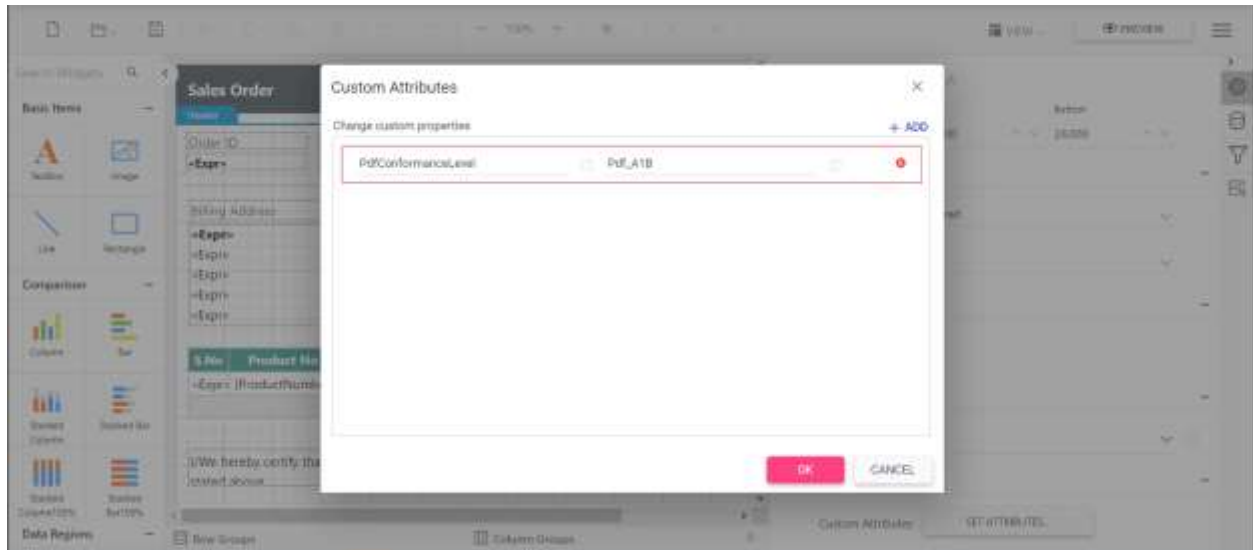
Set pdf conformance level

The `PdfConformanceLevel` allows you set PDF document versions.

You can set anyone of the following PDF conformance option.

- **PdfA1B** - You can create a PdfA1B document by specifying the conformance level as Pdf_A1B through PdfConformanceLevel Enum when creating the new PDF document.
- **PdfA2B** - You can create a PdfA2B document by specifying the conformance level as Pdf_A2B through PdfConformanceLevel Enum when creating the new PDF document
- **PdfA3B** - The PDF/A3B conformance supports the external files as attachment to the PDF document, so you can attach any document format such as Excel, Word, HTML, CAD, or XML files.
- **PdfA1A** - This conformance includes all PdfA1B requirements in addition to the features intended to improve a document's accessibility. PDF/A-1a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2A** - This conformance includes all PDF/A2B requirements in addition to the features intended to improve a document's accessibility. PDF/A-2a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2U** - This conformance includes all PdfA2U requirements, and additionally Unicode mapping for all text in the document.
- **PdfX1A2001** - You can create a PDF/X-1a document by specifying the conformance level as PdfX1A2001 through PdfConformanceLevel Enum when creating the new PDF document.

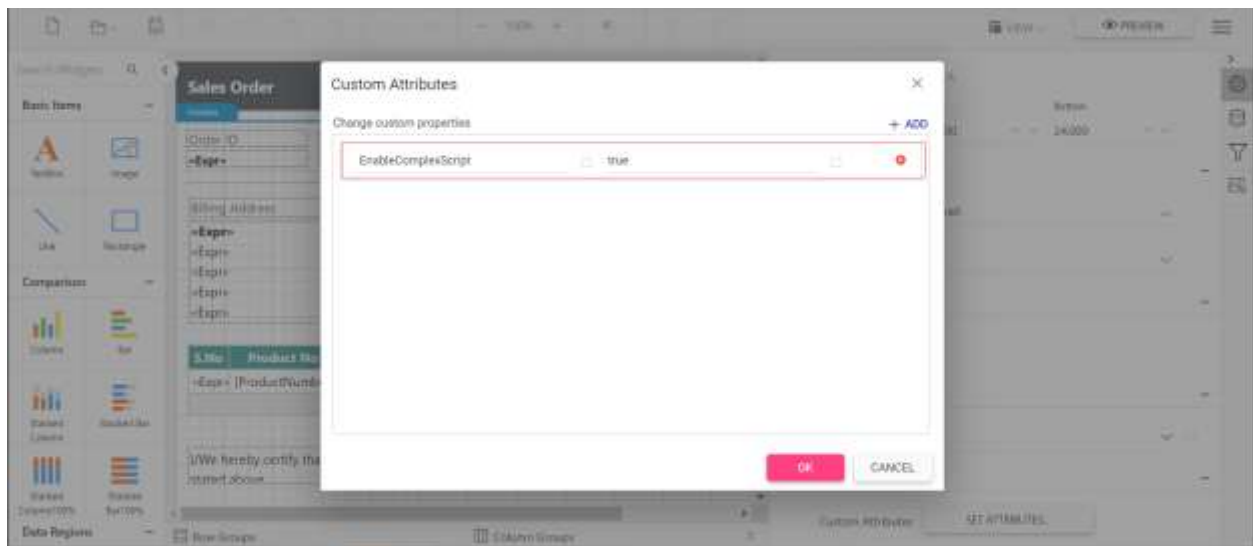
You can set the `PdfConformanceLevel` custom property as shown below,



Export pdf document with complex script

The `EnableComplexScript` custom property allows you to export pdf documents with complex script language texts.

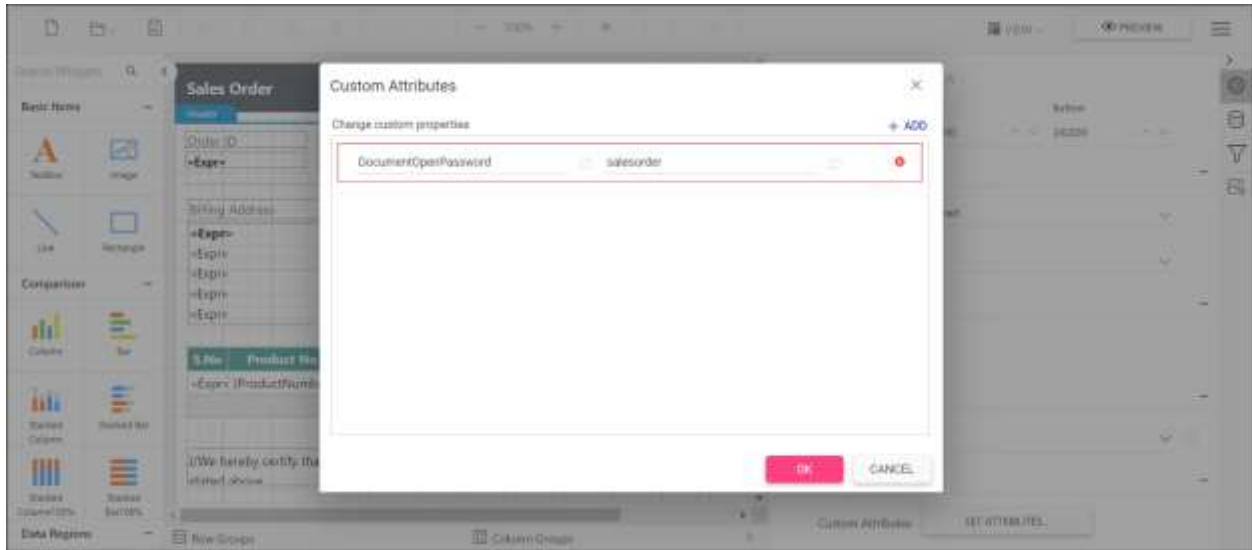
You can set the property value, as shown below,



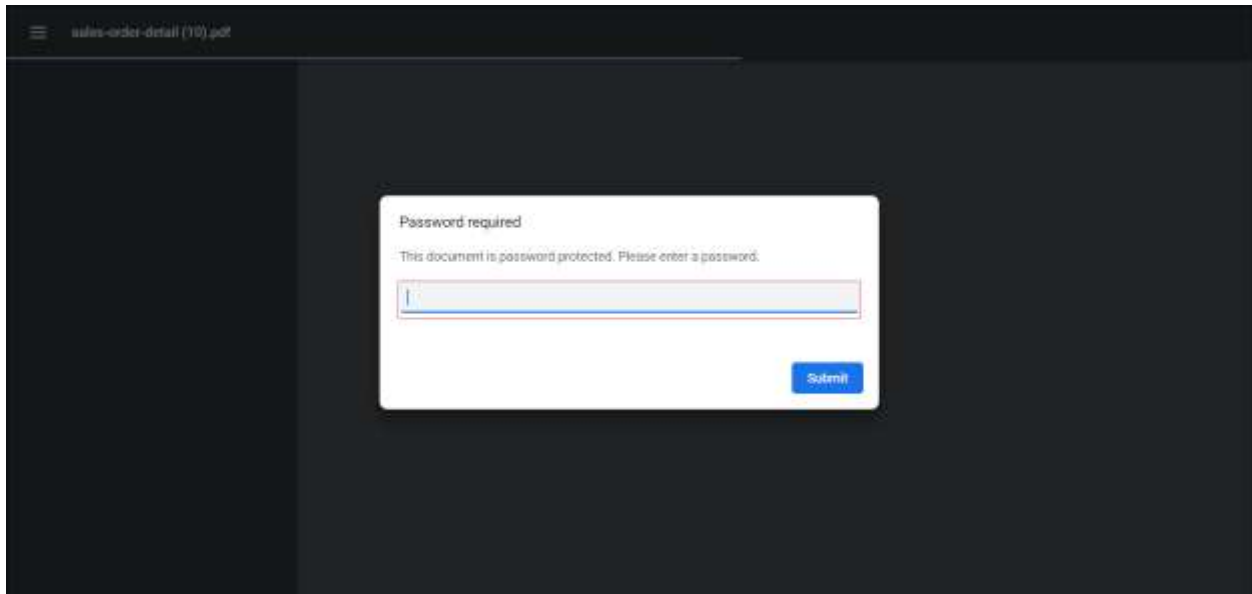
Encrypt and secure export documents

The custom property `DocumentOpenPassword` allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password.

You can set the property value, as shown below,



open the pdf document and see the below output.

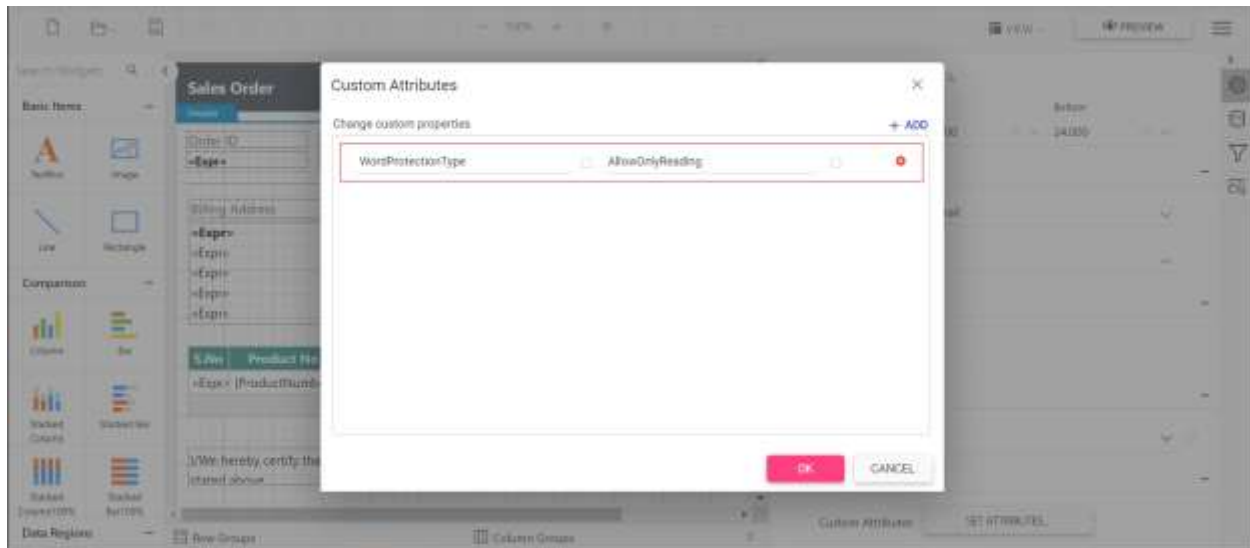


Protecting Word document from editing

The custom property `WordProtectionType` allows you to restrict a Word document from editing either by providing a password or without a password by using following types of protection.

- `AllowOnlyComments`- You can add or modify only the comments in the Word document.
- `AllowOnlyFormFields`- You can modify the form field values in the Word document.
- `AllowOnlyRevisions`- You can accept or reject the revisions in the Word document.
- `AllowOnlyReading`- You can only view the content in the Word document.
- `NoProtection`- You can access or edit the Word document contents as normally.

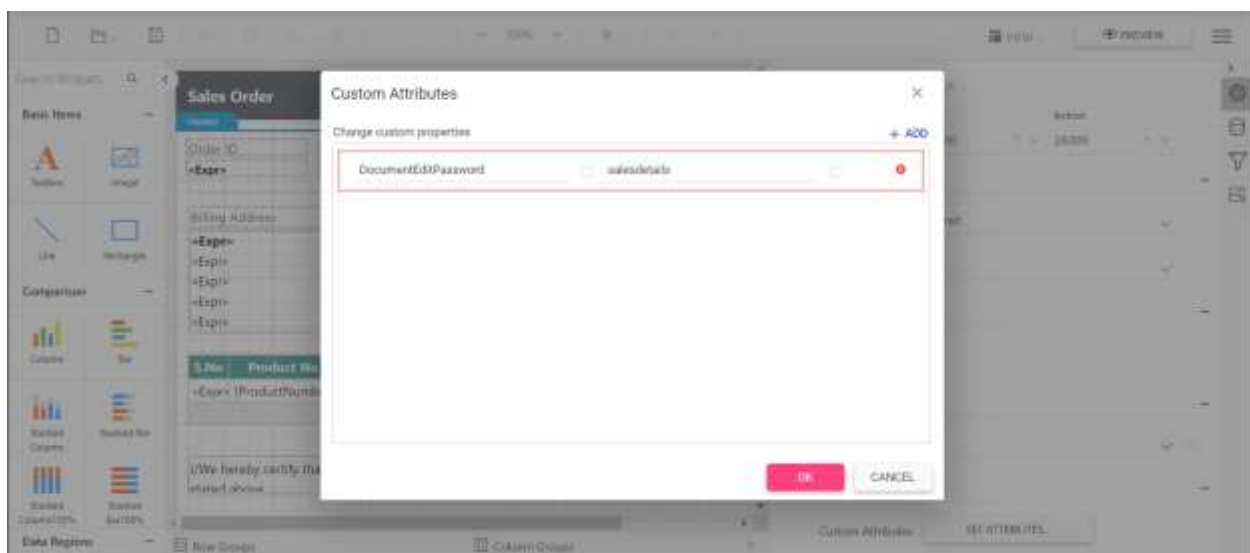
You can set the `WordProtectionType` custom property as shown below,



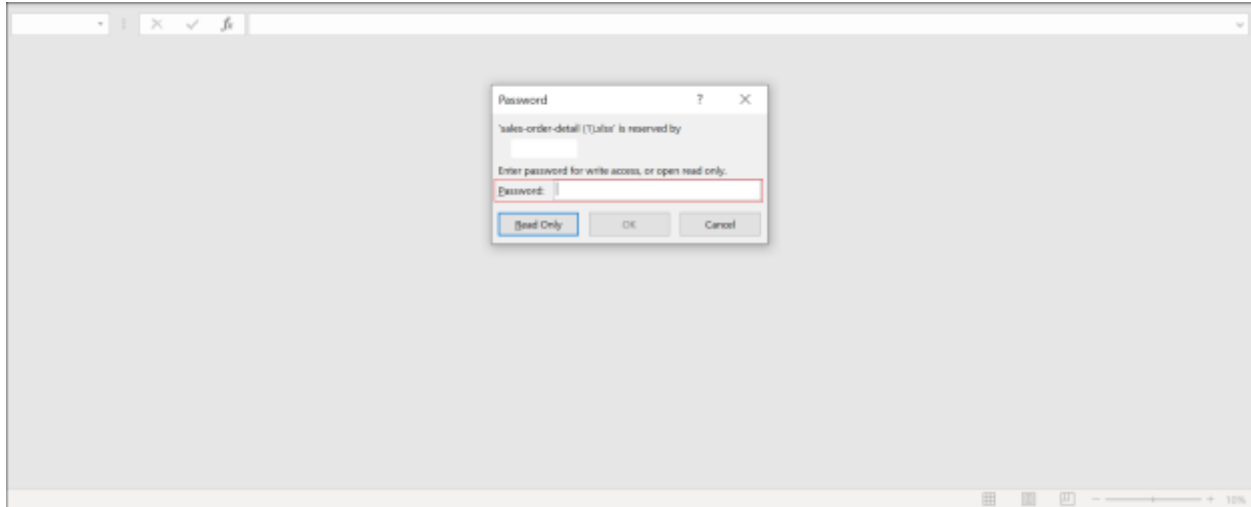
Set excel document edit password

The custom property `DocumentEditPassword` helps to allow specific users permission to modify the workbook data and save changes to the file in the excel document.

You can set the property value, as shown below,



open the excel document and see the below output.

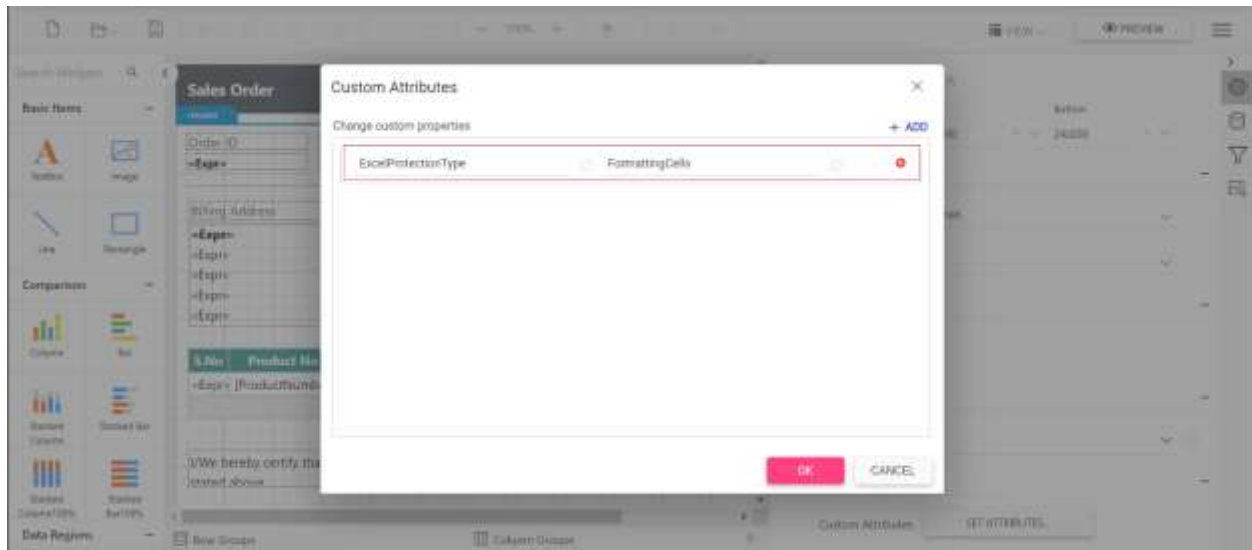


Set excel document protection

The custom property `ExcelProtectionType` allows you to protect the worksheet of excel document either by providing a password or without a password by using following types of protection.

- `None` - Represents no protection in excel sheet
- `Objects` - allows to protect shapes in excel sheet
- `Scenarios` - allows to protect scenarios.
- `FormattingCells` - allows the user to format any cell on a protected worksheet.
- `FormattingColumns` - allows the user to format any column on a protected worksheet.
- `FormattingRows` - allows the user to format any rows on a protected worksheet.
- `InsertingColumns` - allows the user to insert columns on the protected worksheet.
- `InsertingRows` - allows the user to insert rows on the protected worksheet.
- `InsertingHyperlinks` - allows the user to insert hyperlinks on the worksheet.
- `DeletingColumns` - allows the user to delete columns on the protected worksheet, where every cell in the column to be deleted is unlocked.
- `DeletingRows` - allows the user to delete rows on the protected worksheet, where every cell in the row to be deleted is unlocked.
- `LockedCells` - allows to protect locked cells.
- `Sorting` - allows the user to sort on the protected worksheet
- `Filtering` - allows the user to set filters on the protected worksheet. Users can change filter criteria but can not enable or disable an auto filter.
- `UsingPivotTables` - allows the user to use pivot table reports on the protected worksheet.
- `UnLockedCells` - allows to protect the user interface, but not macros.
- `Content` - allows to protect the contents in the excel sheet.
- `All` - allows to protect all type of protection.

You can set the `ExcelProtectionType` custom property as shown below,



Localization of Bold Reports HTML5 JavaScript Report Viewer

Localization of HTML5 JavaScript Report Viewer allows you to localize the static text such as tooltip, parameter block, and dialog text based on a specific culture. To render the static text with specific culture, refer to the following corresponding culture script files and set culture name to the [locale](#) property of the Report Viewer.

- ej.localetexts.fr-FR.min.js
- ej.culture.fr-FR.min.js

Refer this [CDN links for Localization and Culture](#) to get the Localization and Culture scripts for available Culture Code

1. Refer the ej.localetexts.fr-FR.min.js from cdn using the following code.

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.fr-FR.min.js"></script>
```

```
,
```

2. Refer the ej.culture.fr-FR.min.js from cdn using the following code.

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.fr-FR.min.js"></script>
```

```
,
```

Whether you want to get the localization script as local then install the **BoldReports.Global** NuGet package in your application.

```
`html
```

```
<script src="scripts/l10n/ej.localetexts.fr-FR.min.js"></script>
```

```
<script src="scripts/i18n/ej.culture.fr-FR.min.js"></script>
```

```
,
```

3. You can edit and preview the Report Viewer localization using the following.

```
{% tab demoPath="javascript-reporting/report-viewer/localization" files="index.html,index.js" %}
{% endtab %}
```

Responsive layout rendering of HTML5 JavaScript Report Viewer

Report Viewer will adaptively render itself with optimal user interfaces for phone, tablet, or desktop form factors. This helps your application to scale elegantly on all form factors with ease. You can enable responsive layout rendering in Report Viewer by setting [isResponsive](#) property to true.

```
`html
```

```
<div style="height: 600px; width: 950px;">
```

```
<!-- Creating a div tag which will act as a container for boldReportViewer widget.-->
```

```
<div style="height: 600px; width: 950px; min-height: 400px;" id="viewer"></div>
```

```
<!-- Setting property and initializing boldReportViewer widget.-->
```

```
<script type="text/javascript">
```

```
$(function () {
```

```
$("#viewer").boldReportViewer({
```

```
reportServiceUrl: "https://demos.boldreports.com/services/api/ReportViewer",
```

```
reportPath: '~/Resources/docs/sales-order-detail.rdl',
```

```
isResponsive: true
```

```
});
```

```
});
```

```
</script>
```

```
</div>
```

```
,
```

Normal layout

The following output shows the normal layout rendering of Report Viewer tool bar items.

Sales Order

Order ID: #SO50750, Billing Date: 25-07-2019, Order Date: 01-06-2003, Purchase Order: PO7192170677, Shipment Method: CARGO TRANSPORT 5

Billing Address: Jean Handley, Central Discount Store, 259826 Russell Rd. South, Kent, Washington 98031, United States

Shipping Address: Jean Handley, Central Discount Store, 259826 Russell Rd. South, Kent, Washington 98031, United States

Contact: Jean Handley, Ph: 582-555-0113

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.84	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	BK-M688-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46

Responsive layout

The following output shows the responsive layout rendering of Report Viewer tool bar items.

Sales Order

Order ID: #SO50750, Billing Date: 25-07-2019, Order Date: 01-06-2003, Purchase Order: PO7192170677, Shipment Method: CARGO TRANSPORT 5

Billing Address: Jean Handley, Central Discount Store, 259826 Russell Rd. South, Kent, Washington 98031, United States

Shipping Address: Jean Handley, Central Discount Store, 259826 Russell Rd. South, Kent, Washington 98031, United States

Contact: Jean Handley, Ph: 582-555-0113

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38

Limitations

RDL Specification

The Report Viewer control does not support RDL Specification for SQL Server 2000 and SQL Server 2005.

Report Layout

- In the Tablix cell split layout process when the table cell width value exceeds the page width, the entire cell moves to the next page to display the complete cell items.

Expressions

The object function and VB function do not have complete support.

SSRS

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the server. If the report has any data source that uses credentials to connect with the database, then you must specify the data source credentials for each report data source to establish database connection.

Data Processing Extensions for Report Viewer

Data Processing Extensions providing the additional data source supports, which is not available in built-in of Report Viewer. Providing the built-in data source supports only for the .NET Libraries that built-in of the .NET Framework. If the database assemblies do not come with .NET Framework, then you have to built the data source extension for additional data sources such as MySQL, OData, PostgreSQL etc.

1. [MySQL Data Processing Extension](#)
2. [WebAPI Data Processing Extension](#)

Create a MySQL Data Processing Extension for Report Viewer

This article explains you how to create MySQL Data Source extension with Report Viewer.

Configure dependent Assemblies

Add **BoldReports.Web** reference with the application to provide the extension support for MySQL data source. For this, add the MySQL reference with the application to provide the Data Processing Extensions support.

If you are going to create separate libraries for the Data Processing Extensions, then you have to add **BoldReports.Web** and MySQL references with your class library.

Implementing Data Processing Extensions

IDataExtension interface needs to be implemented with your Data Processing Extension class to provide the extension support and **IDataExtension** members will be invoked from the Report Viewer.

```
`csharp
namespace BoldReports.DataExtensions.MySQL
{
    public class MySQLDataExtension : BoldReports.Data.IDataExtension
    {
        ,
```

TestConnection

Data Processing Extensions **TestConnection** member will be invoked from Report Viewer for configured data source when Test Connection performed with Data source creation. You should write the implementation in this method to perform the connection testing of the configured data source.

Implementation for MySQL Data Processing Extension.

```
`csharp
```

```

public bool TestConnection(string connectionString)
{
    MySqlConnection connection = new MySqlConnection(connectionString);
    try
    {
        connection.Open();
        return true;
    }
    catch
    {
        return false;
    }
    finally
    {
        connection.Close();
        connection.Dispose();
    }
}

```

GetSchemaData

Data Processing Extensions **GetSchemaData** member will be invoked from Report Viewer for configured data source while changing the CommandText/Query in DataSet designer. It is invoked to generate the fields for DataSet based on dataset query.

```

`csharp
public object GetSchemaData()
{
    return null;
}

```

GetData

Data Processing Extensions **GetData** member will be invoked from ReportViewer to get the data for configured data source. We should write the implementation in this method to execute the data source query and return the query result with this method.

```

`csharp
public object GetData()

```

```
{  
var connectionString = this.GetConnectionString();  
using (MySQLConnection connection = new MySQLConnection(connectionString))  
{  
try  
{  
connection.Open();  
DataTable dataTable = this.GetTable(connection, this.CommandText);  
return dataTable;  
}  
catch (Exception ex)  
{  
throw ex;  
}  
finally  
{  
connection.Close();  
connection.Dispose();  
}  
}  
}
```

Deploy Data Processing Extensions in ReportViewer

Data Processing Extensions Configuration

To configure the prepared Data Processing Extensions with Report Viewer, add the Data Processing Extensions information with following format in the application configuration,

```
`xml  
<configSections>  
  <section name="ReportingExtensions" type="BoldReports.Configuration.Extensions, BoldReports.Web"  
    allowLocation="true" allowDefinition="Everywhere" />  
</configSections>  
<ReportingExtensions>  
  <DataExtensions>
```



```
<Extension Name="MySQL" Assembly="BoldReports.DataExtensions.MySQL"
Type="BoldReports.DataExtensions.MySQL.MySQLDataExtension"/>
</DataExtensions>
</ReportingExtensions>
`
```

You should add the dependent assemblies with application or it should be there in the application location.

Configuration with extension assembly

You can use the following format, if the Data Extension prepared with Class library,

```
`xml
<configSections>

<section name="ReportingExtensions" type="BoldReports.Configuration.Extensions, BoldReports.Web"
allowLocation="true" allowDefinition="Everywhere" />

</configSections>

<ReportingExtensions>

<DataExtension>

<Extension Name="MySQL" Assembly="BoldReports.DataExtensions.MySQL"
Type="BoldReports.DataExtensions.MySQL.MySQLDataExtension"/>

</DataExtension>

</ReportingExtensions>
`
```

You should add the Data Processing Extensions assemblies and dependent assemblies with application or it should be there the application location.

[MySQL Data Processing Extensions sample](#)

Create a WebAPI Data Processing Extension for Report Viewer

This article explains how to create WebAPI Data Source extension with Report Viewer.

Configure dependent Assemblies

Add the **BoldReports.Web** reference with the application to provide the extension support for WebAPI data source. For this, add the WebAPI reference with the application to provide the Data Processing Extensions support.

If you are going to create separate libraries for the Data Processing Extensions, then you have to add the **BoldReports.Web** and WebAPI references with your class library.

Implementing Data Processing Extensions

The **IDataExtension** interface needs to be implemented with your Data Processing Extension class to provide the extension support and **IDataExtension** members will be invoked from the Report Viewer.

```
`csharp
```

```
namespace BoldReports.DataExtensions.WebAPI
{
    public class WebAPIDataExtension : BoldReports.Data.IDataExtension
    {
        ,
```

TestConnection

The Data Processing Extensions **TestConnection** member will be invoked from the Report Viewer for configured data source when Test Connection performed with Data source creation. You should write the implementation in this method to perform the connection testing of the configured data source.

Implementation for WebAPI Data Processing Extension.

```
`csharp
public bool TestConnection(out string error)
{
    error = string.Empty;
    return true;
}
,
```

GetData

The Data Processing Extensions **GetData** member will be invoked from ReportViewer to get the data for the configured data source. We should write the implementation in this method to execute the data source query and return the query result with this method.

```
`csharp
public object GetData(out string error)
{
    try
    {
        error = string.Empty;
        this.updateQueryData(this.Command.Text);
        var settings = new JsonSerializerSettings
        {
            DateParseHandling = DateParseHandling.DateTimeOffset
        };
        return this.GetReportData(
            Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(this.InvokeServiceRequest(this.ConnectionPr
            operties.ConnectionString).ToString(), settings));
```

```
}  
catch (Exception ex)  
{  
    error = ex.Message;  
    return null;  
}  
}  
,
```

Deploy Data Processing Extensions in ReportViewer

Data Processing Extensions Configuration

To configure the prepared Data Processing Extensions with the Report Viewer, add the Data Processing Extensions information with following format in the application configuration,

```
`xml  
<configSections>  
    <section name="ReportingExtensions" type="BoldReports.Configuration.Extensions, BoldReports.Web" allowLocation="true" allowDefinition="Everywhere" />  
</configSections>  
<ReportingExtensions>  
    <DataExtensions>  
        <Extension Name="WebAPI" Assembly="BoldReports.DataExtensions.WebAPI" Type="BoldReports.DataExtensions.WebAPI.WebAPIDataExtension"/>  
    </DataExtensions>  
</ReportingExtensions>  
,
```

You should add the dependent assemblies with the application or it should be there in the application location.

Configuration with extension assembly

You can use the following format, if the Data Extension prepared with Class library,

```
`xml  
<configSections>  
    <section name="ReportingExtensions" type="BoldReports.Configuration.Extensions, BoldReports.Web" allowLocation="true" allowDefinition="Everywhere" />  
</configSections>  
<ReportingExtensions>  
    <DataExtension>
```

```
<Extension Name="WebAPI" Assembly="BoldReports.DataExtensions.WebAPI"
Type="BoldReports.DataExtensions.WebAPI.WebAPIDataExtension"/>
</DataExtension>
</ReportingExtensions>
```

You should add the Data Processing Extensions assemblies and dependent assemblies with the application or it should be there the application location.

[WebAPI Data Processing Extensions sample](#)

Samples and demos

Browse and explore our ready-to-use RDL, RDLC reports, samples, online and offline demos.

Locally installed reports

You can obtain sample rdl and rdlc files from Bold Reports installed location

```
%localappdata%\Bold Reports\Embedded Reporting\Samples\Common\Data\ReportTemplate.
```

Offline demos

The offline samples are provided in the Bold Reporting Tools setup. For more details, refer to the [Bold Reporting Tools sample deployment](#).

Online demos

You can view the JavaScript Report Viewer online demo samples from [here](#).

GitHub demo samples

Click [here](#) to view the GitHub Report Viewer demo samples.

Reporting Service application

Click [here](#) to view the GitHub source location of the reporting services that are used in our documentation.

Migrate Report Viewer application

In our Bold Reports new assemblies are introduced for both client and server-side to resolve the compatibility problem between Essential Studio Report Viewer versions. It has changes in both Web API service and client-side scripts.

This section provides step-by-step instructions for migrating Report Viewer from Syncfusion Essential Studio release version to Bold Reports version of JavaScript Report Viewer application:

Client-side migration

To refer the latest scripts and CSS references of Report Viewer do the following steps,

1. Remove the following scripts and CSS references from the Report Viewer page.
 - o ej.web.all.min.css
 - o ej.web.all.min.js
2. Replace the following scripts and CSS references in the <head> tag of the Report Viewer page.

```
`html
```

```

<link href="Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="http://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<!-- Report Viewer component script-->
<script src="Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<script src="Scripts/bold-reports/data-visualization/ej.chart.min.js"></script>
<script src="Scripts/bold-reports/bold.report-viewer.min.js"></script>
`

```

Adding data visualization scripts

To render the report with data visualization components such as chart, gauge, and map items, add scripts of the visualization element. The following table shows the script reference that needs to be added in Report Viewer page for data visualization elements.

Visualization item | Script file

Gauge | ej2-base.min.js, ej2-data.min.js, ej2-pdf-export.min.js, ej2-svg-base.min.js, ej2-lineargauge.min.js and ej2-circulargauge.min.js

Map | ej2-maps.min.js

Chart | ej.chart.min.js

To render the chart report item, add chart control script `ej.chart.min.js` before the `bold.report-viewer.min.js` reference in the `\Views\Shared_Layout.cshtml` page as demonstrated in the following code sample.

```

`html
<link href="Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<!-- Report Viewer component script-->
<script src="Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<script src="Scripts/bold-reports/data-visualization/ej.chart.min.js"></script>
<script src="Scripts/bold-reports/bold.report-viewer.min.js"></script>
`

```

The following code can be used to render the chart, gauge, and map report items in Report Viewer.

```

`html
<link href="Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<!--Used to render the gauge item. Add this script, only if your report contains the gauge report item. -->
<script src="Scripts/bold-reports/common/ej2-base.min.js"></script>

```

```

<script src="Scripts/bold-reports/common/ej2-data.min.js"></script>
<script src="Scripts/bold-reports/common/ej2-pdf-export.min.js"></script>
<script src="Scripts/bold-reports/common/ej2-svg-base.min.js"></script>
<script src="Scripts/bold-reports/data-visualization/ej2-circulargauge.min.js"></script>
<script src="Scripts/bold-reports/data-visualization/ej2-lineargauge.min.js"></script>
<!--Used to render the map item. Add this script, only if your report contains the map report item.-->
<script src="Scripts/bold-reports/data-visualization/ej2-maps.min.js"></script>
<!-- Report Viewer component script-->
<script src="Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<!--Used to render the chart item. Add this script, only if your report contains the chart report item.-->
<script src="Scripts/bold-reports/data-visualization/ej.chart.min.js"></script>
<script src="Scripts/bold-reports/bold.report-viewer.min.js"></script>

```

Server-side migration

1. In the Solution Explorer, right-click the **References** and remove the **Syncfusion.EJ.ReportViewer** assembly reference.
2. Add the assembly from the Bold Reporting NuGet package **BoldReports.Web**. To add from NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search for **BoldReports.Web** NuGet package, and then install in your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Web API Controller

1. The **IReportController** interface is moved to **BoldReports.Web.ReportViewer**. Open the Report Viewer Web API Controller file and remove the following using statement.

```

`csharp
using Syncfusion.EJ.ReportViewer;

```

2. Add the following using statement.

```

`csharp
using BoldReports.Web.ReportViewer;

```

Your application is successfully upgraded to the latest version of Report Viewer, and you can run the application with new assemblies.

Report export configuration

Now, the **BoldReports.Web** can export the reports with data visualization components only using web components. It is mandatory to configure the web scripts in Report Viewer Web API controller for exporting data visualization components such as chart, gauge, and map that are used in report definition. To configure the scripts in Web API, refer to the following steps:

1. Open the Report Viewer Web API controller.
2. Configure the following scripts and styles in **OnInitReportOptions** on Web API controller:
 - **jquery-1.10.2.min.js**
 - **ej2-base.min.js**, **ej2-data.min.js**, **ej2-pdf-export.min.js**, **ej2-svg-base.min.js**, **ej2-lineargauge.min.js** and **ej2-circulargauge.min.js** - Exports the gauge item. Add this script only if your report contains the gauge report item.
 - **ej2-maps.min.js** - Exports the map item. Add this script only if your report contains the map report item.
 - **bold.reports.common.min.js**
 - **bold.reports.widgets.min.js**
 - **ej.chart.min.js** - Exports the chart item. Add this script only if your report contains the chart report item.
 - **bold.report-viewer.min.js**
3. Replace the following codes in Report Viewer Web API controller.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    var resourcesPath = System.Web.Hosting.HostingEnvironment.MapPath("~/Scripts");
    reportOption.ReportModel.ExportResources.Scripts = new List<string>
    {
        //Gauge component scripts
        resourcesPath + @"\"bold-reports\\common\\ej2-base.min.js",
        resourcesPath + @"\"bold-reports\\common\\ej2-data.min.js",
        resourcesPath + @"\"bold-reports\\common\\ej2-pdf-export.min.js",
        resourcesPath + @"\"bold-reports\\common\\ej2-svg-base.min.js",
        resourcesPath + @"\"bold-reports\\data-visualization\\ej2-lineargauge.min.js",
        resourcesPath + @"\"bold-reports\\data-visualization\\ej2-circulargauge.min.js",
        //Map component script
    }
```

```

resourcesPath + @"\"bold-reports\data-visualization\ej2-maps.min.js",
resourcesPath + @"\"bold-reports\common\bold.reports.common.min.js",
resourcesPath + @"\"bold-reports\common\bold.reports.widgets.min.js",
//Chart Component Script
resourcesPath + @"\"bold-reports\data-visualization\ej.chart.min.js",
//Report Viewer Component Script
resourcesPath + @"\"bold-reports\bold.report-viewer.min.js"
};
reportOption.ReportModel.ExportResources.DependentScripts = new List<string>
{
resourcesPath + @"\"jquery-1.7.1.min.js"
};
}
,

```

The data visulization components will not export without the above script configurations.

NuGet Packages for JavaScript

Refer to the following steps to configure Bold Reports NuGet packages for JavaScript application.

Configure NuGet feed URL

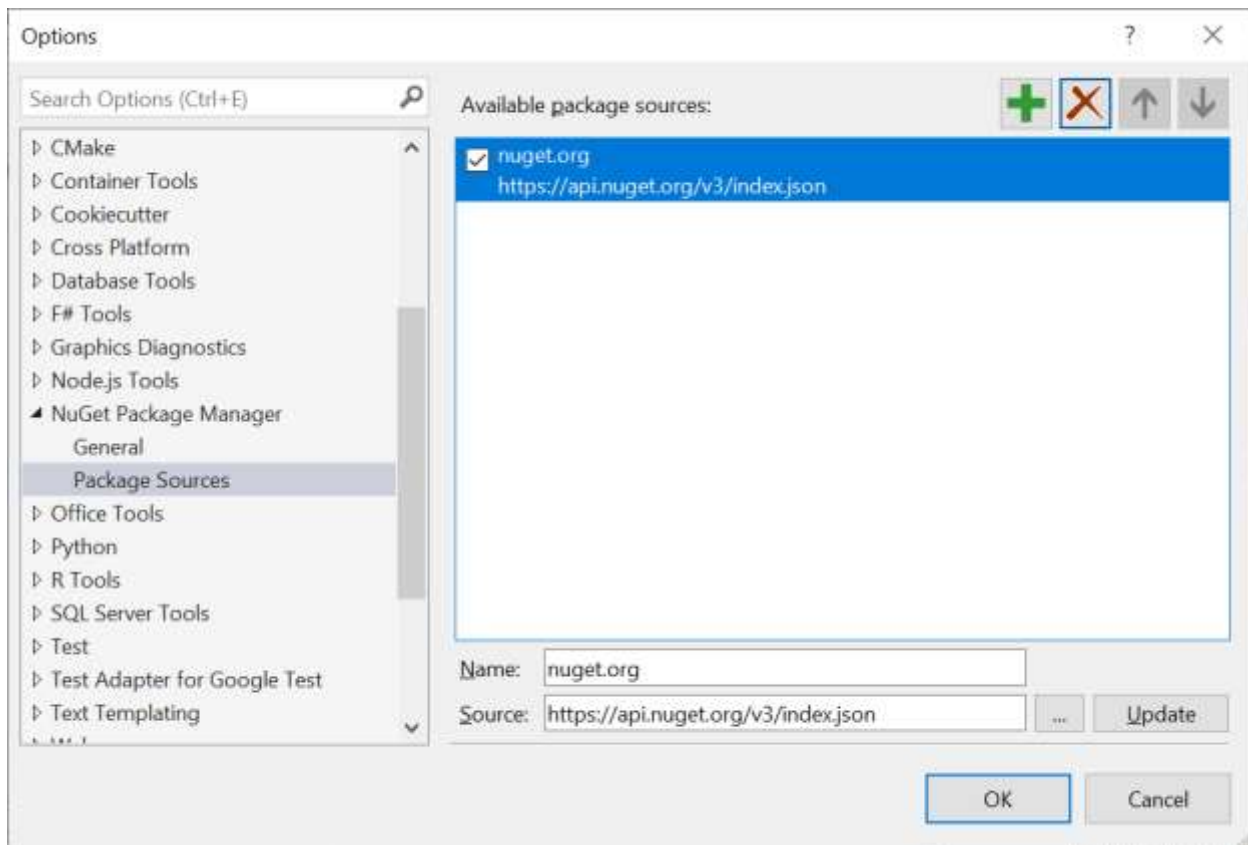
Online NuGet feed URL

The Bold Reporting NuGet packages are published in **Nuget.org**. To configure the online packages, use the following steps:

1. Open Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager** and select **Package Sources**.
4. Click the **Add** button, enter the following Package Name and Package Source URL, and then click **Update**.

Name: NuGet.org

Source: <https://api.nuget.org/v3/index.json>



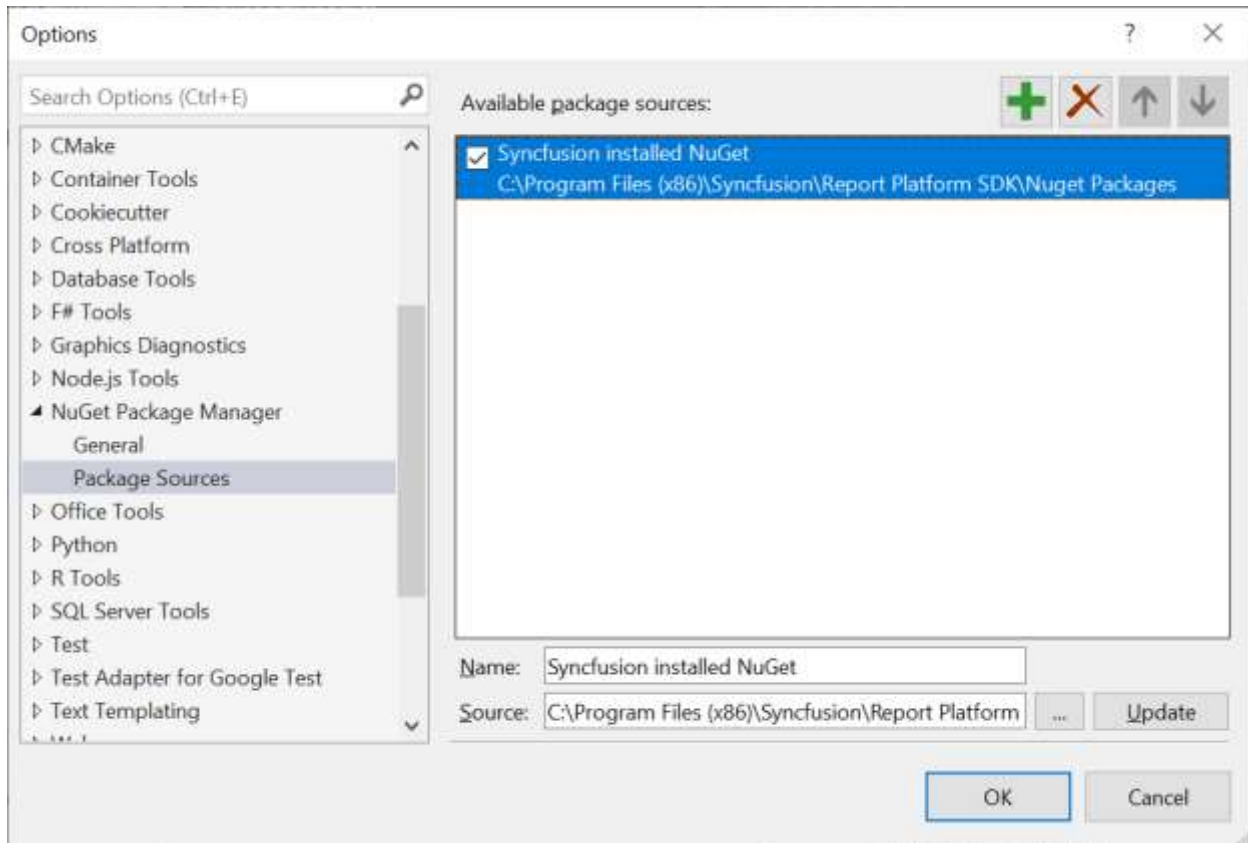
Offline NuGet feed URL

Bold Reporting NuGet packages are shipped into our Bold Reporting Tools build. To configure the packages from Bold Reports installed location, use the following steps:

1. Open your Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager**, and then select **Package Sources**.
4. Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: Bold Reports installed NuGet

Source: {System Drive}:\Program Files (x86)\Bold Reports\Reporting Tools\Nuget Packages.



The system drive varies based on the installed location in your machine.

Installing NuGet packages

Install using NuGet Package Manager

The NuGet Package Manager can be used to search and install NuGet packages in Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution**. Alternatively, right-click the project/solution in Solution Explorer tab, and choose **Manage NuGet Packages**.
2. By default, the **NuGet.org** package is selected in the **Package source** drop-down. Select package source, search for the packages **BoldReports.Web**, and then click **Install** button.

Install using Package Manager Console

To install the Reporting component using the Package Manager Console as NuGet packages,

1. On the **Tools** menu, select **NuGet Package Manager**, and then click **Package Manager Console**.
2. Run the following NuGet installation commands:

```
`cmd
```

install specified package in default project

```
Install-Package <Package Name>
```

install specified package in default project with specified package source

Install-Package <Package Name> -Source <Source Location>

install specified package in specified project

Install-Package <Package Name> -ProjectName <Project Name>

`

For example:

`cmd

install specified package in default project

Install-Package BoldReports.Web

install specified package in default project with specified Package Source

Install-Package BoldReports.Web -Source "https://api.nuget.org/v3/index.json"

install specified package in specified project

Install-Package BoldReports.Web -ProjectName BoldReportsApplication

`

Upgrading NuGet packages

Upgrading using NuGet Package Manager

NuGet packages can be updated to their specific version or latest version available in the Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution....** Alternatively, right-click the project/solution in the Solution Explorer tab, and then choose **Manage NuGet Packages**.
2. Select the **Updates** tab to see the packages available for update from the desired package sources, select the required packages and specific version from the drop-down, and then click the **Update** button.

Upgrading using Package Manger Console

To update the installed Bold Reporting NuGet packages using the Package Manager Console:

1. On the **Tools** menu, select **NuGet Package Manager**, and then select **Package Manager Console**.
2. Run the following NuGet installation commands:

`cmd

Update specific NuGet package in default project

Update-Package <Package Name>

Update all the packages in default project

Update-Package

Update specified package in default project with specified package source

Update-Package <Package Name> -Source <Source Location>

Update specified package in specified project

Update-Package <Package Name> -ProjectName <Project Name>

,

For example:

```
`cmd
```

Update specified Bold Reporting NuGet package

Update-Package BoldReports.Web

Update specified package in default project with specified Package Source

Update-Package BoldReports.Web -Source "https://api.nuget.org/v3/index.json"

Update specified package in specified project

Update-Package BoldReports.Web -ProjectName BoldReportsApplication

,

Upgrading using NuGet CLI

Using the NuGet CLI, all the NuGet packages in the project can be updated to the available latest version:

1. Download the latest [NuGet CLI](#).

To update the existing `nuget.exe` to latest version use the following command:

```
`cmd
```

```
nuget update -self
```

,

2. Open the downloaded executable location in the command window. Run the following "update commands" to update the Bold Reporting NuGet packages.

```
`cmd
```

update all NuGet packages from config file

nuget update <configPath> [options]

update all NuGet packages from specified Packages Source

nuget update -Source <Source Location> [optional]

`configPath` is optional. It identifies the `package.config` or solutions file lists the packages utilized in the project.

For example:

```
`cmd
```

Update all NuGet packages from config file

```
nuget update "C:\Users\BoldReportsApplication\package.config"
```

Update all NuGet packages from specified Packages Source

```
nuget update -Source "https://api.nuget.org/v3/index.json"
```

The Update command is not working as expected in Mono (Mac and Linux) and projects using PackageReference format.

CDN

The [CDN](#) links are provided individually for all the scripts and style sheets of Bold Reports JavaScript Reporting component.

CDN scripts links

Bold Reports dependency libraries

The CDN script files are maintained for each version of the Report Platform SDK individually.

The three scripts, namely `bold.reports.common.min.js`, `bold.reports.widgets.min.js`, and `bold.report-viewer.min.js` are mandatory to render the Bold Report Viewer.

Name	Details	CDN link
bold.reports.common.min.js	Common script for reporting widgets.	Secured link: https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js
		Unsecured link: http://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js
bold.reports.widgets.min.js	Supports Syncfusion widgets	Secured link: https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js
		Unsecured

	to render in HTML5 format.	link: http://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js
ej.chart.min.js	Renders the chart item. Add this script only if your report contains the chart report item.	Secured link: https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js Unsecured link: http://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js
ej2-base.min.js	Renders the gauge item. Add this script only if your report contains the gauge report item.	Secured Link: https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js Unsecured Link: https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js
ej2-data.min.js	Renders the gauge item. Add this script only if your report contains the gauge	Secured link: https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js Unsecured Link: https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js

	report item.	
ej2-pdf-export.min.js	Renders the gauge item. Add this script only if your report contains the gauge report item.	Secured link: https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js Unsecured Link: https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js
ej2-svg-base.min.js	Renders the gauge item. Add this script only if your report contains the gauge report item.	Secured link: https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js Unsecured Link: https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js
ej2-lineargauge.min.js	Renders the linear gauge item. Add this script only if your report contains the linear gauge	Secured link: https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-lineargauge.min.js Unsecured link: http://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-lineargauge.min.js

	report item.	
ej2-circulargauge.min.js	Renders the circular gauge item. Add this script only if your report contains the circular gauge report item.	Secured link: https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-circulargauge.min.js Unsecured link: http://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-circulargauge.min.js
ej2-maps.min.js	Renders the map item. Add this script only if your report contains the map report item.	Secured link: https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js Unsecured link: http://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js
bold.report-viewer.min.js	Renders the Syncfusion JavaScript Report Viewer widget.	Secured link: https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js Unsecured link: http://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js

External dependency libraries

The basic syntax is,

<https://cdn.boldreports.com/external/fileName>

Example: <https://cdn.boldreports.com/external/jquery-1.10.2.min.js>

Name	Details	CDN link
jQuery 1.10.2	Common jQuery script to render the Syncfusion JavaScript Reporting widgets	Secured link: https://cdn.boldreports.com/external/jquery-1.10.2.min.js Unsecured link: http://cdn.boldreports.com/external/jquery-1.10.2.min.js

CDN style sheet links

The CDN links for all the CSS files are provided in the following table. Refer to the following syntax:

[https://cdn.boldreports.com/\[version\]/content/\[theme-name\]/\[fileName\]](https://cdn.boldreports.com/[version]/content/[theme-name]/[fileName])

Name	Details	CDN link
Material (default theme)	Includes the CSS properties defined for the JavaScript Reporting component in material. (Default-theme)	Secured link: https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css Unsecured link: http://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css
Office-365	Includes the CSS properties defined for the JavaScript Reporting component in office-365 theme.	Secured Link: https://cdn.boldreports.com/5.4.20/content/office-365/bold.reports.all.min.css Unsecured Link: http://cdn.boldreports.com/5.4.20/content/office-365/bold.reports.all.min.css
High-contrast-01	Includes the CSS properties defined for the JavaScript Reporting component	Secured Link: https://cdn.boldreports.com/5.4.20/content/high-contrast-01/bold.reports.all.min.css Unsecured Link: http://cdn.boldreports.com/5.4.20/content/high-contrast-01/bold.reports.all.min.css

	t in high-contrast-01 theme.	
High-contrast-02	Includes the CSS properties defined for the JavaScript Reporting component in high-contrast-02 theme.	Secured Link: https://cdn.boldreports.com/5.4.20/content/high-contrast-02/bold.reports.all.min.css Unsecured Link: http://cdn.boldreports.com/5.4.20/content/high-contrast-02/bold.reports.all.min.css
Bootstrap-theme	Includes the CSS properties defined for the JavaScript Reporting component in bootstrap theme.	Secured link: https://cdn.boldreports.com/5.4.20/content/bootstrap-theme/bold.reports.all.min.css Unsecured link: http://cdn.boldreports.com/5.4.20/content/bootstrap-theme/bold.reports.all.min.css

All the provided CDN links can be accessed either through `http` or `https`.

Refer local Scripts and CSS when CDN fails

One of the major risks with CDN links is, sometimes, it may go down due to the network or connection problems. On such scenarios, you can refer the local scripts and CSS files dynamically in application by checking if the scripts and CSS files are loaded through CDN that returns `undefined` as demonstrated in the following code sample.

```
`html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>My first HTML page</title>
// CDN LINK references
<link
href="http://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="http://cdn.boldreports.com/external/jquery-1.10.2.min.js" type="text/javascript"></script>
```

```

<script
src="http://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script src="http://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<script src="http://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
<script type="text/javascript">
if (typeof jQuery == "undefined") { // If CDN fails, jQuery returns undefined
// Referring local scripts - Specify the path where the below files are located in your machine
document.write(decodeURIComponent('%3Cscript src="Scripts/jquery-1.10.2.min.js"
%3E%3C/script%3E'));
}
if (typeof ej == "undefined") { // If CDN fails, ej returns undefined.
// Refer the Syncfusion stylesheets and scripts from the local path here
// StyleSheet reference from the local system path
document.write(decodeURIComponent('%3Clink rel="stylesheet" href="Content/bold-
reports/material/bold.reports.all.min.css" %3C/%3E'));
document.write(decodeURIComponent('%3Cscript src="Scripts/bold-
reports/common/bold.reports.common.min.js" %3E%3C/script%3E'));
document.write(decodeURIComponent('%3Cscript src="Scripts/bold-
reports/common/bold.reports.widgets.min.js" %3E%3C/script%3E'));
// Script reference from the local system path
document.write(decodeURIComponent('%3Cscript src="Scripts/bold-reports/bold.report-viewer.min.js"
%3E%3C/script%3E'));
}
</script>
</head>
<body>
<script type="text/javascript">
$(function () {
// initialization boldReportViewer
$("#viewer").boldReportViewer();
});
</script>
</body>
</html>

```

JavaScript Report Viewer Reporting Service

The JavaScript Report Viewer requires a Web API service to process the report files. The following topics explains how to create reporting Web API service to preview the reports.

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

IReportController

The interface **IReportController** has the declaration of action methods that is defined in Web API Controller for processing the RDL, RDLC, SSRS report and handling request from Report Viewer control. The IReportController has the following action methods declaration.

Methods | Description

GetResource | Returns the report resource for the requested key.

ProcessReport | Processes the report request and returns the result.

`csharp

```
public class ReportsController: ApiController, IReportController
{
    /// <summary>
    /// Action (HttpGet) method for getting resource for report.
    /// </summary>
    /// <param name="key">The unique key to get the required resource.</param>
    /// <param name="resourceType">The type of the requested resource.</param>
    /// <param name="isPrinting">If set to <see langword="true"/>, then the resource is generated for
    printing.</param>
    /// <returns>The object data.</returns>
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        //Returns the report resource for the requested key.
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }
    /// <summary>
    /// Report Initialization method that is triggered when report begin processed.
    /// </summary>
    /// <param name="reportOptions">The ReportViewer options.</param>
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOptions)
```

```

{
//You can update report options here
}
/// <summary>
/// Report loaded method that is triggered when report and sub report begins to be loaded.
/// </summary>
/// <param name="reportOptions">The ReportViewer options.</param>
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOptions)
{
//You can update report options here
}
/// <summary>
/// Action (HttpPost) method for posting the request for report process.
/// </summary>
/// <param name="jsonData">The JSON data posted for processing report.</param>
/// <returns>The object data.</returns>
public object PostReportAction(Dictionary < string, object > jsonData)
{
//Processes the report request and returns the result.
return ReportHelper.ProcessReport(jsonData, this);
}
}
,

```

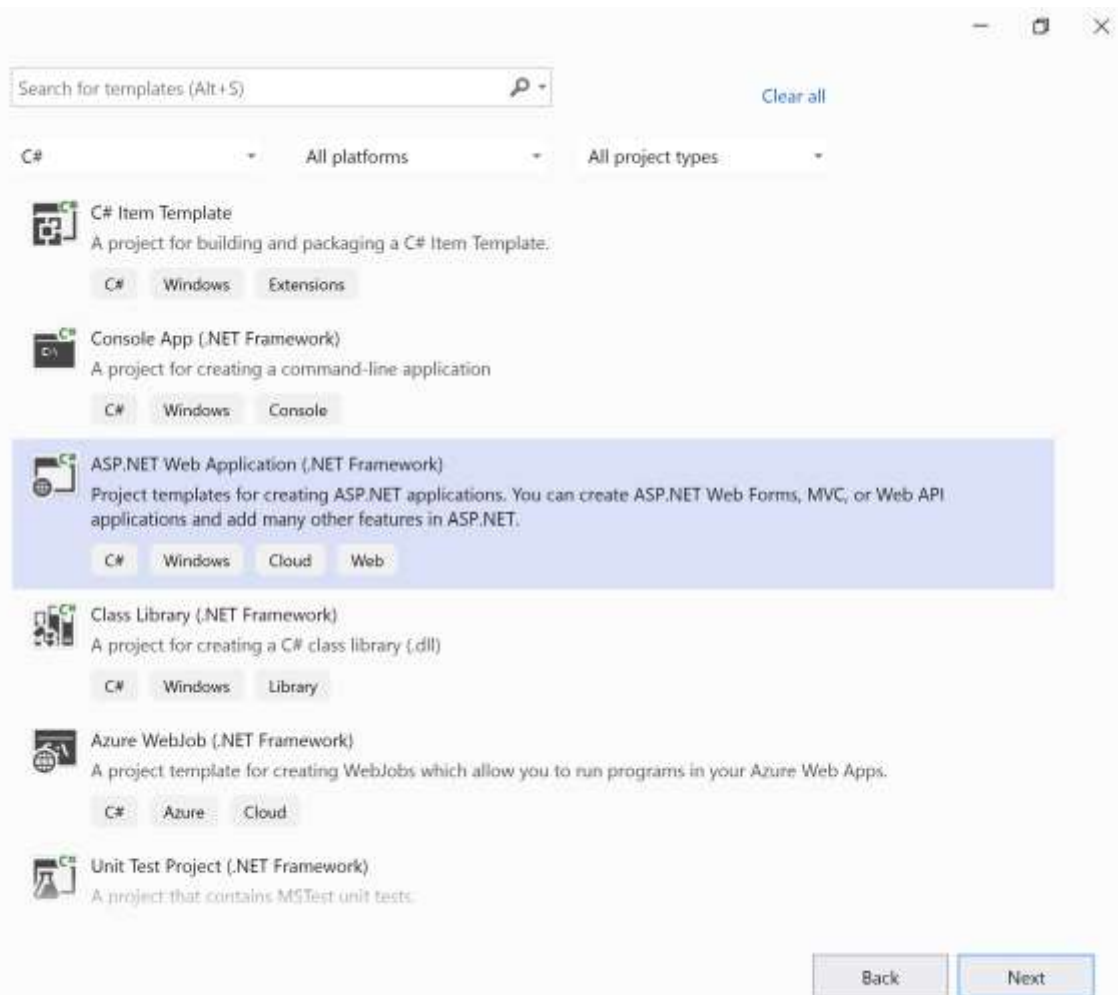
Create ASP.NET Web API service

In this section, you will learn how to create a Web API Service for Report Viewer using the new ASP.NET Empty Web Application template.

To get start quickly with Web API Service for Report Viewer, you can check on this video:

youtube: https://youtu.be/Qbho_8cnhJ8

1. Start Visual Studio 2022 and click **Create new project**.
2. Select **ASP.NET Web Application (.NET Framework)**.



3. Change the application name, and then select the required **.NET Framework** in the drop-down.

Configure your new project

ASP.NET Web Application (.NET Framework) **CR** Windows Cloud Web

Project name
ASP.NET Web API service

Location
C:\

Solution name
ASP.NET Web API service

☐ Place solution and project in the same directory

Framework
.NET Framework 4.5.1

Back Create

4. Choose **Empty, Web API** and then click **OK**. Now, the Web application project is created with default ASP.NET Web template.

Create a new ASP.NET Web Application

Empty
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Web Forms
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

MVC
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Web API
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Single Page Application
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
None

Add folders & core references

- ☐ Web Forms
- ☐ MVC
- ☒ Web API

Advanced

- ☒ Configure for HTTPS
- ☐ Docker support
(Requires [Docker Desktop](#))
- ☐ Also create a project for unit tests

ASP.NET Web API service Tests

Back Create

Configure Report Viewer Web API

1. Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

2. Search for **BoldReports.Web** NuGet packages, and install them in your Web application.

Package | Purpose

PostReportAction | Action (HttpPost) method for posting the request in report process.

OnInitReportOptions | Report initialization method that occurs when the report is about to be processed.

OnReportLoaded | Report loaded method that occurs when the report and sub report start loading.

GetResource | Action (HttpGet) method to get resource for the report.

ReportHelper

The class **ReportHelper** contains helper methods that help to process a Post or Get request from the Report Viewer control and return the response to the Report Viewer control. It has the following methods:

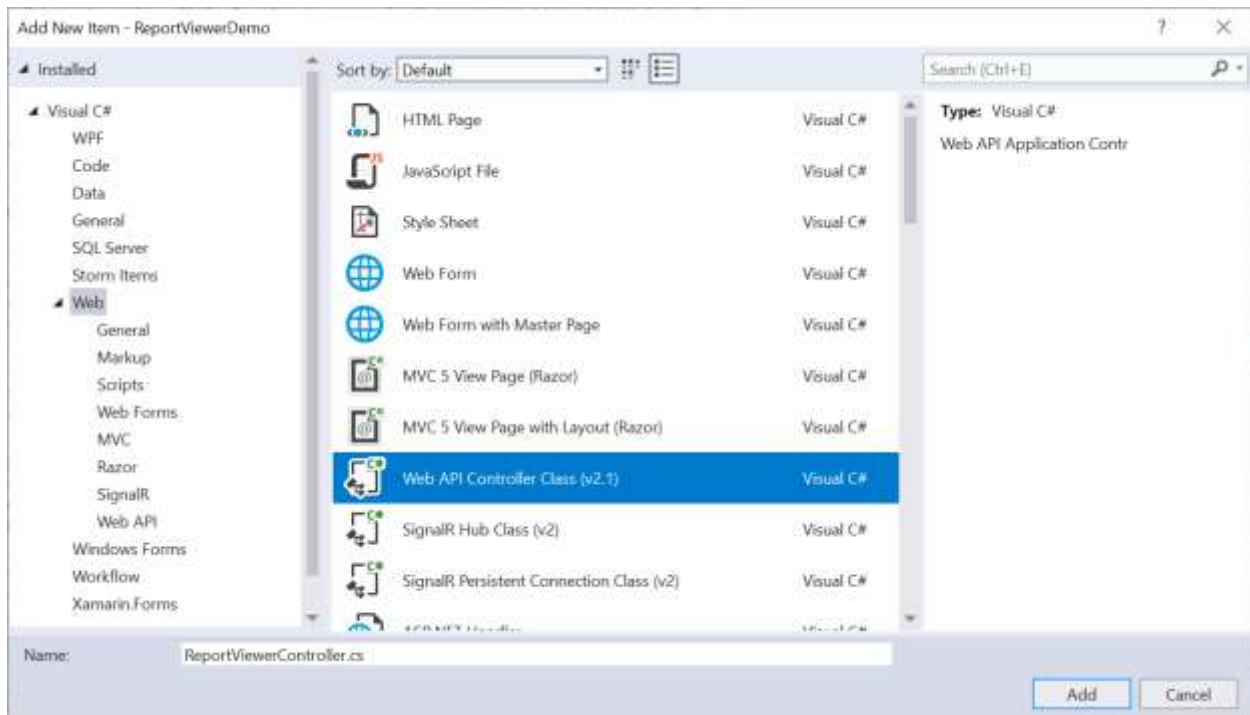
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click **Controller** folder in your project and select **Add > New Item** from the context menu.
2. Select Web API Controller Class from the listed templates and name it as **ReportViewerController.cs**



3. Click **Add**.

While adding Web API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and implement its methods (replace the following code in newly created Web API controller).

```
`csharp
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
}
```

```
[System.Web.Http.ActionName("GetResource")]
[AcceptVerbs("GET")]
public object GetResource(string key, string resourcetype, bool isPrint)
{
    return ReportHelper.GetResource(key, resourcetype, isPrint);
}
// Method that will be called when initialize the report options before start processing the report
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    // You can update report options here
}
// Method that will be called when reported is loaded
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    // You can update report options here
}
}
```

Add routing information

1. To configure routing to include an action name in the URI, open the **WebApiConfig.cs** file and change the **routeTemplate** in the **Register** method as follows,

```
`csharp
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services
        // Web API routes
        config.MapHttpAttributeRoutes();
        config.Routes.MapHttpRoute(
```

```

name: "DefaultApi",
routeTemplate: "api/{controller}/{action}/{id}",
defaults: new { id = RouteParameter.Optional }
);
}
}
,

```

2. Compile and run the Web API service application.

Enable Cross-Origin Requests

Browser security prevents Report Viewer from making requests to your Web API Service when both runs in a different domain. To allow access to your Web API service from a different domain, you must enable cross-origin requests.

1. Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, go to **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for `Microsoft.AspNet.WebApi.Cors` NuGet packages, and install them in your Web API application.
3. Call `EnableCors` in `WebApiConfig` to add CORS services to the **Register** method. Replace the following code to allow any origin requests.

```

`csharp
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Add Enable Cors
        config.EnableCors();

        // Web API configuration and services

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{action}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}

```

```

}
}
`

```

4. To specify the CORS policy for API controller, add the `[EnableCors]` attribute to the controller class. Specify the policy name.

```

`csharp
[EnableCors(origins: "", headers: "", methods: "*")]
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
    [System.Web.Http.ActionName("GetResource")]
    [AcceptVerbs("GET")]
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }
    // Method that will be called when initialize the report options before start processing the report
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
    // Method that will be called when reported is loaded
    [NonAction]
    public void OnReportLoaded(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
}

```

```
}  
,
```

Create ASP.NET Core Web API Service

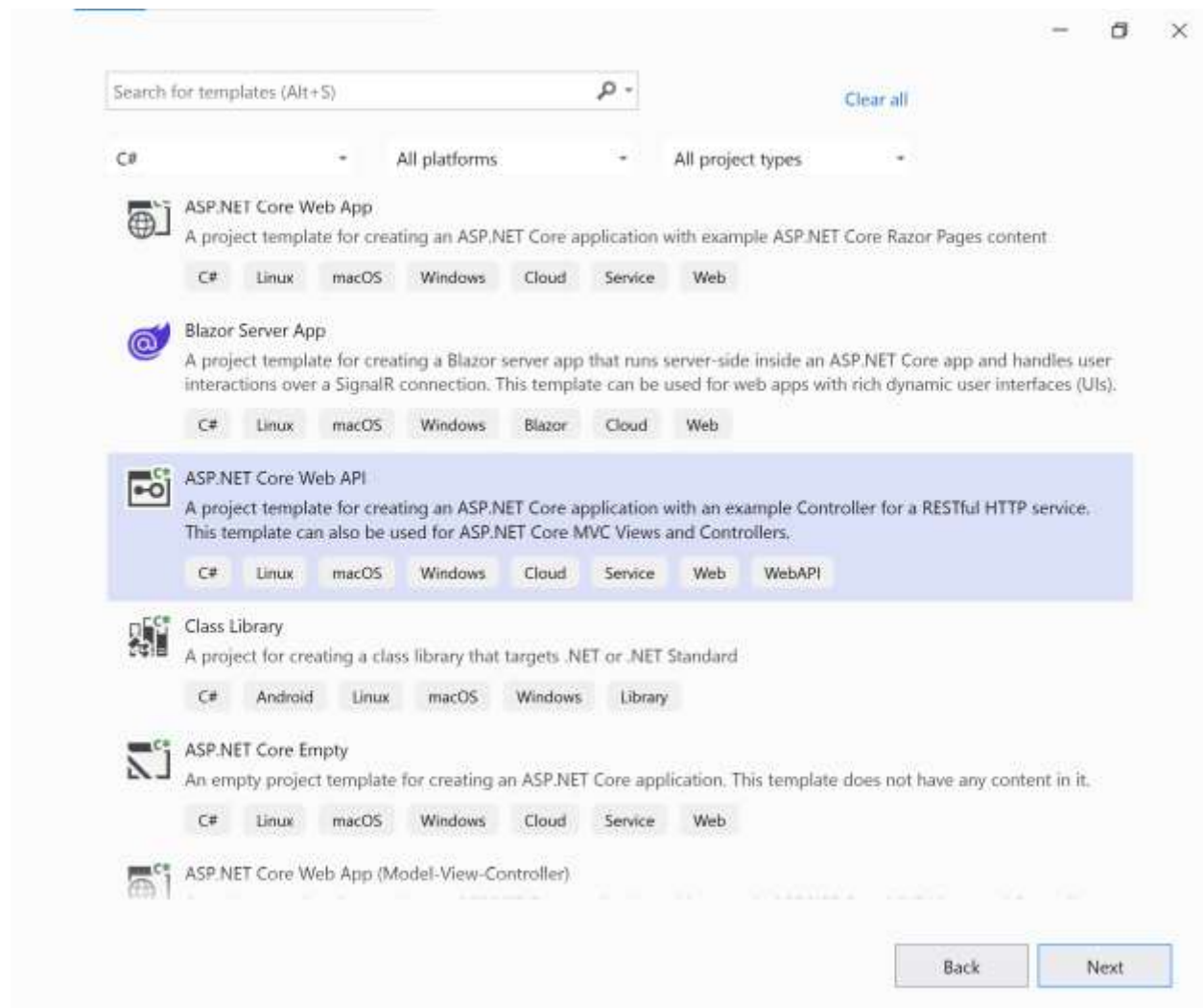
To create an ASP.NET Core Web API for Report Viewer using a new ASP.NET Core Web Application template, follow these steps:

Bold Reports ASP.NET Core supports from **.NET Core 2.1** only. So, choose the .NET Core version **ASP.NET Core 2.1** or higher versions for Viewer API creation.

To get start quickly with ASP.NET Core Web API for Report Viewer, you can check on this video:

youtube: <https://youtu.be/b1yZGAeWoHQ>

1. Start Visual Studio 2022 and click **Create new project**.
2. Choose **ASP.NET Core Web API**, and then click **Next**.



3. Change the project name, and then click **Next**.

4. In the dropdown for the **ASP.NET Core version**, choose **ASP.NET Core 6.0**, then click **Create**.

The screenshot shows the 'Additional information' page for creating an ASP.NET Core Web API service. At the top, there are tabs for 'ASP.NET Core Web API', 'CR', 'Linux', 'macOS', 'Windows', 'Cloud', 'Service', 'Web', and 'WebAPI'. Below the tabs, the 'Framework' dropdown is set to '.NET 6.0 (Long Term Support)'. The 'Authentication type' is set to 'None'. There are checkboxes for 'Configure for HTTPS' (checked), 'Enable Docker' (unchecked), and 'Docker OS' (set to 'Linux'). At the bottom, there are checkboxes for 'Use controllers (unchecked to use minimal APIs)' (checked), 'Enable OpenAPI support' (checked), and 'Do not use top-level statements' (unchecked). At the bottom right, there are 'Back' and 'Create' buttons.

If you need to use Bold Reports with ASP.NET Core on Linux or macOS, then refer to this [Can Bold Reports be used with ASP.NET Core on Linux and macOS](#) section.

List of dependency Libraries

The Web API service configuration requires the following reporting server-side packages. Right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages** and then search for **BoldReports.Net.Core** package, and install to the application. The following provides detail of the packages and its usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Supports for exporting the report to PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the packages **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core** and **Syncfusion.XlsIO.Net.Core**.

Syncfusion.Pdf.Net.Core | Supports for exporting the report to a PDF.

Syncfusion.DocIO.Net.Core | Supports for exporting the report to a Word.

Syncfusion.XlsIO.Net.Core | Supports for exporting the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serialize and deserialize the data for report viewer. It is a mandatory package for the report viewer, and the package version should be higher of 10.0.1 for NET Core 2.0 and others should be higher of 9.0.1.

System.Data.SqlClient | This is an optional package for the report viewer. It should be referred in project when renders the RDL report and which contains the SQL Server and SQL Azure data source. Also, the package version should be higher of 4.1.0.

Configure Web API

The interface **IReportController** has declaration of action methods that are defined in the Web API Controller for processing the RDL, RDLC, and SSRS reports and for handling request from the Report Viewer control. The IReportController has the following action methods declaration:

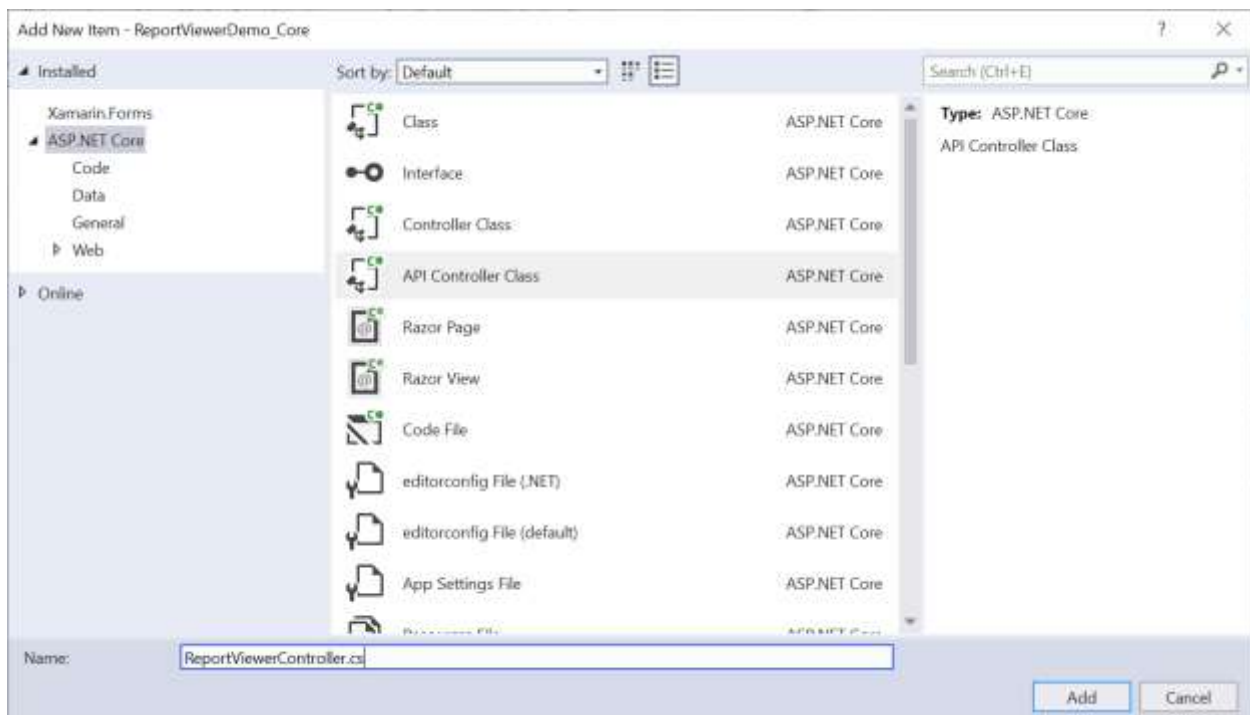
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller** class and name it as **ReportViewerController.cs**



3. Click **Add**.

While adding API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the `IReportController` interface, and then implement its methods.
6. Create local references for the interfaces given in following table.

Interface | Purpose

`IMemoryCache` | Report Viewer requires a memory cache to store the information of consecutive client request and have the rendered report viewer information in server.

`IWebHostEnvironment` | `IWebHostEnvironment` used to get the report stream from application `wwwroot\Resources` folder.

7. Next, add the `[EnableCors]` attribute to the `ReportViewerController` class and specify the policy name which given in `Program.cs`.

```
`csharp
[Route("api/[controller]/[action]")]
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
}
`
```

8. You cannot load the application report with path information in ASP.NET Core service. So, you should load the report as stream in `OnInitReportOptions`.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    System.IO.FileStream reportStream = new System.IO.FileStream(basePath + @"\Resources\sales-order-
    detail.rdl", System.IO.FileMode.Open, System.IO.FileAccess.Read);
    reportOption.ReportModel.Stream = reportStream;
}
`
```

9. You can replace the template code with the following code.

```
`csharp
```



```
[Route("api/[controller]/[action]")]
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered report viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
    public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
    {
        return ReportHelper.ProcessReport(jsonArray, this, this._cache);
    }
    // Method will be called to initialize the report information to load the report with ReportHelper for
    // processing.
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sales-order-detail.rdl report from application the folder
        // wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
        System.IO.FileStream reportStream = new System.IO.FileStream(basePath + @"\Resources\sales-order-
        detail.rdl", System.IO.FileMode.Open, System.IO.FileAccess.Read);
        reportOption.ReportModel.Stream = reportStream;
    }
}
```

```

}
// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}
//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
return ReportHelper.GetResource(resource, this, _cache);
}
[HttpPost]
public object PostFormReportAction()
{
return ReportHelper.ProcessReport(null, this, _cache);
}
}
`

```

The `sales-order-detail.rdl` report can be downloaded from [here](#). Also, you can add the report from Bold Reports installation location. For more information on installed sample location, see [Samples and demos](#).

10. Run the application and use the API URL in the Report Viewer `reportServiceUrl` property.

Enable Cross-Origin requests

Browser security prevents Report Viewer from making requests to your Web API Service when both runs in a different domain. To allow access to your Web API service from a different domain, you must enable cross-origin requests.

Call `AddCors` in `Program.cs` to add the CORS services to the app's service container. Replace the following code to allow any origin requests.

```

`csharp
builder.Services.AddCors(o => o.AddPolicy("AllowAllOrigins", builder =>

```

```
{
builder.AllowAnyOrigin()
.AllowAnyMethod()
.AllowAnyHeader();
});
.....
.....
app.UseHttpsRedirection();
app.UseCors();
app.UseAuthorization();
`
```

For more information about CORS, see [Configure CORS for API](#)

How to queries for Bold Reports ReportViewer

This section helps to get the answer for the frequently asked how to queries in Bold Reports JavaScript ReportViewer.

- [Create a RDL report](#)
- [Create a RDL report](#)
- [Render data visualization items](#)
- [Use Report Viewer in Blazor Web Assembly \(ASP.NET Core Hosted\) application](#)
- [Use Report Viewer in Blazor Server application](#)
- [Change the exporting document file name based on parameter](#)
- [Change the connection string datasource dynamically](#)
- [Disable the vertical scrollbar in parameter panel](#)
- [How to customize the boolean parameter UI with parameter pane?](#)
- [How to add the Null parameter with DatePicker for DateTime parameter?](#)
- [How to group the values in parameter drop down?](#)
- [How to set Min and Max value for DateTime parameter?](#)
- [How to pass custom authentication in headers while exporting a report?](#)
- [How to pass multiple values using custom data?](#)
- [How to clear cache when closing the Report Viewer?](#)

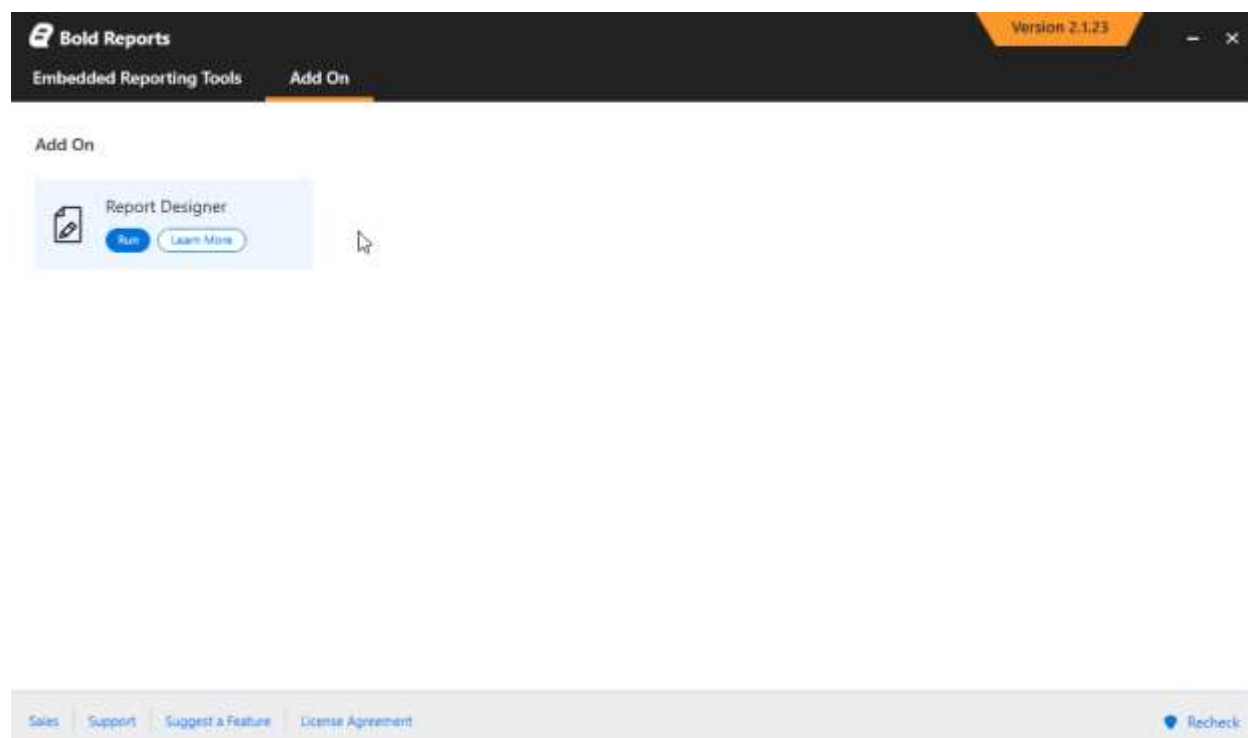
Create a SSRS RDL report

You can create an RDL report using any of the following reporting tools:

- Bold Reports Web Report Designer.
- Microsoft Report Builder.
- Visual Studio Report Server project template.

Bold Reports Report Designer

Bold Reports Report Designer provides the intuitive user interface to create and edit the RDL reports, which is available in Bold Embedded Reporting Tools Control Panel Add On.



Microsoft SQL Report Builder

You can create an RDL report using the Microsoft stand-alone Report Builder. For more details, refer to this [online documentation](#).

Visual Studio Report Server template

To create an RDL report in Visual Studio, a Report Server project is required where you can save your report definition (.rdl) file. For more details, refer to this [Visual Studio documentation](#).

If you do not have the Business Intelligence or Report Server Project options, you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create a RDLC report using business object data source

This section describes step by step procedure to create an RDLC report using Visual Studio Reporting project type.

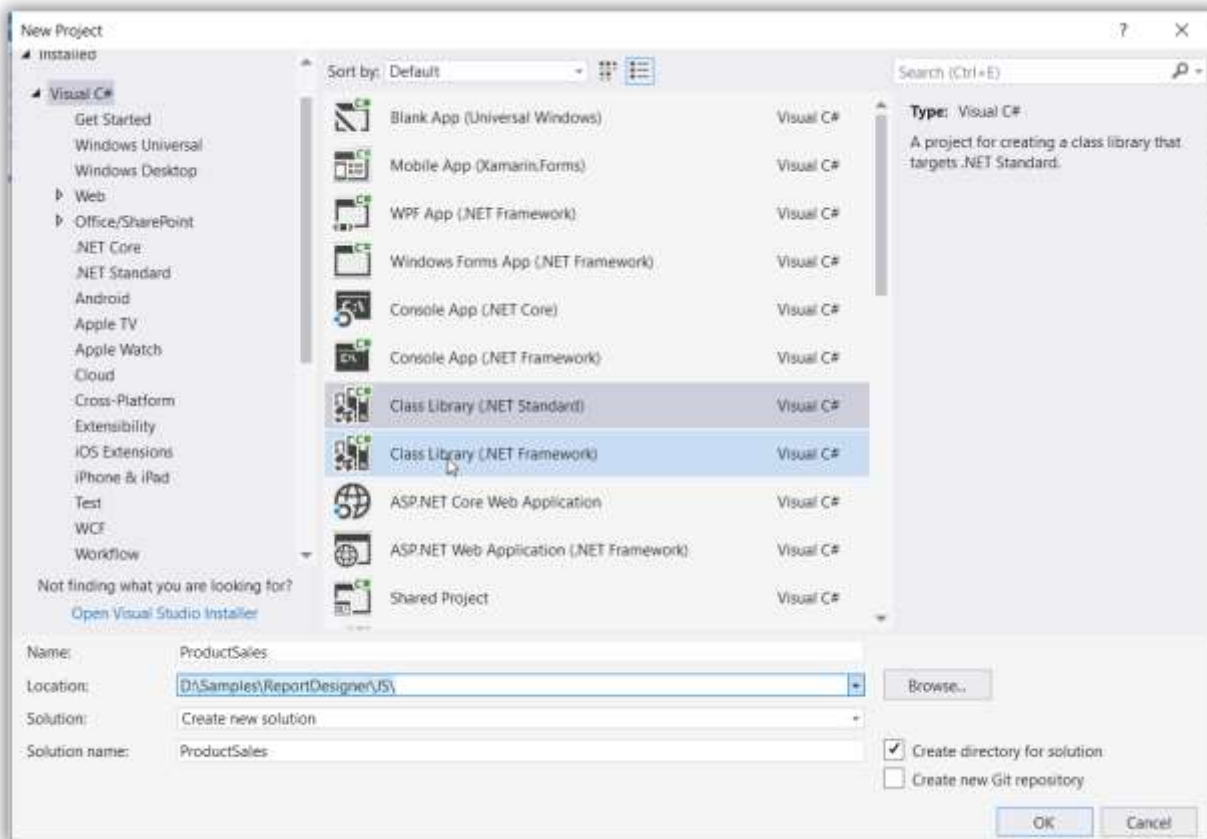
Prerequisites

- Microsoft Visual Studio 2017 or higher
- [Microsoft RDLC Report Designer](#)

If you are using Microsoft Visual Studio lower to 2017 version then you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create business object class

1. Open Visual Studio from the File menu and select **New Project**.
2. Create project with class library type from the project type list.



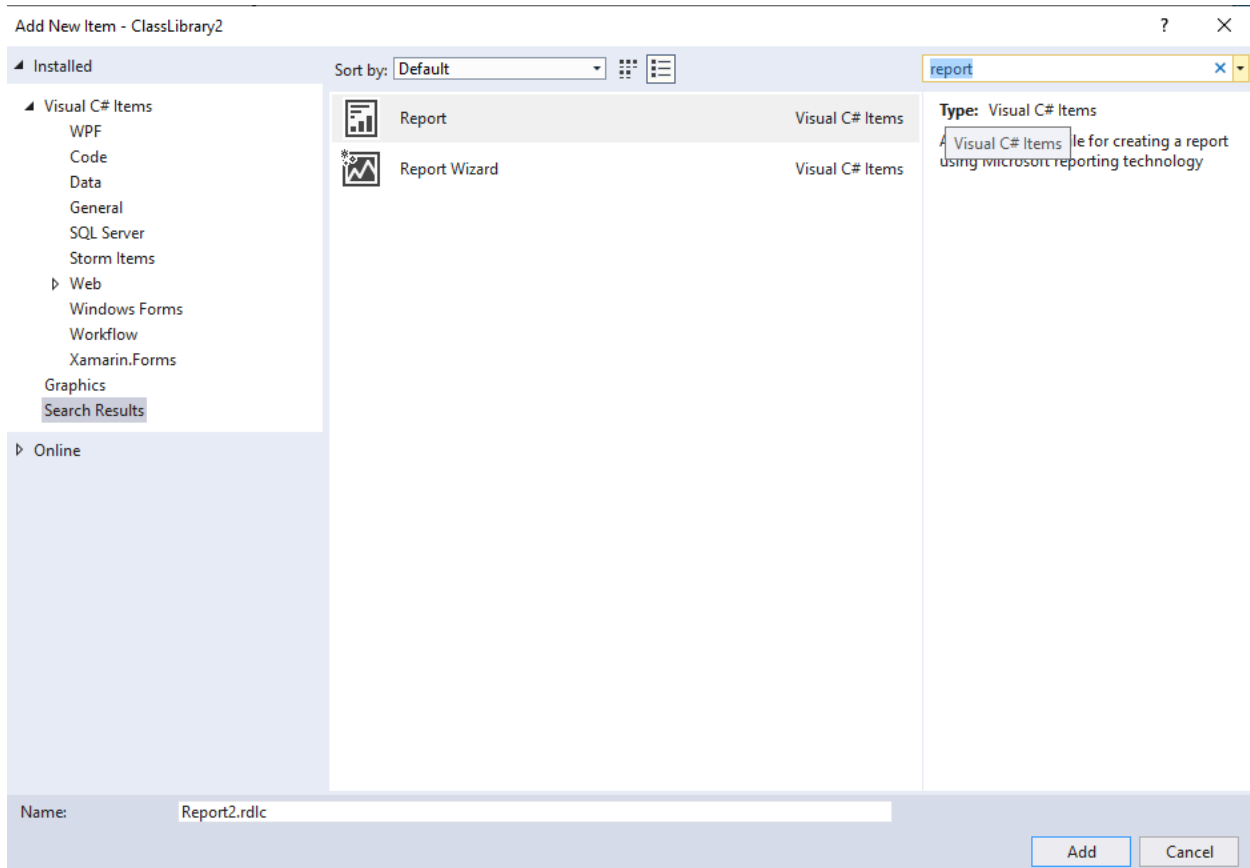
3. Create the class with necessary properties. You can find the reference below,

```
`csharp
public class ProductSales
{
    public string ProdCat { get; set; }
    public string SubCat { get; set; }
    public string OrderYear { get; set; }
    public string OrderQtr { get; set; }
    public double Sales { get; set; }
}
`
```

4. Clean and build the application.

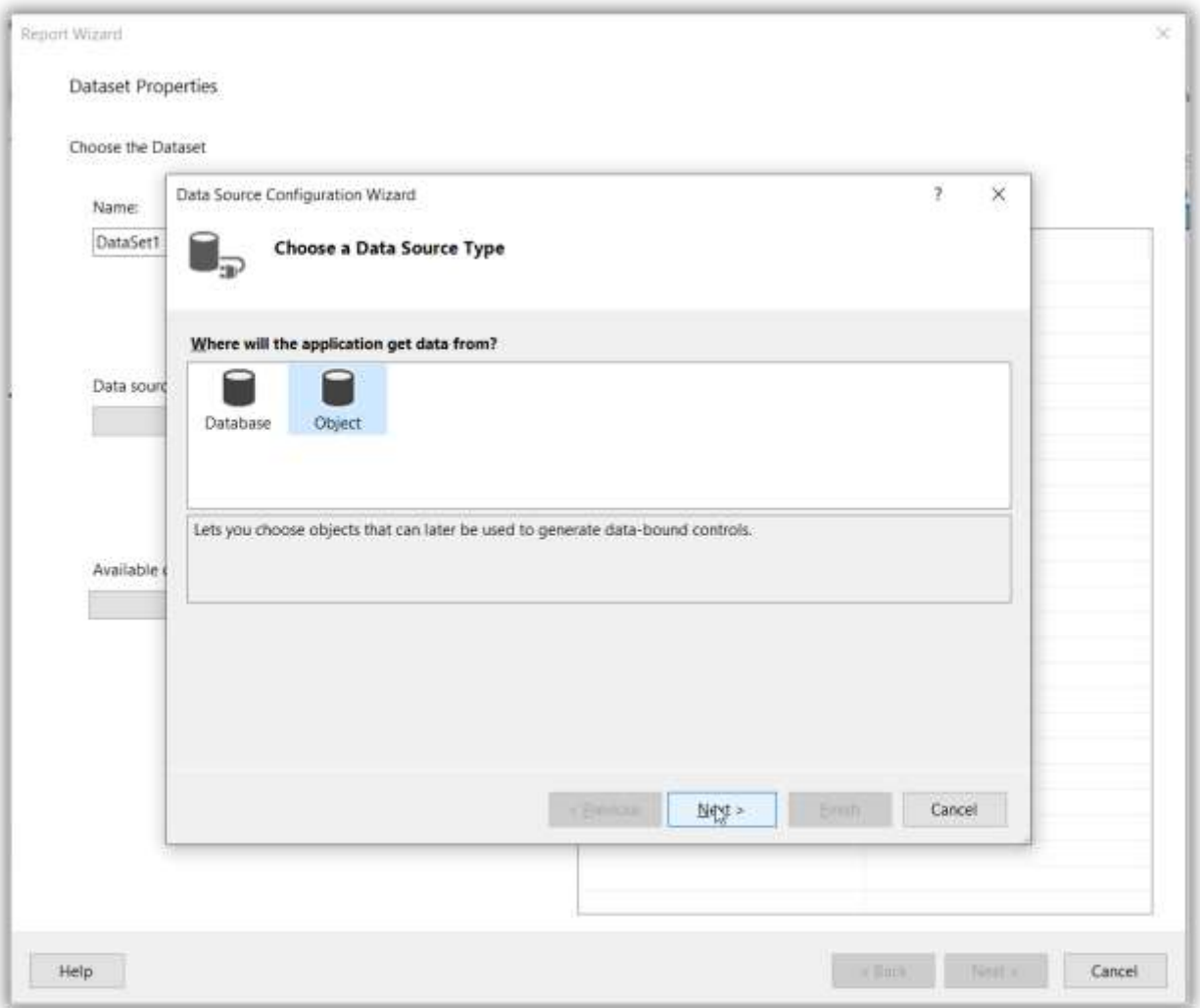
Add an RDLC report

1. Right-click the project and click **Add > New Item**.
2. Search Report with new item and select **Report Wizard** to start the report creation with dataset selection.
3. Click **Add**.

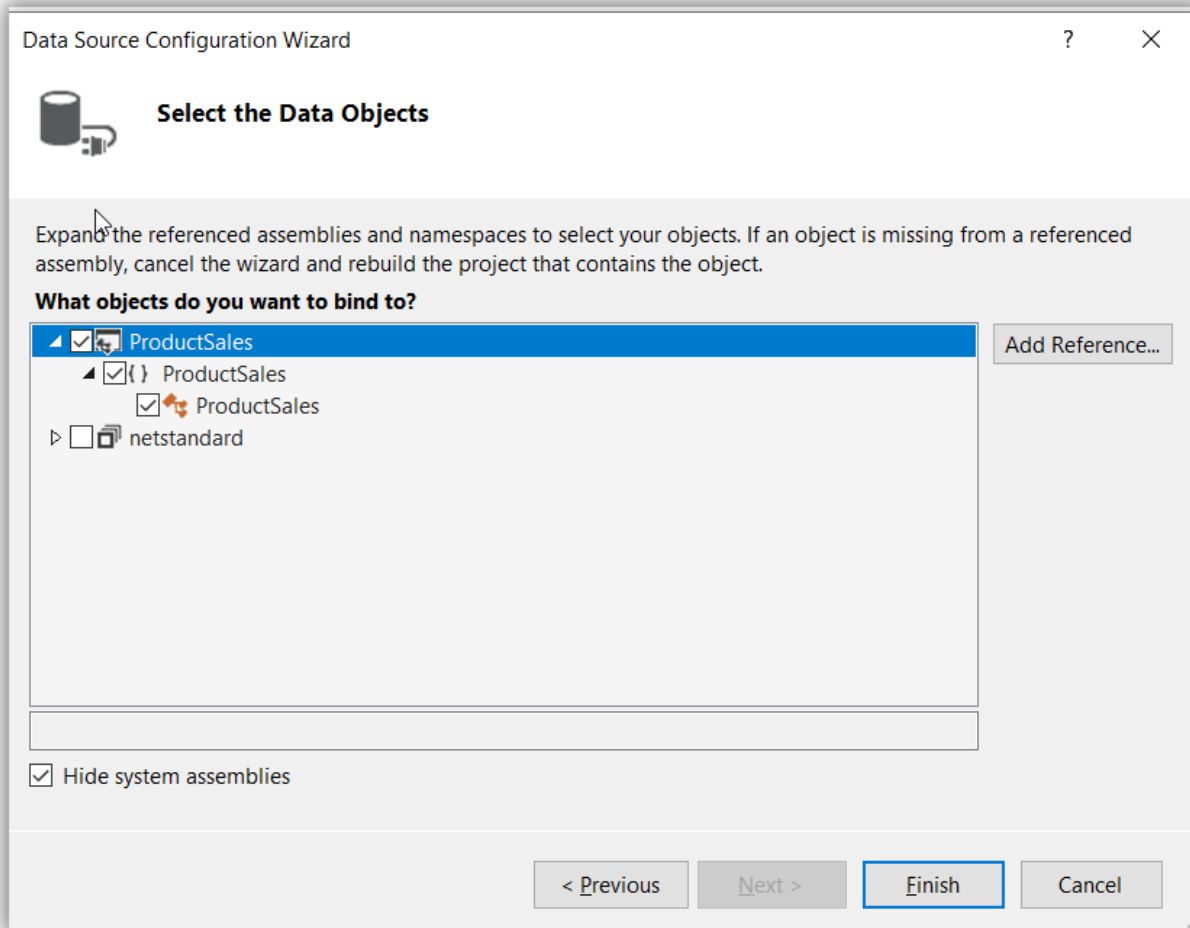


Data source and table configuration wizard

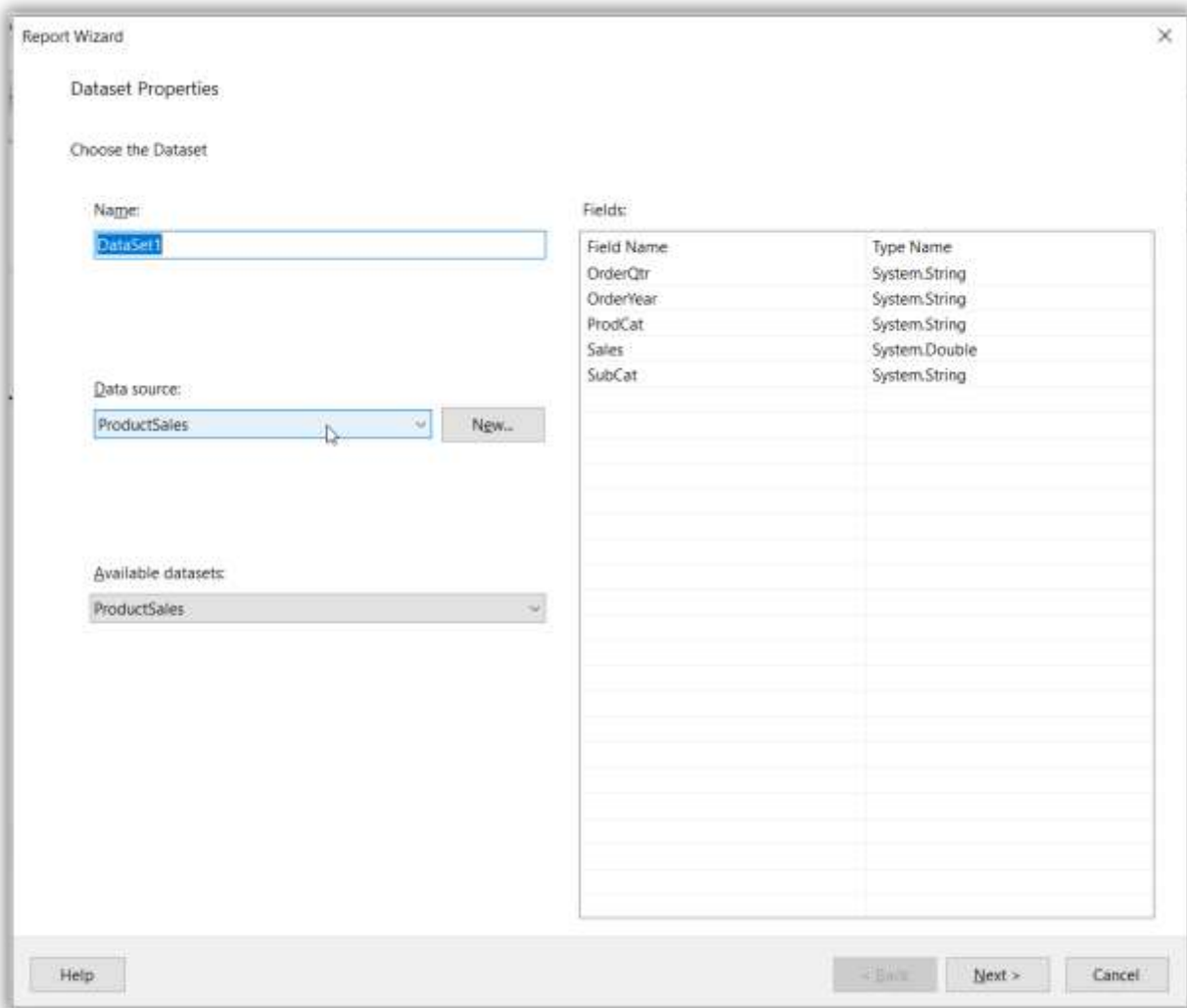
1. Choose object type from the Data Source Configuration wizard and click **Next**.



2. Expand the tree view and select **ProductSales**, and then click **Finish**.



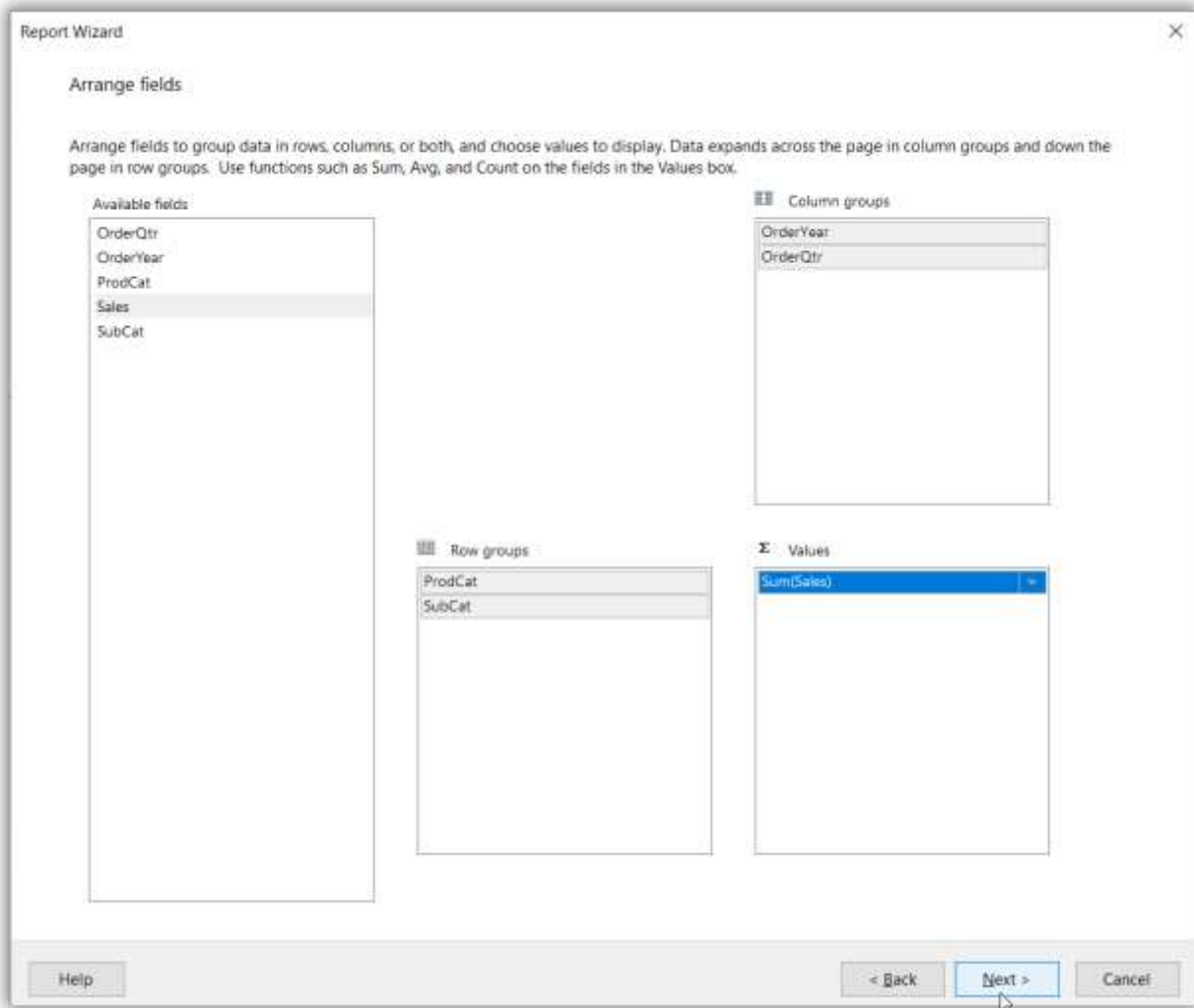
3. In the DataSet Properties wizard, specify the dataset name as `SalesData`.



The image shows the 'Report Wizard' dialog box, specifically the 'Dataset Properties' tab. The 'Choose the Dataset' section contains a 'Name:' label followed by a text box containing 'DataSet1'. Below this is a 'Data source:' label followed by a dropdown menu showing 'ProductSales' and a 'New...' button. At the bottom left, there is an 'Available datasets:' label followed by a dropdown menu also showing 'ProductSales'. On the right side, there is a 'Fields:' label above a table. The table has two columns: 'Field Name' and 'Type Name'. It lists five fields: 'OrderQtr' (System.String), 'OrderYear' (System.String), 'ProdCat' (System.String), 'Sales' (System.Double), and 'SubCat' (System.String). At the bottom of the dialog, there are three buttons: 'Help', '< Back', and 'Next >', and a 'Cancel' button on the far right.

Field Name	Type Name
OrderQtr	System.String
OrderYear	System.String
ProdCat	System.String
Sales	System.Double
SubCat	System.String

4. Drag the fields into Values, Row, and Column groups, and then click **Next**.



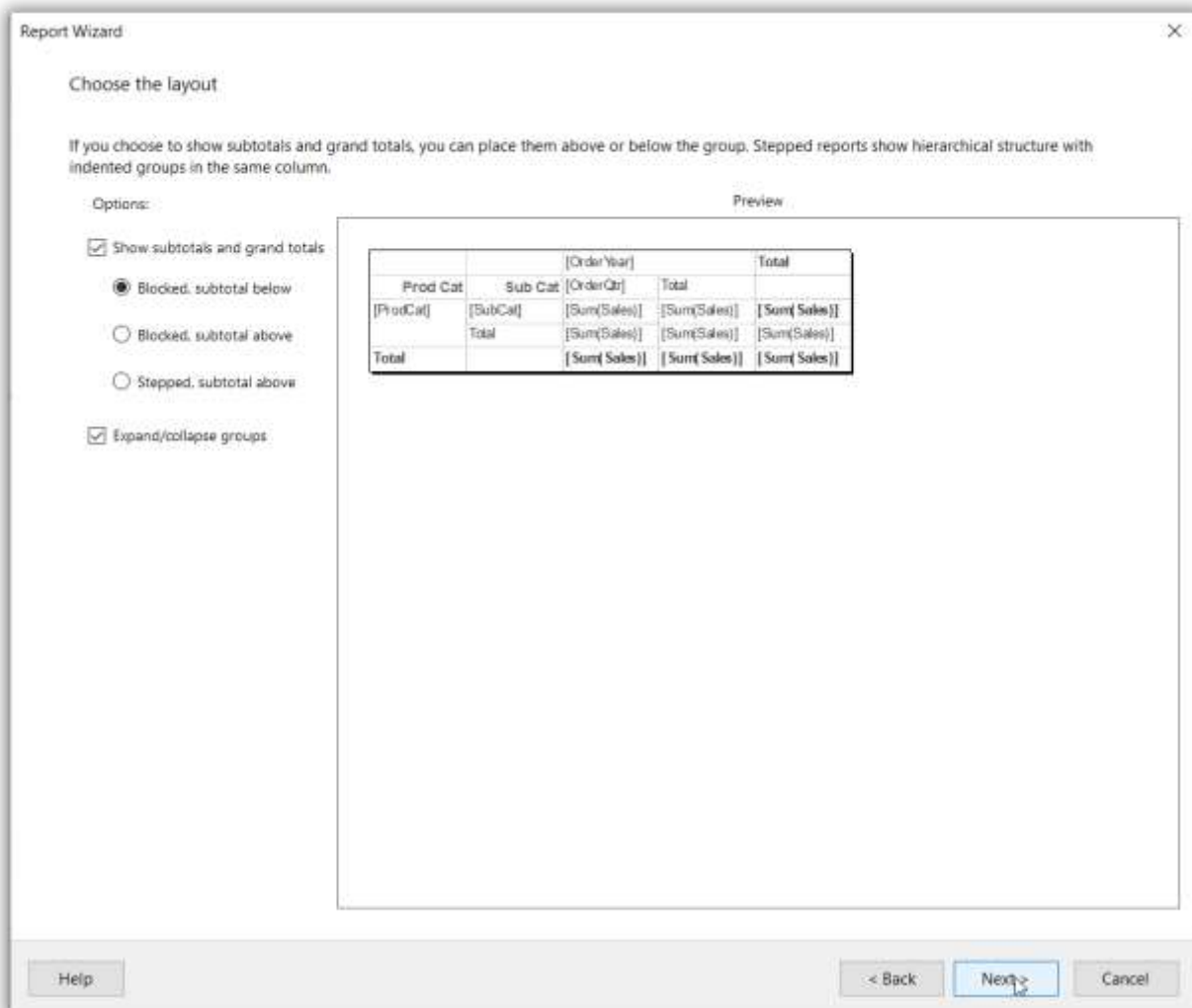
The image shows the 'Report Wizard' dialog box, specifically the 'Arrange fields' step. The dialog has a title bar with 'Report Wizard' and a close button. Below the title bar is the section 'Arrange fields' with a descriptive text: 'Arrange fields to group data in rows, columns, or both, and choose values to display. Data expands across the page in column groups and down the page in row groups. Use functions such as Sum, Avg, and Count on the fields in the Values box.'

The dialog is divided into four main sections:

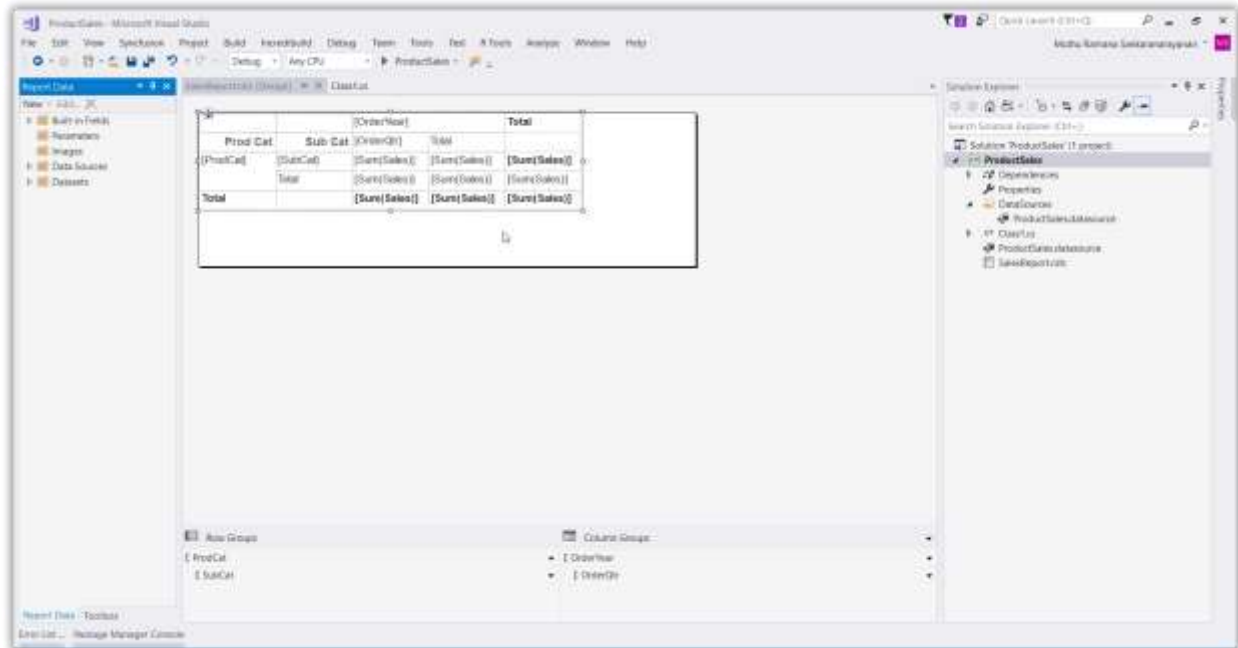
- Available fields:** A list box containing 'OrderQty', 'OrderYear', 'ProdCat', 'Sales', and 'SubCat'. 'Sales' is currently selected.
- Column groups:** A list box containing 'OrderYear' and 'OrderQty'.
- Row groups:** A list box containing 'ProdCat' and 'SubCat'.
- Values:** A list box containing 'Sum(Sales)'.

At the bottom of the dialog, there are four buttons: 'Help', '< Back', 'Next >', and 'Cancel'. The 'Next >' button is highlighted with a mouse cursor.

5. Choose the table layout and click **Next**.
6. Select table style and click **Finish**.



Now, the RDLC report is displayed in the Visual Studio as follows.



To render the RDLC using Report Viewer, refer to the [RDLC Report](#) section.

Adding data visualization scripts

To render the report with data visualization components such as chart, gauge and map items, must add scripts of the visualization element. The following table shows the script reference that need to be added in Report Viewer page for data visualization elements.

Visualization Item | Script File

Gauge | ej2-base.min.js, ej2-data.min.js, ej2-pdf-export.min.js, ej2-svg-base.min.js, ej2-lineargauge.min.js and ej2-circulargauge.min.js

Map | ej2-maps.min.js

Chart | ej.chart.min.js

To render the chart report item add chart control script `ej.chart.min.js` before the `bold.report-viewer.min.js` in Report Viewer page as in following code sample.

```

`html
<link
href="http://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="http://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<!-- Report Viewer component script-->
<script
src="http://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script src="http://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<script src="http://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>

```

How to use JavaScript ReportViewer in `Blazor Web Assembly` App application **Data** source and table configuration wizard

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
```

The following code can be used to render the chart, gauge and map report items in Report Viewer.

`html

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />

<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>

<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>

<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>

<!-- Report Viewer component script-->
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
```

How to use JavaScript ReportViewer in `Blazor Web Assembly` App application

Bold Reports JavaScript ReportViewer can be used with Blazor applications using [ASP.NET Core Blazor JavaScript interoperability](#). Refer to the following steps to use Bold Reports JavaScript ReportViewer with Blazor Web Assembly App template application.

ReportViewer requires a Web API service for report processing. So, you have to create the Blazor Web Assembly App along with ASP.NET Core Hosted configuration to have the ASP.NET Core back end with Blazor application itself.

Refer a scripts and CSS

- Add the following styles and scripts in client project `wwwroot\index.html`.

```
`html
<link href="https://cdn.syncfusion.com/5.4.20/bold-reports/material/bold.reports.all.min.css"
rel="stylesheet" />
<script src="https://cdn.syncfusion.com/js/assets/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
<!-- Report Viewer component script-->
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
`
```

Adding a ReportViewer

- Add the following method in `index.html` to add our [JavaScript ReportViewer](#) with element using jQuery. This will be invoked using [IJSRuntime](#) from razor pages along with name of the report and element need to be created with ReportViewer.

```
`html
<script>
function RenderReportViewer (reportPathInfo, elementID) {
```

```

$("#"+ elementID).boldReportViewer({
reportServiceUrl: '/api/ReportApi',
reportPath: reportPathInfo
});
};
</script>
`

```

- Create div tag and code with razor page where you want to render the ReportViewer.

```

`csharp
@page "/reporting"
@using Microsoft.JSInterop
@using Microsoft.AspNetCore.Components
@inject IJSRuntime JSRuntime
<div id="reportViewerControl" style="width: 100%;height:800px" />
@code {
private string reportPathInput = "Report1";
// You have to call this method based on your need. It can called while loading the page or after
selection the report.
public async void RenderReport()
{
await JSRuntime.InvokeVoidAsync("RenderReportViewer", reportPathInput , "reportViewerControl");
}
/// https://docs.microsoft.com/en-us/aspnet/core/blazor-reporting/components?view=aspnetcore-3.0
/// https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/blazor-reporting/common/samples/3.x/BlazorSample
protected override void OnAfterRender(bool firstRender)
{
// If you want to load in intialization you can make use of this.
//RenderReport();
}
}
`

```

Create a Web API service

- The Report Viewer requires a Web API service to process the report files, and you should create ASP.NET Core Web API Service for server interaction and do the processing in API using Report Helper.

Refer [ASP.NET Core Web API Service](#)

You can ignore Enable Cross origin topic with documentation. since, the service with Blazor application and there is no need to enable cores.

- Also, ensure the following steps while creating the Web API service.
 1. You should route the API properly for interaction with attribute `[Route("api/{controller}/{action}/{id?}")]`.
 2. `[FromBody]` should be used for `PostReportAction` for action method.
 3. `[HttpPost]` and `[HttpGet]` attributes should be used in respective ReportViewer interaction methods.
- You will get compatibility issues with ReportViewer service, which is created from default template. So, following changes required in application StartUp is based on application template type.
 1. You will get the [400 Bad request](#) with service interaction due to the validation of null values with new version of ASP.NET Core. To avoid this issue, you have to remove the `[ApiController]` attribute from Report API service. Otherwise, you have to set `SuppressModelStateInvalidFilter` to `true` in `ConfigureApiBehaviorOptions` to avoid this bad request. API behavior options need to updated in `ConfigureServices` method in `Startup.cs` of server project.

```
`csharp
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
    .ConfigureApiBehaviorOptions(options =>
    {
        options.SuppressModelStateInvalidFilter = true;
    });
    services.AddResponseCompression(opts =>
    {
        opts.MimeTypes = ResponseCompressionDefaults.MimeTypes.Concat(
            new[] { "application/octet-stream" });
    });
}
```


- The following option will be required to avoid the bad request. If you are making the file download application with your Blazor application using `location.rel='nofollow' href`.

```
`csharp
options.SuppressConsumesConstraintForFormFileParameters = true;
options.SuppressInferBindingSourcesForParameters = true;
```

How to use Bold Reports JavaScript ReportViewer with `Blazor Server` App

Bold Reports JavaScript ReportViewer can be used with Blazor applications using [ASP.NET Core Blazor JavaScript interoperability](#). Refer to the following steps to use Bold Reports JavaScript ReportViewer with Blazor Server App template application.

Refer a scripts and CSS

- Add the following styles and scripts `_host.cshtml` available in pages folder.

```
`html
<link href="https://cdn.syncfusion.com/5.4.20/bold-
reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="https://cdn.syncfusion.com/js/assets/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
<!--Used to render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.syncfusion.com/5.4.20/bold-reports/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.syncfusion.com/5.4.20/bold-reports/data-visualization/ej2-
circulargauge.min.js"></script>
<!--Used to render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.syncfusion.com/5.4.20/bold-reports/data-visualization/ej2-
maps.min.js"></script>
<!-- Report Viewer component script-->
<script src="https://cdn.syncfusion.com/5.4.20/bold-
reports/common/bold.reports.common.min.js"></script>
```

```
<script src="https://cdn.syncfusion.com/5.4.20/bold-reports/common/bold.reports.widgets.min.js"></script>
<script src="https://cdn.syncfusion.com/5.4.20/bold-reports/data-visualization/ej.chart.min.js"></script>
<script src="https://cdn.syncfusion.com/5.4.20/bold-reports/bold.report-viewer.min.js"></script>
```

Adding a ReportViewer

- Add the following method in `_host.cshtml` to add our [JavaScript ReportViewer](#) with element using jQuery. This will be invoked using [IJSRuntime](#) from razor pages along with name of the report and element need to be created with ReportViewer.

```
`html
<script>
function RenderReportViewer (reportPathInfo, elementID) {
$("#"+ elementID).boldReportViewer({
reportServiceUrl: '/api/ReportApi',
reportPath: reportPathInfo
});
};
</script>
```

- Create div tag and code with razor page where you want to render the Report Viewer.

```
`csharp
@page "/reporting"
@using Microsoft.JSInterop
@using Microsoft.AspNetCore.Components
@inject IJSRuntime JSRuntime
<div id="reportViewerControl" style="width: 100%;height:800px" />
@code {
private string reportPathInput = "Report1";
// You have to call this method based on your need. It can called while loading the page or after
selection the report.
public async void RenderReport()
{
```

```

await JSRuntime.InvokeVoidAsync("RenderReportViewer", reportPathInput , "reportViewerControl");
}

/// https://docs.microsoft.com/en-us/aspnet/core/blazor-reporting/components?view=aspnetcore-3.0
/// https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/blazor-reporting/common/samples/3.x/BlazorSample
protected override void OnAfterRender(bool firstRender)
{
    // If you want to load in initialization you can make use of this.
    //RenderReport();
}
}
,

```

Create a Web API service

- The ReportViewer requires a Web API service to process the report files, and you should create ASP.NET Core Web API Service for server interaction and do the processing in API using Report Helper.

Refer [ASP.NET Core Web API Service](#)

You can ignore Enable Cross origin topic with documentation. since, the service with Blazor application and there is no need to enable cores.

- Also, ensure the following steps while creating the Web API service.
 1. You should route the API properly for interaction with attribute `[Route("api/{controller}/{action}/{id?}")]`.
 2. `[FromBody]` should be used for `PostReportAction` for action method.
 3. `[HttpPost]` and `[HttpGet]` attributes should be used in respective report viewer interaction methods.
- You will get compatibility issues with ReportViewer service, which is created from default template. So, following changes required in application Startup is based on application template type.
 1. By default, this template will not map controller in your application. So, you have to map controller manually using `MapControllers()` in app end points as follows in `startup.cs`.

```

`csharp
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {

```

```

app.UseDeveloperExceptionPage();
}
else
{
app.UseExceptionHandler("/Error");
// The default HSTS value is 30 days. You may want to change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.
app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseEndpoints(endpoints =>
{
endpoints.MapControllers();
endpoints.MapBlazorHub();
endpoints.MapFallbackToPage("/_Host");
});
}
`

```

How to change the exporting document file name based on parameter

Find the following steps to change the file based on parameter values in Report.

1. Create the file exporting file name using the parameters in **onRenderingComplete** event and store it in local variable.

```

`html
function onRenderingComplete(event) {
var parameters = event.reportParameters;
if(parameters){
for (var i = 0; i < parameters.length; i++) {
if(parameters[i].Name == "Department"){
this.exportFileName = "Sales for " + parameters[i].Value;
}
}
}
}

```

2. Use the file Name property with export, click event to change the file using the value stored in local variable used for having the file using the parameters.

```
`html
function onExportItemClick(event) {
event.fileName = this.exportFileName ;
}
`
```

How to change the data source dynamically

You have to use the `reportOption.ReportModel.DataSourceCredentials` available with the `OnInitReportOptions` method to dynamically change the data source in the web API controller. The following code sample shows how to change the connection string of the `<database>` data source in the report.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
DataSourceCredentials dataSourceCredentials = new DataSourceCredentials();
string connectionString = "Data Source = <instancename>; Initial Catalog = <database>; User ID =
'<username>; Password = '<password>'";
//You have to provide the shared data source name used with the report or the data source name
available with the report.
dataSourceCredentials.Name = "<database>";
dataSourceCredentials.ConnectionString = connectionString;
reportOption.ReportModel.DataSourceCredentials = new List<DataSourceCredentials> {
dataSourceCredentials };
}
`
```

How to disable the vertical scrollbar in parameter panel

To disable the vertical scrollbar in parameter panel, set the `enableparameterblockscroller` property to false.

```
<span style="font-weight:bold">Example</span>
`js
<div id="report viewer"></div>
```

```

<script>
$('#report viewer').boldReportViewer(
{
enableParameterBlockScroller: false
});
</script>
`

```

How to customize the Boolean parameter UI with parameter pane

You have to use `beforeParameterAdd` event for this customization. You can refer the below code snippet for an reference to customize the Boolean parameter type as drop-down instead of selecting the Radio button.

```

`html
<div id="viewer"></div>
<script>
$('#viewer').boldReportViewer({
beforeParameterAdd: "onBeforeParameterAdd"
});
</script>
</html>
`

`html
<script type="text/javascript">
function onBeforeParameterAdd(args) {
var $targetTag = $('#' + args.containerId);
if (args.parameterModel.ParameterElementType == "RadioButton") {
var comboBoxTag = ej.buildTag("input", "", { 'id': args.parameterModel.Name, 'type': 'Text' });
$targetTag.append(comboBoxTag);
var data = [];
if (args.parameterModel.IsNullable) {
data.push({ label: 'Null', value: 'null' })
}
data.push({ label: 'True', value: 'true' });
data.push({ label: 'False', value: 'false' })
}

```

```
comboBoxTag.ejComboBox({
  dataSource: data,
  allowCustom: true,
  fields: { text: 'label', value: 'value' },
  placeholder: "Select a Value",
  select: function (argument) {
    if (argument.itemData) {
      var data = argument;
      var param = {
        name: this._id,
        labels: [],
        values: []
      }
      if (argument.itemData.value == 'null') {
        param.labels.push(null);
        param.values.push(null);
      }
      else if (argument.itemData.value == 'true') {
        param.labels.push(true);
        param.values.push(true);
      }
      else if (argument.itemData.value == 'false') {
        param.labels.push(false);
        param.values.push(false);
      }
      $('#viewer').data('boldReportViewer').updateParameter(param);
    }
  }
});
args.handled = true;
}
}
</script>
```

```
</html>
```

```
,
```

[See Also](#)

- [How to add the Null parameter with DatePicker for DateTime parameter?](#)
- [How to group the values in parameter drop down?](#)

How to add null value within DatePicker for DateTime Parameter

You have to use `beforeParameterAdd` event for this customization. You can refer the following code snippet for a reference to customize the date picker along with null value in drop-down, instead of adding in right side of DatePicker.

```
`html
```

```
<div id="viewer"></div>
```

```
<script>
```

```
$("#viewer").boldReportViewer({
```

```
beforeParameterAdd: "onBeforeParameterAdd"
```

```
});
```

```
</script>
```

```
</html>
```

```
,
```

```
`html
```

```
<script type="text/javascript">
```

```
function onBeforeParameterAdd(args) {
```

```
var $targetTag = $('#' + args.containerId);
```

```
if (args.parameterModel.ParameterElementType == "DateTime") {
```

```
var $dateTime = ej.buildTag("input", "", "", { 'id': args.parameterModel.ControlId, 'type': 'text', 'style':  
'width: 100%' });
```

```
$targetTag.append($dateTime);
```

```
var name = args.parameterModel.Name;
```

```
$dateTime.ejDateTimePicker({
```

```
timePopupWidth: 150,
```

```
value: args.parameterModel._dateTimeValue,
```

```
open: function (args) {
```

```
var picker = this;
```

```
var container = $('#' + this.id + 'popup');
```



```

if ($(container).find('#null-btn').length == 0) {
var btn = ej.buildTag("div.e-dt-button e-btn e-dt-button e-btn e-select e-flate-flat", "NULL", "", { id: "null-
btn", style: "margin-left:4px;margin-right:4px;display:inline" });
btn.click(function (args) {
picker._doneClick();
picker.setModel({ 'value': null, 'watermarkText': 'Null' });
});
$(container).find('.e-button-container').append(btn);
}
},
change: function (args) {
var data = this.getValue();
var updateParam = {
name: name,
labels: [data],
values: [data]
};
$('#viewer').data('boldReportViewer').updateParameter(updateParam);
}
});
args.handled = true;
}
}
</script>
</html>
`

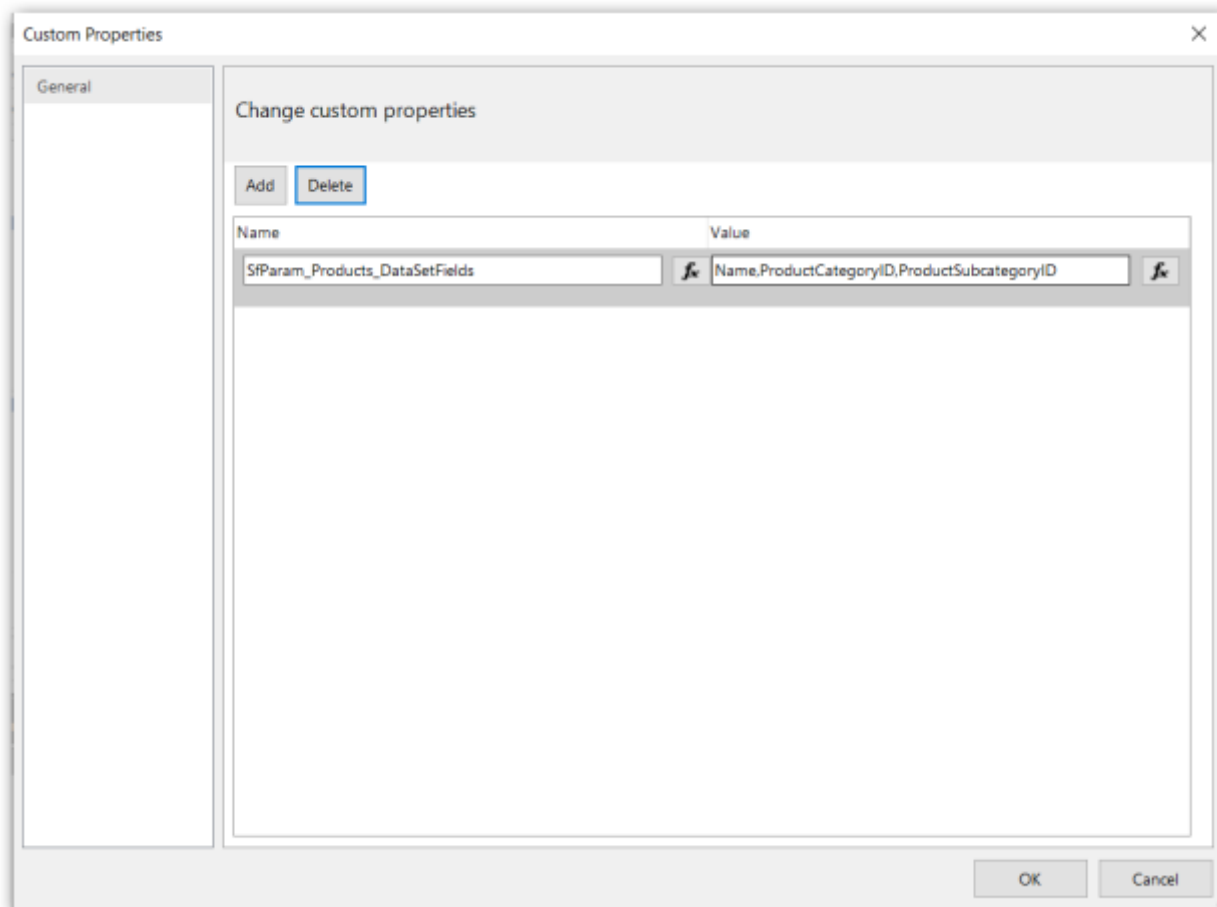
```

See Also

- [How to customize the boolean parameter UI with parameter pane?](#)
- [How to group the values in parameter drop down?](#)

How to group the values in parameter drop-down

You have to use `usebeforeParameterAdd` event for this customization. Additionally, you have to use the report `CustomProperties` to get the values of grouping field.



Set the custom properties in the mentioned format `SfParamProductsDataSetFields`. `Products` as a parameter name.

You can refer the following code snippet.

```
`html
<div id="viewer"></div>
<script>
$("#viewer").boldReportViewer({
beforeParameterAdd: "onBeforeParameterAdd"
});
</script>
</html>
`html
<script type="text/javascript">
var $targetTag = $('#' + args.containerId);
```

```

function onBeforeParameterAdd(args) {
  if (args.customProperties != null && args.parameterModel.Name == "Products") {
    var $targetTag = $('#' + args.containerId);
    if ($("#" + args.parameterModel.ControlId + "popupwrapper").length > 0) {
      $("#" + args.parameterModel.ControlId + "popupwrapper").remove();
    }
    var $paramMultiValuedropdown = ej.buildTag("select.e-rptviewer-multivalue", "", {}, { 'id':
      args.parameterModel.ControlId + '_element', 'type': 'text', 'data-sf-name': args.parameterModel.Name
    });
    $targetTag.append($paramMultiValuedropdown);
    $paramMultiValuedropdown.ejDropDownList({
      cssClass: 'e-js e-reportviewer-param e-rptviewer-drpdwn',
      width: "100%",
      dataSource: args.parameterModel.DataSource,
      change: function (sender) {
        var paramId = (this.id.indexOf("Param") < 0) ? sender.target.id : this.id;
        var targetTag = $('#' + paramId);
        var updateParam = {
          name: "",
          labels: [],
          values: []
        };
        updateParam.name = targetTag.attr('data-sf-name');
        updateParam.labels.push(sender.text);
        updateParam.values.push(sender.text);
        $('#Viewer').data('boldReportViewer').updateParameter(updateParam);
      },
      fields: { text: "Name", category: "ProductCategoryID" }, allowGrouping: true, enableloadOnDemand:
      false
    });
    args.handled = true;
  }
}
</script>

```

```
</html>
```

```
,
```

See Also

- [How to customize the boolean parameter UI with parameter pane?](#)
- [How to add the Null parameter with DatePicker for DateTime parameter?](#)

How to set Maximum and Minimum value for DateTime Parameter

You have to use `parametersetting` property to set the maximum and minimum value for DateTime parameter.

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report viewer"></div>
```

```
<script>
```

```
$("#report viewer").boldReportViewer({
```

```
parameterSettings:
```

```
{
```

```
minDateTime: "3/1/2003",
```

```
maxDateTime: "4/30/9999"
```

```
}
```

```
});
```

```
</script>
```

```
,
```

See Also

[How to use other parameter setting.](#)

How to pass authorization header for exporting request

By default, Report Viewer and Report Designer do not have the option to pass the authorization header for export request. You can find more details about the reason from [How to resolve the image rendering and exporting issue with ASP.NET Core Authentication middleware.](#)

You can add customized export functionality in application with authorization headers using `exportItemClick` event, which will be invoked on exporting the reports from Report Viewer and Report Designer preview. You need to follow the below steps:

1. You need to set the "exportItemClick" on Bold Report Viewer initialization as shown in the following code sample.

```
`javascript
```

```

$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: 'sales-report.rdl',
exportItemClick: onExportItemClick
});
});
`

```

2. Implement the function and replace the following code samples to send export type to server side.

```

`javascript
function onExportItemClick(args) {
var exportType;
var exportTypeFormat;
if (args.value != null) {
if (args.value == 'PDF') {
exportType = 'PDF';
exportTypeFormat = 'PDF';
} else if (args.value == 'Word') {
exportType = 'Word_' + args.model.exportSettings.wordFormat;
exportTypeFormat = 'Word';
} else if (args.value == 'Excel') {
exportType = 'Excel_' + args.model.exportSettings.excelFormat;
exportTypeFormat = 'Excel';
} else if (args.value == 'CSV') {
exportType = 'CSV';
exportTypeFormat = 'CSV';
} else {
var text = $(args.target.textContent);
exportTypeFormat = $(args.target).text();
}
}
if (args.value != null) {

```

```

var _this = this;
args.cancel = true;
var proxy = $('#viewer').data('boldReportViewer');
var requrl = '/api/ReportViewer/ExportReport';
var _json = {
  exportType: exportType,
  reportViewerToken: this['reportViewerToken']
};
$.ajax({
  contentType: "application/json; charset=utf-8",
  dataType: "json",
  type: "POST",
  url: requrl,
  data: JSON.stringify(_json),
  crossDomain: true,
  success: (data) => {
    if (data) {
      //Implement function to start the export
    }
  },
  error: (data) => {
    alert("Ajax failure");
  }
})
}
}
,

```

3. Implement the function **ExportReport** and replace the following code samples to get the report in the specified export type.

```

`csharp
public Dictionary<string, object> ExportReport([FromBody] Dictionary<string, object> jsonResult)
{
  string types = null;

```

```
string fileName = null;
Guid fileId = Guid.NewGuid();
if (jsonResult.ContainsKey("exportType"))
{
    type = null;
    types = type = jsonResult.Single(jr => jr.Key == "exportType").Value.ToString();
}
token = jsonResult["reportViewerToken"].ToString();
if (types.Contains("PDF"))
{
    fileName = "Name" + fileId.ToString() + ".pdf";
}
else if (types.Contains("Word"))
{
    fileName = "Name" + fileId.ToString() + ".doc";
}
else if (types.Contains("Excel"))
{
    fileName = "Name" + fileId.ToString() + ".xls";
}
else if (types.Contains("CSV"))
{
    fileName = "Name" + fileId.ToString() + ".csv";
}
string _token = jsonResult["reportViewerToken"].ToString();
string type1 = jsonResult["exportType"].ToString();
var stream = ReportHelper.GetReport(_token, type1);
var reportLocation = AppDomain.CurrentDomain.BaseDirectory;
using (var fileStream = System.IO.File.Create(reportLocation + @"App_Data\\" + fileName))
{
    stream.Seek(0, SeekOrigin.Begin);
    stream.CopyTo(fileStream);
}
```

```
Dictionary<string, object> jsonData = new Dictionary<string, object>();
filename = null;
filename = fileName;
jsonData.Add("fileName", fileName);
return jsonData;
}
,
```

4. After the ajax request gets successfully completed, getting the report data from the server side. You need to implement the function `exportReport` at client side and replace the following code samples to pass the custom authentication in headers along with file data using blob.

```
`javascript
function exportItemClick() {
...
...
$.ajax({
  contentType: "application/json; charset=utf-8",
  dataType: "json",
  type: "POST",
  url: requrl,
  data: JSON.stringify(_json),
  crossDomain: true,
  success: (data) => {
    if (data) {
      //Implement function to start the export
      exportReport(data);
    }
  },
  error: (data) => {
    alert("Ajax failure");
  }
})
}

function exportReport(jsonData) {
```



```

let url = /api/ReportViewer/ExportStart;
fetch(url, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json', 'authorization': 'authenticationtoken' },
  responseType: 'json',
  body: JSON.stringify(jsonData)
}).then(function (resp) {
  return resp.blob();
}).then(function (data) {
  var filename = null;
  Object.keys(jsonData).forEach(function (fileName) {
    filename = jsonData[fileName];
  });
  //Invoke the download file at client side
});
},
);

```

5. Implement the function **ExportReport** at server side and replace the following code samples to start the export of the report with the file data.

```

`csharp
[HttpPost]
public object ExportReport([FromBody] Dictionary<string, object> jsonResult)
{
  string fileId = null;
  if (jsonResult.ContainsKey("fileName"))
  {
    fileId = jsonResult["fileName"].ToString();
  }
  System.Net.Http.HttpResponseMessage response = new System.Net.Http.HttpResponseMessage();
  try
  {
    string path = HttpContext.Current.Server.MapPath("~/App_Data/");
    var stream = File.Open(path + fileId, FileMode.Open);
  }
}

```

```

stream.Position = 0;
System.Web.HttpResponse httpResponse = System.Web.HttpContext.Current.Response;
httpResponse.ClearContent();
httpResponse.Expires = 0;
httpResponse.Buffer = true;
httpResponse.AddHeader("Content-Disposition", "attachment;filename=\"\" + fileId + "\"");
httpResponse.AddHeader("Content-Type", "Application/" + type);
httpResponse.Clear();
byte[] buffer = new byte[stream.Length + 1];
int bytesRead;
while ((bytesRead = stream.Read(buffer, 0, buffer.Length)) > 0)
{
    httpResponse.OutputStream.Write(buffer, 0, bytesRead);
}
response.StatusCode = System.Net.HttpStatusCode.OK;
}
catch (Exception ex)
{
    response.StatusCode = System.Net.HttpStatusCode.NotFound;
}
return response;
}

```

6. You need to download the report at client side as shown in the following code sample.

```

`javascript
function exportReport(jsonData) {
let url = /api/ReportViewer/ExportStart;
fetch(url, {
method: 'POST',
headers: { 'Content-Type': 'application/json', 'authorization': 'authenticationtoken' },
responseType: 'json',
body: JSON.stringify(jsonData)

```

```

}).then(function (resp) {
return resp.blob();
}).then(function (data) {
var filename = null;
Object.keys(jsonData).forEach(function (fileName) {
filename = jsonData[fileName];
//Invoke the downloaf file at client side
downloadFile(data, filename);
});
});
}

function downloadFile(fileData, filename) {
// Create an object URL for the blob object
const url = URL.createObjectURL(fileData);
const a = document.createElement('a');
a.href = url;
a.download = filename || 'download';
const clickHandler = () => {
setTimeout(() => {
URL.revokeObjectURL(url);
this.removeEventListener('click', clickHandler);
}, 150);
};
a.addEventListener('click', clickHandler, false);
a.click();
return a;
}

```

See also

[How to resolve the image rendering and exporting issue with ASP.NET Core Authentication middleware](#)

How to pass multiple values using custom data

Pass the multiple data values as JSON in `ajaxBeforeLoad` event using the 'data' property, which needs to be serialized in the server side API using known class type and `JsonConvert.DeserializeObject`.

1. Map the [ajaxBeforeLoad](#) event with `onAjaxRequest` function in the script to pass custom data to the server.

```
`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/sales-order-detail.rdl',
ajaxBeforeLoad: onAjaxRequest
});
});
function onAjaxRequest(args) {
//Passing custom data to server
}
</script>
`
```

2. You need to pass the multiple custom data values as JSON and use the `data` property to send custom data to the server in the Ajax request.

```
`js
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer({
reportServiceUrl: "/api/ReportViewer",
reportPath: '~/Resources/docs/sales-order-detail.rdl',
ajaxBeforeLoad: onAjaxRequest
});
});
function onAjaxRequest(args) {
//Passing custom data to server
var jsonData = {
customerID: "CI0021",
productID: "PO0022"
}
```

```
};
args.data = jsonData;
}
</script>
`
```

3. The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates deserialization of the JSON string and change the data source connection strings based on Customer ID and Product ID in the `OnInitReportOptions` method.

```
`csharp
public SampleData customDatas = new SampleData();
public class SampleData
{
    public string customerID { get; set; }
    public string productID { get; set; }
}
[HttpPost]
public object PostReportAction([FromBody]Dictionary<string, object> jsonResult)
{
    if (jsonResult.ContainsKey("customData"))
    {
        //Get client side JSON custom data, deserialize it and store in local variable.
        customDatas = JsonConvert.DeserializeObject<SampleData>(jsonResult["customData"].ToString());
    }
    return ReportHelper.ProcessReport(jsonResult, this, this._cache);
}
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (customDatas != null)
    {
        if (customDatas.customerID == "CI0021" && customDatas.productID == "PO0022") {
            //If you are changing the connection string based on customer id then could you please change the
            connection string as below.
        }
    }
}
```

```
//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>;"));

}

}

}

`
```

How to clear cache when closing the Report Viewer

By using the `clearReportCache` and `destroy` method, you can clear the server side cache. You have to use this method when closing report viewer and switching to another report within your application.

Javascript

```
`js
<div id="reportviewer"></div>

<script>
var isSubmit = true;
$(document.body).bind('submit', $.proxy(this.formSubmit, this));
function formSubmit(args)
{
isSubmit = false;
}
window.onbeforeunload = function () {
if (isSubmit) {
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.clearReportCache();
reportviewerObj.destroy();
}
isSubmit = true;
};
</script>
`
```

Web API

In the `PostReportAction` method, you have to collect the GC with `ClearCache` as shown in the following code sample.

```
`csharp
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
{
    bool isclearcache = false;

    if (jsonArray != null && jsonArray.ContainsKey("reportAction") && jsonArray["reportAction"].ToString()
    == "ClearCache")
    {
        isclearcache = true;
    }

    var reportresult = ReportHelper.ProcessReport(jsonArray, this, this._cache);
    if (isclearcache)
    {
        GC.Collect(); isclearcache = false;
    }

    return reportresult;
}
`
```

Migrate to Report Viewer v2.0 component

This section provides simple step-by-step instructions to update your existing Report Viewer application to our latest modern v2.0 scripts, styles and components.

1. Open the html page that contains Report Viewer and its scripts.
2. Remove the following older scripts and CSS references from the page.
 - o bold.reports.all.min.css
 - o jquery.min.js
 - o ej2-base.min.js
 - o ej2-data.min.js
 - o ej2-pdf-export.min.js
 - o ej2-svg-base.min.js
 - o ej2-lineargauge.min.js
 - o ej2-circulargauge.min.js
 - o ej2-maps.min.js
 - o ej.chart.min.js
 - o bold.reports.common.min.js
 - o bold.reports.widgets.min.js
 - o bold.report-viewer.min.js
3. Add the following v2.0 scripts and CSS references in the <head> tag of the html page.

```
`html
<!-- Report Viewer component styles -->
<link href="https://cdn.boldreports.com/5.4.20/content/v2.0/tailwind-light/bold.report-viewer.min.css"
rel="stylesheet" />
<script src="https://cdn.jsdelivr.net/npm/jquery/3.6.0/jquery.min.js"></script>
<!-- Report Viewer component dependent script -->
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.widgets.min.js"></script>
<!-- Report Viewer component script -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/bold.report-viewer.min.js"></script>
`
```

The final updated html page for v2.0 Report Viewer, looks as follows.

```
`html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<link href="https://cdn.boldreports.com/5.4.20/content/v2.0/tailwind-light/bold.report-viewer.min.css"
rel="stylesheet" />
<script src="https://cdn.jsdelivr.net/npm/jquery/3.6.0/jquery.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.widgets.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/bold.report-viewer.min.js"></script>
</head>
<body>
<script type="text/javascript">
$(function () {
$("#viewer").boldReportViewer();
});
</script>
</body>
</html>
```


JavaScript Report Viewer API reference

Using the Report Viewer public properties, events, and members you can change the report options programmatically. The API gives you simple access to the functionality behind the Report Viewer. You can use this to create your own custom applications or to script interactions with Report Viewer. In this documentation contains the Report Viewer public API's details with code snippets.

- [Properties](#)
- [Events](#)
- [Members](#)
- [Methods](#)

Members

```
<h3 class="doc-prop-wrapper" id="datasources" data-Path="datasources-dataSources">
<a href="#datasources" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45-1-1.27-1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>dataSources</span>
<span class="doc-prop-type"> dataSources
</span>
</h3>
```

Gets or sets the list of data sources for the RDLC report.

Defaults to `[]`

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({ dataSources: [{name:"Menu Items",
values:[{ OrderId: "21D60", FoodName: "Burger", Price: 20, Category: "Veg" },
{ OrderId: "21D61", FoodName: "Pizza", Price: 25, Category: "Non-Veg" },
{ OrderId: "21D63", FoodName: "Sandwiches", Price: 30, Category: "Non-Veg" },
{ OrderId: "21D65", FoodName: "Chicken Drum Sticks", Price: 23, Category: "Non-Veg" },
{ OrderId: "21D64", FoodName: "Fulka", Price: 15, Category: "Veg" } ]}]}
```

```

});
</script>
`
<h3 class="doc-prop-wrapper" id="exportsettings" data-Path="exportsettings-exportSettings">
<a href="#exportsettings" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>exportSettings</span>
<span class="doc-prop-type"> exportSettings
</span>
</h3>
Specifies the export settings.
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer(
{
exportSettings:{ excelFormat: ej.ReportViewer.ExcelFormats.Excel2013,
wordFormat: ej.ReportViewer.WordFormats.Docx
}
});
</script>
`
<h3 class="doc-prop-wrapper" id="isresponsive" data-Path="isresponsive-isResponsive">
<a href="#isresponsive" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>

```

```

</svg>
</a><span class='doc-prop-name'>isResponsive</span>
<span class="doc-prop-type"> boolean
</span>
</h3>

```

When set to true, adapts the report layout to fit the screen size of devices on which it renders.

Defaults to *true*

```

<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer({ isResponsive: true });
</script>
,

<h3 class="doc-prop-wrapper" id="locale" data-Path="locale-locale">
<a href="#locale" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91.72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>locale</span>
<span class="doc-prop-type"> string
</span>
</h3>

```

Specifies the locale for report viewer.

Defaults to *en-US*

```

<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer(
{

```

```
locale: "it-IT"
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="pagesettings" data-Path="pagesettings-pageSettings">
```

```
<a href="#pagesettings" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>pageSettings</span>
```

```
<span class="doc-prop-type"> pageSettings
```

```
</span>
```

```
</h3>
```

Specifies the page settings.

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
{
```

```
pageSettings:{ paperSize: ej.ReportViewer.PaperSize.A4,
```

```
height: 11.69,
```

```
width: 8.27
```

```
}
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="parameters" data-Path="parameters-parameters">
```

```
<a href="#parameters" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>parameters</span>
```

```
<span class="doc-prop-type"> parameters
```

```
</span>
```

```
</h3>
```

Gets or sets the list of parameters associated with the report.

Defaults to `[]`

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
parameters: [{
```

```
name:"Vehicle",
```

```
labels:["Motor Bikes"],
```

```
prompt:"Please select the color",
```

```
values:["Red","Green","Blue","Yellow","Black"],
```

```
nullable:false
```

```
}]
```

```
});
```

```
</script>
```

```
`
```

```
<h3 class="doc-prop-wrapper" id="toolbarsettings" data-Path="toolbarsettings-toolbarSettings">
```

```
<a href="#toolbarsettings" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>toolbarSettings</span>
```

```
<span class="doc-prop-type"> toolbarSettings
```

```
</span>
```

```
</h3>
```

Specifies the toolbar settings.

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
toolbarSettings:{ showTooltip: true }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="parametersettings" data-Path="parametersettings-  
parameterSettings">
```

```
<a href="#parametersettings" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-  
.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-  
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2  
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>parameterSettings</span>
```

```
<span class="doc-prop-type"> parameterSettings
```

```
</span>
```

```
</h3>
```

Specifies the parameter settings.

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings: {
  delimiterChar: ",",
  popupHeight: "200px",
  popupWidth: "150px",
  itemWidth: '250px',
  labelWidth: 'auto',
  attributes: { 'autocomplete': 'off' },
  hideParameterBlock: false,
  timeDisplayFormat: "h:mm tt",
  dateTimeFormat: "M/d/yyyy",
  timeInterval: 30,
  accessInternalValue: true
}
});
</script>
`
```

```
<h3 class="doc-prop-wrapper" id="printmode" data-Path="printmode-printMode">
<a href="#printmode" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>printMode</span>
<span class="doc-prop-type"> boolean
</span>
</h3>
```

Enables and disables the print mode.

Defaults to *false*

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
```

```
$("#reportviewer").boldReportViewer(
{
printMode:true
});
</script>
`
```

```
<h3 class="doc-prop-wrapper" id="printoption" data-Path="printoption-printOption">
<a href="#printoption" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91
.72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2
.5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>printOption</span>
<span class="doc-prop-type"> enum
</span>
</h3>
<ts name = "ej.ReportViewer.PrintOptions" style = "display:none"></ts>
```

Specifies the print option of the report.

Name	Description
Default	Specifies the Default property in printOptions.
NewTab	Specifies the NewTab property in printOptions.
None	Specifies the None property in printOptions.

Defaults to *ej.ReportViewer.PrintOptions.Default*

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer({ printOption: ej.ReportViewer.PrintOptions.Default });
</script>
`

<h3 class="doc-prop-wrapper" id="processingmode" data-Path="processingmode-processingMode">
```



```

<a href="#processingmode" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>processingMode</span>
<span class="doc-prop-type"> enum
</span>
</h3>

```

```
<ts name = "ej.ReportViewer.ProcessingMode" style = "display:none"></ts>
```

Specifies the processing mode of the report.

Name	Description
Remote	Specifies the Remote property in processingMode.
Local	Specifies the Local property in processingMode.

Defaults to *ej.ReportViewer.ProcessingMode.Remote*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({ processingMode: ej.ReportViewer.ProcessingMode.Remote });
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="rendermode" data-Path="rendermode-renderMode">
```

```
<a href="#rendermode" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```

<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>

```

```
</svg>
```

```
</a><span class='doc-prop-name'>renderMode</span>
```

```
<span class="doc-prop-type"> enum
```


</h3>

<ts name = "ej.ReportViewer.RenderMode" style = "display:none"></ts>

Specifies the render layout.

Name	Description
Default	Specifies the Default property in RenderMode to get default output.
Mobile	Specifies the Mobile property in RenderMode to get specified output.
Desktop	Specifies the Desktop property in RenderMode to get specified output.

Defaults to *ej.ReportViewer.RenderMode.Default*

Example

`js

<div id="reportviewer"></div>

<script>

\$("#reportviewer").boldReportViewer({ renderMode: ej.ReportViewer.RenderMode.Default });

</script>

,

<h3 class="doc-prop-wrapper" id="reportpath" data-Path="reportpath-reportPath">

<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">

<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91.72-2.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>

</svg>

reportPath

 string

</h3>

Gets or sets the path of the report file.

Defaults to *empty*

Example

`js

<div id="reportviewer"></div>

```

<script>
$(("#reportviewer").boldReportViewer({ reportPath: "~/App_Data/Sample.rdl" }));
</script>
`

<h3 class="doc-prop-wrapper" id="reportserverurl" data-Path="reportserverurl-reportServerUrl">
<a href="#reportserverurl" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>reportServerUrl</span>
<span class="doc-prop-type"> string
</span>
</h3>

```

Gets or sets the reports server URL.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$(("#reportviewer").boldReportViewer({ reportServerUrl: "http://mvc.syncfusion.com/reportserver" }));
```

```
</script>
```

```
`
```

```

<h3 class="doc-prop-wrapper" id="reportserviceurl" data-Path="reportserviceurl-reportServiceUrl">
<a href="#reportserviceurl" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>reportServiceUrl</span>
<span class="doc-prop-type"> string

```


</h3>

Specifies the report Web API service URL.

Defaults to *empty*

Example

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({ reportServiceUrl: "../api/RDLReport" });
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="zoomfactor" data-Path="zoomfactor-zoomFactor">
```

```
<a href="#zoomfactor" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>zoomFactor</span>
```

```
<span class="doc-prop-type"> number
```

```
</span>
```

</h3>

Gets or sets the zoom factor for report viewer.

Defaults to *1*

Example

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({ zoomFactor: 2 });
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="serviceauthorizationtoken" data-Path="serviceauthorizationtoken-serviceAuthorizationToken">
```

```

<a href="#serviceauthorizationtoken" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>serviceAuthorizationToken</span>
<span class="doc-prop-type"> string
</span>
</h3>

```

Specifies the token for authorizing reporting service URL to process the reports.

Defaults to *empty*

```

<span style="font-weight:bold">Example</span>
`js
<div id="report-viewer"></div>
<script>
$(function () {
var dataValue = "";
var apiRequest = new Object();
apiRequest.password = "demo";
apiRequest.userid = "guest";
$.ajax({
type: "POST",
url: "http://reportserver.syncfusion.com/api/get-user-key",
data: apiRequest,
success: function (data) {
dataValue = data.Token;
var token = JSON.parse(dataValue);
$("#report-viewer").boldReportViewer(
{
reportServiceUrl: "http://reportserver.syncfusion.com/ReportService/api/Viewer",
serviceAuthorizationToken: token["tokentype"] + " " + token["accesstoken"],
reportPath: '/Sample Reports/Company Sales'

```

```

});
}
});
});
</script>
`

<h3 class="doc-prop-wrapper" id="toolbarrendermode" data-Path="toolbarrendermode-
toolbarRenderMode">

<a href="#toolbarrendermode" aria-hidden="true" class="anchor">

<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">

<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-
.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>

</svg>

</a><span class='doc-prop-name'>toolbarRenderMode</span>

<span class="doc-prop-type"> enum

</span>

</h3>

<ts name = "ej.ReportViewer.ToolbarRenderMode" style = "display:none"></ts>

```

Specifies the toolbar render mode of the report.

Name	Description
Native	Specifies the Native property in toolbarRenderMode.
Classic	Specifies the Classic property in toolbarRenderMode.

Defaults to *ej.ReportViewer.ToolbarRenderMode.Native*

```

<span style="font-weight:bold">Example</span>

`js

<div id="reportviewer"></div>

<script>

$("#reportviewer").boldReportViewer({ toolbarRenderMode:
ej.ReportViewer.ToolbarRenderMode.Classic });

</script>
`

```

```

<h3 class="doc-prop-wrapper" id="enableparameterblockscroller" data-
Path="enableparameterblockscroller-enableParameterBlockScroller">
<a href="#enableparameterblockscroller" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-
.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>enableParameterBlockScroller</span>
<span class="doc-prop-type"> boolean
</span>
</h3>

```

Enables and disables the parameter block scroller.

Defaults to *true*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer(
```

```
{
```

```
enableParameterBlockScroller: false
```

```
});
```

```
</script>
```

```
`
```

```

<h3 class="doc-prop-wrapper" id="enabledatasourceblockscroller" data-
Path="enabledatasourceblockscroller-enableDatasourceBlockScroller">

```

```
<a href="#enabledatasourceblockscroller" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```

<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-
.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>

```

```
</svg>
```

```
</a><span class='doc-prop-name'>enableDatasourceBlockScroller</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Enables and disables the data source credential block scroller.

Defaults to *true*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer(
```

```
{
```

```
enableDatasourceBlockScroller: false
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="sizetoreportcontent" data-Path="sizetoreportcontent-sizeToReportContent">
```

```
<a href="#sizetoreportcontent" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91.72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>sizeToReportContent</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Render the Report Viewer height based on the report content size.

Defaults to *false*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer(
```



```
{
sizeToReportContent: true
});
</script>
`
<h3 class="doc-prop-wrapper" id="autorender" data-Path="autorender-autoRender">
<a href="#autorender" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>autoRender</span>
<span class="doc-prop-type"> boolean
</span>
</h3>
```

Enables and disables the rendering of Viewer when default values are specified for the parameters.

Defaults to *true*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer(
```

```
{
```

```
autoRender: false
```

```
});
```

```
</script>
```

```
`
```

```
<h3 class="doc-prop-wrapper" id="enablenotificationbar" data-Path="enablenotificationbar-enableNotificationBar">
```

```
<a href="#enablenotificationbar" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-
```

```
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>enableNotificationBar</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Enables and disables the Error Notification bar.

Defaults to *true*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer(
```

```
{
```

```
enableNotificationBar: false
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="enabledropdownsearch" data-Path="enabledropdownsearch-
enableDropDownSearch">
```

```
<a href="#enabledropdownsearch" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-
.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>enableDropDownSearch</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Enables and disables the drop-down parameter search.

Defaults to *false*

```
<span style="font-weight:bold">Example</span>
```

```

`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer(
{
enableDropDownSearch: true
});
</script>
`

<h3 class="doc-prop-wrapper" id="enablevirtualevaluation" data-Path="enablevirtualevaluation-
enableVirtualEvaluation">
<a href="#enablevirtualevaluation" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-
.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>enableVirtualEvaluation</span>
<span class="doc-prop-type"> boolean
</span>
</h3>

```

Set the property value as true to enable the processing of a large amount of data with a lesser memory footprint and without performance degradation.

Defaults to *false*

Example

```

`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer(
{
enableVirtualEvaluation: true
});
</script>
`

```

```

<h3 class="doc-prop-wrapper" id="waitingpopuptemplate" data-Path="waitingpopuptemplate-
waitingpopuptemplate">
<a href="#waitingpopuptemplate" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-
.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>waitingPopupTemplate </span>
<span class="doc-prop-type"> String
</span>
</h3>

```

Gets or sets the waiting popup template for the Report viewer.

Defaults to *null*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer(
```

```
{
```

```
waitingPopupTemplate: '<div id="customLoadingtemplate" style="background-
image:url(https://js.syncfusion.com/demos/web/content/images/waitingpopup/load_light.gif); width:
200px; height: 15px"><p>Loading report... </p></div>'
```

```
});
```

```
</script>
```

```
,
```

```

<h3 class="doc-prop-wrapper" id="enableOnScrollNavigation" data-Path="enableOnScrollNavigation-
enableOnScrollNavigation">

```

```

<a href="#enableOnScrollNavigation" aria-hidden="true" class="anchor">

```

```

<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">

```

```

<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-
.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>

```

```
</svg>
```

```

</a><span class='doc-prop-name'>enableOnScrollNavigation</span>
<span class="doc-prop-type"> boolean
</span>
</h3>

```

Set the property value as true to enable the processing of a navigating next and previous page using scroller.

Defaults to *false*

```

<span style="font-weight:bold">Example</span>

```

```

`js

```

```

<div id="report-viewer"></div>

```

```

<script>

```

```

$("#report-viewer").boldReportViewer(

```

```

{

```

```

enableOnScrollNavigation: true

```

```

});

```

```

</script>

```

```

`

```

```

<h3 class="doc-prop-wrapper" id="customBrandSettings" data-Path="customBrandSettings-
customBrandSettings">

```

```

<a href="#customBrandSettings" aria-hidden="true" class="anchor">

```

```

<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">

```

```

<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-
.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>

```

```

</svg>

```

```

</a><span class='doc-prop-name'>customBrandSettings</span>

```

```

<span class="doc-prop-type"> customBrandSettings

```

```

</span>

```

```

</h3>

```

Specifies the customBrandSettings.

```

<span style="font-weight:bold">Example</span>

```

```

`js

```

```

<div id="reportviewer"></div>

```

```

<script>

```

```
$("#reportviewer").boldReportViewer(  
{  
  customBrandSettings: {  
    hideHelpLink: false,  
    customDomain: 'https://help.boldreports.com',  
    customBrandName: 'Bold Reports',  
    customLinks: [{ name: 'ESLicenseMessage', url: '/licensing/license-token/' }]  
  }  
});  
</script>  
,
```

Methods

destroy()

Destroy the client and server side report viewer processing objects.

Example

```
`js  
<div id="report-viewer"></div>  
<script>  
var reportviewerObj = $("#report-viewer").data("boldReportViewer");  
reportviewerObj.destroy(); //Destroy the report viewer processing objects.  
</script>  
,
```

exportReport()

Export the report to the specified format.

Parameter	Type	Description
show	string	Indicates to choose the export type while exporting like PDF or Excel or Word and etc...

Example

```
`js  
<div id="reportviewer">ReportViewer</div>  
<script>  
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
```

reportviewerObj.exportReport("PDF"); //Exports the report into PDF format.

</script>

,

fitToPage()

Fit the report page to the container.

Example

`js

<div id="reportviewer">ReportViewer</div>

<script>

var reportviewerObj = \$("#reportviewer").data("boldReportViewer");

reportviewerObj.fitToPage(); // To fit the report page.

</script>

,

fitToPageHeight()

Fit the report page height to the container.

Example

`js

<div id="reportviewer">ReportViewer</div>

<script>

var reportviewerObj = \$("#reportviewer").data("boldReportViewer");

reportviewerObj.fitToPageHeight(); // To fit the report page height.

</script>

,

fitToPageWidth()

Fit the report page width to the container.

Example

`js

<div id="reportviewer">ReportViewer</div>

<script>

var reportviewerObj = \$("#reportviewer").data("boldReportViewer");

reportviewerObj.fitToPageWidth(); // To fit the report page width.

</script>

,

getDataSetNames()

Get the available datasets name of the rdlc report.

Example

`js

<div id="reportviewer">ReportViewer</div>

<script>

var reportviewerObj = \$("#reportviewer").data("boldReportViewer");

reportviewerObj.getDataSetNames(); // To get the dataset names.

</script>

,

getParameters()

Get the available parameters of the report.

Example

`js

<div id="reportviewer">ReportViewer</div>

<script>

var reportviewerObj = \$("#reportviewer").data("boldReportViewer");

reportviewerObj.getParameters(); // To get the parameters.

</script>

,

gotoFirstPage()

Navigate to first page of report.

Example

`js

<div id="reportviewer">ReportViewer</div>

<script>

var reportviewerObj = \$("#reportviewer").data("boldReportViewer");

reportviewerObj.gotoFirstPage(); // To navigate to first page

</script>

,

gotoLastPage()

Navigate to last page of the report.

Example

`js


```
<div id="reportviewer">ReportViewer</div>
<script>
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.gotoLastPage(); // Navigate to the last page
</script>
`js
```

gotoNextPage()

Navigate to next page from the current page.

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer">ReportViewer</div>
<script>
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.gotoNextPage(); //To navigate to the next page
</script>
`js
```

gotoPageIndex()

Go to specific page index of the report.

Parameter	Type	Description
show	number	Indicates to pass the page number to show the page directly instead of starting from first page.

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer">ReportViewer</div>
<script>
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.gotoPageIndex(5); // To navigate the specific page
</script>
`js
```

gotoPreviousPage()

Navigate to previous page from the current page.

```
<span style="font-weight:bold">Example</span>
`js
```

```
<div id="reportviewer">ReportViewer</div>
<script>
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.gotoPreviousPage(); // To navigate to the previous page
</script>
```

,

print()

Print the report.

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer">ReportViewer</div>
<script>
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.print(); //To perform print operation.
</script>
```

,

printLayout()

Apply print layout to the report.

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer">ReportViewer</div>
<script>
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.printLayout(); //Changes between print layout and normal modes.
</script>
```

,

refresh()

Refresh the report.

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer">ReportViewer</div>
<script>
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.refresh(); // To refresh the report.
```

```
</script>
```

```
,
```

setParameterBlockVisibility()

Specify the visibility of parameter block.

Parameter	Type	Description
show	boolean	Indicates whether the parameter block needs to be shown or not. true if you have to show the parameter block; otherwise, false.

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
```

```
reportviewerObj.setParameterBlockVisibility(true);
```

```
</script>
```

```
,
```

cancelRendering()

Cancel the report processing.

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer">ReportViewer</div>
```

```
<script>
```

```
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
```

```
reportviewerObj.cancelRendering(); // To cancel the report processing.
```

```
</script>
```

```
,
```

Events

drillThrough

Fires during drill through action done in report. If you want to perform any operation when a drill through action is performed, you can make use of the drillThrough event.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
actionInfo	Object	returns the actionInfo's parameters bookmarkLink, reportName, parameters.

Name	Type	Description
model	object	returns the report model.
type	string	returns the name of the event.

Example

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer({
```

```
  drillThrough: function (args) {
```

```
    // Write a code block to perform any operation when drill through action occurs in report.
```

```
  }
```

```
});
```

```
</script>
```

```
,
```

renderingBegin

Fires before report rendering is completed. If you want to perform any operation before the rendering of report, you can make use of the renderingBegin event.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
model	object	returns the report model.
type	string	returns the name of the event.

Example

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
//rendering begin event for report.
```

```
$("#report-viewer").boldReportViewer({
```

```
  renderingBegin:function(args) {
```

```
    // Write a code block to perform any operation before rendering.
```

```
  }
```

```
});
```

```
</script>
```

```
,
```

renderingComplete

Fires after report rendering completed. If you want to perform any operation after the rendering of report, you can make use of this renderingComplete event.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
model	object	returns the report model.
type	string	returns the name of the event.
reportParameters	object	returns the collection of parameters.

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
//rendering complete event for report viewer control.
```

```
$("#report-viewer").boldReportViewer({
```

```
renderingComplete:function(args) {
```

```
// Write a code block to perform any operation after rendering completed.
```

```
}
```

```
});
```

```
</script>
```

```
,
```

reportError

Fires when any error occurred while rendering the report. If you want to perform any operation when an error occurs in the report, you can make use of the reportError event.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
error	string	returns the error details.
model	object	returns the report model.
type	string	returns the name of the event.

```
<span style="font-weight:bold">Example</span>
```

```
`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
reportError: function (args) {
// Write a code block to perform any operation when report error occurs.
}
});
</script>
`
```

reportExport

Fires when the report is being exported. If you want to perform any operation before exporting of report, you can make use of the reportExport event.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
model	object	returns the report model.
type	string	returns the name of the event.

>Example

```
`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
reportExport: function (args) {
// Write a code block to perform any action before exporting of report.
}
});
</script>
`
```

reportLoaded

Fires when the report is loaded. If you want to perform any operation after the successful loading of report, you can make use of the reportLoaded event.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
model	object	returns the report model.
type	string	returns the name of the event.

Example

```
`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
reportLoaded: function(args) {
// Write a code block to perform any action when the report is loaded successfully.
}
});
</script>
`
```

showError

Fires when user clicks on a failed report item in the rendered report, before displaying error details dialog. If you want to show custom error detail or perform any action before viewing error detail, you can make use of the showError event.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
errorCode	string	returns the error code.
message	string	returns the error message.
detail	string	returns the detailed error information.

Example

```
`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
showError: function (args) {
// Write a code block to perform any operation when user clicks a failed item in a report.
}
```

```

}
});
</script>
`

```

viewReportClick

Fires when click the View Report Button.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
parameters	Object	returns the parameter collection.
model	object	returns the report model.
type	string	returns the name of the event.

Example

```

`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
viewReportClick: function (args) {
// Write a code block to perform any operation after destroy of report viewer.
}
});
</script>
`

```

ajaxBeforeLoad

Fires before the AJAX request process started.

Name	Type	Description
reportViewerToken	string	returns the reportViewerToken.
serviceAuthorizationToken	string	returns the serviceAuthorizationToken.
headerReq	Object	Send the header request collection.
headers	Object	Send the headers collection.
data	string	Send the custom data.
actionName	string	Change the Web API action with custom endpoint.


```
<span style="font-weight:bold">Example</span>
`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
  ajaxBeforeLoad: function (args) {
    // Write a code block to perform any operation after destroy of report viewer.
  }
});
</script>
`
```

ajaxSuccess

Fires when AJAX post call succeed.

Name	Type	Description
data	object	returns the success data.

```
<span style="font-weight:bold">Example</span>
`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
  ajaxSuccess: function (args) {
    // Write a code block to perform any operation after destroy of report viewer.
  }
});
</script>
`
```

ajaxError

Fires when AJAX request failed.

Name	Type	Description
msg	string	returns the error details

```
<span style="font-weight:bold">Example</span>
`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
  ajaxError: function (args) {
    // Write a code block to perform any operation after destroy of report viewer.
  }
});
</script>
`
```

toolbarRendering

This event will be triggered on rendering the report viewer toolbar.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
model	object	returns the report model.
type	string	returns the name of the event.
target	JQuery	returns the toolbar container.

```
<span style="font-weight:bold">Example</span>
`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
  toolbarRendering: function(args) {
    // Write a code block to customize the report viewer toolbar.
  }
});
</script>
`
```

exportProgressChanged

Fires when the export progress is changed. To perform any operation when the export progress is changed, use the exportProgressChanged event.

Name	Type	Description
format	string	returns the export format
stage	string	returns the stage of export processing.
handled	boolean	true if the event should be handled; otherwise, false.
containerId	string	returns report viewer container Id.
waitingPopupTemplate	string	Set the waitingPopupTemplate in report viewer.
actionName	string	Change the Web API action with custom endpoint.

Example

```
`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
  exportProgressChanged : function(args) {
    // Write a code block to perform any action when the export progress changed.
  }
});
</script>
`
```

printProgressChanged

Fires when the print progress is changed. To perform any operation when the print progress is changed, use the printProgressChanged event.

Name	Type	Description
stage	string	returns the stage of export processing.
currentPage	string	returns the currentPage value
totalPages	string	returns the totalPages value
handled	boolean	true if the event should be handled; otherwise, false.
containerId	string	returns report viewer container Id.
waitingPopupTemplate	string	Set the waitingPopupTemplate in report viewer.

Example

```
`js
```

```

<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
printProgressChanged : function(args) {
// Write a code block to perform any action when the print progress changed.
}
});
</script>
`

```

exportItemClick

Fires when the export items are clicked. To perform any operation when the export items are clicked, use the exportItemClick event.

Name	Type	Description
value	string	returns the export format value.

Example

```

`js
<div id="report-viewer"></div>
<script>
$("#report-viewer").boldReportViewer({
exportItemClick : function(args) {
// Write a code block to perform any action when the export item clicked.
}
});
</script>
`

```

toolBarItemClick

Fires when the toolbar items are clicked. To perform any operation when the toolbar items are clicked, use the toolBarItemClick event.

Name	Type	Description
target	string	returns the toolbar clicked item name .
cssClass	string	returns the CSS class name specified for the toolbar item
groupIndex	string	returns the Toolbar item rendered group index

Name	Type	Description
Index	string	returns the Toolbar item rendered index
value	string	returns the Toolbar item value.

Example

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer({
```

```
toolBarItemClick : function(args) {
```

```
// Write a code block to perform any action when the toolbar item clicked.
```

```
}
```

```
});
```

```
</script>
```

```
,
```

hyperlink

Fires when the hyperlink action is performed in the report. To perform any operation during the hyperlink action, use the hyperlink event.

Name	Type	Description
cancel	boolean	true if the event should be canceled; otherwise, false.
actionInfo	Object	returns the actionInfo's hyperLink detail
model	object	returns the report model.
type	string	returns the name of the event.

Example

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer({
```

```
hyperlink: function (args) {
```

```
// Write a code block to perform any operation when hyperlink action occurs in report.
```

```
}
```

```
});
```

```
</script>
```

```
,
```

reportPrint

Fires when the report print action is performed in the report. To perform any operation during the report print action, use the ReportPrint event.

Name	Type	Description
isStyleLoad	boolean	true if you have to load the external style file; otherwise, false.

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```
$("#report-viewer").boldReportViewer({
```

```
reportPrint : function(args) {
```

```
// Write a code block to perform any action when the export item clicked.
```

```
}
```

```
});
```

```
</script>
```

```
,
```

beforeParameterAdd

Fires when begin to add the parameter element in the report viewer parameter block. To perform any changes in parameter element, use the beforeParameterAdd event.

Name	Type	Description
parameterModel	object	returns the parameter model.
containerId	string	returns the current report parameter container id
handled	boolean	true if the event should be handled; otherwise, false.
parameterSettings	object	returns the parameter settings.
customProperties	object	set custom properties to the report parameter.

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="report-viewer"></div>
```

```
<script>
```

```

$("#report-viewer").boldReportViewer({
  beforeParameterAdd : function(args) {
    // Write a code block to perform any change in report parameter element before add it to the viewer
    block.
  }
});
</script>
`

```

JavaScript Report Viewer Properties

This section provides the JavaScript Report Viewer properties with example code snippets used to customize the report preview, export, and much more.

- [Page Settings](#)
- [Parameters](#)
- [Export Settings](#)
- [Toolbar Settings](#)
- [Parameter Settings](#)
- [Data Sources](#)

pageSettings

Properties

```

<h3 class="doc-prop-wrapper" id="orientation" data-Path="orientation-orientation">
  <a href="#orientation" aria-hidden="true" class="anchor">
    <svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
      <path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
    </svg>
  </a><span class='doc-prop-name'>orientation</span>
  <span class="doc-prop-type"> enum
</span>
</h3>
<ts name = "ej.ReportViewer.Orientation" style = "display:none"></ts>

```

Specifies the print layout orientation.

Name	Description
Landscape	Specifies the Landscape property in pageSettings.orientation to get specified layout.

Name	Description
portrait	Specifies the portrait property in pageSettings.orientation to get specified layout.

Defaults to *null*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
pageSettings:{ orientation: ej.ReportViewer.Orientation.Landscape }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="papersize" data-Path="papersize-paperSize">
```

```
<a href="#papersize" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>paperSize</span>
```

```
<span class="doc-prop-type"> enum
```

```
</span>
```

```
</h3>
```

```
<ts name = "ej.ReportViewer.PaperSize" style = "display:none"></ts>
```

Specifies the paper size of print layout.

Name	Description
A3	Specifies the A3 as value in pageSettings.paperSize to get specified size.
A4	Specifies the A4 as value in pageSettings.paperSize to get specified size.
B4_JIS	Specifies the B4(JIS) as value in pageSettings.paperSize to get specified size.
B5_JIS	Specifies the B5(JIS) as value in pageSettings.paperSize to get specified size.

Name	Description
Envelope_10	Specifies the Envelope #10 as value in pageSettings.paperSize to get specified size.
Envelope_Monarch	Specifies the Envelope as value in pageSettings.paperSize to get specified size.
Executive	Specifies the Executive as value in pageSettings.paperSize to get specified size.
Legal	Specifies the Legal as value in pageSettings.paperSize to get specified size.
Letter	Specifies the Letter as value in pageSettings.paperSize to get specified size.
Tabloid	Specifies the Tabloid as value in pageSettings.paperSize to get specified size.
Custom	Specifies the Custom as value in pageSettings.paperSize to get specified size.

Defaults to *null*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
pageSettings:{ paperSize: ej.ReportViewer.PaperSize.A4 }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="height" data-Path="height-height">
```

```
<a href="#height" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>height</span>
```

```
<span class="doc-prop-type"> number
```

```
</span>
```

```
</h3>
```

Specifies the height of print layout.

Defaults to 0

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
{
```

```
pageSettings: { height: 11.69 }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="width" data-Path="width-width">
```

```
<a href="#width" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>width</span>
```

```
<span class="doc-prop-type"> number
```

```
</span>
```

```
</h3>
```

Specifies the width of print layout.

Defaults to 0

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
{
```

```
pageSettings: { width: 8.27 }
```

```
});
```

```
</script>
```

```

<h3 class="doc-prop-wrapper" id="margins" data-Path="margins-margins">
<a href="#margins" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>margins</span>
<span class="doc-prop-type"> object
</span>
</h3>

```

Specifies the margins of print layout.

Name	Description
top	Specifies the top margins of print layout.
right	Specifies the right margins of print layout.
bottom	Specifies the bottom margins of print layout.
left	Specifies the left margins of print layout.

```

<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer(
{
pageSettings: {
margins: { top: 0.5, right: 0.25, bottom: 0.25, left: 0.25 }
}
});
</script>

```

parameters `array`

Properties

```

<h3 class="doc-prop-wrapper" id="labels" data-Path="labels-labels">
<a href="#labels" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>labels</span>
<span class="doc-prop-type"> array
</span>
</h3>

```

Gets or sets the parameter labels.

Defaults to *null*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```

$("#reportviewer").boldReportViewer({
parameters: [{
name:"Vehicle",
labels:["Motor Bikes"],
prompt:"Please select the color",
values:["Red","Green","Blue","Yellow","Black"],
nullable:false
}]
});
</script>

```

```

<h3 class="doc-prop-wrapper" id="name" data-Path="name-name">
<a href="#name" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">

```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>name</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Gets or sets the name of the parameter.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
parameters: [{
```

```
name:"Vehicle",
```

```
labels:["Motor Bikes"],
```

```
prompt:"Please select the color",
```

```
values:["Red","Green","Blue","Yellow","Black"],
```

```
nullable:false
```

```
}]
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="nullable" data-Path="nullable-nullable">
```

```
<a href="#nullable" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>nullable</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Gets or sets whether the parameter allows nullable value or not.

Defaults to *false*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
parameters: [{
```

```
name:"Vehicle",
```

```
labels:["Motor Bikes"],
```

```
prompt:"Please select the color",
```

```
values:["Red","Green","Blue","Yellow","Black"],
```

```
nullable:false
```

```
}]
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="prompt" data-Path="prompt-prompt">
```

```
<a href="#prompt" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>prompt</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Gets or sets the prompt message associated with the specified parameter.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
parameters: [{
```

```
name:"Vehicle",
```

```
labels:["Motor Bikes"],
```

```
prompt:"Please select the Color",
```

```
values:["Red","Green","Blue","Yellow","Black"],
```

```
nullable:false
```

```
}]
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="values" data-Path="values-values">
```

```
<a href="#values" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>values</span>
```

```
<span class="doc-prop-type"> array
```

```
</span>
```

```
</h3>
```

Gets or sets the parameter values.

Defaults to `[]`

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```

parameters: [{
  name:"Vehicle",
  labels:["Motor Bikes"],
  prompt:"Please select the color",
  values:["Red","Green","Blue","Yellow","Black"],
  nullable:false
}]
});
</script>
`

```

exportSettings

Properties

```

<h3 class="doc-prop-wrapper" id="exportoptions" data-Path="exportoptions-exportOptions">
<a href="#exportoptions" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>exportOptions</span>
<span class="doc-prop-type"> enum
</span>
</h3>
<ts name = "ej.ReportViewer.ExportOptions" style = "display:none"></ts>

```

Specifies the export formats.

Name	Description
All	Specifies the All property in ExportOptions to get all available options.
Pdf	Specifies the Pdf property in ExportOptions to get Pdf option.
Word	Specifies the Word property in ExportOptions to get Word option.
Excel	Specifies the Excel property in ExportOptions to get Excel option.
Html	Specifies the Html property in ExportOptions to get Html option.
PPT	Specifies the PPT property in ExportOptions to get PPT option.

Name	Description
CSV	Specifies the CSV property in ExportOptions to get CSV option.
XML	Specifies the XML property in ExportOptions to get XML option.
CustomItems	Specifies the customItems property in ExportOptions to get customItems option.

Defaults to *ej.ReportViewer.ExportOptions.All*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
exportSettings:{ exportOptions: ej.ReportViewer.ExportOptions.Html |  
ej.ReportViewer.ExportOptions.Pdf }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="excelformat" data-Path="excelformat-excelFormat">
```

```
<a href="#excelformat" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-  
.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-  
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2  
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>excelFormat</span>
```

```
<span class="doc-prop-type"> enum
```

```
</span>
```

```
</h3>
```

```
<ts name = "ej.ReportViewer.ExcelFormats" style = "display:none"></ts>
```

Specifies the excel export format.

Name	Description
Excel97to2003	Specifies the Excel97to2003 property in ExcelFormats to get specified version of exported format.

Name	Description
Excel2007	Specifies the Excel2007 property in ExcelFormats to get specified version of exported format.
Excel2010	Specifies the Excel2010 property in ExcelFormats to get specified version of exported format.
Excel2013	Specifies the Excel2013 property in ExcelFormats to get specified version of exported format.

Defaults to *ej.ReportViewer.ExcelFormats.Excel2013*

Example

```
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer(
{
exportSettings:{ excelFormat: ej.ReportViewer.ExcelFormats.Excel2013}
});
</script>
`
<h3 class="doc-prop-wrapper" id="wordformat" data-Path="wordformat-wordFormat">
<a href="#wordformat" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>wordFormat</span>
<span class="doc-prop-type"> enum
</span>
</h3>
<ts name = "ej.ReportViewer.WordFormats" style = "display:none"></ts>
Specifies the word export format.
```

Name	Description
Doc	Specifies the Doc property in WordFormats to get specified version of exported format.
Dot	Specifies the Dot property in WordFormats to get specified version of exported format.
DOCX	Specifies the DOCX property in WordFormats to get specified version of exported format.
Word2007	Specifies the Word2007 property in WordFormats to get specified version of exported format.
Word2010	Specifies the Word2010 property in WordFormats to get specified version of exported format.
Word2013	Specifies the Word2013 property in WordFormats to get specified version of exported format.
Word2007Dotx	Specifies the Word2007Dotx property in WordFormats to get specified version of exported format.
Word2010Dotx	Specifies the Word2010Dotx property in WordFormats to get specified version of exported format.
Word2013Dotx	Specifies the Word2013Dotx property in WordFormats to get specified version of exported format.
Word2007Docm	Specifies the Word2007Docm property in WordFormats to get specified version of exported format.
Word2010Docm	Specifies the Word2010Docm property in WordFormats to get specified version of exported format.
Word2013Docm	Specifies the Word2013Docm property in WordFormats to get specified version of exported format.
Word2007Dotm	Specifies the Word2007Dotm property in WordFormats to get specified version of exported format.
Word2010Dotm	Specifies the Word2010Dotm property in WordFormats to get specified version of exported format.
Word2013Dotm	Specifies the Word2013Dotm property in WordFormats to get specified version of exported format.
RTF	Specifies the RTF property in WordFormats to get specified version of exported format.
Txt	Specifies the Txt property in WordFormats to get specified version of exported format.

Name	Description
EPUB	Specifies the EPUB property in WordFormats to get specified version of exported format.
HTML	Specifies the HTML property in WordFormats to get specified version of exported format.
XML	Specifies the XML property in WordFormats to get specified version of exported format.

Defaults to *ej.ReportViewer.WordFormats.Docx*

Example

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
exportSettings:{ wordFormat: ej.ReportViewer.WordFormats.Docx}
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="customitems" data-Path="customitems-customItems">
```

```
<a href="#customitems" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>customItems</span>
```

```
<span class="doc-prop-type"> array
```

```
</span>
```

```
</h3>
```

Add the custom icon item to the export options.

Defaults to *empty*

Example

```

`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer(
{
  exportSettings: {
    customItems: [{
      index: 2,
      cssClass: "",
      value: 'exportCustom',
    },
    {
      index: 4,
      cssClass: "",
      value: 'exportCustom3',
    }
  ]
},
});
</script>
`

<h3 class="doc-prop-wrapper" id="imagequality" data-Path="imagequality-imageQuality">
<a href="#imagequality" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>ImageQuality</span>
<span class="doc-prop-type"> number
</span>
</h3>

```

Sets the image quality of data visualization items in report export. It allows value range from 1 to 10.

Defaults to 2

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
exportSettings:{
```

```
CommonOptions:{
```

```
ImageQuality: 4
```

```
}
```

```
}
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="useprintsizes" data-Path="useprintsizes-usePrintSizes">
```

```
<a href="#useprintsizes" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91.72-2 .25V8.59c-.58-.45-1-1.27-1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>UsePrintSizes</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Set the property value as true to reflect Report Viewer print page settings in the report export document.

Defaults to *false*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
  exportSettings:{
    CommonOptions:{
      UsePrintSizes: true
    }
  }
};
</script>
```

toolbarSettings

Properties

```
<h3 class="doc-prop-wrapper" id="click" data-Path="click-click">
  <a href="#click" aria-hidden="true" class="anchor">
    <svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
      <path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
    </svg>
  </a><span class='doc-prop-name'>click</span>
  <span class="doc-prop-type"> string
</span>
</h3>
```

Fires when user click on toolbar item in the toolbar.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer(
{
  toolbarSettings:{click: "onToolbarClick"}
});
</script>
```

```

<h3 class="doc-prop-wrapper" id="items" data-Path="items-items">
<a href="#items" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>items</span>
<span class="doc-prop-type"> enum
</span>
</h3>

<ts name = "ej.ReportViewer.ToolbarItems" style = "display:none"></ts>

```

Specifies the toolbar items.

Name	Description
Print	Specifies the Print as value in ToolbarItems to get specified item.
Refresh	Specifies the Refresh as value in ToolbarItems to get specified item.
Stop	Specifies the Stop as value in ToolbarItems to get specified item.
Zoom	Specifies the Zoom as value in ToolbarItems to get specified item.
FittoPage	Specifies the FittoPage as value in ToolbarItems to get specified item.
Export	Specifies the Export as value in ToolbarItems to get specified item.
PageNavigation	Specifies the PageNavigation as value in ToolbarItems to get specified item.
Parameters	Specifies the Parameters as value in ToolbarItems to get specified item.
PrintLayout	Specifies the PrintLayout as value in ToolbarItems to get specified item.
PageSetup	Specifies the PageSetup as value in ToolbarItems to get specified item.
ExportSetup	Specifies the ExportSetup as value in ToolbarItems to get specified item.
Find	Specifies the Find as value in ToolbarItems to get specified item.
Analytics	Specifies the Analytics as value in ToolbarItems to get specified item.
Settings	Specifies the Settings as value in ToolbarItems to get specified item.
All	Shows all toolbar items.

Defaults to *null*

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer(
{
toolbarSettings:{ items: ej.ReportViewer.ToolbarItems.All }
});
</script>
,

<h3 class="doc-prop-wrapper" id="toolbars" data-Path="toolbars-toolbars">
<a href="#toolbars" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>toolbars</span>
<span class="doc-prop-type"> enum
</span>
</h3>
<ts name = "ej.ReportViewer.Toolbars" style = "display:none"></ts>
```

Specifies the toolbars.

Name	Description
Horizontal	Specifies the Horizontal as value in Toolbars to display horizontal toolbar.
Vertical	Specifies the Vertical as value in Toolbars to display vertical toolbar.
All	Shows both vertical and horizontal toolbars.

Defaults to *All*

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
```

```

<script>
$("#reportviewer").boldReportViewer(
{
toolbarSettings:{ toolbars: ej.ReportViewer.Toolbars.All & ~ej.ReportViewer.Toolbars.Horizontal }
});
</script>
`

<h3 class="doc-prop-wrapper" id="showtoolbar" data-Path="showtoolbar-showToolbar">
<a href="#showtoolbar" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>showToolbar</span>
<span class="doc-prop-type"> boolean
</span>
</h3>
Shows or hides the toolbar.
Defaults to true

<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer(
{
toolbarSettings:{ showToolbar: true }
});
</script>
`

<h3 class="doc-prop-wrapper" id="showtooltip" data-Path="showtooltip-showTooltip">
<a href="#showtooltip" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">

```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>showTooltip</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Shows or hides the tooltip of toolbar items.

Defaults to *true*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
toolbarSettings:{ showTooltip: true }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="autohide" data-Path="autoHide-autoHide">
```

```
<a href="#autohide" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>autoHide</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Enable or Disable auto hide for horizontal toolbar.

Defaults to *true*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
toolbarSettings:{ autoHide: true }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="autohidedelay" data-Path="autoHideDelay-autoHideDelay">
```

```
<a href="#autohidedelay" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91.72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>autoHideDelay</span>
```

```
<span class="doc-prop-type"> number
```

```
</span>
```

```
</h3>
```

Set auto hide time delay for horizontal toolbar.

Defaults to 5

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
toolbarSettings:{ autoHideDelay: 5 }
```

```
});
```

```
</script>
```

```
,
```

```

<h3 class="doc-prop-wrapper" id="templateid" data-Path="templateid-templateId">
<a href="#templateid" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>templateId</span>
<span class="doc-prop-type"> string
</span>
</h3>

```

Specifies the toolbar template ID.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
toolbarSettings:{ templateId: "customtoolbarId" }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="customitems" data-Path="customitems-customItems">
```

```
<a href="#customitems" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>customItems</span>
```

```
<span class="doc-prop-type"> array
```

```
</span>
```

</h3>

Add the custom icon item to the toolbar.

Defaults to *empty*

Example

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
  toolbarSettings:{
```

```
    customItems: [{
```

```
      groupId: 1,
```

```
      index: 1,
```

```
      type: 'Default',
```

```
      prefixIcon: "e-icons e-save",
```

```
      appendTo: 'HorizontalToolbar',
```

```
      id: "customItem",
```

```
      cssClass: "CustomItem",
```

```
      tooltip: { header: 'CustomItem', content: 'toolbaritems', value: 'CustomItem' },
```

```
      sidePanel: { isEnabled: true, title: 'Custom Panel', id: 'custom-panel' }
```

```
    ]}
```

```
  }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="customgroups" data-Path="customgroups-customGroups">
```

```
<a href="#customgroups" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91.72-2 .25V8.59c-.58-.45-1-1.27-1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13 9.8 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09-.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>customGroups</span>
```

```
<span class="doc-prop-type"> array
```

```
</span>
```

```
</h3>
```

Add the custom icon groups to the toolbar.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
{
```

```
  toolbarSettings:{
```

```
    customGroups: [{
```

```
      items: [{
```

```
        type: 'Default',
```

```
        prefixIcon: "e-icons e-save",
```

```
        id: "customGroup",
```

```
        cssClass: "CustomGroup",
```

```
        tooltip: { header: 'CustomItem', content: 'toolbaritems', value: 'CustomItem' }
```

```
      },
```

```
      {
```

```
        type: 'Default',
```

```
        prefixIcon: "e-icons e-save",
```

```
        id: "customsubGroup",
```

```
        cssClass: "subCustomGroup",
```

```
        tooltip: { header: 'subCustomGroup', content: 'subtoolbaritems', value: 'subCustomGroup' }
```

```
      }],
```

```
      appendTo: 'VerticalToolbar',
```

```
      groupIndex: 3
```

```
    }],
```

```
  }
```

```
});
```

```
</script>
```

parameterSettings

Properties

```
<h3 class="doc-prop-wrapper" id="delimiterchar" data-Path="delimiterchar-delimiterChar">
<a href="#delimiterchar" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>delimiterChar</span>
<span class="doc-prop-type"> string
</span>
</h3>
```

Sets the separator when the multiSelectMode with delimiter option or checkbox is enabled with the dropdown. When you enter the delimiter value, the texts after the delimiter are considered as a separate word or query. The delimiter string is a single character and must be a symbol. Mostly, the delimiter symbol is used as comma (,) or semi-colon (;) or any other special character.

Defaults to ','

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings:{ delimiterChar: "," }
```

```
});
```

```
</script>
```

```
<h3 class="doc-prop-wrapper" id="position" data-Path="position-position">
```

```
<a href="#position" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
```



```
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>position</span>
```

```
<span class="doc-prop-type"> enum
```

```
</span>
```

```
</h3>
```

```
<ts name = "ej.ReportViewer.Position" style = "display:none"></ts>
```

Specifies the parameter panel docking position.

This support applicable only in Classic view.

Name	Description
Top	Specify the value to dock parameter panel in top side.
Bottom	Specify the value to dock parameter panel in bottom side.
Right	Specify the value to dock parameter panel in right side.
Left	Specify the value to dock parameter panel in left side.

Defaults to *ej.ReportViewer.Position.Top*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
toolbarRenderMode: ej.ReportViewer.ToolbarRenderMode.Classic,
```

```
parameterSetting: {
```

```
position: ej.ReportViewer.Position.Right;
```

```
}
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="popupheight" data-Path="popupheight-popupHeight">
```

```
<a href="#popupheight" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>popupHeight</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Specifies the height of the combobox parameter popup list. By default, the popup height value is **152px**.

Defaults to *152px*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings:{ popupHeight: "200px" }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="popupwidth" data-Path="popupwidth-popupWidth">
```

```
<a href="#popupwidth" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>popupWidth</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Specifies the width of the combobox parameter popup list. By default, the popup width sets based on the width of the component.

Defaults to *'auto'*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
{
```

```
parameterSettings:{ popupWidth: "150px" }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="itemwidth" data-Path="itemwidth-itemWidth">
```

```
<a href="#itemwidth" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>itemWidth</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Specifies the width of the parameter item. By default, the item width value is set as **185px**.

Defaults to *185px*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
{
```

```
parameterSettings:{ itemWidth: "250px" }
```

```
});
```

```
</script>
```

```

<h3 class="doc-prop-wrapper" id="labelwidth" data-Path="labelwidth-labelWidth">
<a href="#labelwidth" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>labelWidth</span>
<span class="doc-prop-type"> string
</span>
</h3>

```

Specifies the width of the parameter label. By default, the parameter label width value is set as **110px**.

Defaults to *110px*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings:{ labelWidth: "auto" }
```

```
});
```

```
</script>
```

```

<h3 class="doc-prop-wrapper" id="mindatetime" data-Path="mindatetime-minDateTime">

```

```

<a href="#mindatetime" aria-hidden="true" class="anchor">

```

```

<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">

```

```

<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2 1.22-2 2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>

```

```
</svg>
```

```
</a><span class='doc-prop-name'>minDateTime</span>
```

```
<span class="doc-prop-type"> string
```


</h3>

Specifies the minimum date in the date parameter that the user can select. By default, the `minDateTime` value is set as `null`

Defaults to `null`

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings:{ minDateTime: "3/1/2003" }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="maxdatetime" data-Path="maxdatetime-maxDateTime">
```

```
<a href="#maxdatetime" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>maxDateTime</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Specifies the maximum date in the date parameter that the user can select. By default, the `maxDateTime` value is set as `null`

Defaults to `null`

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
parameterSettings:{ maxDateTime: "4/30/2003" }
});
</script>
`
<h3 class="doc-prop-wrapper" id="hidetooltip" data-Path="hidetooltip-hideTooltip">
<a href="#hidetooltip" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>hideTooltip</span>
<span class="doc-prop-type"> boolean
</span>
</h3>
```

Show or hide the parameter tooltip on report initial rendering.

Defaults to *false*

```
<span style="font-weight:bold">Example</span>
`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer(
{
parameterSettings:{ hideTooltip: true }
});
</script>
`
<h3 class="doc-prop-wrapper" id="hideparameterblock" data-Path="hideparameterblock-hideParameterBlock">
<a href="#hideparameterblock" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-
```

```
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>hideParameterBlock</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Show or hide the parameter block on report initial rendering.

Defaults to *false*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings:{ hideParameterBlock: true }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="datetimeformat" data-Path="datetimeformat-dateTimeFormat">
```

```
<a href="#datetimeformat" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-
.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-
1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2
3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>dateTimeFormat</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Defines the date time format displayed in the `DateTimePicker`. By default, the `dateTimeFormat` value is set as `empty`.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings:{ dateTimeFormat: "d/M/yyyy tt h:mm" }
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="timedisplayformat" data-Path="timedisplayformat-timeDisplayFormat">
```

```
<a href="#timedisplayformat" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>timeDisplayFormat</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Defines the time format displayed in the time dropdown inside the DateTimePicker popup. By default, the `timeDisplayFormat` value is set as `empty`.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings:{ timeDisplayFormat: "HH:mm" }
```

```
});
```

```
</script>
```



```

<h3 class="doc-prop-wrapper" id="timeinterval" data-Path="timeinterval-timeInterval">
<a href="#timeinterval" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>timeInterval</span>
<span class="doc-prop-type"> number
</span>
</h3>

```

Sets the time interval between the two adjacent time values in the `DateTimePicker` time popup. By default, the `timeInterval` value is set as `30`.

Defaults to `30`

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings:{ timeInterval: 60 }
```

```
});
```

```
</script>
```

```

<h3 class="doc-prop-wrapper" id="accessinternalvalue" data-Path="accessinternalvalue-
accessInternalValue">

```

```
<a href="#accessinternalvalue" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```

<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.55S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .55S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>

```

```
</svg>
```

```
</a><span class='doc-prop-name'>accessInternalValue</span>
```

```
<span class="doc-prop-type"> boolean
```

```
</span>
```

```
</h3>
```

Specifies whether the hidden or internal report parameters can be exposed to user or not. By default, the value is set as `false` and only the public visibility parameters are exposed outside.

Defaults to *false*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
parameterSettings:{ accessInternalValue: true }
```

```
});
```

```
</script>
```

```
,
```

dataSources `array`

Properties

```
<h3 class="doc-prop-wrapper" id="name" data-Path="name-name">
```

```
<a href="#name" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>name</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Gets or sets the name of the data source.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```

`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer({ dataSources: [name:"Menu Items",
value:[{ OrderId: "21D60", FoodName: "Burger", Price: 20, Category: "Veg" },
{ OrderId: "21D61", FoodName: "Pizza", Price: 25, Category: "Non-Veg" },
{ OrderId: "21D63", FoodName: "Sandwiches", Price: 30, Category: "Non-Veg" },
{ OrderId: "21D65", FoodName: "Chicken Drum Sticks", Price: 23, Category: "Non-Veg" },
{ OrderId: "21D64", FoodName: "Fulka", Price: 15, Category: "Veg" }]]
});
</script>
`

<h3 class="doc-prop-wrapper" id="value" data-Path="value-value">
<a href="#value" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>value</span>
<span class="doc-prop-type"> array
</span>
</h3>

```

Gets or sets the values of data source.

Defaults to `[]`

`Example`

```

`js
<div id="reportviewer"></div>
<script>
$("#reportviewer").boldReportViewer({ dataSources: [name:"Menu Items",
value:[{ OrderId: "21D60", FoodName: "Burger", Price: 20, Category: "Veg" },
{ OrderId: "21D61", FoodName: "Pizza", Price: 25, Category: "Non-Veg" },
{ OrderId: "21D63", FoodName: "Sandwiches", Price: 30, Category: "Non-Veg" },

```

```
{ OrderId: "21D65", FoodName: "Chicken Drum Sticks", Price: 23, Category: "Non-Veg" },
{ OrderId: "21D64", FoodName: "Fulka", Price: 15, Category: "Veg" }]]
});
</script>
`
```

customBrandSettings

Properties

```
<h3 class="doc-prop-wrapper" id="hidehelplink" data-Path="hidehelplink-hidehelplink">
<a href="#hidehelplink" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91.72-2 .25V8.59c-.58-.45-1-1.27-1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
</svg>
</a><span class='doc-prop-name'>link</span>
<span class="doc-prop-type"> boolean
</span>
</h3>
```

Shows or hides the link of customBrandSettings.

Defaults to *false*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
{
customBrandSettings:{ hidehelplink: false }
});
```

```
</script>
```

```
`
```

```
<h3 class="doc-prop-wrapper" id="customdomain" data-Path="customdomain-customdomain">
<a href="#customdomain" aria-hidden="true" class="anchor">
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>domain</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Gets or sets the domain of the customBrandSettings.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
customBrandSettings:{
```

```
customdomain: 'https://help.boldreports.com'
```

```
}
```

```
});
```

```
</script>
```

```
`
```

```
<h3 class="doc-prop-wrapper" id="custombrandname" data-Path="custombrandname-custombrandname">
```

```
<a href="#custombrandname" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>name</span>
```

```
<span class="doc-prop-type"> string
```

```
</span>
```

```
</h3>
```

Gets or sets the name of the customBrandSettings.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer({
```

```
customBrandSettings:{
```

```
custombrandname: 'Bold Reports'
```

```
}
```

```
});
```

```
</script>
```

```
,
```

```
<h3 class="doc-prop-wrapper" id="customlinks" data-Path="customlinks-customlinks">
```

```
<a href="#customlinks" aria-hidden="true" class="anchor">
```

```
<svg aria-hidden="true" height="16" version="1.1" viewBox="0 0 16 16" width="16">
```

```
<path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 .72-2 .25V8.59c-.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 .5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 3h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path>
```

```
</svg>
```

```
</a><span class='doc-prop-name'>links</span>
```

```
<span class="doc-prop-type"> array
```

```
</span>
```

```
</h3>
```

Add the links to the customBrandSettings.

Defaults to *empty*

```
<span style="font-weight:bold">Example</span>
```

```
`js
```

```
<div id="reportviewer"></div>
```

```
<script>
```

```
$("#reportviewer").boldReportViewer(
```

```
{
```

```
customBrandSettings:{
```

```

customlinks: [{
name: 'ESLicenseMessage',
url: '/licensing/license-token/',
}]
}
});
</script>
`

```

Frequently asked questions

This section helps to get the answer for the frequently asked questions in Bold Reports JavaScript Report Viewer.

1. [How can improve the performance and handle the large amounts of data with Report Viewer?](#)
2. [Is Report Viewer compatible with latest version of JQuery library?](#)
3. [Is the back action from drillthrough report will load the report again with Report Viewer?](#)
4. [Is possible to change the culture of the date time parameter?](#)
5. [Is it possible to hide report parameters?](#)
6. [Is it possible to hide the export options in Report Viewer?](#)
7. [Is it possible to load reports providing parameter values at runtime?](#)
8. [How to Add Web Report Viewer to a Vue application?](#)

How can improve the performance and handle the large amounts of data with Report Viewer

You have to use the **EnableVirtualEvaluation** and **DisablePageSplitting** in `ReportViewerController.cs` to handle the larger amount data. This will helpful to process the report with less memory footprint

```

`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.EnableVirtualEvaluation = true;
reportOption.ReportModel.DisablePageSplitting = true;
}
`

```

To improve the loading performance, you are requesting to set **CanGrow** true for necessary textbox alone in report. Because, if you set the **CanGrow** for textboxes and report having larger amount, then viewer will takes time, measure, height, and width for each textbox cells based on amount of data.

See Also

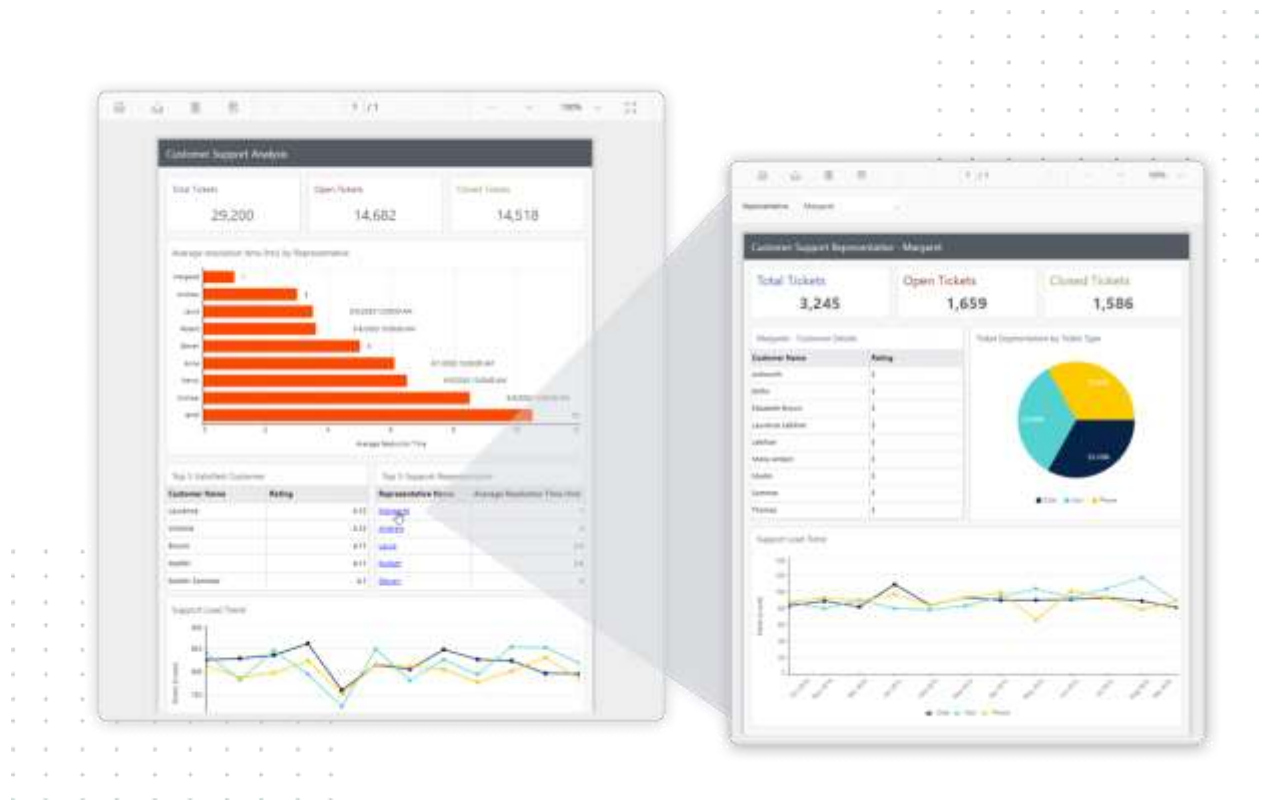
[Render report with huge data in Reportserver](#)

Is Report Viewer compatible with latest version of JQuery library

Yes, Report Viewer is compatible with latest version of JQuery library. You can make use of the latest JQuery library based on your need.

Is the back action from drillthrough report will load the report again with Report Viewer

No, Report Viewer will not load or process the report again when back from drillthrough report.



You can refer the following help documentation for creating the drill through report using the Bold Reports Report Designer.

[Create SSRS drill through report](#)

Is possible to change the culture of the date time parameter

Yes, we can change the culture of the date time parameter for Report Viewer by changing the locale. You can make use the following reference to change the locale of Report Viewer.

[ReportViewer Localization](#)

In Report Viewer, we could not change the culture of specific parameter or UI. So, we have to achieve the requirements only by changing the culture.

Is it possible to hide the parameters in Report Viewer

Yes, it is possible to hide the parameters in Report Viewer. On hiding the parameters, the users will not be able to have the parameter block in the Report Viewer. Refer to this [Hide parameter block and toolbar items](#) section.

Is it possible to hide the export options in Report Viewer

Yes, it is possible to hide the export options in Report Viewer. The Report Viewer provides the `exportOptions` property to show or hide the default export types available in the component. Refer to this [Decide or Hide the export options](#) section.

Is it possible to load reports providing parameter values at runtime

Yes, it is possible to load reports with application inputs as parameters in Report Viewer. You need to provide the input values to the Report Viewer from client side at runtime using the `parameters` property, which accepts JSON array values. Refer to this [Set parameter at client](#) section.

How to Add Web Report Viewer to a Vue application

This section explains the steps required to Add web Report Viewer to a Vue application.

Prerequisites

Before getting started with the bold report viewer, make sure that your development environment includes the following commands.

- [Node JS](#) (version 8.x or 10.x)
- [NPM](#) (v3.x.x or higher)

Install the Vue CLI

Vue provides the easiest way to set Vue CLI projects using the [Vue CLI](#) tool. To install the CLI application globally to your machine, run the following command in the Command Prompt.

```
` typescript
npm install -g @vue/cli
`
```

Create a new Vue application

To create a new Vue application, run the following command in Command Prompt.

```
` typescript
vue create project-name
E.g : vue create report-viewer-app
`
```

The [vue create](#) command prompts you for information about features to include in the initial app project. Accept the defaults by pressing the Enter.

Configure Bold Report Viewer in Vue CLI

To configure the Report Viewer component, change the directory to your application's root folder.

```
` typescript
cd project-name
E.g : cd report-viewer-app
`
```

Run the following commands to install the Bold Reporting packages are distributed in npm as [@boldreports/javascript-reporting-controls](https://www.npmjs.com/package/@boldreports/javascript-reporting-controls).

```
` typescript
npm install @boldreports/javascript-reporting-controls
`
```

Adding Report Viewer component

Hence create a file named 'ReportViewer.vue' inside the src folder

```
` js
<template>
<div id="viewer"></div>
</template>
<script>
import $ from 'jquery';
export default {
name: 'ReportViewer',
mounted()
{
$("#viewer").boldReportViewer({
reportServiceUrl:"https://demos.boldreports.com/services/api/ReportViewer",
reportPath:'~/Resources/docs/sales-order-detail.rdl'
});
}
}
</script>
`
```

Adding scripts reference

Bold Reports needs 'window.jQuery' object to render the Vue components. Hence create a file named 'globals.js' inside the src folder and import jQuery in 'globals.js' file as like the below code snippet.

```
` js
import jquery from 'jquery';
```

```
window['jQuery'] = jquery;  
window['$'] = jquery;  
`
```

Refer the `globals.js` file in `main.js` file as like below code snippet.

The Bold Report Viewer script and style files need to be imported, in order to run the web Report Viewer.

```
` js  
import './globals'  
import Vue from 'vue'  
import ReportViewer from './ReportViewer.vue'  
//Report Viewer source  
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';  
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-designer.min';  
//Data-Visualization  
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';  
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';  
Vue.config.productionTip = false  
new Vue({  
  render:h => h(ReportViewer),  
}).$mount('#viewer')  
`
```

Hence create a file named 'style.css' inside the public folder

```
` css  
html {  
  height: 100%;  
}  
body {  
  height: 100%;  
  margin: 0;  
  width: 100%;  
  overflow: hidden;  
}  
ej-sample,
```

```
ej-reportdesigner {  
  display: block;  
  height: inherit;  
  width: inherit;  
}  
.splash {  
  position: absolute;  
  left: 50%;  
  top: 50%;  
  transform: translateY(-50%) translateX(-50%);  
}  
.message {  
  text-align: center;  
  font-size: 40px;  
  margin-bottom: 20px;  
}  
.loader {  
  margin: auto;  
  width: 70px;  
  text-align: center;  
  position: relative;  
}  
.loader>div {  
  width: 18px;  
  height: 18px;  
  background-color: black;  
  border-radius: 100%;  
  display: inline-block;  
  -webkit-animation: sk-bouncedelay 1.4s infinite ease-in-out both;  
  animation: sk-bouncedelay 1.4s infinite ease-in-out both;  
}  
.loader .bounce1 {  
  -webkit-animation-delay: -0.32s;  
  animation-delay: -0.32s;  
}
```

```

animation-delay: -0.32s;
}
.loader .bounce2 {
-webkit-animation-delay: -0.16s;
animation-delay: -0.16s;
}
@-webkit-keyframes sk-bouncedelay {
0%, 80%, 100% {
-webkit-transform: scale(0);
}
40% {
-webkit-transform: scale(1);
}
}
@keyframes sk-bouncedelay {
0%, 80%, 100% {
-webkit-transform: scale(0);
transform: scale(0);
}
40% {
-webkit-transform: scale(1);
transform: scale(1);
}
}
`

```

Open the 'public\index.html' and refer the following scripts in <head> tag.

` html

```

<link href="style.css" rel="stylesheet">
<!-- Data-Visualization -->
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reportdesigner.min.css"
rel="stylesheet" />
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>

```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
```

Add the following code in the <body> tag.

```
` html
<div class="splash">
<div class="message">Vue.js Reports</div>
<div class="loader">
<div class="bounce1"></div>
<div class="bounce2"></div>
<div class="bounce3"></div>
</div>
</div>
<div id="viewer"></div>
<!-- built files will be auto injected -->
<script>
document.addEventListener('DOMContentLoaded', function () {
document.getElementsByClassName("splash")[0].style.display = "None";
});
</script>
```

Run the Application

To run the application, run the following command in command prompt.

```
` typescript
npm run serve
```

Bold reporting tools for React

Enterprise-class reporting tools for React development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your React applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

Bold reporting tools for React

Enterprise-class reporting tools for React development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your React applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

System Requirements

This topic describes the software requirements for setting up the development environment of Bold Reports React.

Supported Operating Systems

- Windows 7+, 8+
- Windows Server 2008 R2+

Development Environments

The following development environments are necessary to run the Bold Reports React.

- [Microsoft Visual Studio Code](#)
- [Node JS](#) (version 8.x or 10.x)
- [NPM](#) (v3.x.x or higher)

Browser Compatibility

- IE 9+

- Microsoft Edge
- Mozilla Firefox 22+
- Chrome 17+
- Opera 12+
- Safari 5+

See Also

- [Licensing procedure for deployment](#)

Overview

The Report Viewer is a visualization control used to display SSRS, RDL, RDLC, and Bold Report Server reports within web applications. It allows you to view RDL/RDLC reports with or without using SSRS or Bold Report Server. You can bind data sources, parameters, and render reports with all major capabilities of RDL reporting and export the report to PDF, Excel, CSV, Word, PowerPoint, and HTML formats. Some of the key features are:

- Renders interactive reports with drill down, drill through, hyperlinks, and interactive sorting.
- Easily customize each element of the Report Viewer and provide events for report processing customization.
- Supports jQuery, Angular, React, Blazor, ASP.NET Core, ASP.NET MVC, ASP.NET WebForms, WPF, and UWP.

Introducing the newly redesigned Report Viewer! We have given it a refreshing new look that embraces sleek and modern design, aligning perfectly with the latest web design trends. It allows seamless integration into your application. For detailed guidance on the migration process, we recommend you to refer our [Report Viewer v2.0 migration document](#).

Add Web Report Viewer to a React application

This section explains the steps required to add a web Report Viewer to a React application.

To get start quickly with Report Viewer, you can check on this video:

youtube: <https://youtu.be/VXEL50MN85M>

Prerequisites

Before getting started with the bold report viewer, make sure that your development environment includes the following commands.

- [Node JS](#) (version 8.x or 10.x)
- [NPM](#) (v3.x.x or higher)

Install the Create React App package

Create React App is a simple way to create a single-page React application which provides a build setup with no configuration. To install the Create React App globally in your machine, run the following command in the Command Prompt.

```
`typescript
```



```
npm install create-react-app -g
```

```
,
```

To learn more about Create React App package, refer [here](#).

Create a new React application

To create a new React application, run the following command in the Command Prompt.

```
`typescript
```

```
create-react-app reports
```

```
,
```

The `create-react-app` command adds the `react`, `react-dom`, `react-scripts`, and other dependencies required for your React application.

The source code for this React reporting components app is available on [GitHub](#).

Install the Create React Class

To configure the Report Viewer component, change the directory to your application's root folder.

```
`typescript
```

```
cd reports
```

```
,
```

To install the type definitions for `create-react-class`, run the following command in the Command Prompt.

```
`typescript
```

```
npm install create-react-class --save
```

```
,
```

Install the Bold Reports React package

To install the Bold Reports React package, run the following command in the Command Prompt.

```
`typescript
```

```
npm install @boldreports/react-reporting-components --save
```

```
,
```

Adding scripts reference

Bold Reports needs `window.jQuery` object to render the React components. Hence, create a file named `globals.js` in the `src` folder and import `jQuery` into the `globals.js` file, like in the below code snippet.

```
`typescript
```

```
import jquery from 'jquery';
```

```
import React from 'react';
```

```
import createReactClass from 'create-react-class';
```

```
import ReactDOM from 'react-dom';
```

```

window.React = React;
window.createReactClass = createReactClass;
window.ReactDOM = ReactDOM;
window.$ = window.jQuery = jquery;
`

```

Refer to the `globals.js` file in the `index.js` file, like in the below code snippet.

```

`typescript
import './globals'
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
ReactDOM.render(<App />, document.getElementById('root'));
`

```

Adding Report Viewer component

The Bold Report Viewer script and style files need to be imported in order to run the Web Report Viewer. Hence, import the following files into the `App.js` file.

```

`typescript
/ eslint-disable /
import React from 'react';
import './App.css';
//Report Viewer source
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
import '@boldreports/javascript-reporting-controls/Content/material/bold.reports.all.min.css';
//Data-Visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
var viewerStyle = {'height': '700px', 'width': '100%'};
function App() {
  return (
    <div style={viewerStyle}>
      <BoldReportViewerComponent

```

```

id="reportviewer-container">
</BoldReportViewerComponent>
</div>
);
}
export default App;
`

```

Open the `public\index.html` and refer the following scripts in the `<head>` tag.

```

`html
<!-- Data-Visualization -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
`

```

Create a Web API service

The Web Report Viewer requires a Web API service to process the data and file actions. You can skip this step and use our online [Web API services](#) to create, edit, and browse reports or you must create any one of the following Web API services.

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

If you are looking to load the report directly from SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#).

Set a Web API service URL

To set a Web API service, open the `App.js` file and replace the existing code with the below code snippet.

```

`javascript
/ eslint-disable /
import React from 'react';
import './App.css';

```

```
//Report Viewer source
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
import '@boldreports/javascript-reporting-controls/Content/material/bold.reports.all.min.css';
//Data-Visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
var viewerStyle = {'height': '700px', 'width': '100%'};
function App() {
  return (
    <div style={viewerStyle}>
      <BoldReportViewerComponent
        id="reportviewer-container"
        reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
        reportPath = {'~/Resources/docs/sales-order-detail.rdl'} >
      </BoldReportViewerComponent>
    </div>
  );
}
export default App;
`
```

In the above code, the `reportServiceUrl` is used from online URL. You can host the Bold Reports service at any Azure, AWS, or own domain URL and use it in the Report Viewer. You can view the already created Web API service from the [Reporting Service](#) git hub location.

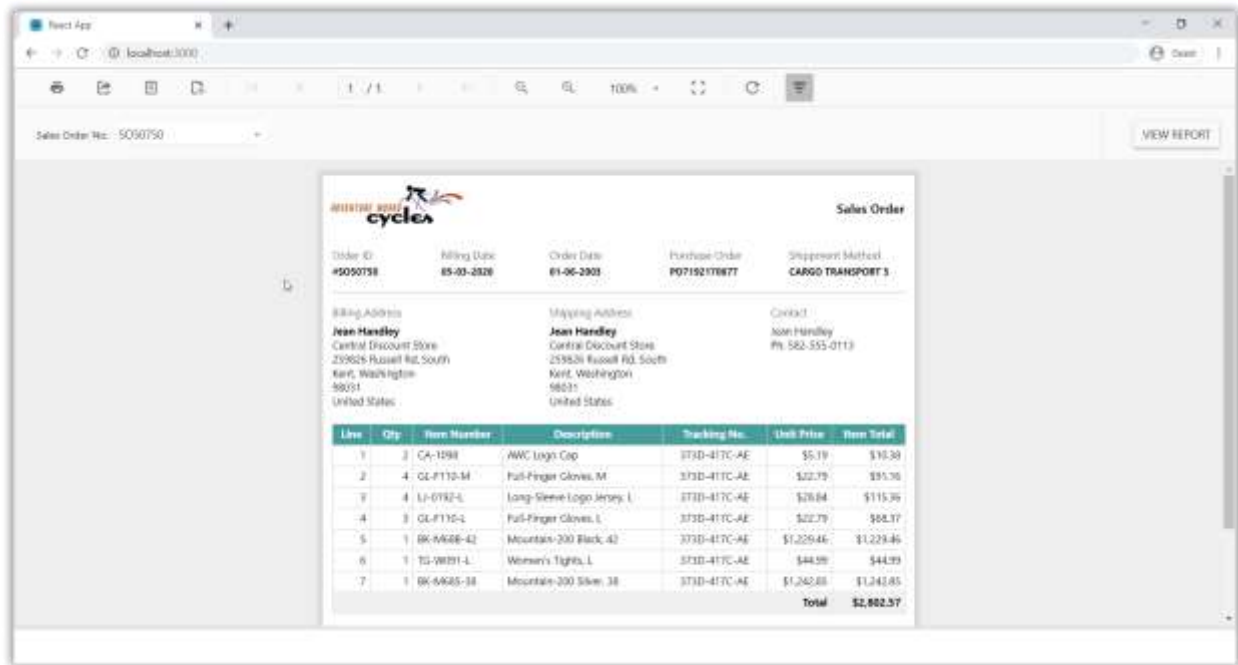
Run the Application

To run the application, run the following command at the command prompt.

```
`typescript
npm run start
`
```

While running the above command, if you are getting an error like `'BoldReportViewerComponent' is not defined` then you need to disable the `eslint` by adding the line `/ eslint-disable /` at the top of [App.js](#).

The `npm run start` command automatically opens your browser to `http://localhost:3000/`.



Deploying the application in production

To deploy the application in production, run the following command in the command prompt, which will create a folder named **build** to generate the build files.

```
`typescript
npm run build
`
```

Note: You can refer to our feature tour page for the [React Report Viewer](#) to see its innovative features. Additionally, you can view our [React Report Viewer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

Add Web Report Viewer with Typescript React application

This section explains the steps required to add a web Report Viewer to a React Typescript application.

Prerequisites

Before getting started with a bold web report viewer, make sure your development environment includes the following.

- [Node JS](#) (version 8.x or 10.x)
- [NPM](#) (v3.x.x or higher)

Install the Create React App Package

Create React App is a simple way to create a single-page React application which provides a build setup with no configuration. To install the Create React App globally on your machine, run the following command in Command Prompt.

```
`typescript
```

```
npm install create-react-app -g
```

```
,
```

To learn more about Create React App package, refer [here](#).

Create a new React Typescript Application

To create a new React typescript application, run the below command in the Command Prompt.

```
`typescript
```

```
create-react-app reports --template typescript
```

```
,
```

The `create-react-app` command adds the `react`, `react-dom`, `react-scripts`, and other dependencies required to your react typescript application.

Install Create React Class

To configure the Report Viewer component, change the directory to your application's root folder.

```
`typescript
```

```
cd reports
```

```
,
```

To install the type definitions for create-react-class run the following command in the Command Prompt.

```
`typescript
```

```
npm install create-react-class --save
```

```
npm install @types/create-react-class --save
```

```
,
```

Install Bold Reports React package

To install the Bold Reports React package run the following command in the Command Prompt.

```
`typescript
```

```
npm install @boldreports/react-reporting-components --save
```

```
,
```

Install JQuery package

To install the JQuery package run the following command in the Command Prompt.

```
`typescript
```

```
npm install @types/jquery --save
```

```
,
```

Adding Scripts reference

Bold Reports needs `window.jQuery` object to render the React components. Hence, create a file named `globals.ts` in the `src` folder and import jQuery into the `globals.ts` file, like in the below code snippet.

```
`typescript
import jquery from 'jquery';
import React from 'react';
import createReactClass from 'create-react-class';
import ReactDOM from 'react-dom';
(window as any).React = React;
(window as any).createReactClass = createReactClass;
(window as any).ReactDOM = ReactDOM;
(window as any).$ = (window as any).jQuery = jquery;
`
```

Refer to the `globals.ts` file in the `index.tsx` file, like in the below code snippet.

```
`typescript
import './globals';
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
ReactDOM.render(<App />, document.getElementById('root'));
`
```

Adding Report Viewer component

The web Report Viewer script and style files need to be imported in order for the web Report Viewer to run. Hence, import the following files into the `App.tsx` file.

```
`typescript
/ eslint-disable /
import React from 'react';
import './App.css';
//Report designer source
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
import '@boldreports/javascript-reporting-controls/Content/material/bold.reports.all.min.css';
//Data-Visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
```

```

declare let BoldReportViewerComponent: any;
var viewerStyle = {
  'height': '700px',
  'width': '100%'
};
function App() {
  return (
    <div style={viewerStyle}>
      <BoldReportViewerComponent
        id="reportviewer-container">
      </BoldReportViewerComponent>
    </div>
  );
}
export default App;
`

```

Open the `public\index.html` and refer the following scripts in the `<head>` tag.

```

`html
<!-- Data-Visualization -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
`

```

Create a Web API service

The Web Report Viewer requires a Web API service to process the data and file actions. You can skip this step and use our online [Web API services](#) to create, edit, and browse reports or you must create any one of the following Web API services.

- [ASP.NET Web API Service](#)

- [ASP.NET Core Web API Service](#)

If you are looking to load the report directly from the SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#).

Set Web API service URL

To set Web API service, open the `app.component.ts` file and add the code snippet as in the constructor.

```
`javascript
/ eslint-disable /
import React from 'react';
import './App.css';
//Report designer source
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
import '@boldreports/javascript-reporting-controls/Content/material/bold.reports.all.min.css';
//Data-Visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
declare let BoldReportViewerComponent: any;
var viewerStyle = {
  'height': '700px',
  'width': '100%'
};
function App() {
  return (
    <div style={viewerStyle}>
      <BoldReportViewerComponent
        id="reportviewer-container"
        reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
        reportPath = {'~/Resources/docs/sales-order-detail.rdl'} >
      </BoldReportViewerComponent>
    </div>
  );
};
```

```

}
export default App;
`

```

Run the Application

To run the application, run the following command at the command prompt.

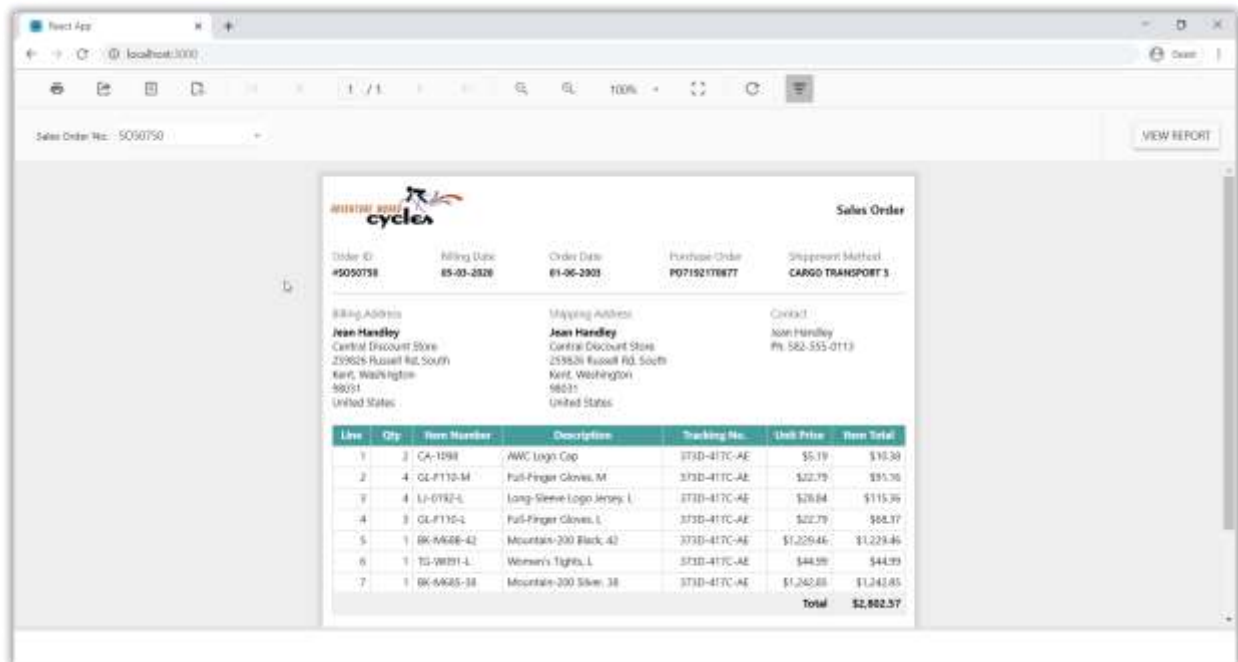
```

`typescript
npm run start
`

```

While running the above command, if you are getting an error like 'BoldReportViewerComponent' is not defined then you need to declare the BoldReportViewerComponent by adding the line declare let BoldReportViewerComponent: any after import section in the [App.tsx](#).

The npm run start command automatically opens your browser to <http://localhost:3000/>.



Deploying the application in production

To deploy the application in production, we need to generate the build files. Hence to generate the build files, run the below command at the command prompt which will create a folder named **build**.

```

`typescript
npm run build
`

```

Load SSRS Report Server reports

Report Viewer has support to load RDL reports from SSRS Report Server. To render SSRS Reports, set the `reportServerUrl`, `reportPath` and `reportServiceUrl` properties as shown in the following code snippet in `App.js` or `app.component.ts` file.

1. To create your first React reporting application, refer to the [Getting-started](#) section.

If you need to know about the difference between `reportServiceUrl` and `reportServerUrl`, then refer to [Difference between Report Service URL and Report Server URL](#).

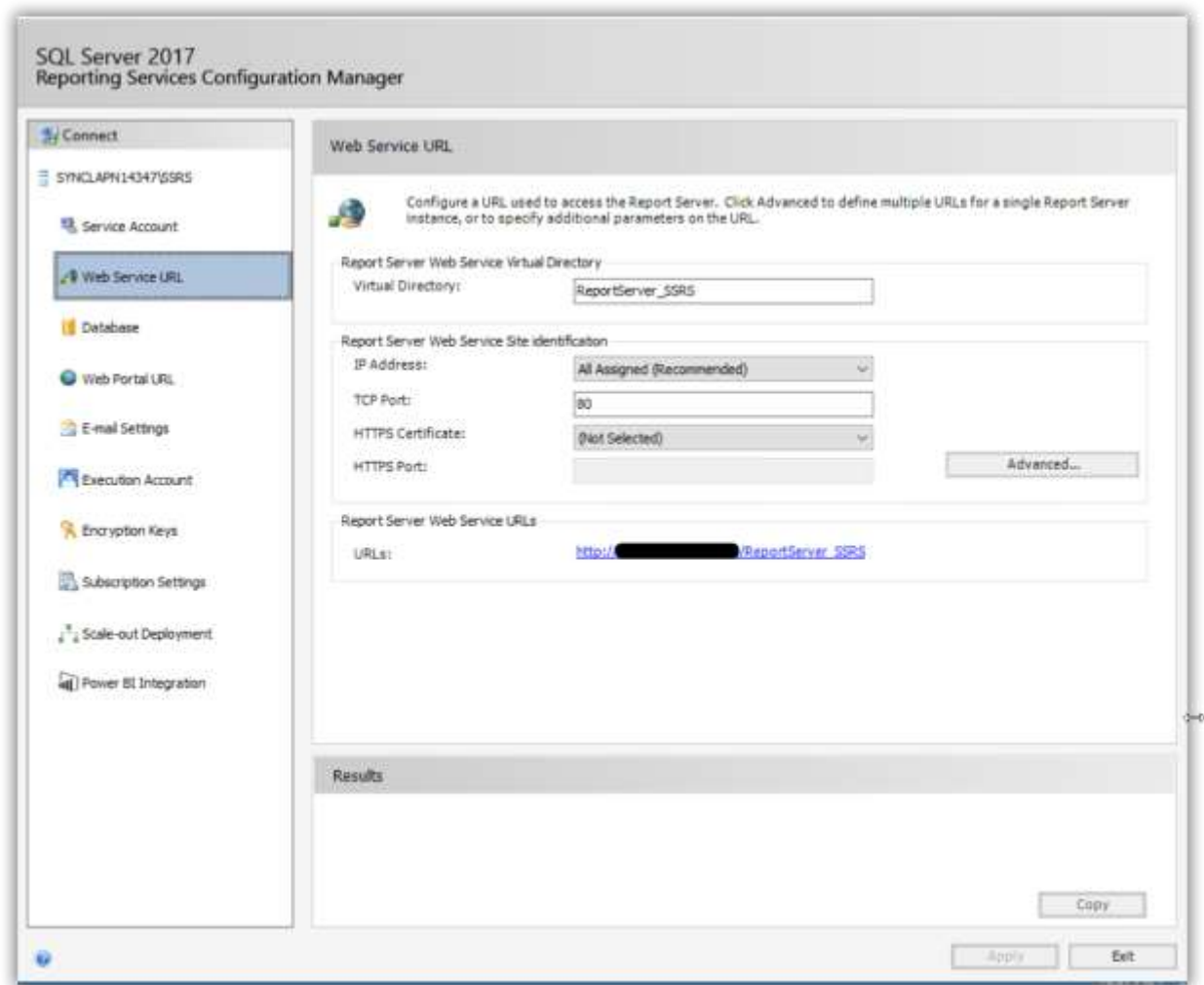
2. Set the `reportServerUrl` API on Bold Report Viewer with `WebServiceURL`. Open the `App.js` or `app.component.ts` and replace the following code example.

```
`javascript
function App() {
  return (<div id="viewer" style={viewerStyle}>
    <BoldReportViewerComponent id="reportviewer-container"
      reportServiceUrl = {'https://localhost:7451/api/Report Viewer'}
      reportServerUrl = {'http://<servername>/Reports_SSRS'}

    </BoldReportViewerComponent>
  </div>);
}
```

The Web Service URL should be set as `reportServerUrl` in the report viewer configuration. The Web Service URL can be found from the Reporting Services Configuration manager under the `Web Service`

URL section, as shown in the following image.



3. Set the report path for loading the reports from the SSRS Report Server. The report path should be in the format of `/folder name/report name`. Open the `App.js` or `app.component.ts` and replace the following code example.

```
`javascript
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/Report Viewer'}
reportPath = {'/SSRSSamples2/Territory Sales_Link'}
reportServerUrl = {'http://<servername>/reportserver$instanceName'}

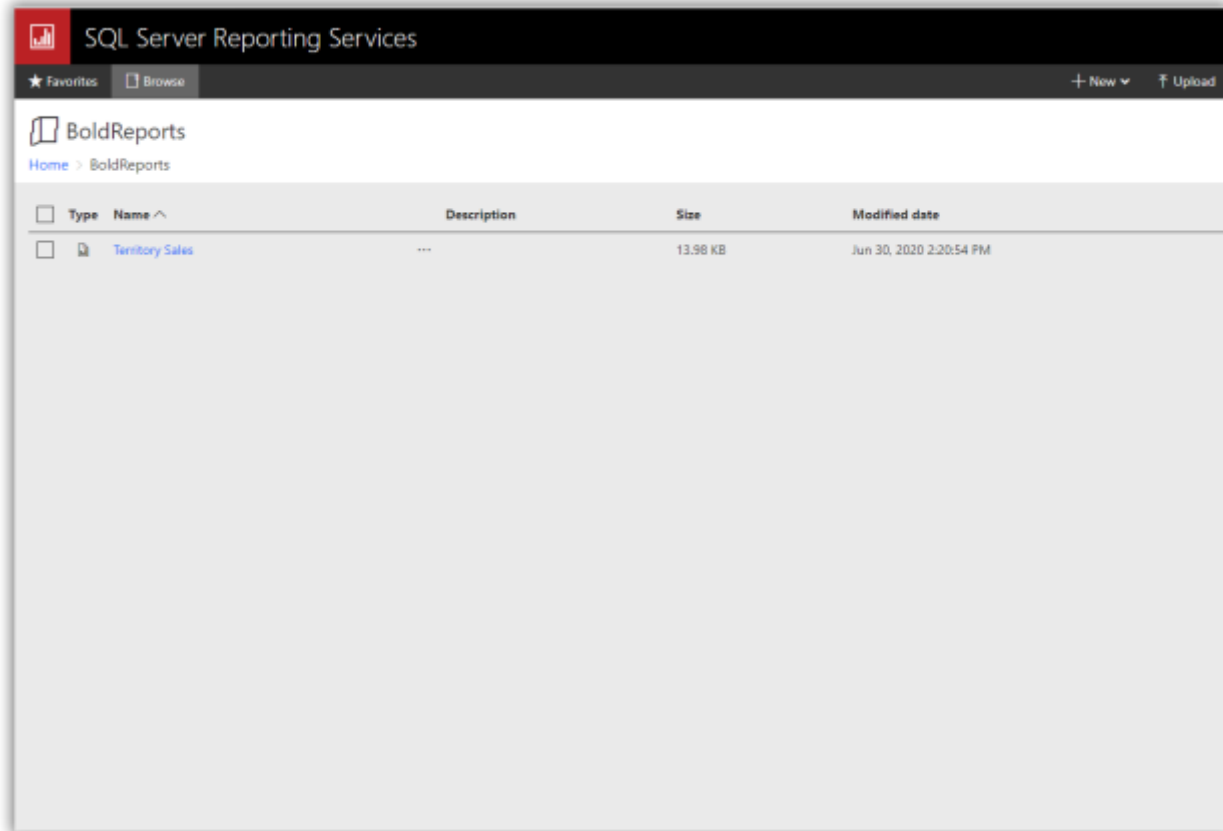
</BoldReportViewerComponent>
```

```
</div>);
```

```
}
```

```
,
```

The report path can be found from the SSRS Report Server by navigating to the path of the report to be loaded, as shown in the following image.



Network credentials for SSRS

The network credentials are required to connect with the specified SSRS Report Server using the Report Viewer. Specify the `ReportServerCredential` property in the Web API Controller `OnInitReportOptions` method.

```
`csharp
```

```
[NonAction]
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
//Add SSRS Report Server credential
```

```
reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",  
"RDLReport1");
```

```
}
```

```
,
```

If you are facing problem to access the SSRS Report server reports, you can refer [How to provide the permission for user to access the SSRS Report Server reports](#).

Set data source credential for shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS server. If the report has any data source that uses credentials to connect with the database, then you should specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server and data source credentials
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
    "RDLReport1");
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "<username>", "<password>"));
}
`
```

Data source credentials should be added to the shared data sources that do not have credentials in the connection strings.

Change data source connection string

You can change the connection string of a report data source before it is loaded in the Report Viewer. The `DataSourceCredentials` class provides the option to set and update the modified connection string as in the following code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "<username>", "<password>", "Data
    Source=<instancename>;Initial Catalog=<database>;"));
}
`
```

The previous code shows an option to change the connection string only, but the class provides multiple options to change data source information. To learn more about this, refer to this `DataSourceCredentials` class.

See also

[Does Bold Report Viewer use SSRS Report processing?](#)

Add Web Report Viewer to a React Boilerplate application

This section explains the steps required to add a web [Bold Report Viewer](#) to a [React Boilerplate application](#).

Prerequisites

Before getting started with the report viewer, make sure that you have the following requirements.

- [Node JS](#) (version 8.x or 10.x)
- [NPM](#) (v3.x.x or higher)

To quick start with the React Boilerplate application, we have already configured our [Bold Reports with React Boilerplate application](#). Execute the following commands to get started with the Bold Reports React Boilerplate application right away.

```
`bash
git clone https://github.com/boldreports/react-boilerplate.git
cd react-boilerplate
npm install
npm start
`
```

React Boilerplate Application

- Download the React Boilerplate application from this [link](#) and extract it.
- From the extracted folder, execute the following command to install the project dependencies.

```
`bash
npm install
`
```

Install the Bold Reports React package

To install the [Bold Reports React](#) package, run the following command at the command prompt.

```
`bash
npm install @boldreports/react-reporting-components --save-dev
`
```

Also, install the [create-react-class](#) package, which is required by the Bold Reports React package.

```
`bash
npm install create-react-class --save-dev
`
```

Adding global scripts reference

- Since, Bold Reports requires global `jquery`, `React`, `createReactClass`, and `ReactDOM` objects to render the Report components, we need to import and assign those to the `window` object in a newly created `app/globals.js` file.

```
`typescript
import jquery from 'jquery';
import React from 'react';
import createReactClass from 'create-react-class';
import ReactDOM from 'react-dom';
window.React = React;
window.createReactClass = createReactClass;
window.ReactDOM = ReactDOM;
window.$ = window.jQuery = jquery;
`
```

- Import the `globals.js` file into the `app/app.js` file, like in the below code snippet.

```
`javascript
/

  • app.js *
  • This is the entry file for the application, only setup and boilerplate
  • code.

*/
import './globals';
// Needed for redux-saga es6 generator support
import '@babel/polyfill';
// Import all the third party stuff
....
....
`
```

Configuring webpack

Since, Bold Reports uses `cur` type files, we need to provide support to load such type of file in the webpack `url-loader` plugin by configuring `internals/webpack/webpack.base.babel.js` file.

```
`javascript
```



```
....
....
test: /\. (jpg|png|gif|cur)$/ ,
use: [
{
  loader: 'url-loader',
  options: {
    // Inline files smaller than 10 kB
    limit: 10 * 1024,
  },
}
....
....
\
```

Adding Report Viewer component

The Bold Report Viewer **script** and **style** files need to be imported in order to run the web viewer. So, import the following scripts and css into the **app/app.js** file. Now, use the tag **BoldReportViewerComponent** to render our viewer in the application.

```
`javascript
....
....
import history from 'utils/history';
import 'sanitize.css/sanitize.css';
//Report Viewer source
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
import '@boldreports/javascript-reporting-controls/Content/material/bold.reports.all.min.css';
//Data-Visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
....
....
const MOUNT_NODE = document.getElementById('app');
```

```
const bounds = { height: '800px', width: '100%'};
const render = messages => {
  ReactDOM.render(
    <div style={bounds}>
      <BoldReportViewerComponent id="reportviewer-container" >
    </BoldReportViewerComponent>
    </div>,
    MOUNT_NODE,
  );
};
....
....
`
```

Open the `app\index.html` and refer the following scripts in the `<head>` tag.

```
`html
<!-- Data-Visualization -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
`
```

Create a Web API service

The Web Report Viewer requires a Web API service to process the data and file actions. You can skip this step and use our online [Web API services](#) to browse reports or you must create any one of the following Web API services.

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

If you are looking to load the report directly from the SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#).

Set a Web API service URL

Bind an online `reportServiceUrl` to the Bold Report viewer component in the `app/app.js` file, like in the below snippet.

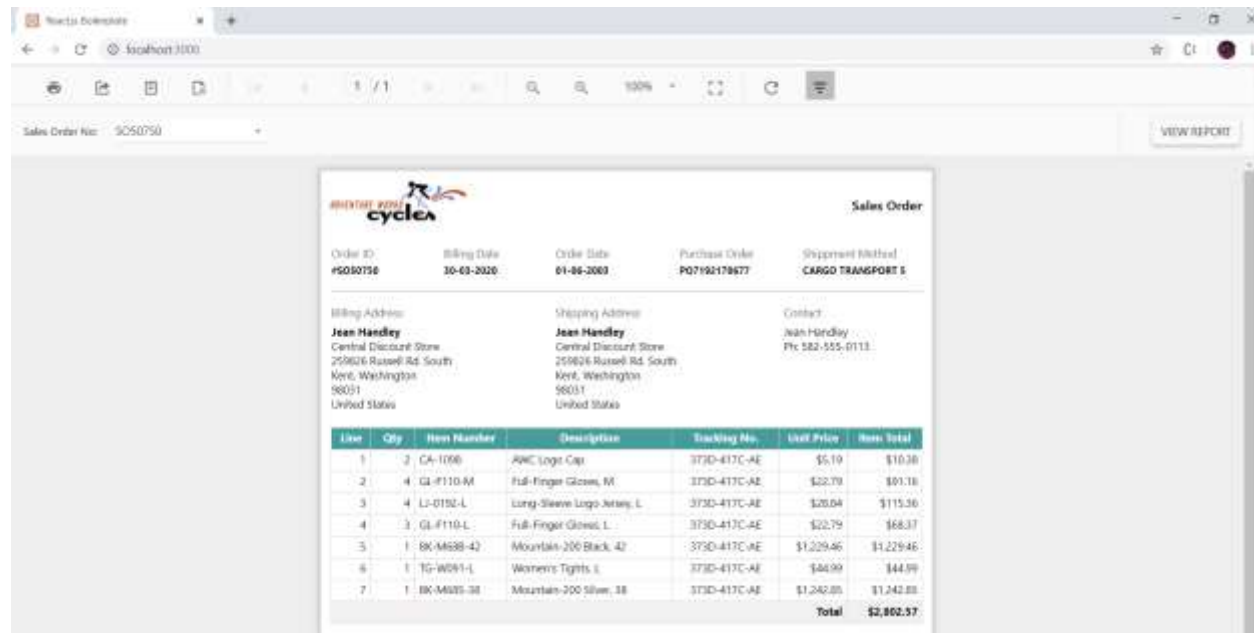
```
`javascript
....
....
import history from 'utils/history';
import 'sanitize.css/sanitize.css';
//Report Viewer source
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
import '@boldreports/javascript-reporting-controls/Content/material/bold.reports.all.min.css';
//Data-Visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
....
....
const MOUNT_NODE = document.getElementById('app');
const bounds = { height: '800px', width: '100%' };
const render = messages => {
  ReactDOM.render(
    <div style={bounds}>
      <BoldReportViewerComponent
        id="reportviewer-container"
        reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
        reportPath = {'~/Resources/docs/sales-order-detail.rdl'} >
      </BoldReportViewerComponent>
    </div>,
    MOUNT_NODE,
  );
};
....
....
```

Run the Application

To run the app, execute the following command and browse to <http://localhost:3000> to see the application.

```
`bash
```

```
npm start
```



Deploying the application in production

To deploy the application in production, run the following command at the command prompt, which will create a folder named `build` to generate the build files.

```
`bash
```

```
npm run build
```

Add Web Report Viewer to a React Boilerplate application

This section explains the steps required to add a web [Report Viewer](#) to a [React Boilerplate TypeScript application](#).

Prerequisites

Before getting started with the report viewer, make sure that you have the following requirements.

- [Node JS](#) (version 8.x or 10.x)
- [NPM](#) (v3.x.x or higher)

To quick start with React Boilerplate application, we have already configured our [Bold Reports with React Boilerplate TypeScript application](#). Execute the following commands to get started with the Bold Reports React Boilerplate TypeScript application.

```
`bash
git clone https://github.com/boldreports/react-boilerplate-typescript.git
cd react-boilerplate
npm install
npm start
`
```

React Boilerplate TypeScript Application

- Download the React Boilerplate application from this [link](#) and extract it.
- From the extracted folder, execute the following command to install project dependencies.

```
`bash
npm install
`
```

Install the Bold Reports React package

To install the [Bold Reports React](#) package, run the following command at the command prompt.

```
`bash
npm install @boldreports/react-reporting-components --save-dev
`
```

Also, install the `create-react-class` package, which is required by the Bold Reports React package.

```
`bash
npm install create-react-class --save-dev
`
```

Adding global scripts reference

- Since, Bold Reports requires global `jquery`, `React`, `createReactClass`, and `ReactDOM` objects to render the Report components, we need to import and assign those to the `window` object in a newly created `app/globals.ts` file.

```
`typescript
import jquery from 'jquery';
import React from 'react';
import createReactClass from 'create-react-class';
import ReactDOM from 'react-dom';
```

```
(window as any).React = React;
(window as any).createReactClass = createReactClass;
(window as any).ReactDOM = ReactDOM;
(window as any).$ = (window as any).jQuery = jquery;
`
```

- Import the `globals.ts` file into the `app/app.tsx` file, like in the below code snippet.

```
`javascript
/

  • app.tsx *
  • This is the entry file for the application, only setup and boilerplate
  • code.

*/
import './globals';
// Needed for redux-saga es6 generator support
import 'react-app-polyfill/ie11';
import 'react-app-polyfill/stable';
// Import all the third party stuff
....
....
`
```

Configuring webpack

- Since, Bold Reports uses `cur` type files, we need to provide support to load such type of file in the webpack `url-loader` plugin by configuring `internals/webpack/webpack.base.babel.js` file.

```
`javascript
....
....
test: /\. (jpg|png|gif|cur)$/,
use: [
{
  loader: 'url-loader',
  options: {
```

```
// Inline files smaller than 10 kB
```

```
limit: 10 * 1024,
```

```
},
```

```
}
```

```
....
```

```
....
```

```
,
```

- Also, change the image quality in `image-webpack-loader` from `65-90` to `[0.65, 0.90]`.

```
`javascript
```

```
....
```

```
....
```

```
{
```

```
  loader: 'image-webpack-loader',
```

```
  options: {
```

```
    mozjpeg: {
```

```
      enabled: false,
```

```
      // NOTE: mozjpeg is disabled as it causes errors in some Linux environments
```

```
      // Try enabling it in your environment by switching the config to:
```

```
      // enabled: true,
```

```
      // progressive: true,
```

```
    },
```

```
    gifsicle: {
```

```
      interlaced: false,
```

```
    },
```

```
    optipng: {
```

```
      optimizationLevel: 7,
```

```
    },
```

```
    pngquant: {
```

```
      quality: [0.65, 0.90],
```

```
      speed: 4,
```

```
    },
```

```
  },
```

```
}
....
....
`
```

Adding Report Viewer component

The Bold Report Designer script and style files need to be imported in order to run the web viewer. So, import the following scripts and css into the `app/app.tsx` file. Now, use the tag `BoldReportViewerComponent` to render our viewer in the application.

```
`javascript
....
....
import history from 'utils/history';
import 'sanitize.css/sanitize.css';
//Report Viewer source
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
import '@boldreports/javascript-reporting-controls/Content/material/bold.reports.all.min.css';
//Data-Visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
....
....
declare let BoldReportViewerComponent: any;
const MOUNT_NODE = document.getElementById('app') as HTMLElement;
const bounds = { height: '800px', width: '100%'};
const ConnectedApp = (props: { messages: any }) => (
<div style={bounds}>
<BoldReportViewerComponent id="reportviewer-container" >
</BoldReportViewerComponent>
</div>
);
const render = (messages: any) => {
ReactDOM.render(<ConnectedApp messages={messages} />, MOUNT_NODE);
```



```
};
....
....
`
```

Open the `app\index.html` and refer to the following scripts in the `<head>` tag.

```
`html
<!-- Data-Visualization -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
`
```

Create a Web API service

The Web Report Viewer requires a Web API service to process the data and file actions. You can skip this step and use our online [Web API services](#) to browse reports or you must create any one of the following Web API services.

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

If you are looking to load the report directly from the SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#).

Set a Web API service URL

Bind an online `reportServiceUrl` to Bold Report viewer component in the `app/app.tsx` file, like in the below snippet.

```
`javascript
....
....
import history from 'utils/history';
import 'sanitize.css/sanitize.css';
//Report Viewer source
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
```

```

import '@boldreports/javascript-reporting-controls/Content/material/bold.reports.all.min.css';
//Data-Visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
....
....
declare let BoldReportViewerComponent: any;
const MOUNT_NODE = document.getElementById('app') as HTMLElement;
const bounds = { height: '800px', width: '100%' };
const ConnectedApp = (props: { messages: any }) => (
  <div style={bounds}>
    <BoldReportViewerComponent
      id="reportviewer-container"
      reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
      reportPath = {'~/Resources/docs/sales-order-detail.rdl'} >
    </BoldReportViewerComponent>
  </div>
);
const render = (messages: any) => {
  ReactDOM.render(<ConnectedApp messages={messages} />, MOUNT_NODE);
};
....
....
`

```

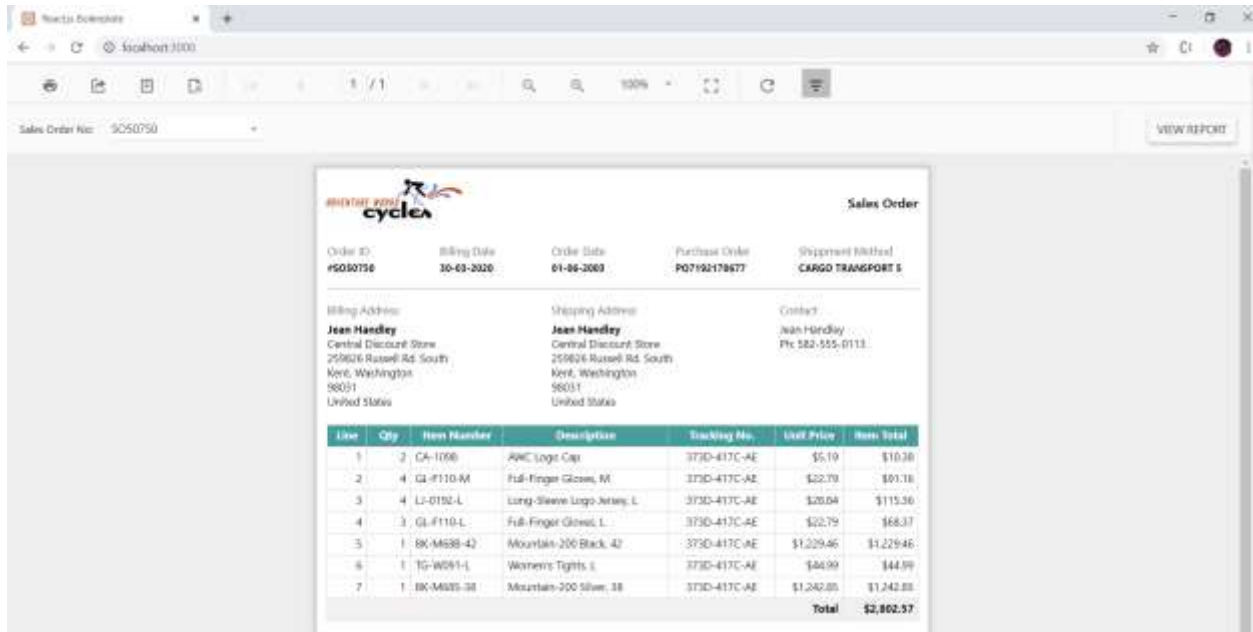
Run the Application

To run the app, execute the following command and browse to <http://localhost:3000> to see the application.

```

`bash
npm start
`

```



Deploying the application in production

To deploy the application in production, run the following command at the command prompt, which will create a folder named `build` to generate the build files.

```
`bash
```

```
npm run build
```

```
`
```

Render RDLC report

The data binding support allows you view the RDLC reports that exist on the local file system with JSON array and custom business object data collection. The following steps demonstrates how to render an RDLC report with JSON array and custom business object data collection.

Add the RDLC report `AreaCharts.rdlc` from the Bold Reports installation location to your application `App_Data` folder. For more information, refer to [Samples and demos](#).

Bind data source at client side

To bind the data source at client side, follow these steps;

- Set the RDLC report path to the `reportPath` property.
- Assign the `processingMode` property to `ProcessingMode.Local`.
- Bind the JSON array collection to the `dataSources` property as shown in following code.

```
`javascript
```

```
var viewerStyle = {'height': '700px', 'width': '100%'};
```

```
var reportPath = 'AreaCharts.rdlc';
```

```
var reportData = [{
```

```

value: [
{ SalesPersonID: 281, FullName: 'Ito', Title: 'Sales Representative', SalesTerritory: 'South West', Y2002: 0,
Y2003: 28000, Y2004: 3018725 },
{ SalesPersonID: 282, FullName: 'Saraiva', Title: 'Sales Representative', SalesTerritory: 'Canada', Y2002:
25000, Y2003: 14000, Y2004: 3189356 },
{ SalesPersonID: 283, FullName: 'Cambell', Title: 'Sales Representative', SalesTerritory: 'North West',
Y2002: 12000, Y2003: 13000, Y2004: 1930885 }
],
name: 'AdventureWorksXMLDataSet'
});
function App() {
return (
<div style={viewerStyle}>
<BoldReportViewerComponent
id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
processingMode = {"Local"}
reportPath = {reportPath}
dataSources = {reportData}>
</BoldReportViewerComponent>
</div>
);
}
export default App;
`

```

- Build and run the application.

Bind data source in Web API controller

The following steps help you configure the Web API to render the RDLC report with business object data collection.

1. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```

`csharp
public class ProductList

```

```
{
public string ProductName { get; set; }
public string OrderId { get; set; }
public double Price { get; set; }
public string Category { get; set; }
public string Ingredients { get; set; }
public string ProductImage { get; set; }
public static IList GetData()
{
List<ProductList> datas = new List<ProductList>();
ProductList data = null;
data = new ProductList()
{
ProductName = "Baked Chicken and Cheese",
OrderId = "323B60",
Price = 55,
Category = "Non-Veg",
Ingredients = "grilled chicken, corn and olives.",
ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
ProductName = "Chicken Delite",
OrderId = "323B61",
Price = 100,
Category = "Non-Veg",
Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
```

```

ProductName = "Chicken Tikka",
OrderId = "323B62",
Price = 64,
Category = "Non-Veg",
Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
ProductImage = ""
};
datas.Add(data);
return datas;
}
}
`

```

2. Set the value of the `ProcessingMode` property to `ProcessingMode.Local` and `ReportPath` in the RDLC report location.
3. Bind the business object data values collection by adding a new item to the `DataSources` as shown in the following code snippet.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ProcessingMode = ProcessingMode.Local;
    reportOption.ReportModel.ReportPath =
    System.Web.Hosting.HostingEnvironment.MapPath(@"~/App_Data/Product List.rdlc");
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
    Value = ProductList.GetData() });
}
`

```

Here the `Name` is case sensitive and it should be same as in the data source name in the report definition.

The `Value` accepts `IList`, `DataSet`, and `DataTable` inputs.

Load report as stream

To load report as a stream, create a report stream using the `FileStream` class and assign the report stream to the `Stream` property.

```

`csharp

```

[NonAction]

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string filePath = System.Web.Hosting.HostingEnvironment.MapPath(@"~/App_Data/Product List.rdlc");
    ;
    // Opens the report from application App_Data folder using FileStream
    FileStream reportStream = new FileStream(filePath, FileMode.Open, FileAccess.Read);
    reportOption.ReportModel.Stream = reportStream;
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
    Value = ProductList.GetData() });
}
`
```

In the previous code, the **Product List.rdlc** report is loaded from the **App_Data** folder location.

[View report click](#)

You can get the user selected parameter details when users clicks the **ViewReport** button in the parameter block. The **viewReportClick** event allows you handle the **ViewReport** button click at client side as shown in the following code.

```
`javascript
var viewerStyle = {'height': '700px', 'width': '100%'};
var reportPath = 'GroupingAgg.rdl';
function viewReportClick(event) {
    var reportParams = [];
    reportParams.push({ name: 'ReportParameter1', labels: ['SO50756'], values: ['SO50756'] });
    event.model.parameters = reportParams;
}
function App() {
    return (
        <div style={viewerStyle}>
        <BoldReportViewerComponent
        id="reportviewer-container"
        reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
        processingMode = {"Local"}
        reportPath = {reportPath}
        viewReportClick = {viewReportClick}>
```

```

</BoldReportViewerComponent>
</div>
);
}
export default App;
`

```

The model property in the event argument has the details of current processing report model.

Render subreport

You can display another report inside the body of a main report using the Report Viewer. The following steps help you to customize the subreport properties such as data source, report path, and parameters.

- Add the subreport and main reports to your application `App_Data` folder. In this tutorial, the already created reports are used. Refer to the [Create RDL Report section](#) or [Create RDLC Report section](#).

Download the `SideBySideMainReport.rdl` and `SideBySideSubReport.rdl` reports from [here](#). You can add the report from the Bold Reports installation location. For more information, refer to [Samples and demos](#). The reports used from the installed location requires the `NorthwindIO_Reports.sdf` database to run, so add it to your application.

- Set the `reportPath` and `reportServiceUrl` properties of the Report Viewer as shown in following code snippet.

```

`javascript
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = { 'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'SideBySideMainReport.rdl' }

</BoldReportViewerComponent>
</div>);
}
`

```

- Build and run the application.

Change subreport path

To change the subreport file path, set the **ReportPath** property of **SubReportModel** in the **OnInitReportOptions** method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.ReportPath =
        System.Web.Hosting.HostingEnvironment.MapPath(@"~/AppData/SubReportDetail.rdl");
    }
}
`
```

Set subreport parameter

You can change the parameter default values of a subreport in the **OnReportLoaded** method of the Web API Controller as given in the following code snippet.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.Parameters = new BoldReports.Web.ReportParameterInfoCollection();
        reportOption.SubReportModel.Parameters.Add(new BoldReports.Web.ReportParameterInfo()
        {
            Name = "SalesPersonID",
            Values = new List<string>() { "2" }
        });
    }
}
`
```

Modify subreport data source connection string

You can change the credential and connection information of the data sources used in the subreport using the **SubReportModel** in the **OnInitReportOptions** method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSourceCredentials = new
        List<BoldReports.Web.DataSourceCredentials>();

        reportOption.SubReportModel.DataSourceCredentials.Add(new
        BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
        Catalog=<database>;user id=<username>;password=<password>"));
    }
}
`
```

Set subreport data source

You can bind local business object data source collection only for RDLC reports. To specify data source of a RDLC subreport, set the `ReportDataSource` property in the `OnReportLoaded` method.

The RDL report has the connection information in report definition itself, so no need to bind the data source.

1. Add the RDLC subreport and main reports to your application `App_Data` folder. You can downloaded it from [here](#).
2. Set the `reportPath` and `reportServiceUrl` properties of the Report Viewer as shown in following code snippet.

```
`javascript
function App() {
    return (<div id="viewer" style={viewerStyle}>
        <BoldReportViewerComponent id="reportviewer-container"
        reportServiceUrl = { 'https://demos.boldreports.com/services/api/ReportViewer'}
        reportPath = { 'SideBySideMainReport.rdl' }

        </BoldReportViewerComponent>
    </div>);
}
`
```

- Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = null;
        data = new ProductList()
        {
            ProductName = "Baked Chicken and Cheese",
            OrderId = "323B60",
            Price = 55,
            Category = "Non-Veg",
            Ingredients = "grilled chicken, corn and olives.",
            ProductImage = ""
        };
        datas.Add(data);
        data = new ProductList()
        {
            ProductName = "Chicken Delite",
            OrderId = "323B61",
            Price = 100,
            Category = "Non-Veg",
            Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
            ProductImage = ""
        };
    }
}
```

```

};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
`

```

- Bind the business object data values collection to subreport by adding a new item to the **DataSources** as shown in the following code snippet.

```

`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'Product List.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
        "list", Value = ProductList.GetData() });
    }
}
}
`

```

The data source name is case sensitive, and it should be same as in the report definition.

Load subreport stream

To load subreport as stream, set the **Stream** property in the **OnInitReportOptions** method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        // Opens the report from application App_Data folder using FileStream and loads the sub report stream.
        FileStream reportStream = new
        FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"~/App_Data/Product List.rdlc"),
        FileMode.Open, FileAccess.Read);
        reportOption.SubReportModel.Stream = reportStream;
    }
}
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'Product List.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
        "list", Value = ProductList.GetData() });
    }
}
`
```

Report parameters

Provides property options to pass or set report parameters default values at run time using the **parameters** property. You can set the report parameters while creating the Report Viewer control in a script or in the Web API Controller.

In this tutorial, the **Sales Order Detail.rdl** report is used, and it can be downloaded from [here](#).

Set parameter at client side

The **parameters** property takes the JSON array value input with parameter details.

- Set the default value data to the `values` property and name of the report parameter to the `name` property.

The parameter name is case sensitive, and it should be same as available in the report definition.

The following code example illustrates how to set a report parameter in the script.

```
`javascript
var parameters = [{
  name: 'SalesOrderNumber',
  labels: ['SO50751'],
  values: ['SO50751'],
  nullable: false
}];
function App() {
  return (<div id="viewer" style={viewerStyle}>
    <BoldReportViewerComponent id="reportviewer-container"
      reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
      reportPath = { 'Sales Order Detail.rdl' }
      parameters = {parameters}

    </BoldReportViewerComponent>
  </div>);
}
```

- Build and run the application.

Set parameters in Web API Controller

To set parameter default value in the Web API Controller, use the following code in the `OnReportLoaded` method.

```
`csharp
public void OnReportLoaded(ReportViewerOptions reportOption)
{
  List<BoldReports.Web.ReportParameter> userParameters = new
  List<BoldReports.Web.ReportParameter>();
  userParameters.Add(new BoldReports.Web.ReportParameter()
  {
```

```

Name = "SalesOrderNumber",
Values = new List<string>() { "SO50756" }
});
reportOption.ReportModel.Parameters = userParameters;
}
`

```

The Report Parameters name should be case sensitive

Get report parameter

The **ReportHelper** class provides methods that help you to get the report parameters used in the report. The following helper methods are used to get parameter with or without values.

Methods | Description

Short Date and Time - d/M/yy h:mm tt | 9/12/2014 2:04 PM

Medium Date - d MMM yy h:mm tt | 12 Sep 14 2:04: PM

Full Date and short time - dddd, MMMM dd, yyyy HH:mm tt | Friday, September 12,2014 2:04 PM

Full Date and Long Time - dddd, MMMM dd, yyyy HH:mm:ss tt | Friday, September 12,2014 2:04:00 PM

UTC - yyyy-MM-dThh:mm:ssz | 2014-09-12T2:04:00+5

```

`javascript
var parameterSettings = {
dateTimePickerType: "DateTime",
dateTimeFormat: "MM/dd/yyyy h:mm tt",
timeDisplayFormat: "HH:mm",
timeInterval: 60,
};
function App() {
return ( <div className = "App" >
<BoldReportViewerComponent id = "reportviewer-container"
parameterSettings = { parameterSettings }>
</BoldReportViewerComponent> </div>
);
}
`

```

The above code sets date and time value to be display for all the date parameters used in the report.

To set different date and time value to be display for each date parameter used in the report, register the event `beforeParameterAdd` and specify date and time value based on parameter name as in below code sample.

```
`javascript
function App() {
  return (
    <div id="viewer" style={viewerStyle}>
      <BoldReportViewerComponent
        id="reportviewer-container"
        reportServiceUrl={"https://demos.boldreports.com/services/api/ReportViewer"}
        reportPath={"~/Resources/demos/Report/product-line-sales.rdl"}
        beforeParameterAdd={onBeforeParameterAdd}>
      </BoldReportViewerComponent>
    </div>);
}

function onBeforeParameterAdd(event) {
  event.parameterSettings.dateTimePickerType = "DateTime";
  if (event.parameterModel.Name === "StartDate") {
    event.parameterSettings.dateTimeFormat = "MM/dd/yyyy h:mm tt";
    event.parameterSettings.timeDisplayFormat = "HH:mm";
    event.parameterSettings.timeInterval = 60;
  }
  if (event.parameterModel.Name === "EndDate") {
    event.parameterSettings.dateTimeFormat = "MM/dd/yyyy h:mm tt";
    event.parameterSettings.timeDisplayFormat = "HH:mm";
    event.parameterSettings.timeInterval = 60;
  }
}
```

[Access the hidden or internal parameter information](#)

The `accessInternalValue` property in the `parameterSettings` helps you to expose the `hidden` or `internal` report parameter information used in report to the user.

```
` javascript
var parameterSettings = {
```



```
accessInternalValue: true
}
function App() {
return ( <div className = "App" >
<BoldReportViewerComponent id = "reportviewer-container"
parameterSettings = { parameterSettings }>
</BoldReportViewerComponent> </div>
);
}
`
```

Set the report parameter visibility in Web API Controller

The `Hidden` property of `ReportParameter` allows you to show or hide the parameter at the top of the report viewer panel. The following code example shows hiding a report parameter in the Web API controller's `OnReportLoaded` method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
var reportParameters = ReportHelper.GetParameters(jsonArray, this);
List<BoldReports.Web.ReportParameter> modifiedParameters = new
List<BoldReports.Web.ReportParameter>();
if (reportParameters != null)
{
foreach (var rptParameter in reportParameters)
{
modifiedParameters.Add(new BoldReports.Web.ReportParameter()
{
Name = rptParameter.Name,
Hidden = true
});
}
reportOption.ReportModel.Parameters = modifiedParameters;
}
}
```

Report interaction events

You can handle the report interaction events with reports using the following events.

- ReportLoaded
- ReportError
- ShowError
- Drill through
- Hyperlink

Report loaded

The `reportLoaded` event occurs after the report is loaded and it is ready to start the processing. You can handle the event and specify the data source and parameters at client side. The following sample code loads a report by assigning the report data source input in the `reportLoaded` event.

In this tutorial, the `IndicatorReport.rdlc` report is used. You can add the report from the Bold Reports installation location. For more information, refer to [Samples and demos](#).

```
`javascript
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = { 'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'IndicatorReport.rdlc' }
processingMode = { 'Local' }
reportLoaded = {reportLoaded}

</BoldReportViewerComponent>
</div>);
}
function reportLoaded(event) {
var desc2013 = [
{
No: 1, Name: "Carlos Slim", NetWorth: 73.0, Age: 73, CitizenShip: "Mexico", Source: "Telmex,America
Movil, Grupo Carso", RankingStatus: 50, ProfitStatus: 75
},
{
```

```
No: 2, Name: "Bill Gates", NetWorth: 67.0, Age: 57, CitizenShip: "United States", Source: "Microsoft",
RankingStatus: 50, ProfitStatus: 75
},
{
No: 3, Name: "Amancio Ortega", NetWorth: 57.0, Age: 57, CitizenShip: "Spain", Source: "Inditex Group",
RankingStatus: 75, ProfitStatus: 75
},
{
No: 4, Name: "Warren Buffett", NetWorth: 53.0, Age: 82, CitizenShip: "United States", Source:
"Berkshire Hathaway", RankingStatus: 25, ProfitStatus: 75
},
{
No: 5, Name: "Larry Ellison", NetWorth: 43.0, Age: 68, CitizenShip: "United States", Source: "Oracle
Corporation", RankingStatus: 75, ProfitStatus: 75
},
{
No: 6, Name: "Charles Koch", NetWorth: 34.0, Age: 77, CitizenShip: "United States", Source: "Koch
Industries", RankingStatus: 75, ProfitStatus: 75
},
{
No: 7, Name: "David Koch", NetWorth: 34.0, Age: 72, CitizenShip: "United States", Source: "Koch
Industries", RankingStatus: 75, ProfitStatus: 75
},
{
No: 8, Name: "Li Ka-shing", NetWorth: 32.0, Age: 84, CitizenShip: "Hong Kong/ Canada", Source:
"Cheung Kong Holdings", RankingStatus: 75, ProfitStatus: 75
},
{
No: 9, Name: "Liliane Bettencourt", NetWorth: 30.0, Age: 90, CitizenShip: "France", Source: "L'Oreal",
RankingStatus: 75, ProfitStatus: 75
},
{
No: 10, Name: "Bernard Arnault", NetWorth: 29.0, Age: 63, CitizenShip: "France", Source: "LVMH Moet
Hennessy Louis Vuitton", RankingStatus: 25, ProfitStatus: 25
}};
var desc2012 = [
```

```
{
No: 1, Name: "Carlos Slim", NetWorth: 69.0, Age: 72, CitizenShip: "Mexico", Source: "Telmex,America
Movil, Grupo Carso", RankingStatus: 50, ProfitStatus: 25
},
{
No: 2, Name: "Bill Gates", NetWorth: 61.0, Age: 56, CitizenShip: "United States", Source: "Microsoft",
RankingStatus: 50, ProfitStatus: 75
},
{
No: 3, Name: "Warren Buffett", NetWorth: 44.0, Age: 81, CitizenShip: "United States", Source:
"Berkshire Hathaway", RankingStatus: 50, ProfitStatus: 25
},
{
No: 4, Name: "Bernard Arnault", NetWorth: 41.0, Age: 63, CitizenShip: "France", Source: "LVMH Moet
Hennessy Louis Vuitton", RankingStatus: 50, ProfitStatus: 75
},
{
No: 5, Name: "Amancio Ortega", NetWorth: 37.5, Age: 75, CitizenShip: "Spain", Source: "Inditex Group",
RankingStatus: 75, ProfitStatus: 75
},
{
No: 6, Name: "Larry Ellison", NetWorth: 36.0, Age: 67, CitizenShip: "United States", Source: "Oracle
Corporation", RankingStatus: 25, ProfitStatus: 75
},
{
No: 7, Name: "Eike Batista", NetWorth: 30.0, Age: 55, CitizenShip: "Brazil", Source: "EBX Group",
RankingStatus: 75, ProfitStatus: 75
},
{
No: 8, Name: "Stefan Persson", NetWorth: 26.5, Age: 64, CitizenShip: "Sweden", Source: "H&M",
RankingStatus: 75, ProfitStatus: 75
},
{
No: 9, Name: "Li Ka-shing", NetWorth: 25.0, Age: 83, CitizenShip: "Hong Kong/ Canada", Source:
"Cheung Kong Holdings", RankingStatus: 75, ProfitStatus: 75
},
}
```

```

{
No: 10, Name: "Karl Albrecht", NetWorth: 25.4, Age: 92, CitizenShip: "Germany", Source: "Aldi",
RankingStatus: 75, ProfitStatus: 25
});
var description = [
{
Status: 25, Description: "Has not changed from the previous ranking."
},
{
Status: 50, Description: "Has increased from the previous ranking."
},
{
Status: 75, Description: "Has decreased from the previous ranking."
}
];
event.model.dataSources = [{
value: ej.DataManager(desc2013).executeLocal(ej.Query()),
name: "DataSet1"
}, {
value: ej.DataManager(desc2012).executeLocal(ej.Query()),
name: "DataSet2"
}, {
value: ej.DataManager(description).executeLocal(ej.Query()),
name: "DataSet3"
}
];
}
`

```

Report error

When an error occurs in the report processing, it raises the `reportError` event. You can handle the event and show the details in your custom dialog instead of component default error detail interface.

```

`javascript
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = { 'https://demos.boldreports.com/services/api/ReportViewer'}

```

```
reportPath = { 'Sales Order Detail.rdl' }
reportError = {onReportError}
```

```
</BoldReportViewerComponent>
</div>);
}
function onReportError(event) {
  alert(event.errmsg);
  event.cancel = true;
}
`
```

To suppress the default error dialog, set the cancel argument to true.

Show error

The `showError` event is invoked whenever users click a report item that contains an error in processing. It provides detailed information about the cause of the error. You can hide the default dialog and show your customized dialog.

```
`javascript
function App() {
  return (<div id="viewer" style={viewerStyle}>
    <BoldReportViewerComponent id="reportviewer-container"
      reportServiceUrl = { 'https://demos.boldreports.com/services/api/ReportViewer' }
      reportPath = { 'Sales Order Detail.rdl' }
      reportError = {onShowError}

    </BoldReportViewerComponent>
  </div>);
}
function onShowError(event) {
  alert("Error code : " + event.errorCode + "\n" +
    "Error Detail : " + event.errorDetail + "\n" +
    "Error Message : " + event.errorMessage);
  event.cancel = true;
}
`
```

Drill through

When a drill through item is selected in a report, it invokes the `drillThrough` event. You can change the drill through arguments such as report parameter and data sources. The following sample code can be used to change the drill through report name and set the parameter value before the drill through report is rendered.

```
`javascript
function App() {
  return (<div id="viewer" style={viewerStyle}>
    <BoldReportViewerComponent id="reportviewer-container"
      reportServiceUrl = { 'https://demos.boldreports.com/services/api/ReportViewer'}
      reportPath = { 'SalesPersonDetails.rdl' }
      drillThrough = {onDrillThrough}

    </BoldReportViewerComponent>
  </div>);
}

function onDrillThrough(event) {
  event.actionInfo.ReportName = "PersonalDetails";
  event.actionInfo.Parameters = [{
    name: 'SalesOrderNumber',
    labels: ['SO50751'],
    values: [SO50751],
    nullable: false
  }];
}
```

Hyperlink

The `hyperlink` event occurs when users click a hyperlink in a report, before the hyperlink is followed. The following sample code redirects to a new custom URL and cancels the component default action.

```
`javascript
function App() {
  return (<div id="viewer" style={viewerStyle}>
    <BoldReportViewerComponent id="reportviewer-container"
      reportServiceUrl = { 'https://demos.boldreports.com/services/api/ReportViewer'}
      reportPath = { 'Customer Support Analysis (Random data).rdl' }
```

```
hyperlink = {onHyperLink}
```

```
</BoldReportViewerComponent>
</div>);
}
function onHyperLink(event) {
  event.cancel = true;
  //You can modify the URL here
  window.open(event.actionInfo.Hyperlink);
}
`
```

Handle post actions

Report processing actions are sent in an Ajax request to exchange data with the Web API service. You can handle post actions event to customize the Ajax requests.

- `AjaxBeforeLoad`
- `AjaxSuccess`
- `AjaxError`

AjaxBeforeLoad

This event can be triggered before an Ajax request is sent to the Report Viewer Web API service. It allows you to set additional headers and custom data in the Ajax request. The following code sample demonstrates adding custom authorization header and passing default parameter values to service.

Add custom header in Ajax request

Initialize the `ajaxBeforeLoad` event in the script and add the authorization token to the `headers` property.

```
`javascript
function App() {
  return (<div id="viewer" style={viewerStyle}>
    <BoldReportViewerComponent id="reportviewer-container"
      reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
      reportPath = { 'Sales Order Detail.rdl' }
      ajaxBeforeLoad = {onAjaxRequest}

    </BoldReportViewerComponent>
  </div>);
`
```



```

}
function onAjaxRequest(event) {
event.headers.push({
Key: 'Authorization', Value: 'demo@123'
});
}
`

```

In this tutorial, the **Sales Order Detail.rdl** report is used, and it can be downloaded from [here](#).

Get the custom header value from the **HttpContext** header collection using the key name specified at client side.

```

`csharp
string authenticationHeader;
public object PostReportAction(Dictionary<string, object> jsonResult)
{
if (jsonResult != null)
{
//Get client side custom ajax header and store in local variable
authenticationHeader = HttpContext.Current.Request.Headers["Authorization"];
//Perform your custom validation here
if (authenticationHeader == "")
{
return new Exception("Authentication failed!!!");
}
else
{
return ReportHelper.ProcessReport(jsonResult, this);
}
}
return null;
}
`

```

Perform your own action to validate the header values.

Pass custom data in Ajax request

Use the `data` property to set custom data to the server in the Ajax request. In the following code sample, parameter values are passed to the server side.

```
`javascript
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'Sales Order Detail.rdl' }
ajaxBeforeLoad = {onAjaxRequest}

</BoldReportViewerComponent>
</div>);
}
function onAjaxRequest(event) {
//Passing custom data to server
var customerID = "CI0021";
event.data = customerID;
}
`
```

The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates to change the datasource connection strings based on Customer ID in the `OnInitReportOptions` method.

```
`csharp
string CustomerID = null;
public object PostReportAction(Dictionary<string, object> jsonResult)
{
if (jsonResult != null)
{
if (jsonResult.ContainsKey("customData"))
{
//Get client side custom data and store in local variable.
CustomerID = jsonResult["customData"].ToString();
}
}
}
```

```

}
return ReportHelper.ProcessReport(jsonResult, this);
}
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (CustomerID != null)
    {
        if (CustomerID.Contains("CI0021"))
        {
            //If you are changing the connection string based on customer id then could you please change the
            connection string as below.

            //reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

            reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
            Catalog=<database>;"));
        }
        else if (CustomerID.Contains("CI0022"))
        {
            //If you are changing the connection string based on customer id then could you please change the
            connection string as below.

            //reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

            reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
            Catalog=<database>;"));
        }
    }
}

```

AjaxSuccess

To perform custom action or show user defined message, use the `ajaxSuccess` event on the successful Ajax request.

```

`javascript
function App() {
    return (<div id="viewer" style={viewerStyle}>

```

```

<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'Sales Order Detail.rdl' }
ajaxSuccess = {onAjaxSuccess}

</BoldReportViewerComponent>
</div>;
}
function onAjaxSuccess(event) {
//Perform your custom success message here
alert("Ajax request success!!!");
}
`

```

AjaxError

The `ajaxError` event is called, if an error occurred with the Ajax request. You can display the customized error details in the event method.

```

`javascript
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'Sales Order Detail.rdl' }
ajaxError = {onAjaxFailure}

</BoldReportViewerComponent>
</div>);
}
function onAjaxFailure(event) {
alert("Status: " + event.status + "\n" +
"Error: " + event.responseText);
}
`

```

You can never have both an error and a success callback with a request.

Print report

The Report Viewer provides print report option in the toolbar to print a copy of the report. The Page Setup dialog allows you to set the paper size or other page setup properties. To see print margins, click **Print Layout** on the toolbar.

You can set values in the Page Setup dialog box for current session only. When you close the report and reopen it, it will have the default values again. The default values for the Page Setup dialog is based on the report properties set in the design view.

View report in print mode

Print margins are displayed in the Print Layout only. To view the report in print mode by default, set the `printMode` property to true.

```
`javascript
var isPrintMode = true;
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }
printMode = {isPrintMode}

</BoldReportViewerComponent>
</div>);
}
```

By default, the Report Viewer renders the report in normal layout in which the print margins are not displayed.

Print in new page

To open the print in a new tab of the current browser, set the `printOption` property to `NewTab`. By default, it shows the print dialog in the same page.

```
`javascript
var printOption = ej.ReportViewer.PrintOptions.NewTab;
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }
```

```
printOption = {printOption}
```

```
</BoldReportViewerComponent>
```

```
</div>);
```

```
}
```

```
,
```

The pop-up blocker should be enabled for the page to open the print view in a new tab.

Set page orientation and paper size

You can specify the print page paper size and orientation at client-side to change the page setup properties by setting the `pageSettings` property.

```
`javascript
```

```
var isPrintMode = true;
```

```
var pageSettings = {
```

```
orientation: ej.ReportViewer.Orientation.Portrait,
```

```
paperSize: ej.ReportViewer.PaperSize.A3,
```

```
};
```

```
function App() {
```

```
return (<div id="viewer" style={viewerStyle}>
```

```
<BoldReportViewerComponent id="reportviewer-container"
```

```
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
```

```
reportPath = { 'GroupingAgg.rdl' }
```

```
printMode = {isPrintMode}
```

```
pageSettings = {pageSettings}
```

```
</BoldReportViewerComponent>
```

```
</div>);
```

```
}
```

```
,
```

Set report margin

To set margin values to the report page setup, use the `margins` property and specify the value to top, right, bottom, and left.

```
`javascript
```

```
var isPrintMode = true;
```

```
var pageSettings = {
```

```

margins: {
top: 0.5,
right: 0.25,
bottom: 0.25,
left: 0.25
}
};
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }
printMode = {isPrintMode}
pageSettings = {pageSettings}

</BoldReportViewerComponent>
</div>);
}
`

```

The values set in the margin property is considered as inches input.

Set page height and width

To set the height and width values to the report page setup, use the `height` and `width` properties.

```

`javascript
var isPrintMode = true;
var pageSettings = {
height: 2.69,
width: 28.27
};
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }

```

```
printMode = {isPrintMode}
pageSettings = {pageSettings}
```

```
</BoldReportViewerComponent>
</div>;
}
`
```

The values set in the height and width properties is considered as inches input.

Print report with images

When the report has more images, the browser will send the report stream to the print dialog before the images are completely loaded. To load the print report stream with complete images, set the `EmbedImageData` property to true in `OnInitReportOptions` as shown in the following code.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.EmbedImageData = true;
}
`
```

Replace the following code sample in client-side HTML file.

```
`javascript
function App() {
    return (<div id="viewer" style={viewerStyle}>
        <BoldReportViewerComponent id="reportviewer-container"
            reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
            reportPath = { 'GroupingAgg.rdl' }

        </BoldReportViewerComponent>
    </div>);
}
`
```

In this tutorial, the `Product Details.rdl` report is used, and it can be downloaded from [here](#).

External styles in report printing

While printing a report, the external styles used in the application overrides the printable page style and prints output with incorrect alignments. To avoid the external script overriding, set the `isStyleLoad` property to false, which will print the page using only the Report Viewer styles.

```
`javascript
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }
reportPrint = {onReportPrint}

</BoldReportViewerComponent>
</div>);
}
function onReportPrint(event) {
event.isStyleLoad = false;
}
`
```

Show print progress

The Report Viewer provides events that help you show the progress information, when the printing process takes a long time to complete.

To show print progress, follow these steps:

1. Set the `printProgressChanged` in Report Viewer initialization.
2. Implement the function and add code samples to show a custom message based on the print progress status as shown in the following code snippet.

```
`javascript
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }
printProgressChanged = {onPrintProgressChanged}
```

```

</BoldReportViewerComponent>
</div>;
}
function onPrintProgressChanged(event) {
if (event.stage === "beginPrint") {
console.log(event.stage);
}
if (event.stage === "printStarted") {
console.log(event.stage);
}
else if (event.stage === "preparation") {
console.log(event.stage);
if (event.preparationStage === "dataPreparation") {
console.log(event.preparationStage);
console.log(event.totalPages);
console.log(event.currentPage);
}
}
event.handled = true;
}
}
`

```

Remove empty spaces in printing

The extra blank page is created when the body of your report is too wide for your page. To make the report appear on a single page, all the content within the report body must fit on the physical page, and the body width should be as the following formula:

Body Width <= Page Width - (Left Margin + Right Margin)

For more details to remove the empty pages in the report while designing, refer to the knowledge base article of [report page sizing](#).

Export report

The Report Viewer provides events and properties to control and customize the report exporting functionality.

Export event handling

You can show the progress information, when the exporting process takes long time to complete using the `exportProgressChanged` event.

1. Set the `exportProgressChanged` event in Report Viewer initialization.
2. Implement the function and replace the following code samples to get the log message based on the progress stage.

```
`javascript
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }
exportProgressChanged = {onExportProgressChanged}

</BoldReportViewerComponent>
</div>);
}
function onExportProgressChanged(event) {
if (event.stage === "beginExport") {
console.log(event.stage);
}
else if (event.stage === "exportStarted") {
console.log(event.stage);
}
else if (event.stage === "preparation") {
console.log(event.stage);
console.log(event.format);
console.log(event.preparationStage);
}
event.handled = true;
}
`
```

Change Excel and Word export format

You can change the default file format to any other file format using the `excelFormat` and `wordFormat` properties. The following code sample changes the default versions.

```
`javascript
var exportSettings = {
```

```

excelFormat: ej.ReportViewer.ExcelFormats.Excel2013,
wordFormat: ej.ReportViewer.WordFormats.Word2013
}
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }
exportSettings = {exportSettings}

</BoldReportViewerComponent>
</div>);
}
`

```

Hide specific export type for report

Show or hide the default export types available in the component using the `exportOptions` property. The following code hides the HTML export type from the default export options.

```

`javascript
var exportSettings = {
exportOptions: ej.ReportViewer.ExportOptions.All & ~ej.ReportViewer.ExportOptions.HTML
}
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }
exportSettings = {exportSettings}

</BoldReportViewerComponent>
</div>);
}
`

```

PDF export options

The `PDFOptions` provides properties to manage PDF export behaviors. You should set the properties in the `OnInitReportOptions` method of the Web API service.

Export with complex scripts

To export reports with the complex scripts, set the `ComplexScript` property of `PDFOptions` instance to true.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
    {
        EnableComplexScript = true
    };
}
`
```

PDF conformance

You can export the report as a PDF/A-1b document by specifying the `PdfConformanceLevel.Pdf_A1B` conformance level in the `PdfConformanceLevel` property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
    {
        PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B
    };
}
`
```

Add custom PDF fonts

You can add custom fonts to the PDF exported document by adding the font streams to `Fonts` collection in `PDFOptions` instance.

To add custom fonts to the PDF exported document, follow these steps:

1. Add the font `.ttf` files into your application `App_Data` folder.
2. In the Solution Explorer, open the properties of the font file and set the property Copy to Output Directory as Copy always.
3. Initialize the `Font` collection and add the font stream to it.

The key value provided in the font collection should be same as in the report item font property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
    {
        //Load Missing font stream
        Fonts = new Dictionary<string, System.IO.Stream>
        {
            { "Segoe UI",
              System.IO.File.OpenRead(System.Web.Hosting.HostingEnvironment.MapPath(@"~/AppData/fontsymbols.ttf")) }
        }
    };
}
```

If any fonts used in the report definition is not installed or available in the local system, then you should load the font stream. In the above code, `font_symbols` font stream is loaded to export the `Sales Order Detail.rdl` report as symbols.

Word export options

The `WordOptions` provides properties to manage Word document export behaviors.

Word document type

You can save the report to the required document version by setting the `FormatType` property.

```
`csharp
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
    FormatType = BoldReports.Writer.WordFormatType.Docx
};
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
```

```
{
LayoutOption = BoldReports.Writer.WordLayoutOptions.TopLevel,
ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
Bottom = 0.5f,
Top = 0.5f
}
};
`
```

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior.

The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, added an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without password. The following are the types of protection:

- **AllowOnlyComments**: Adds or modifies only the comments in the Word document.
- **AllowOnlyFormFields**: Modifies the form field values in the Word document.
- **AllowOnlyRevisions**: Accepts or rejects the revisions in the Word document.
- **AllowOnlyReading**: Allows you to view the content only in the Word document.
- **NoProtection**: Accesses or edits the Word document contents as normally.

```
`csharp
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading
};
`
```

Excel export options

The **ExcelOptions** provides properties to manage Excel document export behaviors.

Excel document type

You can save the report to the required excel version by setting the **ExcelSaveType** property.

```
`csharp
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{

```

```
ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013
```

```
};
```

```
,
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` to `IgnoreCellMerge`.

```
`csharp
```

```
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
```

```
{
```

```
LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
```

```
};
```

```
,
```

Protecting Excel document from editing

You can restrict the Excel document from editing either by providing the `ExcelSheetProtection` or enabling the `ReadOnlyRecommended` properties.

```
`csharp
```

```
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
```

```
{
```

```
ReadOnlyRecommended = true,
```

```
ExcelSheetProtection = ExcelSheetProtection.DeletingColumns
```

```
};
```

```
,
```

PowerPoint export options

You can save the report to the required PowerPoint version by setting the `FormatType` property.

```
`csharp
```

```
reportOption.ReportModel.PPTOptions = new BoldReports.Writer.PPTOptions()
```

```
{
```

```
FormatType = BoldReports.Writer.PPTSaveType.PowerPoint2013
```

```
};
```

```
,
```

CSV export options

The `CsvOptions` allows you change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document.

```
`csharp
```

```
reportOption.ReportModel.CsvOptions = new BoldReports.Writer.CsvOptions()
```



```
{  
Encoding = System.Text.Encoding.Default,  
FieldDelimiter = ",",  
UseFormattedValues = false,  
Qualifier = "#",  
RecordDelimiter = "@",  
SuppressLineBreaks = true,  
FileExtension = ".txt"  
};  
`
```

HTML export options

You can hide the separator added at the end of each page by setting the `HidePageSeparator` property to true.

```
`csharp  
reportOption.ReportModel.HTMLOptions = new BoldReports.Writer.HTMLOptions()  
{  
HidePageSeparator = true  
};  
`
```

Password protect exported document

Allows you protect the exported document such as PDF, Microsoft Word, Microsoft Excel, and PowerPoint from unauthorized users by encrypting the document using encryption password. The following code snippet demonstrates how to encrypt the exported document with user defined password.

```
`csharp  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
//PDF encryption  
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions();  
reportOption.ReportModel.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity()  
{  
UserPassword = "password"  
};  
//Word encryption  
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()  
}
```

```
{
EncryptionPassword = "password"
};
//Excel encryption
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
PasswordToModify = "password",
PasswordToOpen = "password"
};
//PPT encryption
reportOption.ReportModel.PPTOptions = new BoldReports.Writer.PPTOptions()
{
EncryptionPassword = "password"
};
}
```

Password protection is not supported for HTML export format.

Toolbar customization

You can hide the component toolbar to show customized user interface or to customize the toolbar icons and element's appearances using the templates and Report Viewer toolbar customization properties.

In this tutorial, the `Sales Order Detail.rdl` report is used, and it can be downloaded from [here](#). You can add the reports from the Bold Reports installation location. For more information, refer to [Samples and demos](#).

Hide toolbar items

To hide toolbar items, set the `toolbarSettings` property. The following code can be used to remove the parameter option from the toolbar and hide the parameter block. Use the following code snippet in `App.js` file.

Similarly, you can show or hide all other toolbar options with the help of [toolbarSettings.items](#) enum.

Use the following code snippet in `App.js` file.

```
`js
var toolbarSettings = {
items: ~ej.ReportViewer.ToolbarItems.Parameters
}
function App() {
```

```
return (  
<div style={viewerStyle} className="App">  
  <BoldReportViewerComponent  
    id="reportviewer-container"  
    serviceUrl={'https://demos.boldreports.com/services/api/ReportViewer'}  
    reportPath = {'~/Resources/docs/sales-order-detail.rdl'}  
    toolbarSettings = {toolbarSettings}>  
  </BoldReportViewerComponent>  
</div>  
);  
}  
`
```

The following code sample hides the print options from the toolbar items which can be used in the `App.js` file.

```
`js  
var toolbarSettings = {  
  items: ~ej.ReportViewer.ToolbarItems.Print  
}  
function App() {  
  return (  
    <div style={viewerStyle} className="App">  
      <BoldReportViewerComponent  
        id="reportviewer-container"  
        serviceUrl={'https://demos.boldreports.com/services/api/ReportViewer'}  
        reportPath = {'~/Resources/docs/sales-order-detail.rdl'}  
        toolbarSettings = {toolbarSettings}>  
      </BoldReportViewerComponent>  
    </div>  
  );  
}  
`
```

Enable stop option in toolbar

To enable stop option in toolbar, set the `toolbarSettings.items` property to `ej.ReportViewer.ToolbarItems.All`. The following code can be used to enable stop option in toolbar in App.js file.

```
`js
var toolbarSettings = {
items: ej.ReportViewer.ToolbarItems.All
}
function App() {
return (
<div style={viewerStyle} className="App">
<BoldReportViewerComponent
id="reportviewer-container"
serviceUrl={'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = {'~/Resources/docs/sales-order-detail.rdl'}
toolbarSettings = {toolbarSettings}>
</BoldReportViewerComponent>
</div>
);
}
`
```

Enable export setup option in toolbar

To enable export setup option in toolbar, set the `toolbarSettings.items` property to `ej.ReportViewer.ToolbarItems.All`. The following code can be used to enable export setup option in toolbar in App.js file.

```
`js
var toolbarSettings = {
items: ej.ReportViewer.ToolbarItems.All
}
function App() {
return (
<div style={viewerStyle} className="App">
<BoldReportViewerComponent
id="reportviewer-container"
```

```

serviceUrl={'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = {'~/Resources/docs/sales-order-detail.rdl'}
toolbarSettings = {toolbarSettings}>
</BoldReportViewerComponent>
</div>
);
}
`

```

Enable search text option in toolbar

To enable search text option in toolbar, set the `toolbarSettings.items` property to `ej.ReportViewer.ToolbarItems.All`. The following code can be used to enable search text option in toolbar in `App.js` file.

```

`js
var toolbarSettings = {
items: ej.ReportViewer.ToolbarItems.All
}
function App() {
return (
<div style={viewerStyle} className="App">
<BoldReportViewerComponent
id="reportviewer-container"
serviceUrl={'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = {'~/Resources/docs/sales-order-detail.rdl'}
toolbarSettings = {toolbarSettings}>
</BoldReportViewerComponent>
</div>
);
}
`

```

Hide toolbar

To hide the Report Viewer toolbar, set the `showToolbar` property to false.

```

`js
var toolbarSettings = {
showToolbar: false

```

```

}
function App() {
  return (
    <div style={viewerStyle} className="App">
      <BoldReportViewerComponent
        id="reportviewer-container"
        serviceUrl={"https://demos.boldreports.com/services/api/ReportViewer"}
        reportPath = {'~/Resources/docs/sales-order-detail.rdl'}
        toolbarSettings = {toolbarSettings}>
      </BoldReportViewerComponent>
    </div>
  );
}

```

Decide or hide the export option

The Report Viewer provides the `exportOptions` property to show or hide the default export types available in the component. The following code hides the HTML export type from the default export options which is used in `App.js` file.

```

`js
var exportsettings = {
  exportOptions: ej.ReportViewer.ExportOptions.All & ~ej.ReportViewer.ExportOptions.HTML
};
function App() {
  return (
    <div style={viewerStyle} className="App">
      <BoldReportViewerComponent
        id="reportviewer-container"
        serviceUrl={"https://demos.boldreports.com/services/api/ReportViewer"}
        reportPath = {'~/Resources/docs/sales-order-detail.rdl'}
        exportSettings = {exportsettings}>
      </BoldReportViewerComponent>
    </div>
  );
}

```

,

Add custom items to the export drop-down

To add custom items to the export drop-down available in the Report Viewer toolbar, use the `customItems` property and provide the JSON array of collection input with the `index`, `cssClass` name, and `value` properties. Register the `exportItemClick` event to handle the custom item actions as given in following code snippet, which can be used in `App.js` file.

```
`js
var exportsettings = {
  customItems: [{
    index: 2,
    cssClass: "",
    value: 'Text File'
  },
  {
    index: 4,
    cssClass: "",
    value: 'DOT'
  }
  ];
  //Export click event handler
  function onExportItemClick(event) {
    if (event.value === "Text File") {
      //Implement the code to export report as Text
      alert("Text File export option clicked");
    } else if (event.value === "DOT") {
      //Implement the code to export report as DOT
      alert("DOT export option clicked");
    }
  }
  function App() {
    return (
      <div style={viewerStyle} className="App">
        <BoldReportViewerComponent
          id="reportviewer-container"
```

```

serviceUrl={'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = {'~/Resources/docs/sales-order-detail.rdl'}
exportSettings = {exportsettings}
exportItemClick = {onExportItemClick}>
</BoldReportViewerComponent>
</div>
);
}
`

```

Add custom toolbar item

You can add custom items to Report Viewer toolbar using the `toolbarSettings` property. You should register the `toolbarItemClick` event to handle the newly added custom items action.

Add custom item to exiting toolbar group

To add a custom item to existing toolbar group use the `customItems` property in `toolbarSettings` and provide the JSON array of collection input with the `groupIndex`, `index`, `itemType`, `cssClass` name, and `tooltip` properties as given in following code snippet, which is used in `App.js` file.

```

`js
var toolbarSettings = {
  showToolbar: true,
  items: ~ej.ReportViewer.ToolbarItems.Print,
  customItems: [{
    groupIndex: 1,
    index: 1,
    type: 'Default',
    cssClass: "e-icon e-mail e-reportviewer-icon",
    id: 'E-Mail',
    tooltip: {
      header: 'E-Mail',
      content: 'Send rendered report as mail attachment'
    }
  ]
}
}
//Toolbar click event handler
function onToolBarItemClick(event) {

```



```

if (event.value === "CustomItem") {
//Implement the code to CustomItem toolbar option
alert("Email item toolbar option Clicked");
}
}

function App() {
return (
<div style={viewerStyle} className="App">
<BoldReportViewerComponent
id="reportviewer-container"
serviceUrl={"https://demos.boldreports.com/services/api/ReportViewer"}
reportPath = {'~/Resources/docs/sales-order-detail.rdl'}
toolbarSettings = {toolbarSettings}
toolBarItemClick = {onToolBarItemClick}>
</BoldReportViewerComponent>
</div>
);
}

```

Add new toolbar group

To add a new toolbar group and custom items to it, use the `customGroups` property in the `toolbarSettings` and provide the JSON array of collection input with the `groupIndex` and `items` properties. The `items` should have the `itemType`, `cssClass`, and `tooltip` properties as given in following code snippet, which is used in the `App.js` file.

```

`js
var toolbarSettings = {
showToolbar: true,
items: ~ej.ReportViewer.ToolbarItems.Print,
customGroups: [{
items: [{
type: 'Default',
cssClass: "e-icon e-mail e-reportviewer-icon CustomGroup",
id: 'CustomGroup',
tooltip: { header: 'CustomGroup', content: 'toolbargroups' }

```

```

},
{
  type: 'Default',
  cssClass: "e-icon e-mail e-reportviewer-icon subCustomGroup",
  id: 'subCustomGroup',
  tooltip: { header: 'subCustomGroup', content: 'subtoolbargroups' }
}],
groupIndex: 3
}]
}

//Toolbar click event handler
function onToolBarItemClick(event) {
  if (event.value === "CustomGroup") {
    //Implement the code to CustomGroup toolbar option
    alert("CustomGroup toolbar option clicked");
  }
  if (event.value === "subCustomGroup") {
    //Implement the code to subCustomGroup toolbar option
    alert("SubCustomGroup toolbar option clicked");
  }
}

function App() {
  return (
    <div style={viewerStyle} className="App">
      <BoldReportViewerComponent
        id="reportviewer-container"
        serviceUrl={'https://demos.boldreports.com/services/api/ReportViewer'}
        reportPath = {'~/Resources/docs/sales-order-detail.rdl'}
        toolbarSettings = {toolbarSettings}
        toolBarItemClick = {onToolBarItemClick}>
      </BoldReportViewerComponent>
    </div>
  );
}

```

```
}
,
```

Custom actions

This section explains you the steps required to add user defined buttons in Report Viewer toolbar and invoke custom actions.

Send a report as an email attachment

This topic describes steps required to create custom email option and share a report as an email attachment to other users.

Add email button in Report Viewer

1. Create an email button in the toolbar using the `customItems` property with the values such as `groupIndex`, `index`, `itemType`, `cssClass`, and `tooltip`. The `toolbarItemClick` event triggers when you click the email button.
2. Access the Report Viewer model and create a JSON array for sending requests to the Web API server. You can use the following codes to create an event with custom action.

```
`javascript
var toolbarSettings = {
  showToolbar: true,
  items: ej.ReportViewer.ToolbarItems.All & ~ej.ReportViewer.ToolbarItems.Print,
  customItems: [{
    groupIndex: 1,
    index: 1,
    type: 'Default',
    cssClass: "e-icon e-mail e-reportviewer-icon",
    id: 'E-Mail',
    tooltip: {
      header: 'E-Mail',
      content: 'Send rendered report as mail attachment'
    }
  }]
}

function App() {
  return (<div id="viewer" style={viewerStyle}>
    <BoldReportViewerComponent id="reportviewer-container"
      reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}>
```

```

reportPath = { 'GroupingAgg.rdl' }
toolbarSettings = {toolbarSettings}
toolBarItemClick = {ontoolBarItemClick}

</BoldReportViewerComponent>
</div>;
}
function ontoolBarItemClick(args) {
if (args.value == "E-Mail") {
var proxy = $('#viewer').data('boldReportViewer');
var Report = proxy.model.reportPath;
var lastsIndex = Report.lastIndexOf("/");
var reportName = Report.substring(lastsIndex + 1);
var requrl = proxy.model.reportServiceUrl + '/SendEmail';
var _json = {
exportType: "PDF", reportViewerToken: proxy._reportViewerToken, ReportName: reportName
};
$.ajax({
type: "POST",
contentType: "application/json; charset=utf-8",
url: requrl,
data: JSON.stringify(_json),
dataType: "json",
crossDomain: true
})
}
}
,

```

Create custom email action

To create custom email action, follow these steps:

1. Create a new action method **SendEmail** in the Web API service.
2. Export the report to the required type using the **ReportHelper.GetReport** method to send a report stream as an attachment.

The following code sample exports the report to stream and send it as an attachment to a specified mail address. In the code, the `SmtpClient` method is used to send the report as an email attachment.

```
`csharp
public object SendEmail(Dictionary<string, object> jsonResult)
{
    string _token = jsonResult["reportViewerToken"].ToString();
    var stream = ReportHelper.GetReport(_token, jsonResult["exportType"].ToString());
    stream.Position = 0;
    if (!ComposeEmail(stream, jsonResult["reportName"].ToString()))
    {
        return "Mail not sent !!!";
    }
    return "Mail Sent !!!";
}

public bool ComposeEmail(Stream stream, string reportName)
{
    try
    {
        MailMessage mail = new MailMessage();
        SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
        mail.IsBodyHtml = true;
        mail.From = new MailAddress("xx@gmail.com");
        mail.To.Add("xx@gmail.com");
        mail.Subject = "Report Name : " + reportName;
        stream.Position = 0;
        if (stream != null)
        {
            ContentType ct = new ContentType();
            ct.Name = reportName + DateTime.Now.ToString() + ".pdf";
            System.Net.Mail.Attachment attachment = new System.Net.Mail.Attachment(stream, ct);
            mail.Attachments.Add(attachment);
        }
        SmtpServer.Port = 587;
```

```
SmtpServer.Credentials = new System.Net.NetworkCredential("xx@gmail.com", "xx");
SmtpServer.EnableSsl = true;
SmtpServer.Send(mail);
return true;
}
catch (Exception ex)
{
return ex.ToString();
}
return false;
}
`
```

In the above code sample, the report is exported to PDF format and sent to users using the `SmtpClient` method.

3. Build and run the application.

Custom properties

The custom properties helps you to include additional features that are not natively supported in the RDL reporting. This topic explains the list of custom properties supported in Report Viewer of React application.

See Also

- [Textbox Custom Properties](#)
- [Table Custom Properties](#)
- [Image Custom Properties](#)
- [Chart Custom Properties](#)
- [Report Custom Properties](#)
- [Parameter Custom Properties](#)
- [Export Custom Properties](#)

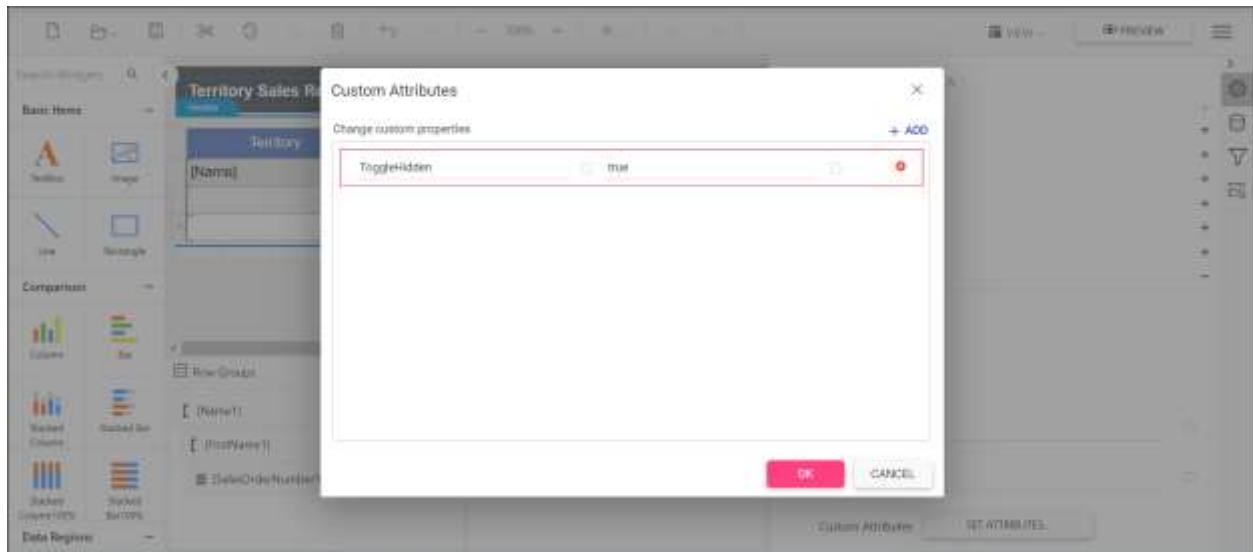
Textbox custom properties

This topic explains about the list of textbox report item custom properties that are supported to render in React Report Viewer.

Show or hide toggle icon in text box report item

The `ToggleHidden` custom property is used to show or hide the toggle icon in the textbox.

You can set the `ToggleHidden` property value, as shown in the below.



Before setting the toggle hidden property, the default value will be displayed as below.

The screenshot shows a preview of the 'Territory Sales Report'. The table has the following data:

Territory	Sales Person	Order Number	Total Sales
Australia	Lynn Tsotlias		\$1,943,016
Canada	Garnett Vargas		\$12,808,458
	José Saravia		\$4,840,689
			\$7,967,769
Central	Jillian Carson		\$13,434,510
			\$13,434,510
France	Ranjit Varkey		\$6,083,691
	Chudukatil		\$6,083,691
Germany	Rachel Valdez		\$2,476,530
			\$2,476,530
Northeast	Michael Blythe		\$12,433,503
			\$12,433,503
Northwest			\$6,305,407

Preview the report and the see the toggle icon is hidden in the report.



Territory	Sales Person	Order Number	Total Sales
Australia	Lynn Tsiftas		\$1,943,016
Canada	Garrett Vargas		\$12,808,458
	José Saraiva		\$4,840,689
Central	Jillian Carson		\$7,967,769
France	Ranjit Varkey Chudakatti		\$13,434,510
Germany	Rachel Valdez		\$13,434,510
Northeast			\$6,083,691
Northwest			\$6,083,691
			\$2,476,530
			\$2,476,530
			\$12,433,503
			\$12,433,503
			\$6,306,407

Table custom properties

This topic explains about the list of table report item custom properties that are supported to render in Report Viewer of React application.

Limit number of table records on each page

The `RowsPerPage` custom property is used to specify the number of table records to display on each page. It supports integer data value greater than zero.

This property is ignored when table rows heights higher than current page size. Increase the report page height or reduce `RowsPerPage` count that fits within the page.

You can set the `RowsPerPage` property value, as shown in the below.

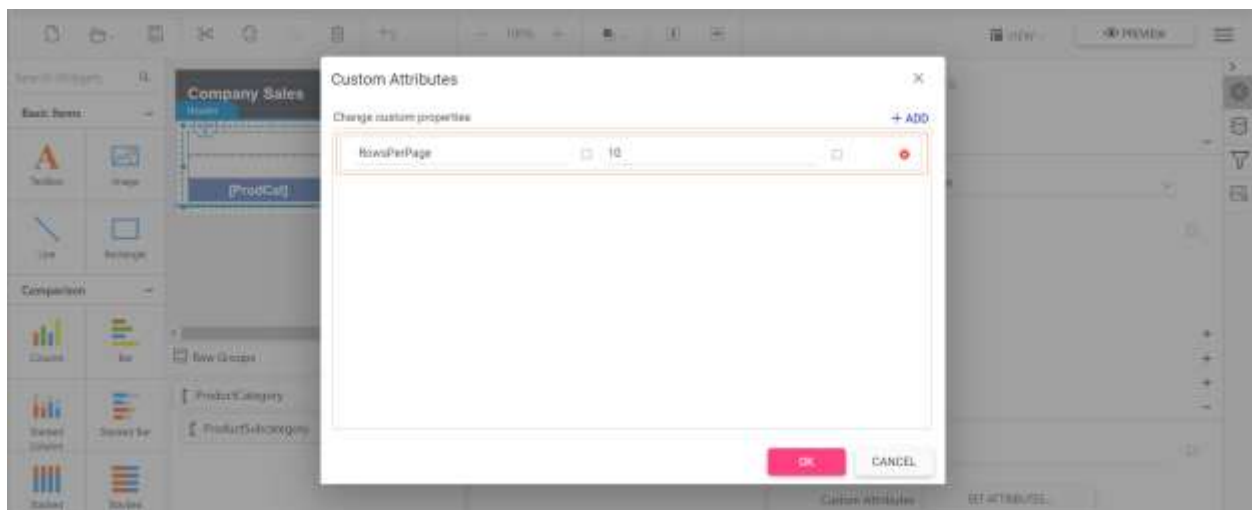
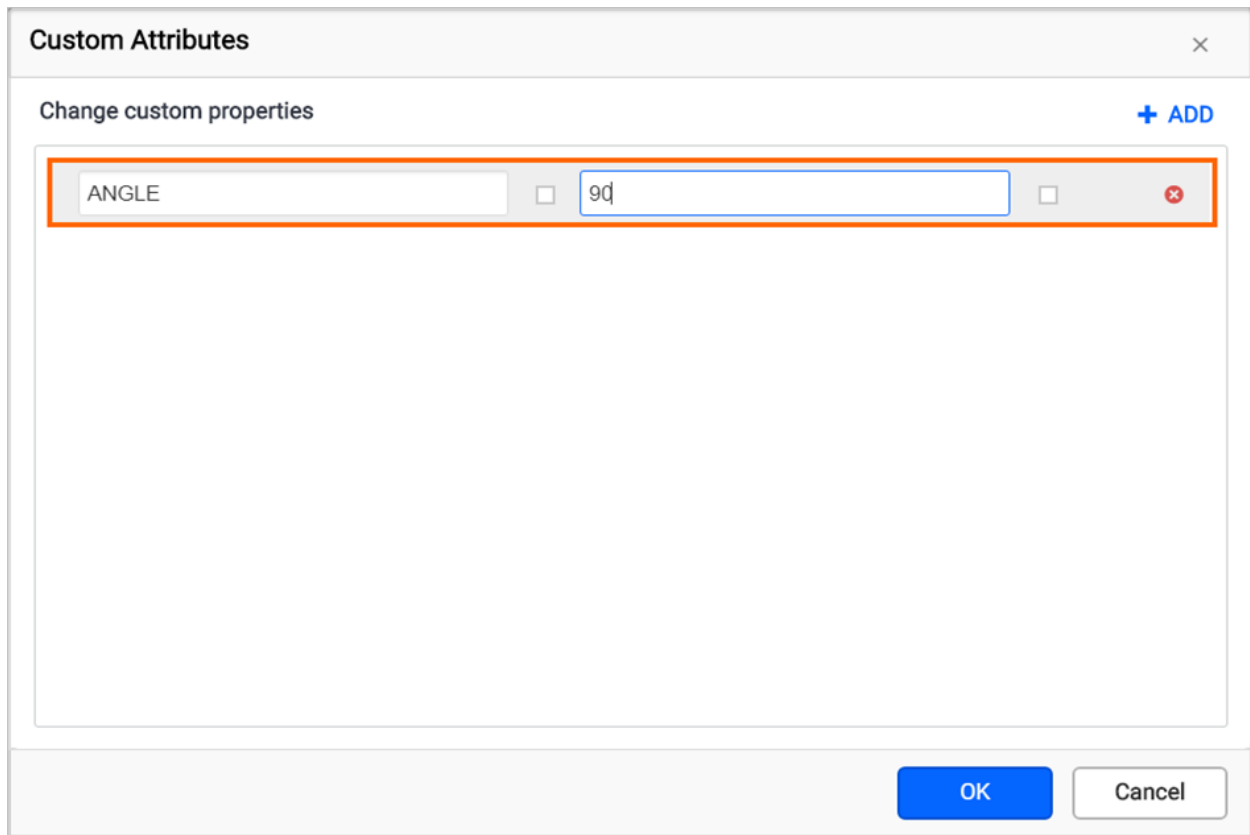


Image custom properties

This topic explains about the list of image custom properties that are supported to render in the React Report Viewer.

Angle

Set **Angle** custom property to image report item to rotate the image on a specified angle. It supports the angle values 0, 90, 180, and 270. You can set the property value, as shown in the below.



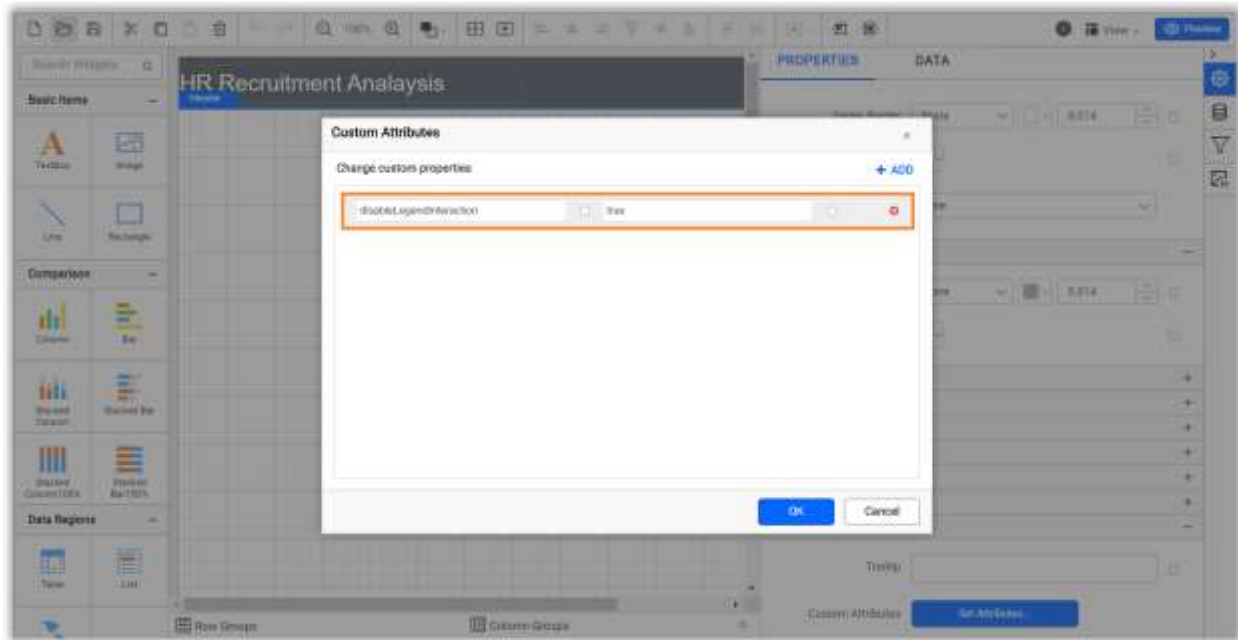
The screenshot shows a 'Custom Attributes' dialog box. At the top, there's a title bar with a close button. Below it, the 'Change custom properties' section is visible, featuring a '+ ADD' button. A table-like structure contains one row with the property name 'ANGLE' and its value '90'. The value '90' is highlighted with a blue border. At the bottom of the dialog, there are 'OK' and 'Cancel' buttons.

Chart custom properties

This topic explains about the list of chart custom properties that are supported to render in the React Report Viewer.

Disable legend item interaction

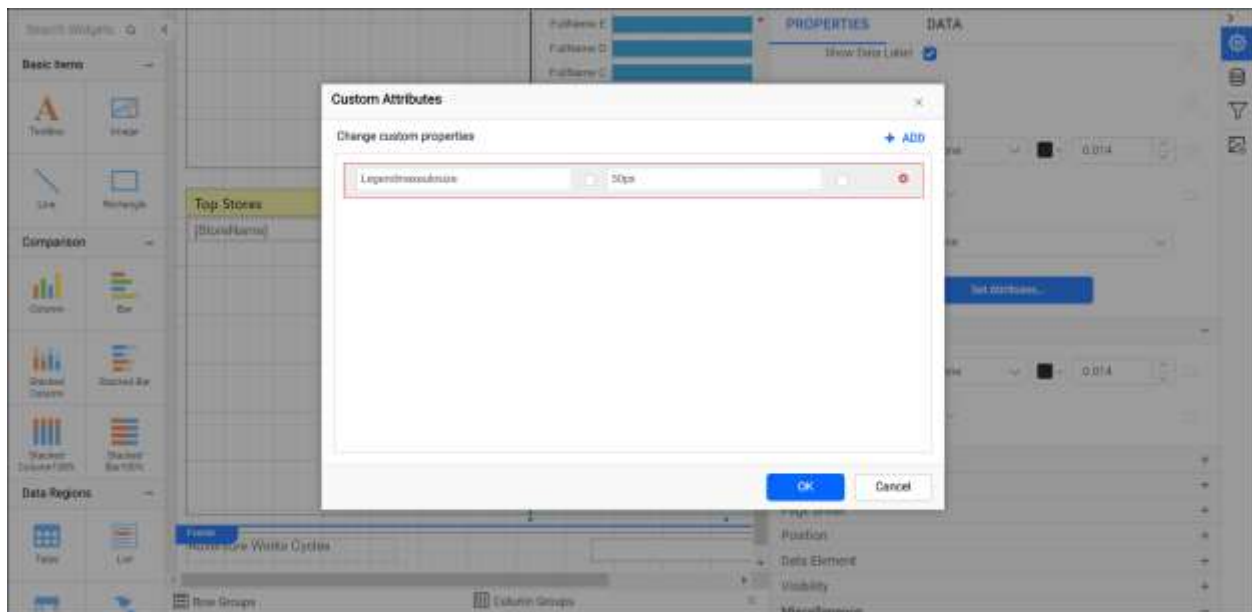
Set **DisableLegendInteraction** custom property value as **true** to stop the legend item interaction. The property value should be boolean. You can set the property value, as shown in the below.



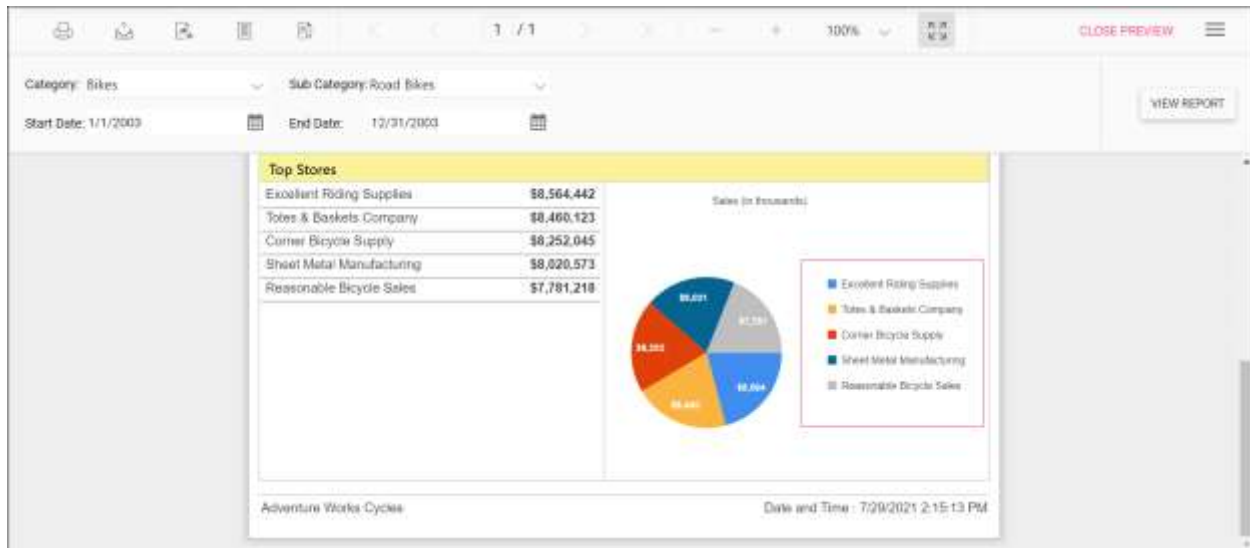
Set maximum size for chart legend

The `LegendMaxAutoSize` custom property specifies the maximum size of the legend container in the report.

You can set the `LegendMaxAutoSize` property value, as shown in the below.



Preview the report and the see legend container size in chart report.

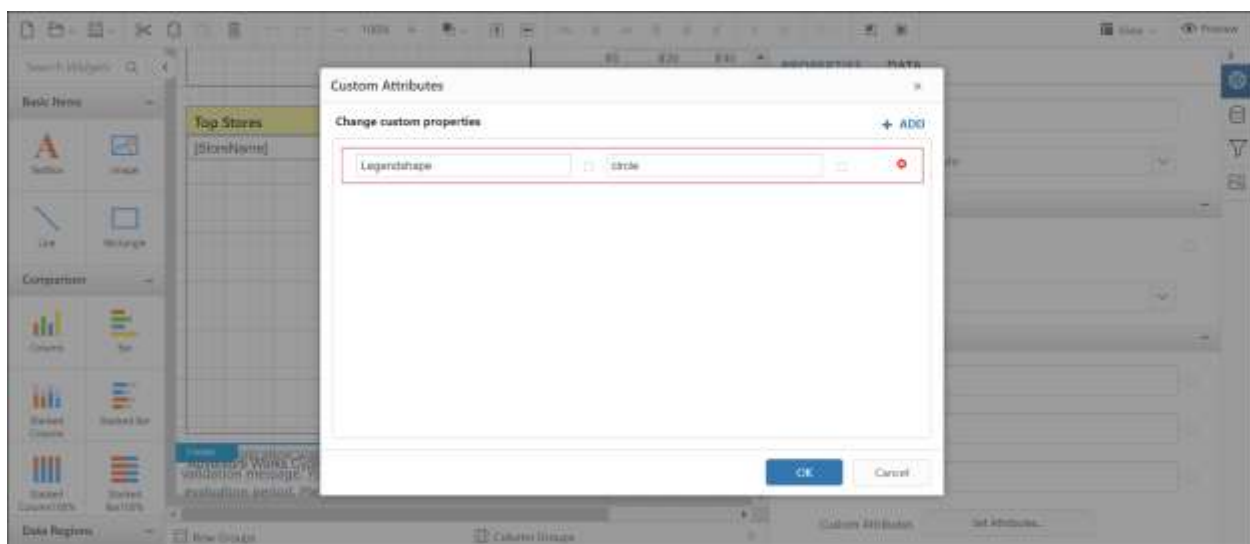


Change chart legend shape

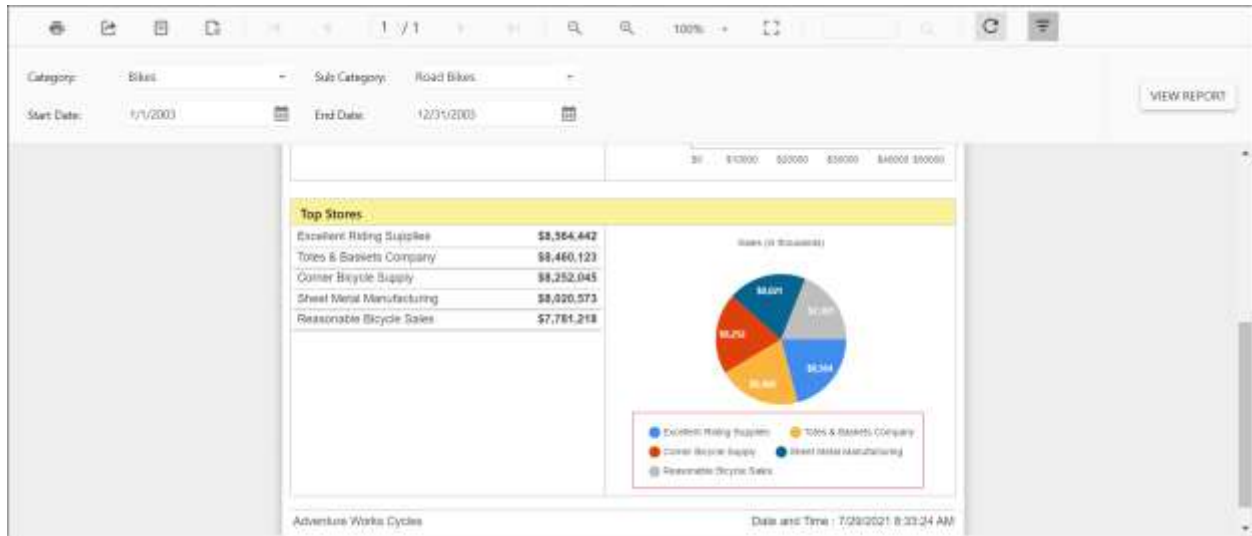
The **LegendShape** custom property allows changing the shape of the legend in the chart report item. By default, the **LegendShape** value is **rectangle**.

- Rectangle - legend shapes displayed in **rectangle**.
- Circle - legends are displayed in **circle**.
- Thumbnail - legends are displayed in **seriestype**.

You can set the **LegendShape** property value, as shown in the below.



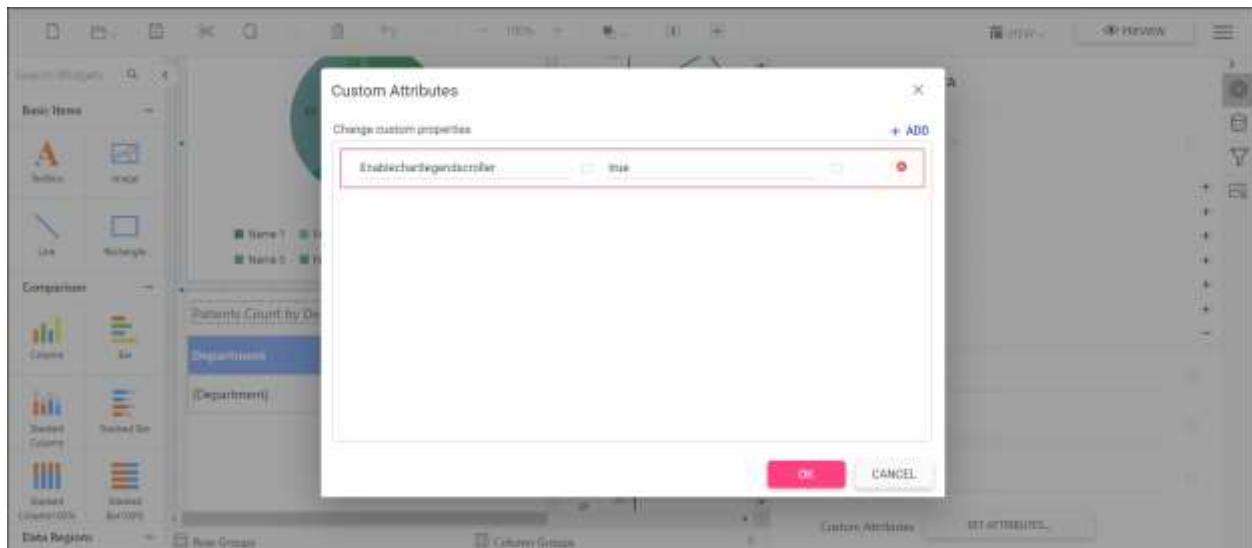
Preview the report and then see the legend shape in the chart report.



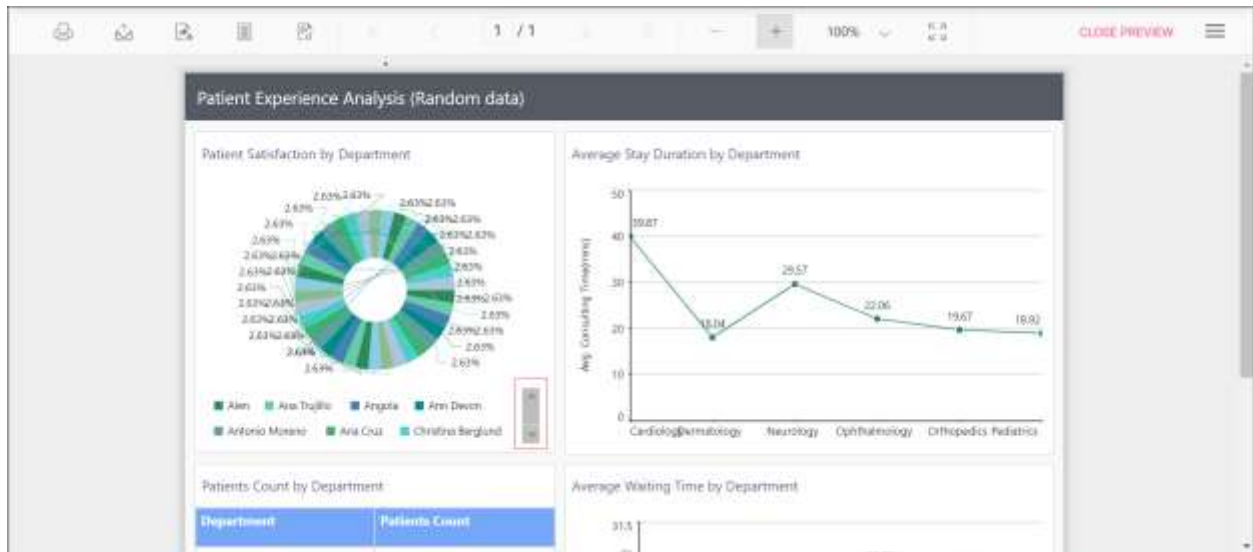
Show or hide chart legend scroller

The `EnableChartLegendScroller` custom property controls whether legend has to use scrollbar or not. The scrollbar appears depending upon size and position properties of legend. By default, the `EnableChartLegendScroller` value is `false`.

You can set the `EnableChartLegendScroller` property value, as shown in the below.



Preview the report and the see the legend scrollbar in chart report.



Set range padding for X-axis and Y-axis

Padding can be applied to the minimum and maximum extremes of the axis range by using the `XAxisRangePadding` and `YAxisRangePadding` property. The default value is `none`.

Numeric axis supports the following types of padding.

Name | Description

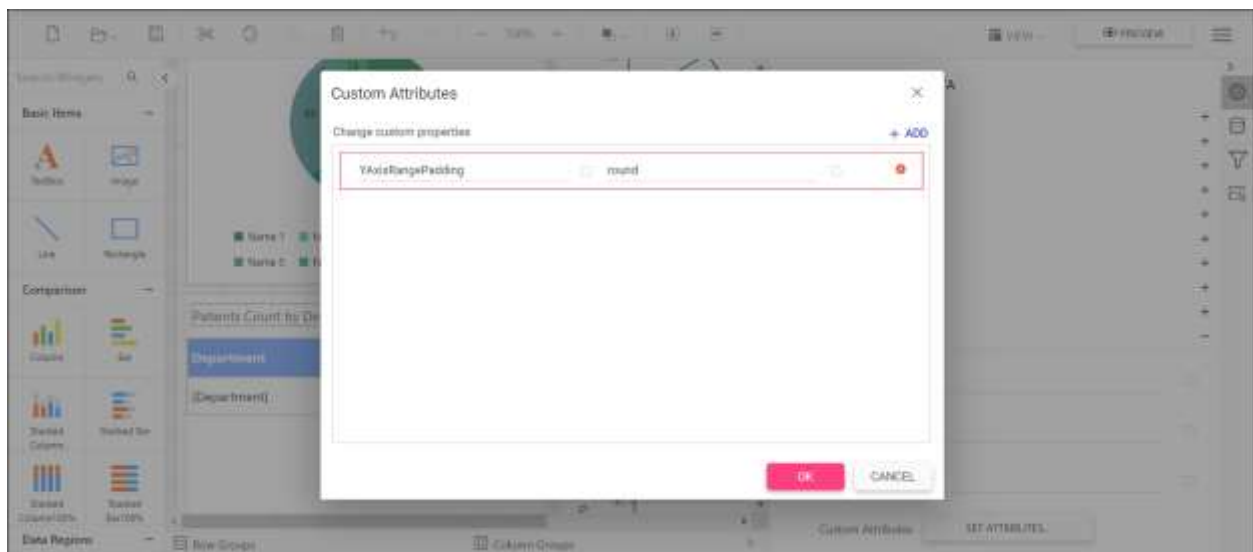
Additional | Interval of the axis is added as padding to the minimum and maximum values of the range

Normal | Padding is applied to the axis based on the range calculation

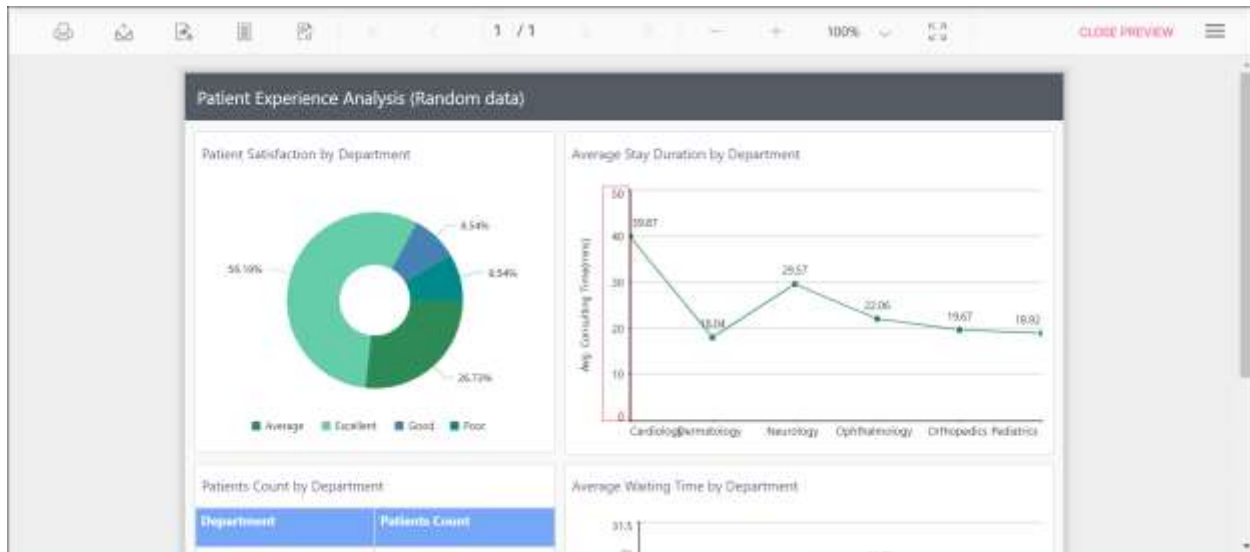
None | Padding cannot be applied to the axis

Round | Axis range is rounded to the nearest possible value divided by the interval

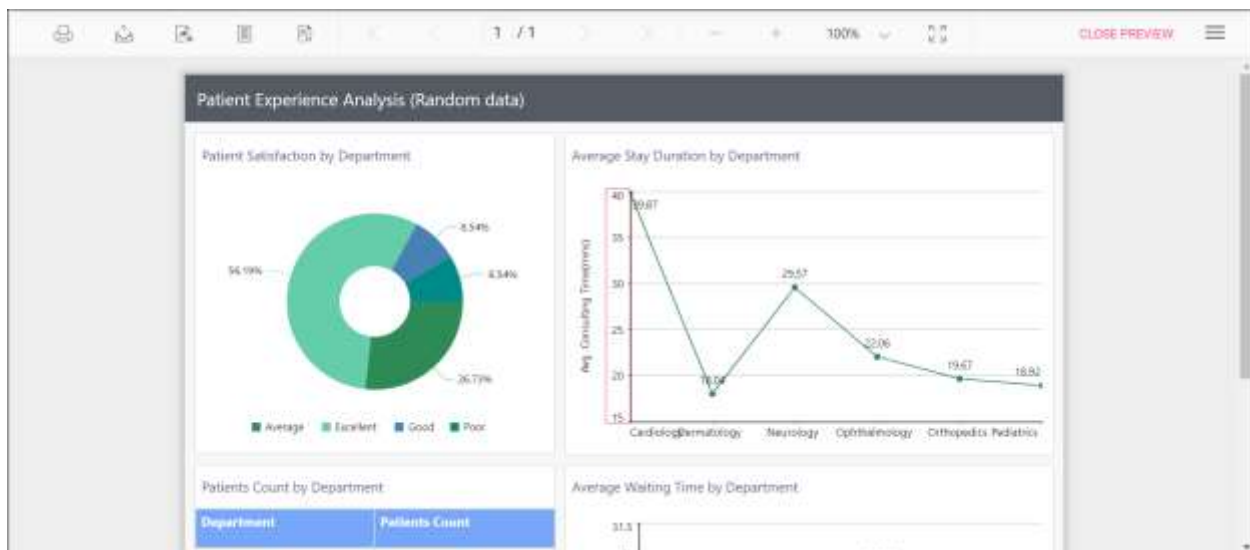
You can set the `XAxisRangePadding` and `YAxisRangePadding` property value, as shown in the below.



Before setting the range padding, the default value will be displayed as below.



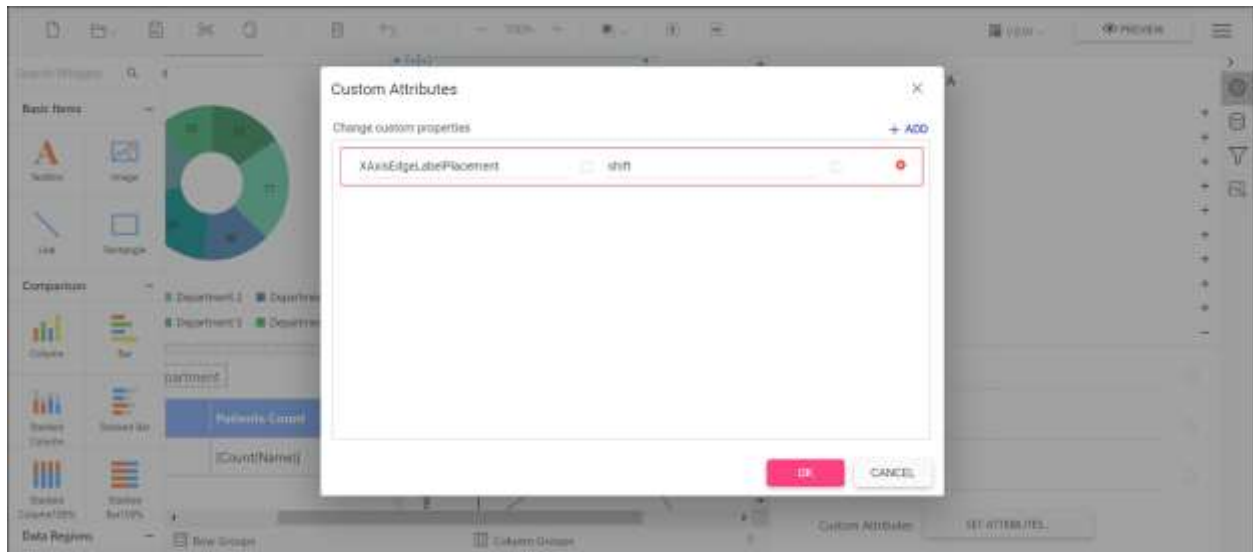
Set the padding range and see the changes in chart report as below.



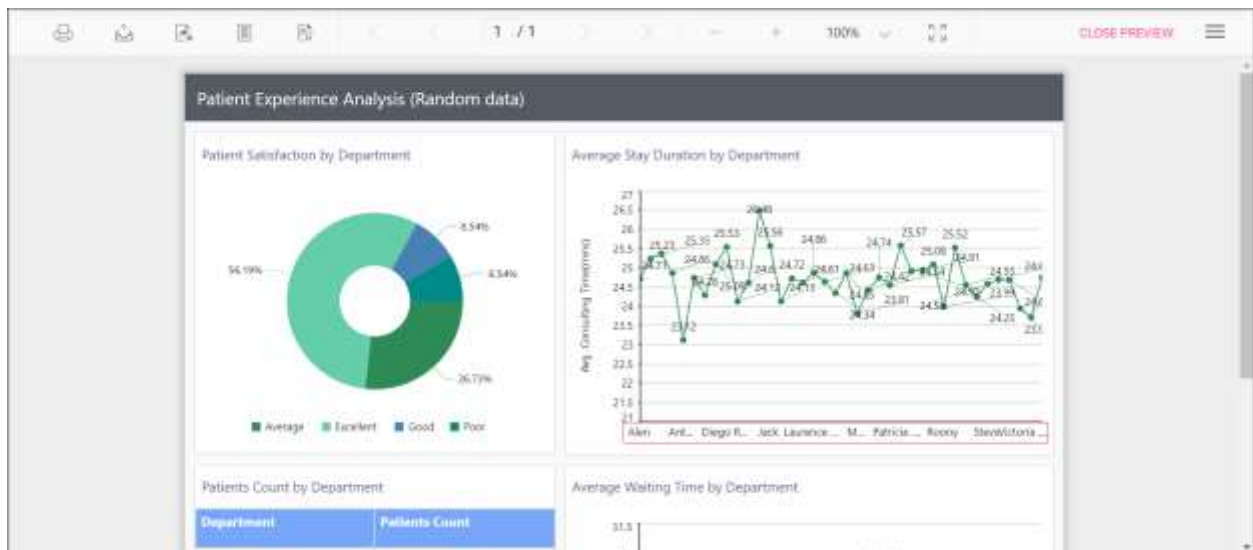
Control edge label placement in chart axis

Labels with long text at the edges of an axis may appear partially outside the chart. The `XAxisEdgeLabelPlacement` and `YAxisEdgeLabelPlacement` custom property can be used to avoid the partial appearance of the labels at the corners. The default value is `none`.

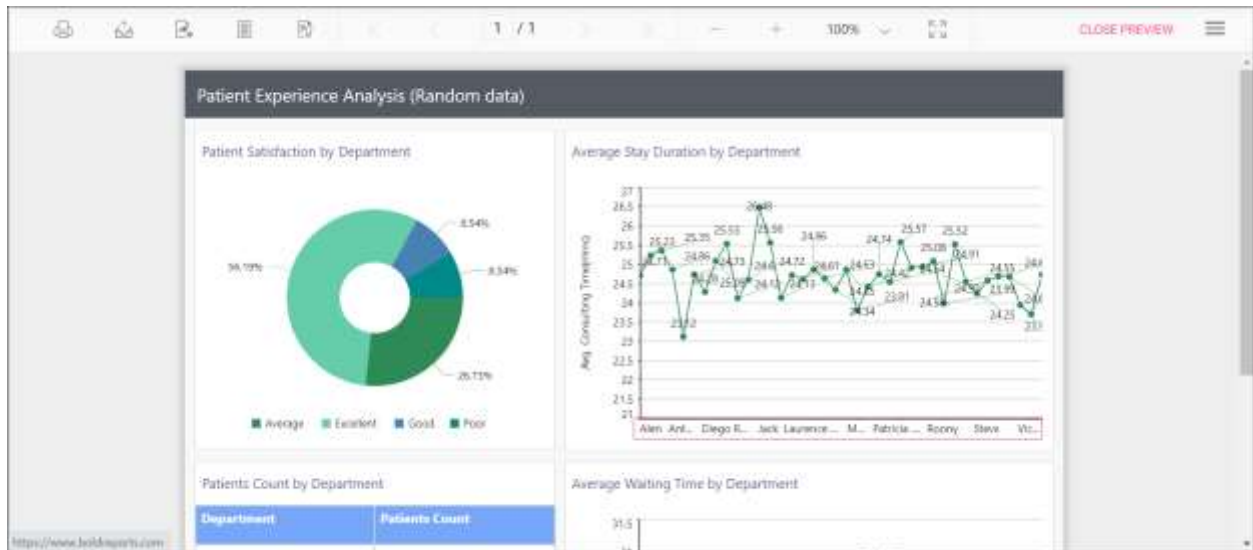
You can set the `XAxisEdgeLabelPlacement` property value, as shown in the below.



Before setting the edge label placement, the default value will be displayed as below.



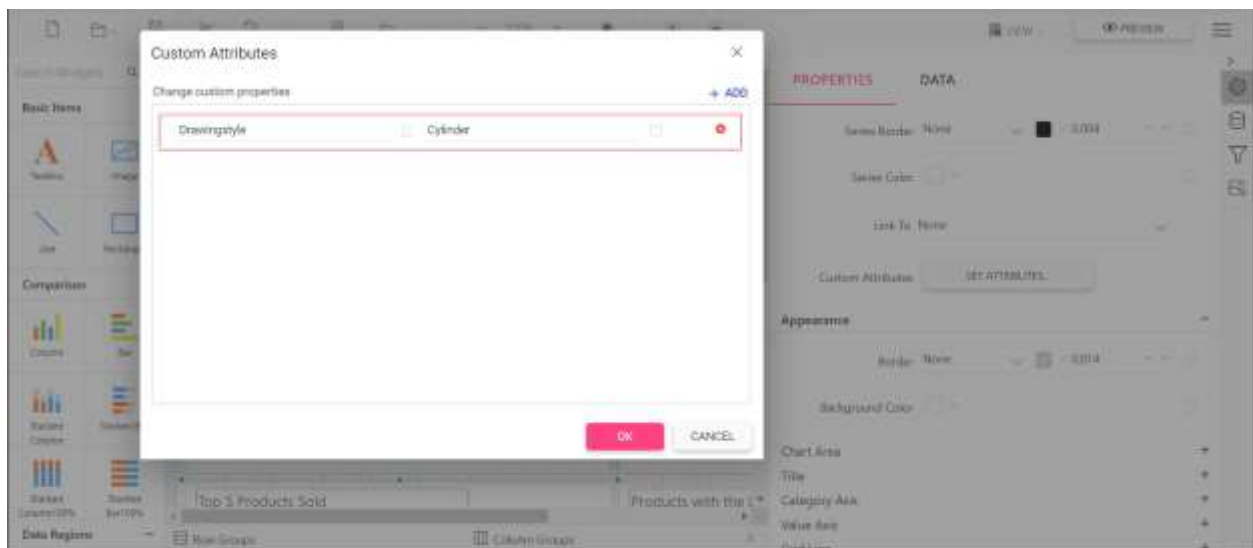
Set the edge label placement and see the changes in chart report as below.



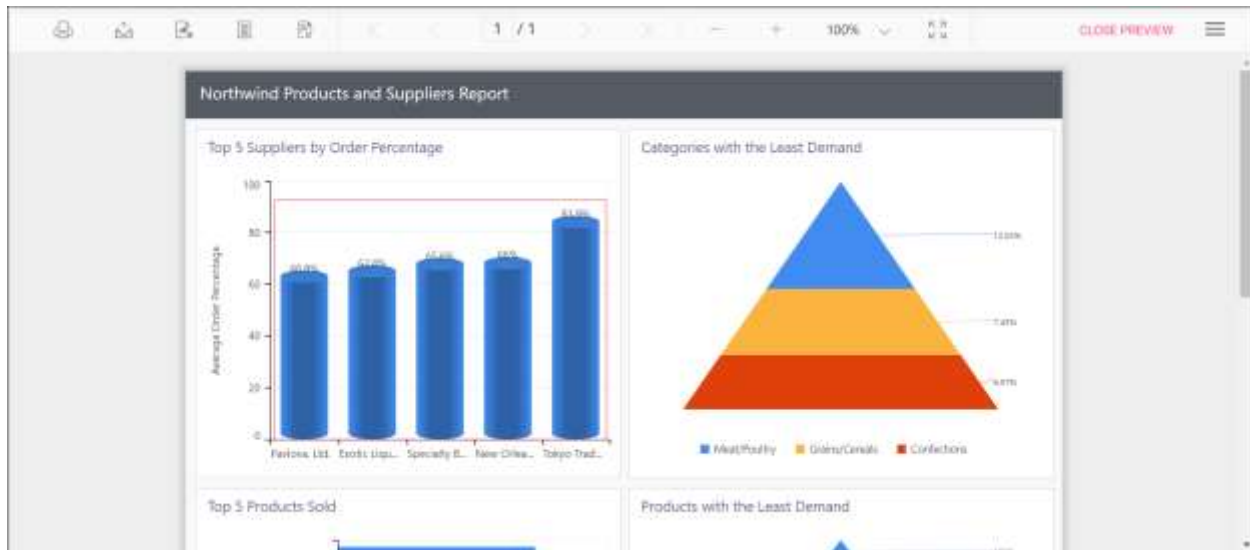
Change the drawing style of a chart column or bar series

The shape of the chart column or bar can be changed using this **Drawingstyle** custom property. By default, the **Drawingstyle** property value is **rectangle**.

You can set the **Drawingstyle** property value, as shown in the below.



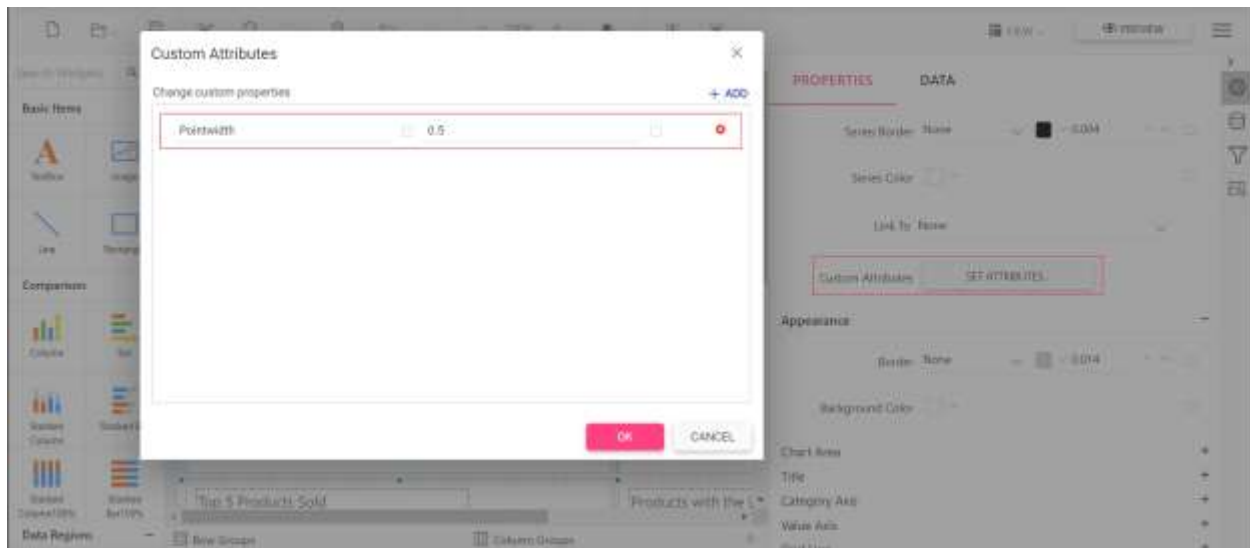
Preview the report and the see the column or bar shape in chart report.



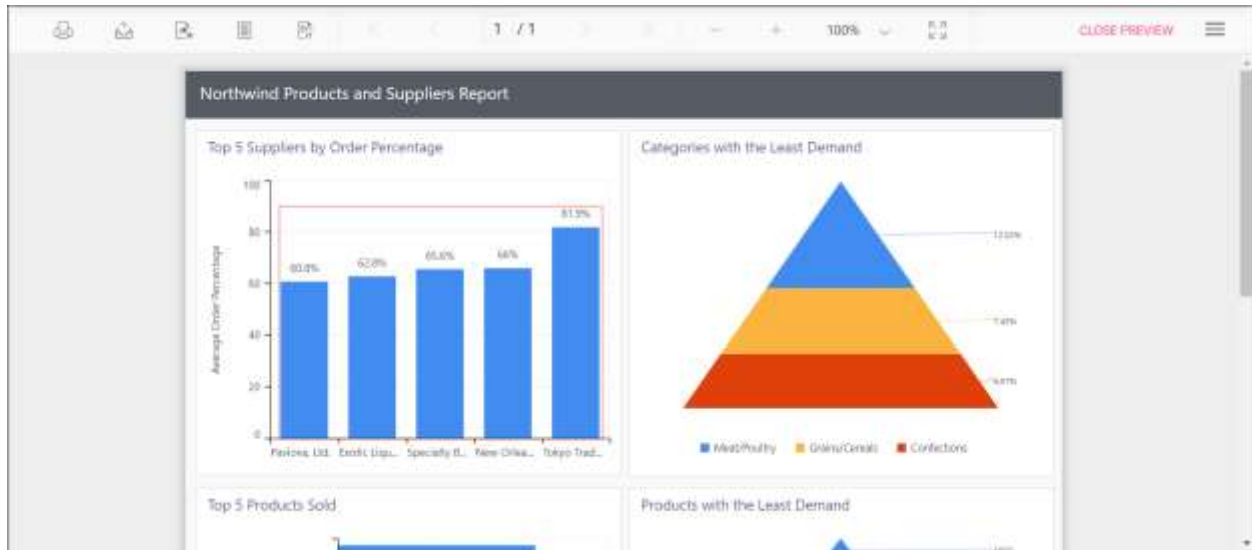
Change width of the column type series in chart

Width of the column type series can be customized by using the **Pointwidth** property. Default value of **Pointwidth** is 0.7. Value ranges from 0 to 1. Here 1 corresponds to 100% of available width and 0 corresponds to 0% of available width.

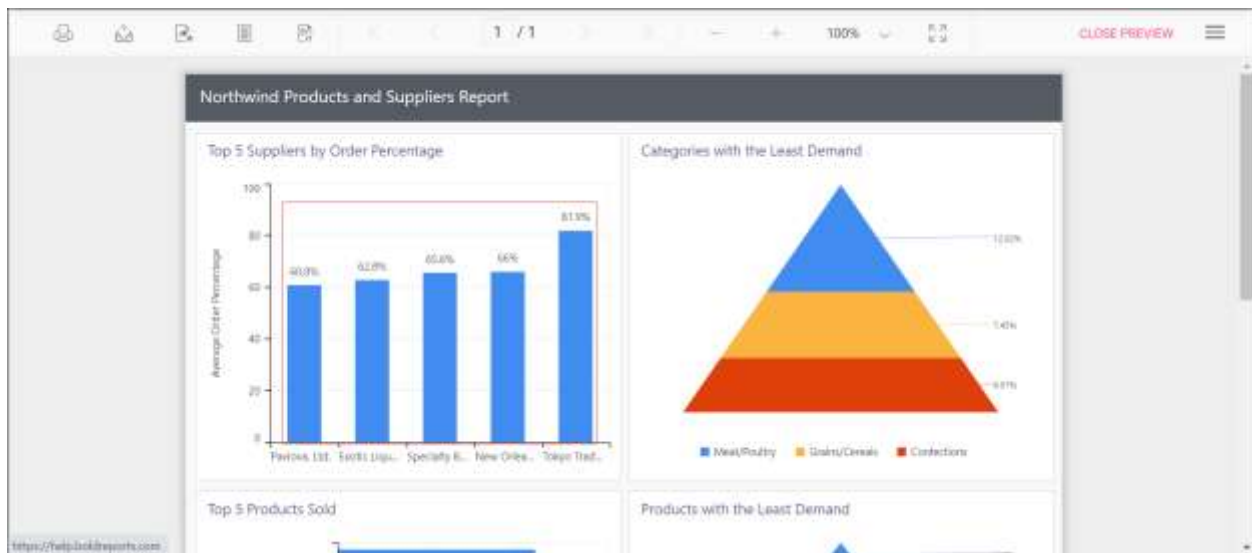
You can set the **Pointwidth** property value, as shown in the below.



Before setting the point width, the default value will be displayed as below.



Preview the report and the see the column width in chart report.



Report custom properties

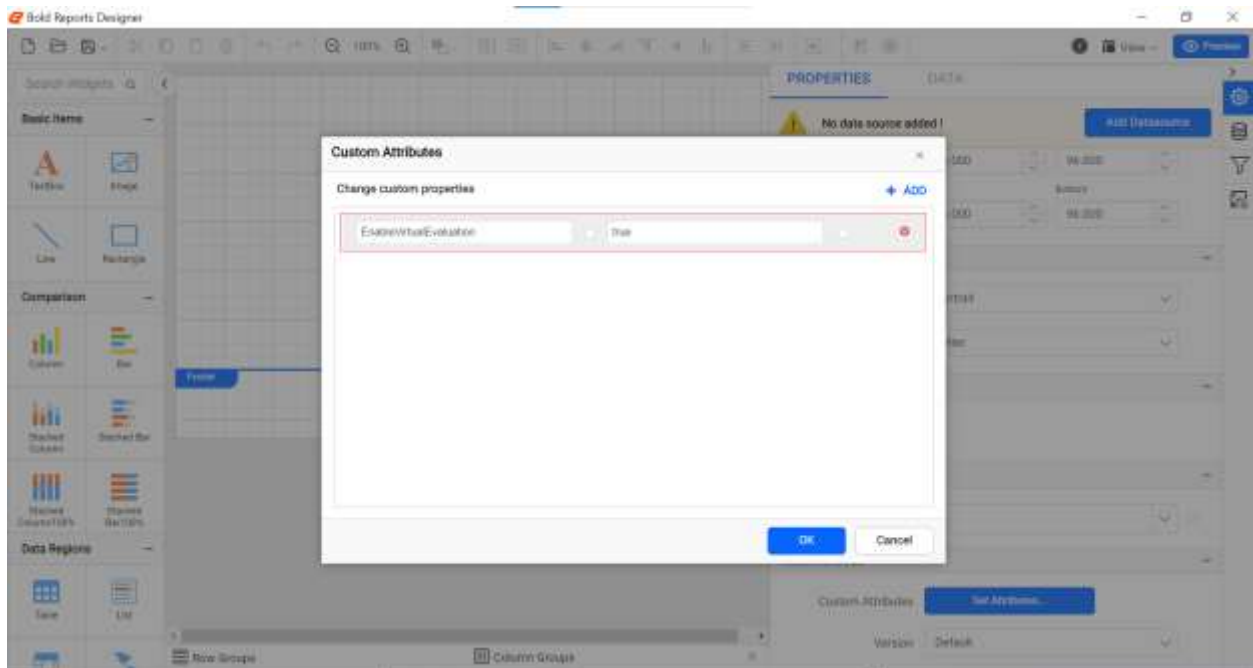
This topic explains about the list of report custom properties that are supported to render in React Report Viewer.

Improve performance and handle large amount of data

Set the `EnableVirtualEvaluation` and `DisablePageSplitting` custom properties in a report to improve performance and handle the larger amount of data with less memory footprint.

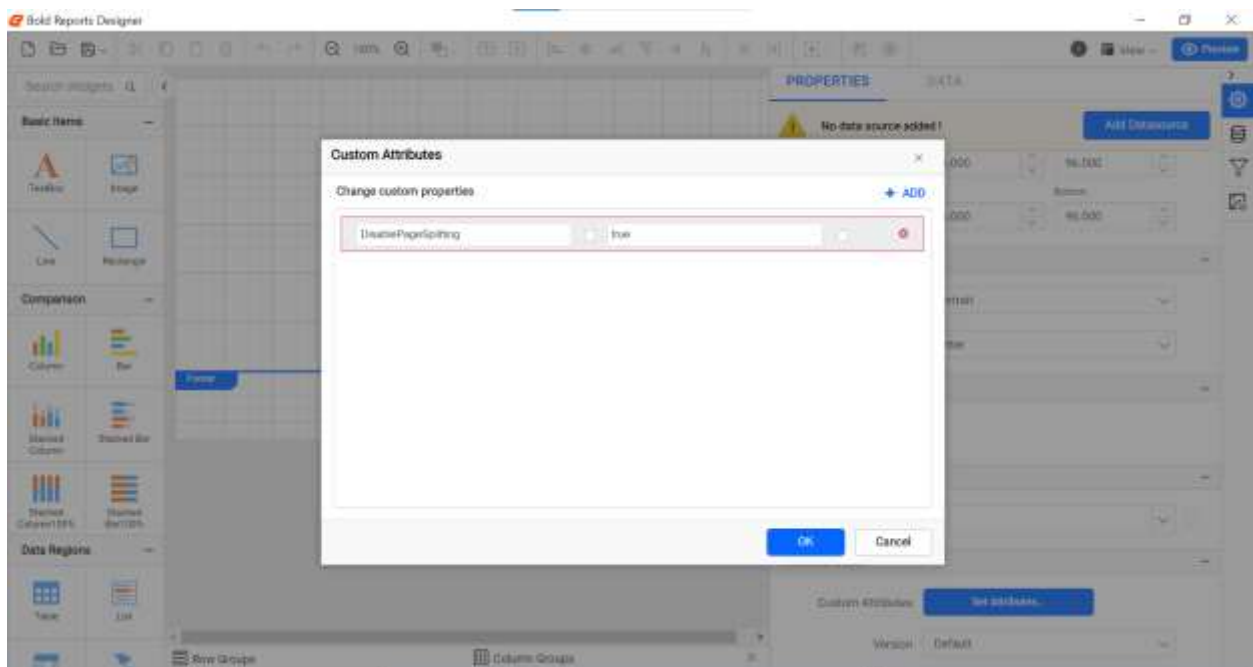
Render large data report faster

The `EnableVirtualEvaluation` custom property is used to render the large data report faster. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Reduce memory footprint for large data report

The `DisablePageSplitting` custom property is used to reduce the memory footprint for large data report. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Improve report items layout and avoid extra blank pages

When the `RoundLayoutMeasures` property is false, all non-integral values that are calculated during the report processing are rounded to whole pixel values. It provides following improvements,

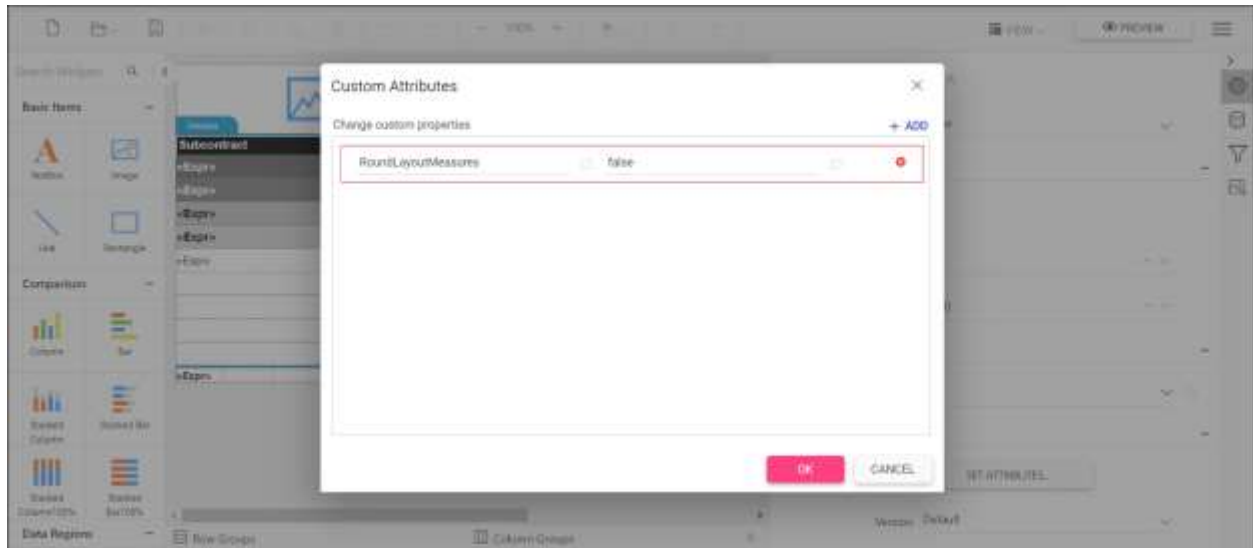
- Report text rendered without cuts.

Report custom properties
timeouts

Handling failure in long-running HTTP requests, report processes, and

- Eliminates extra blank pages.
- Removes item and text overlaps.
- Eliminates the blur semi-transparent edges that are produced by anti-aliasing.
- Produces identical look in report view and export output.

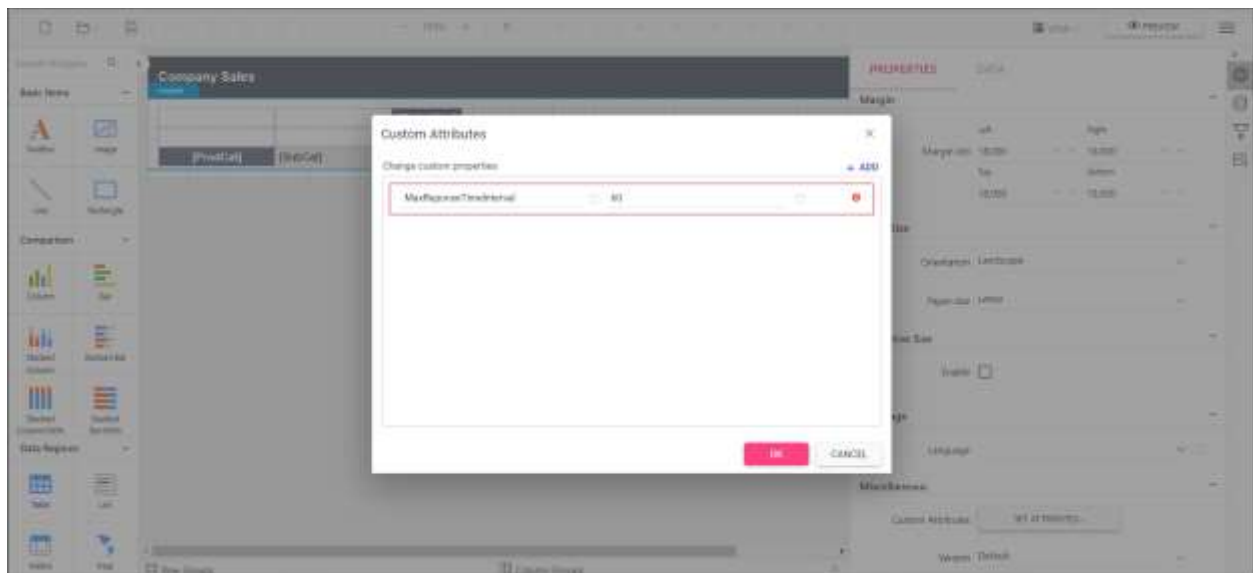
You can set the property value, as shown in the below.



Handling failure in long-running HTTP requests, report processes, and timeouts

When a report process for a long time due to huge records or a HTTP request takes too long to respond then it results in Gateway Timeout or report rendering errors. The `MaxResponseTimeInterval` allows to specify the seconds to process an HTTP request and respond back. It helps to keep the client and server connection live by avoiding the timeout.

You can set the property value as shown in the below.



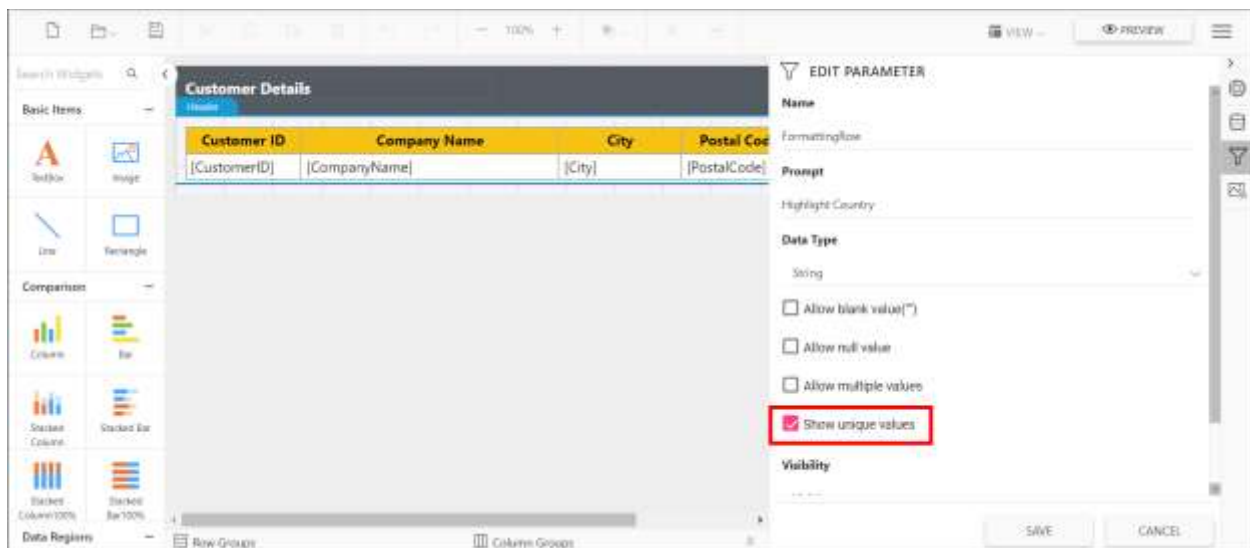
Parameter custom properties

This topic explains about the list of parameter custom properties that are supported to customize the reports preview in Report Viewer.

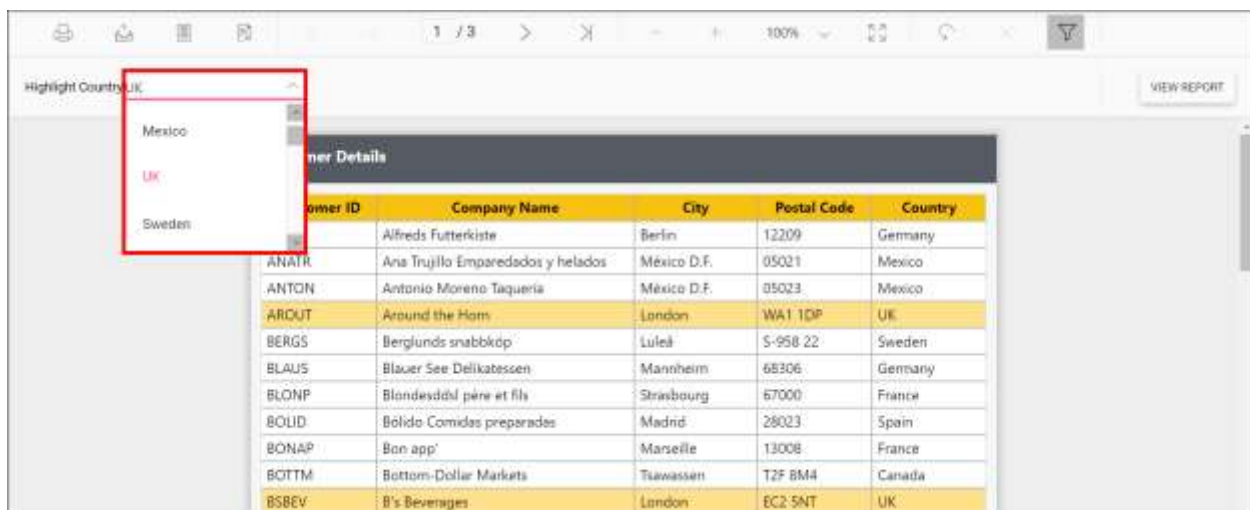
Show only distinct values in parameter

A parameter created with data set query values may contain duplicate values. The Report Viewer supports `UniqueValueParameters` custom property that helps show only unique values in the parameter drop-down while viewing the report, without creating new a data set. Refer to the below image for property setting option.

You can also provide the parameter names with comma (,) separator to set for multiple parameters.



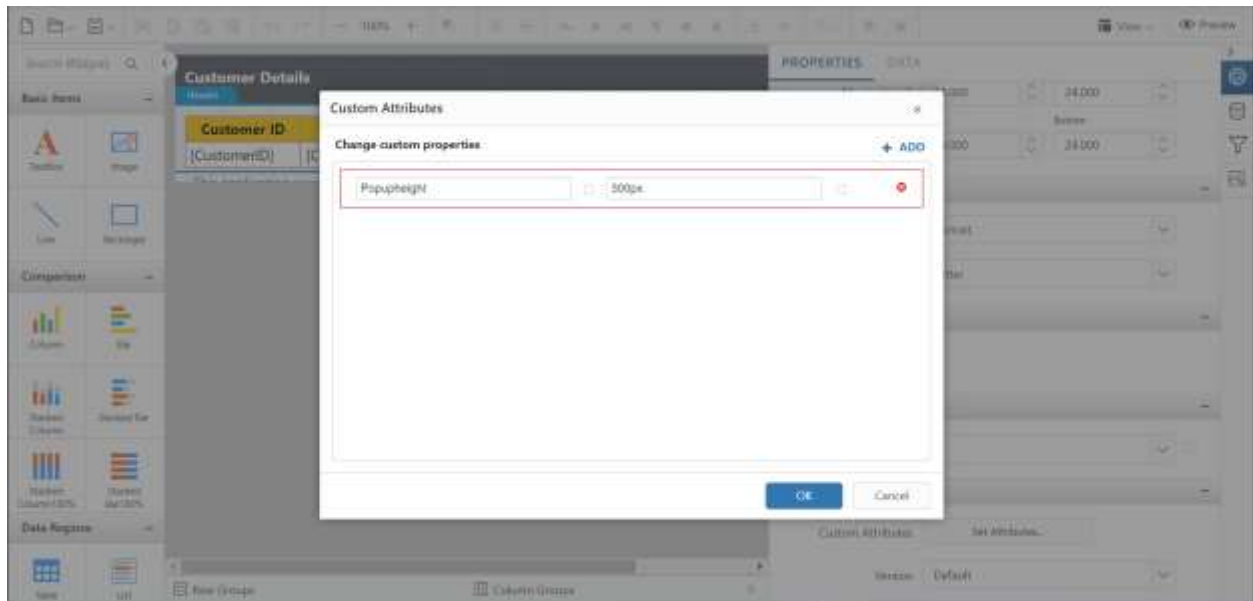
Preview the report and the unique values showed in the parameter drop down.



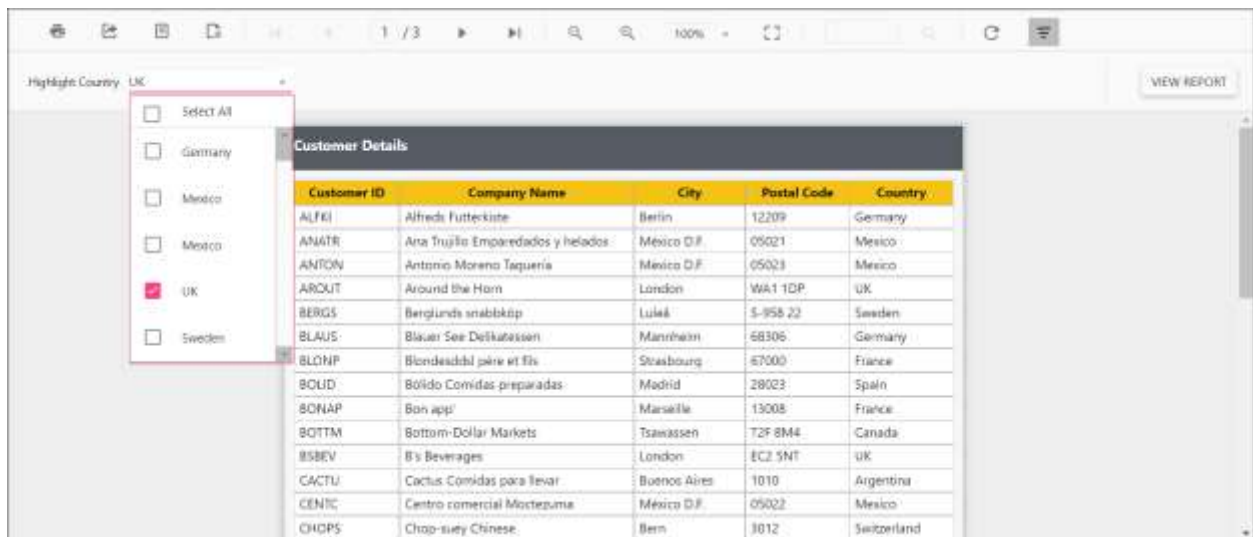
Change the dropdown parameter pop up height value

The `PopupHeight` custom property specifies the height of the parameter combobox popup list in the report. By default, the `PopupHeight` value is 152px.

You can set the `PopupHeight` property value, as shown in the below.



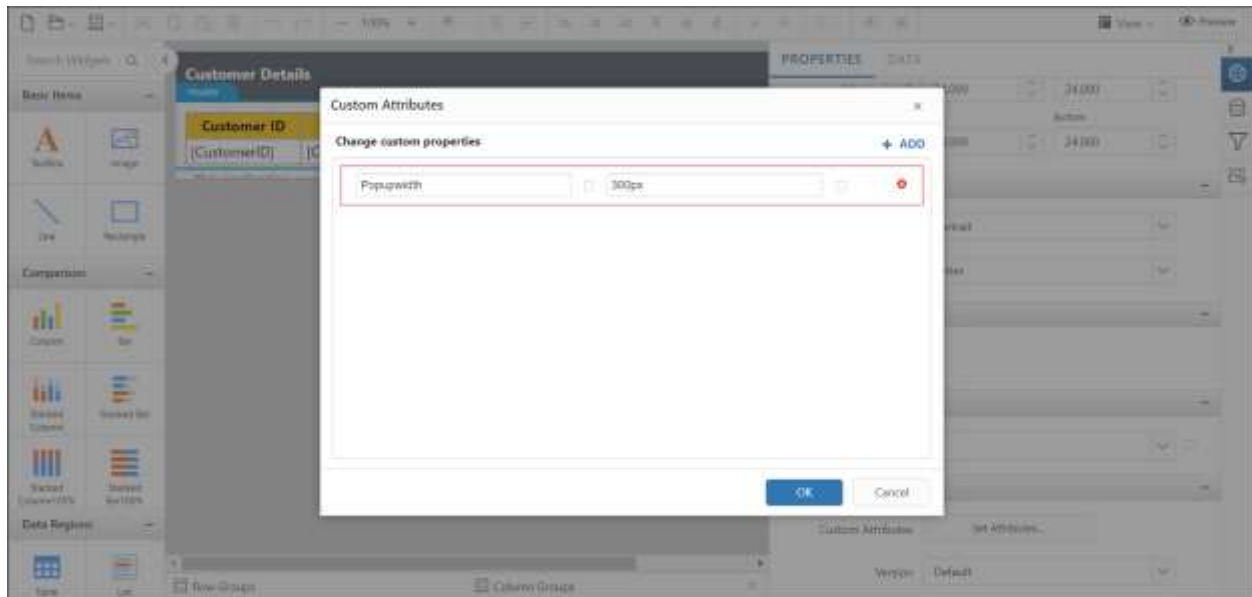
Preview the report and the pop up height showed in the parameter drop down.



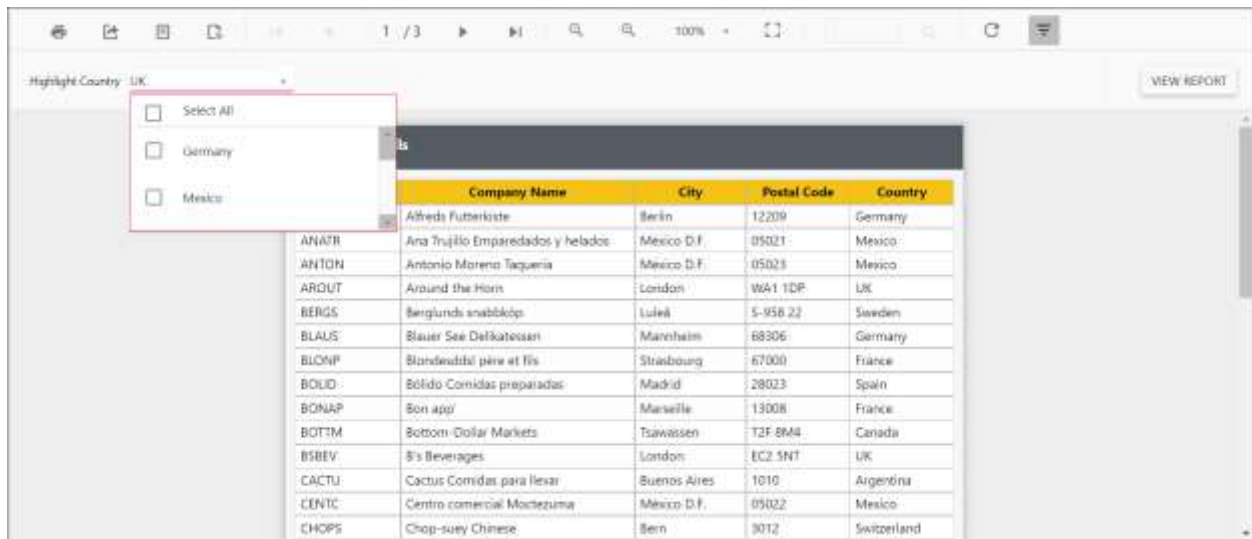
Change the dropdown parameter pop up width value

The **PopupWidth** custom property specifies the width of the parameter combobox popup list in the report. By default, the popup width sets based on the width of the component.

You can set the **PopupWidth** property value, as shown in the below.



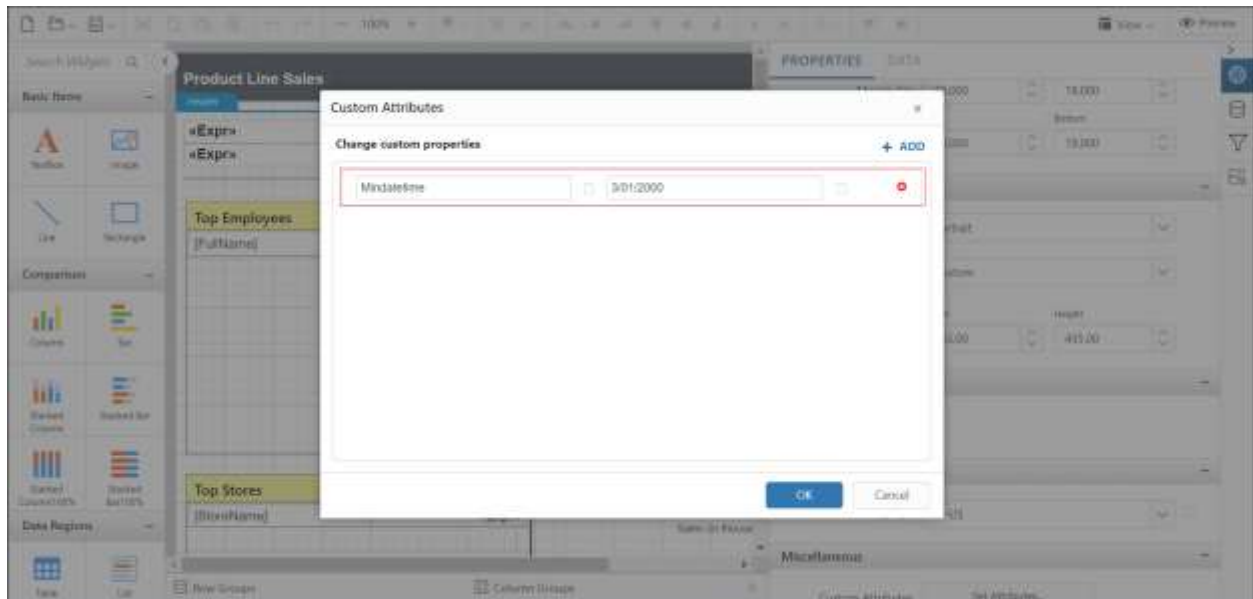
Preview the report and the pop up width showed in the parameter drop down.



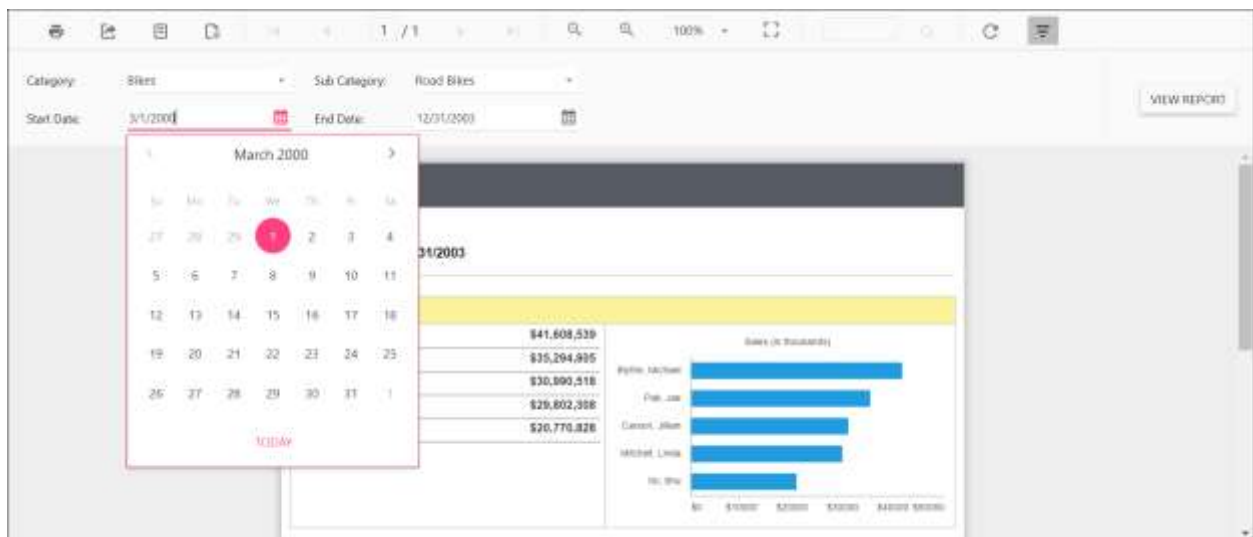
Set the minimum date range for the date report parameter

The `MinDateTime` custom property specifies the minimum date in the datetime parameter item. By default, the `MinDateTime` value is set as `null`.

You can set the `MinDateTime` property value, as shown in the below.



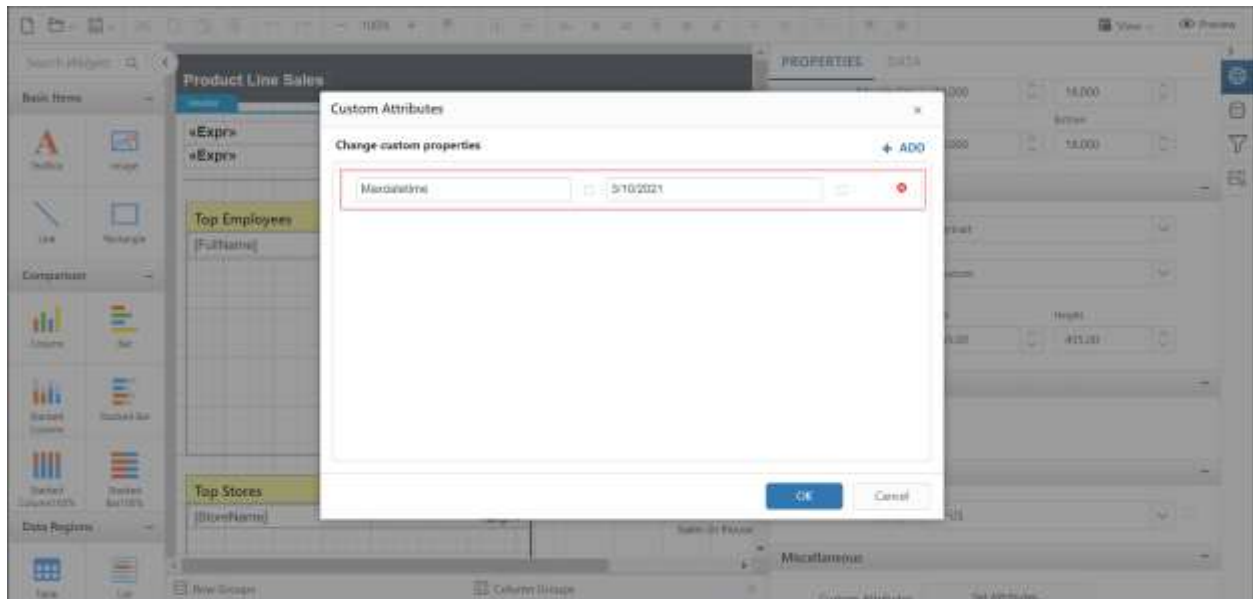
Preview the report and the minimum date showed in the datetime parameter drop down.



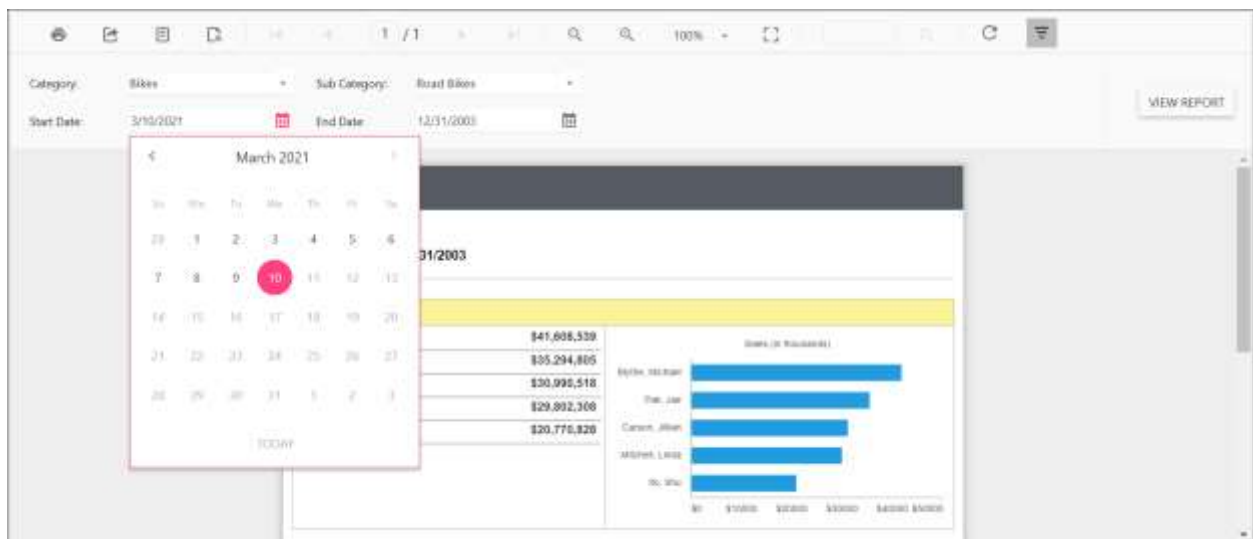
Set the maximum date range for date report parameter

The `MaxDateTime` custom property specifies the maximum date in the datetime parameter item. By default, the `MaxDateTime` value is set as `null`.

You can set the `MaxDateTime` property value, as shown in the below.



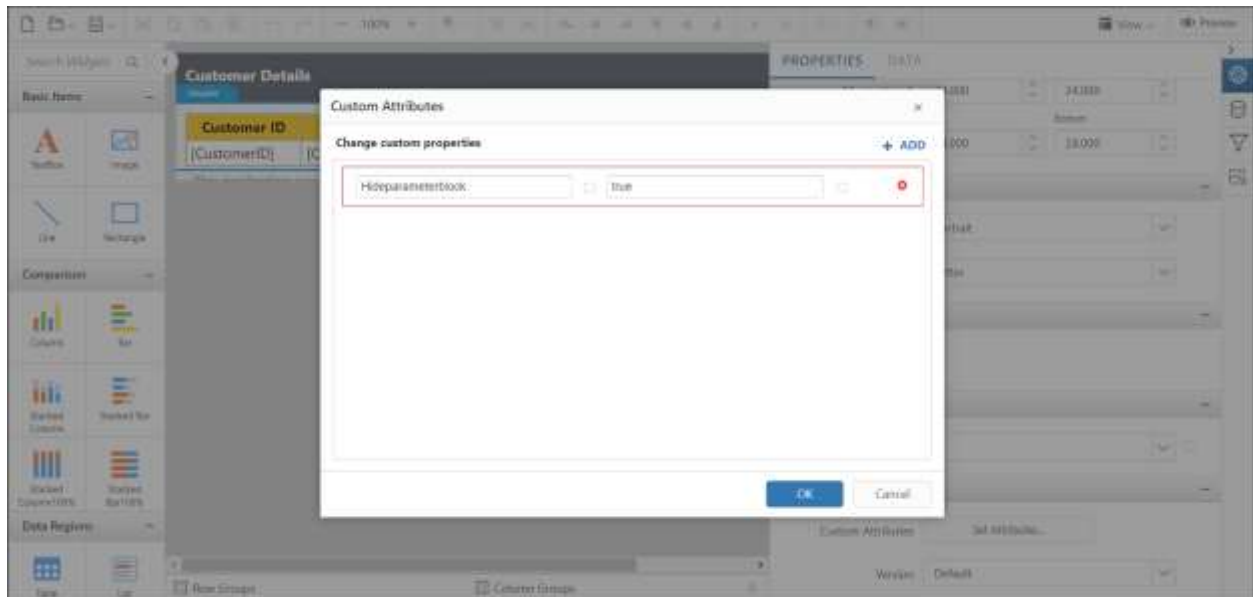
Preview the report and the maximum date showed in the datetime parameter drop down.



Hide the report parameter block

The `HideParameterBlock` custom property is used to hide the parameter block on report initial rendering. The property value should be boolean. By default, the `HideParameterBlock` value is `false`.

You can set the `HideParameterBlock` property value, as shown in the below.



Preview the report and see the parameter block is hidden.

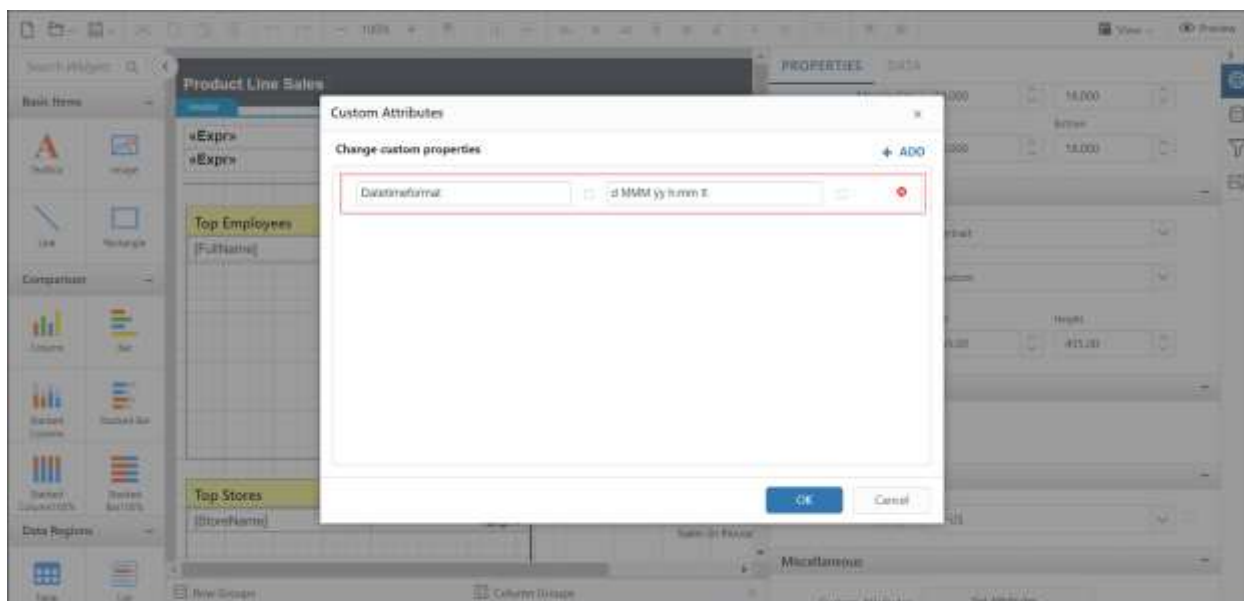
The screenshot shows a report preview titled 'Customer Details'. It displays a table with the following columns: Customer ID, Company Name, City, Postal Code, and Country. The table contains 10 rows of data, showing various companies and their locations across different countries.

Customer ID	Company Name	City	Postal Code	Country
ALFKI	Alfreds Futterkiste	Berlin	12209	Germany
ANATR	Ana Trujillo Emparedados y helados	México D.F.	05021	Mexico
ANTON	Antonio Moreno Taquería	México D.F.	05023	Mexico
AROUT	Around the Horn	London	WA1 1DP	UK
BERGS	Berglunds snabbköp	Luleå	S-958 22	Sweden
BLAUS	Blaauw Sea Delicatessen	Mannheim	68306	Germany
BONAP	Bonaparte's pizzeria	Strasbourg	67000	France
BOLID	Bolid Comidas preparadas	Madrid	28023	Spain
BONAP	Bon app'	Marseille	13008	France
BOITM	Bottom-Dollar Markets	Toronto	M5V 3M4	Canada
BSBEV	B's Beverages	London	EC2 5NT	UK
CACTU	Cactus Comidas para llevar	Buenos Aires	1010	Argentina
CENTC	Centro comercial Mochizuki	México D.F.	05022	Mexico
CHOPS	Chop-icey Chinese	Bern	3012	Switzerland
COMM1	Comércio Mineiro	São Paulo	05432-043	Brazil
CONSH	Consolidated Holdings	London	WX1 6LT	UK
DRACD	Drachendorff & Pfeiffer	Dresden	81068	Germany

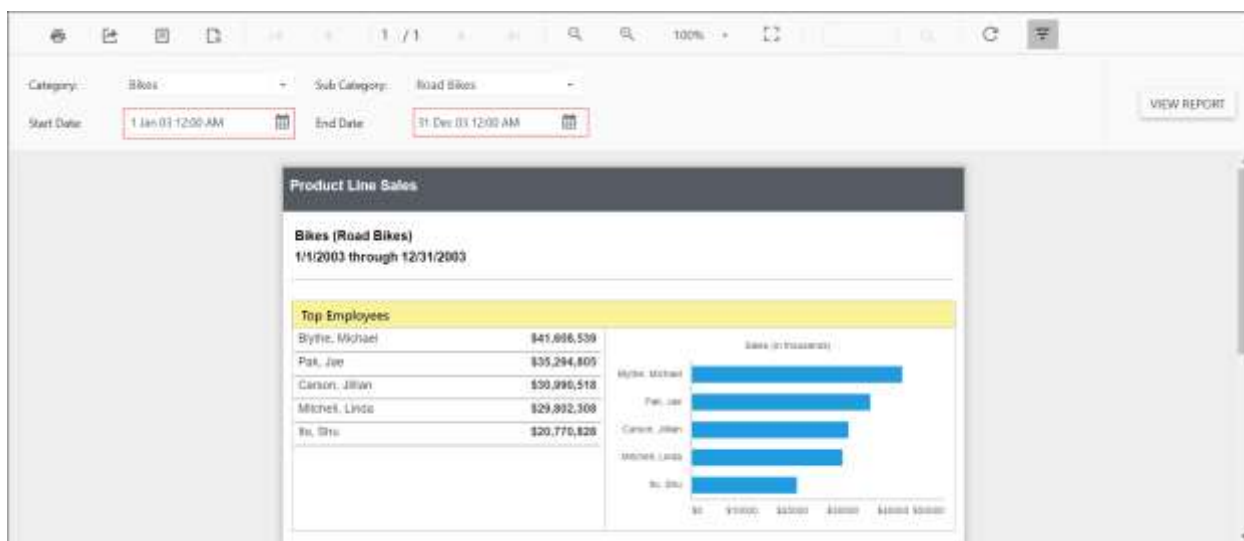
Set date and time format for parameter

The `DateTimeFormat` custom property defines the date time format to be displayed in the `DateTimePicker` popup. By default, the `DateTimeFormat` value is set as `empty`.

You can set the `DateTimeFormat` property value, as shown in the below.



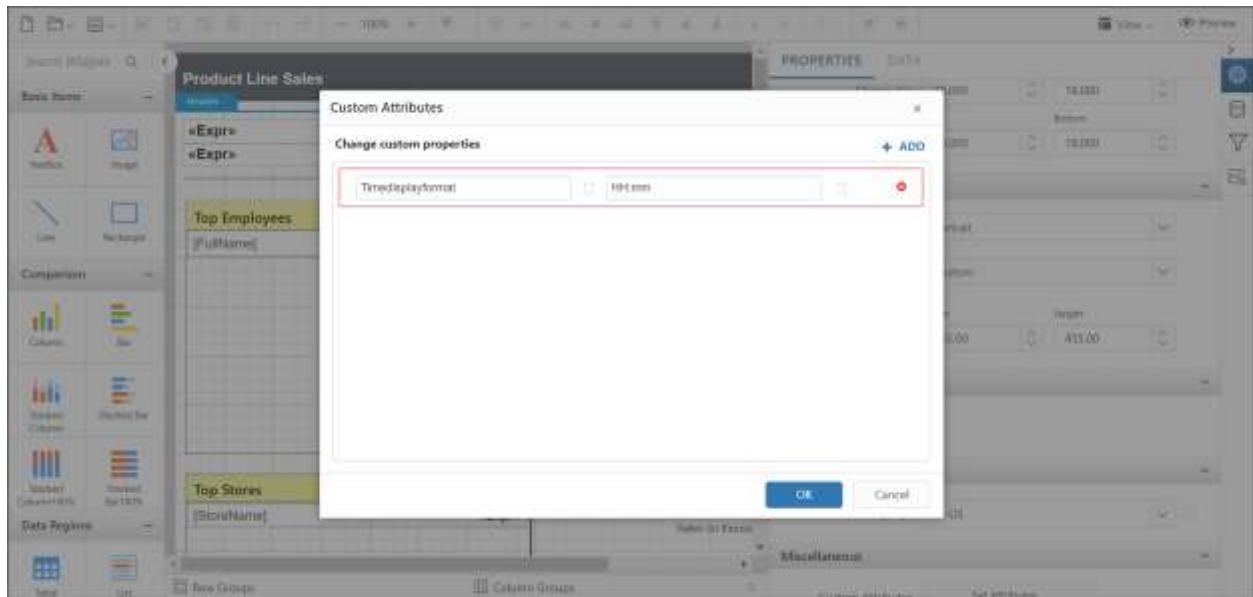
Preview the report and the see date time format in datetime parameter.



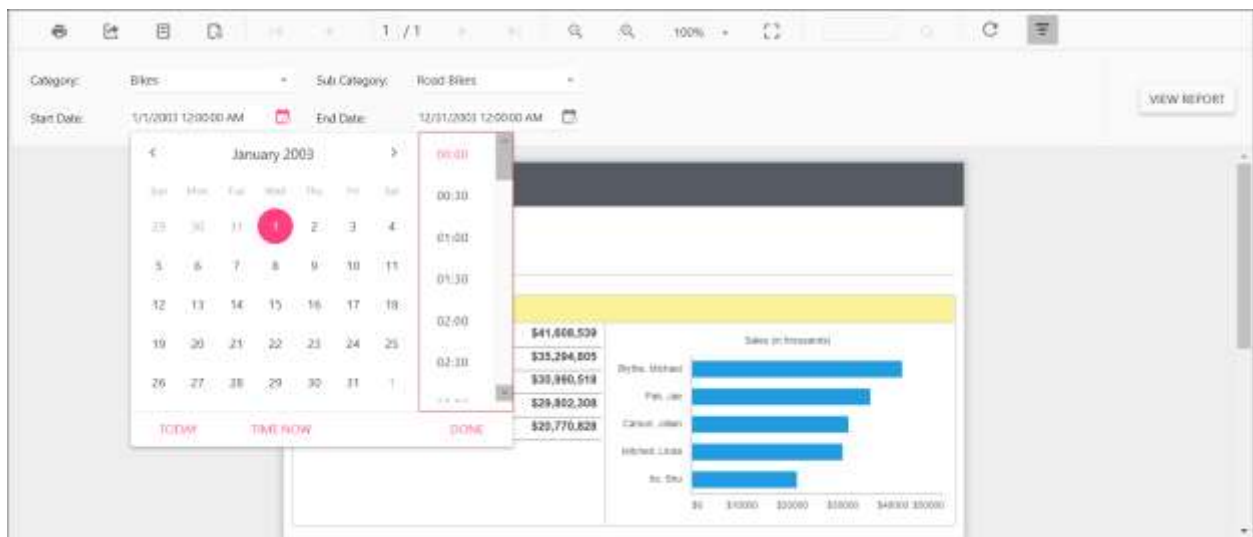
Change time display format for parameter

The `TimeDisplayFormat` custom property defines the time format to be displayed in the time dropdown inside the `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeDisplayFormat`. By default, the `TimeDisplayFormat` value is set as empty.

You can set the `TimeDisplayFormat` property value, as shown in the below.



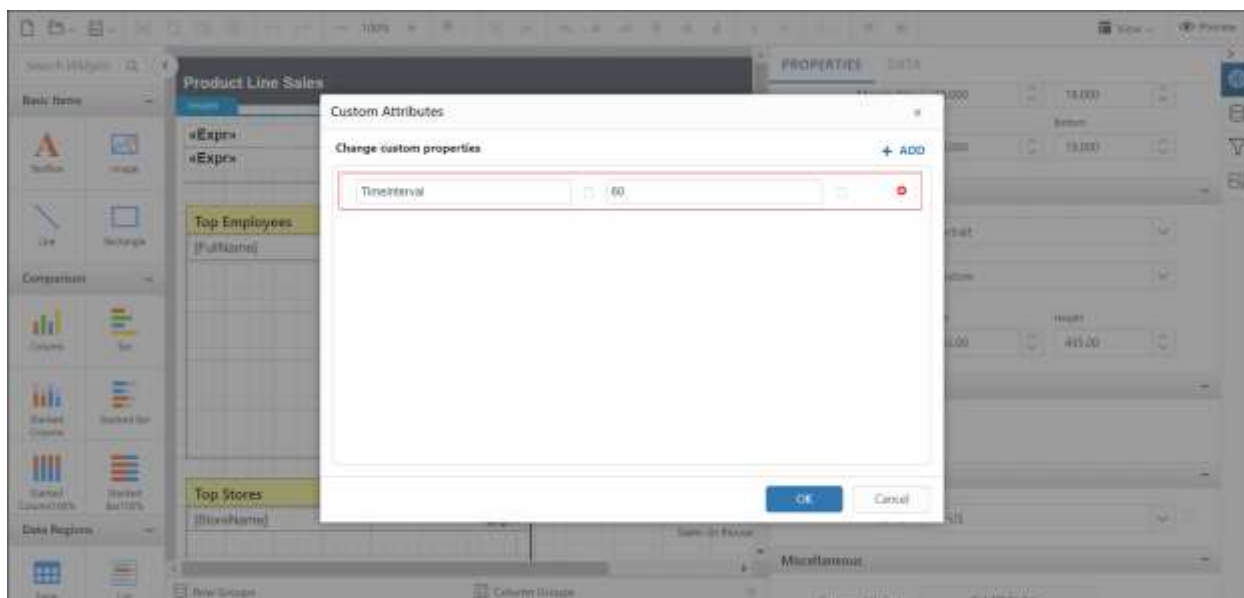
Preview the report and the see time format in datetime parameter.



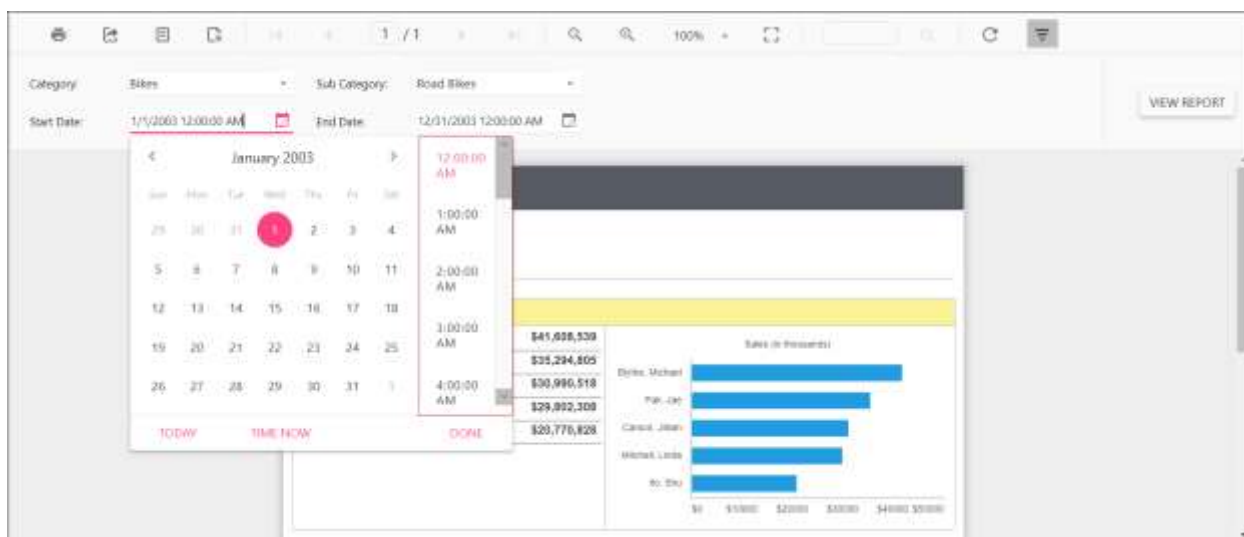
Set time interval in datetime parameter

The `TimeInterval` custom property is used to set the interval between the two adjacent time values in `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeInterval`. By default, the `TimeInterval` value is set as 30.

You can set the `TimeInterval` property value, as shown in the below.



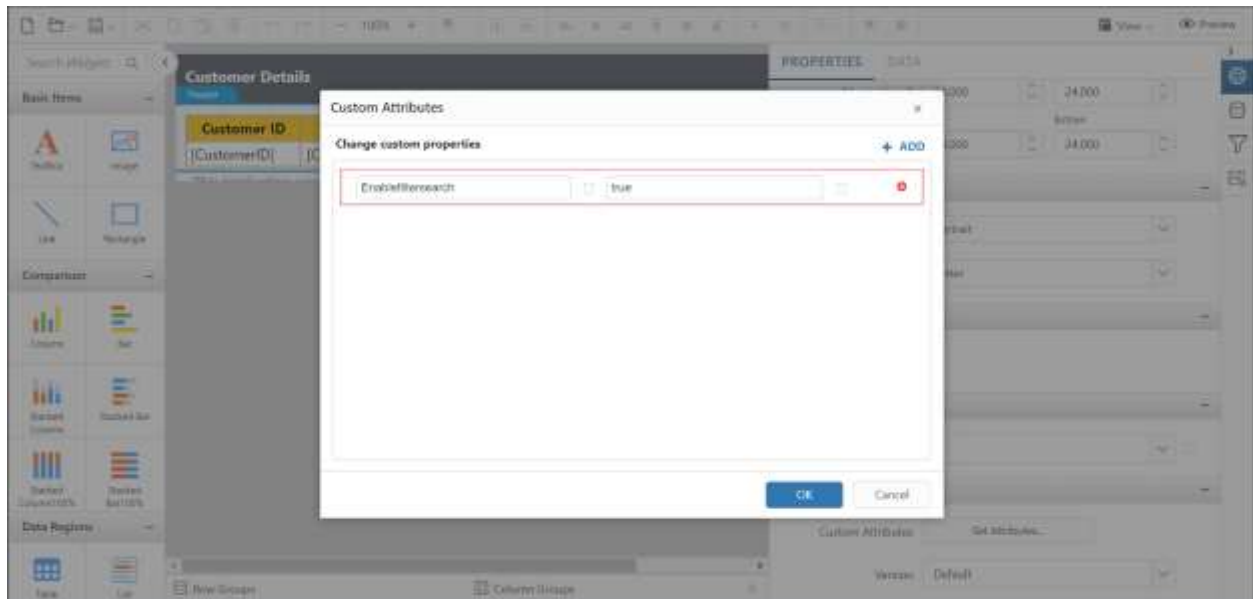
Preview the report and the see time interval in datetime parameter.



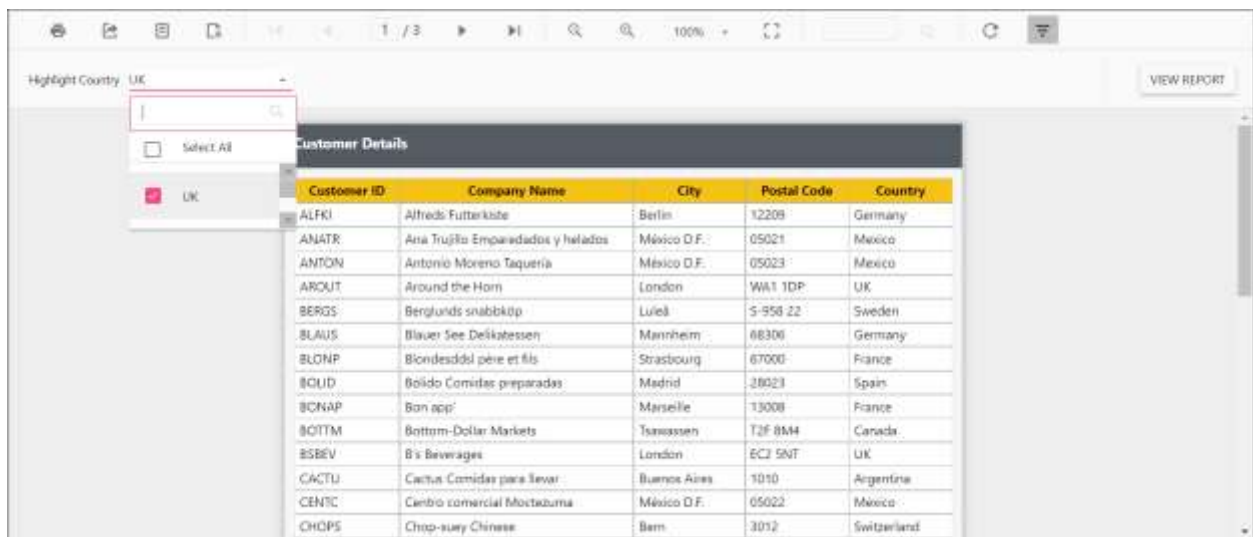
Enable filtering and searching in drop-down report parameter

Setting `EnableFilterSearch` custom property enables search and filtering option in dropdown parameter to easily find the values. The property value should be boolean. By default, the `EnableFilterSearch` value is `false`.

You can set the `EnableFilterSearch` property value, as shown in the below.



Preview the report and the see search filter in parameter.



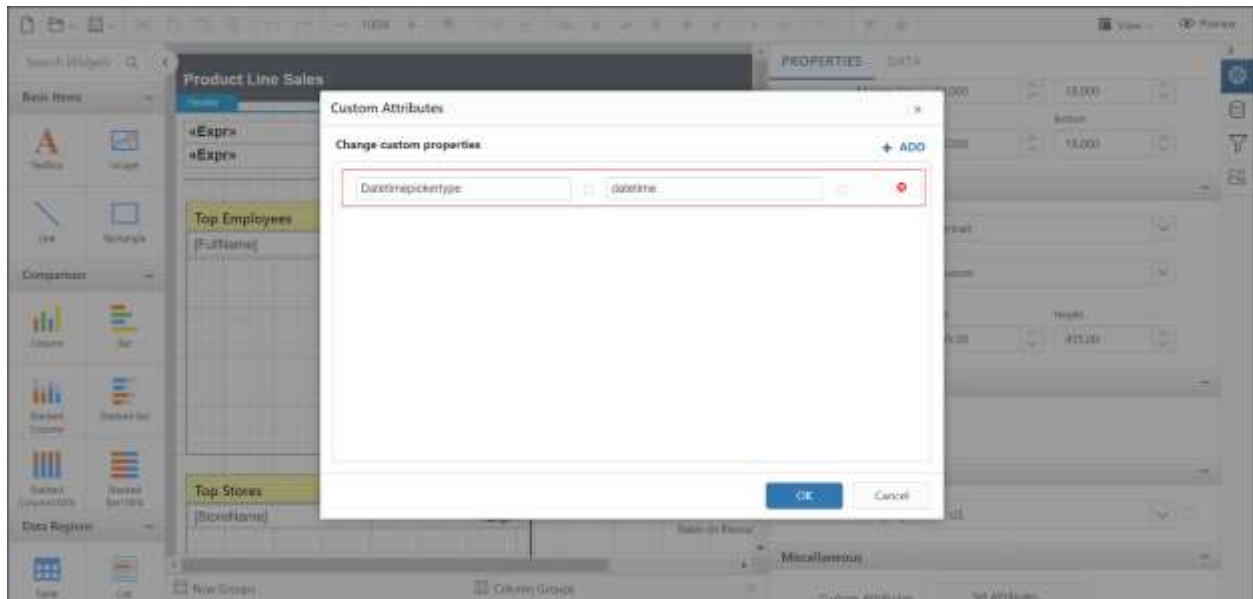
Change date report parameter to display date time picker

You can set `DateTimePickerType` custom property value as `DateTime` to change date parameter item to display `DateTimePicker` for value selection as shown the below.

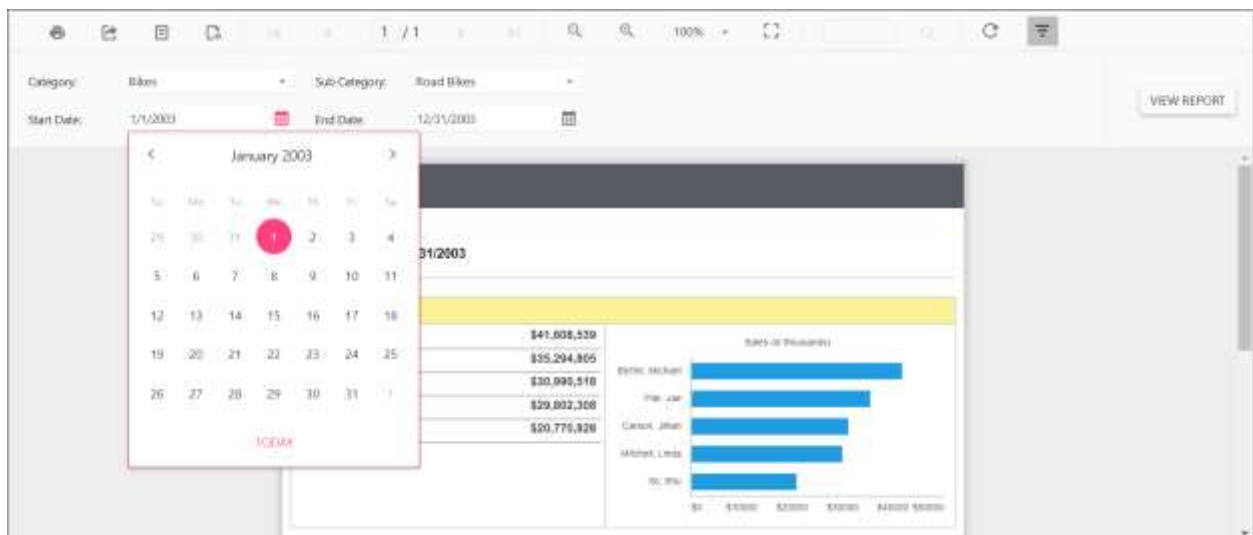
You can set the `DateTimePickerType` property value, as shown in the below.

Parameter custom properties

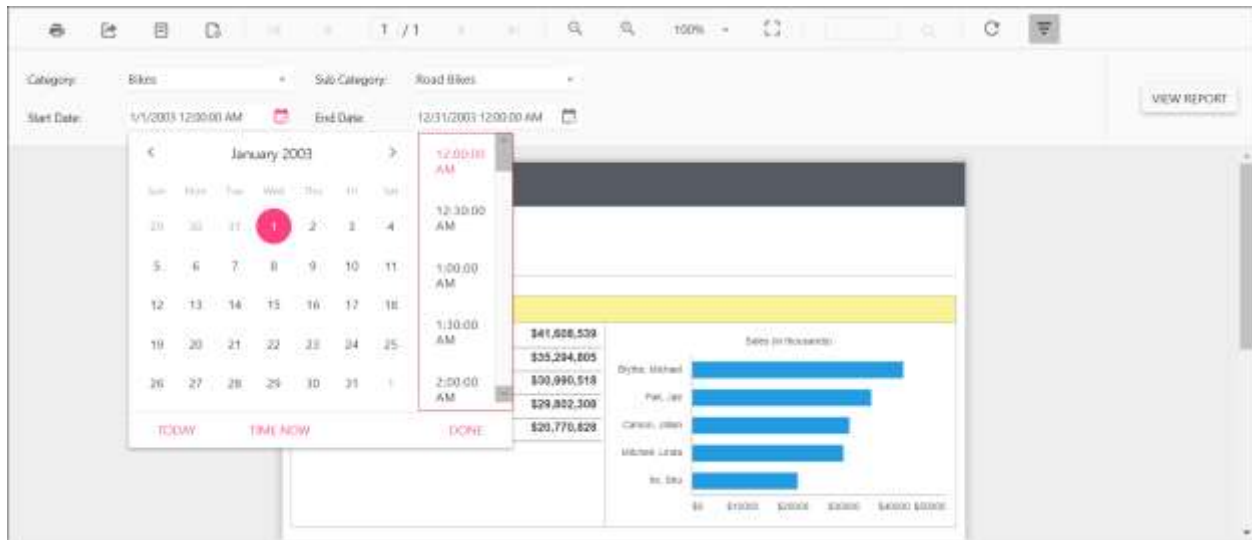
Change date report parameter to display date time picker



Before enabling date time picker type, the default value will be displayed as below..



Enable date time picker type and see the time in `DateTimePicker` as in below output.



Export custom properties

This topic explains the list of custom properties that are supported at the report level to control the export behaviour in React Report Viewer.

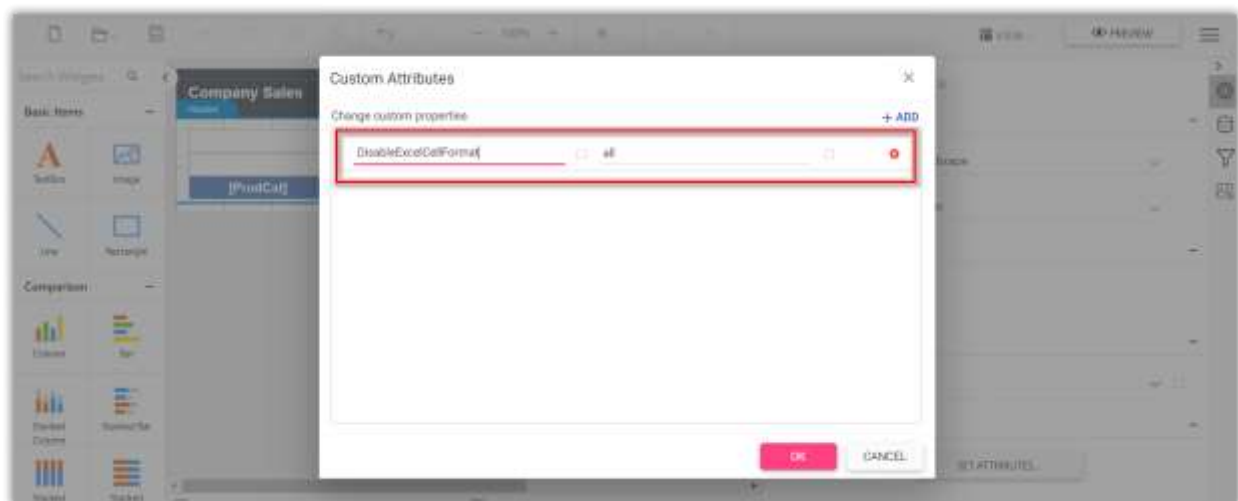
Improve excel export performance

The custom property `DisableExcelCellFormat` helps to improve the excel export performance by ignoring the rendering of report item styles. It allows property value from anyone of the following values.

- **Style** - Disables rendering of the cell styles like padding, background, color, and text style
- **Border** - Disables rendering of the cell border
- **All** - Disables rendering of the cell border, padding, background, color, and text style

Set the property value as **All** to improve maximum excel export performance.

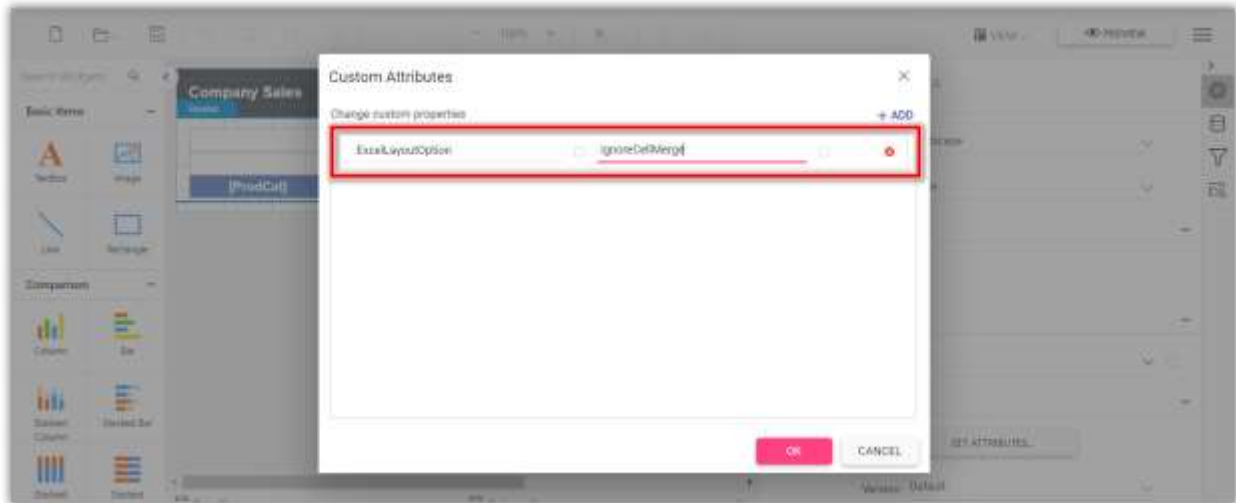
You can set the `DisableExcelCellFormat` custom property as shown below,



The `DisableExcelCellFormat` property must be added to report properties.

Improve excel export readability

Set `ExcelLayoutOption` custom property value as `IgnoreCellMerge` to improve the readability of the excel document by eliminating the tiny columns, rows, and merged cells. You can set the property value, as shown below,



The `ExcelLayoutOption` property must be added to report properties.

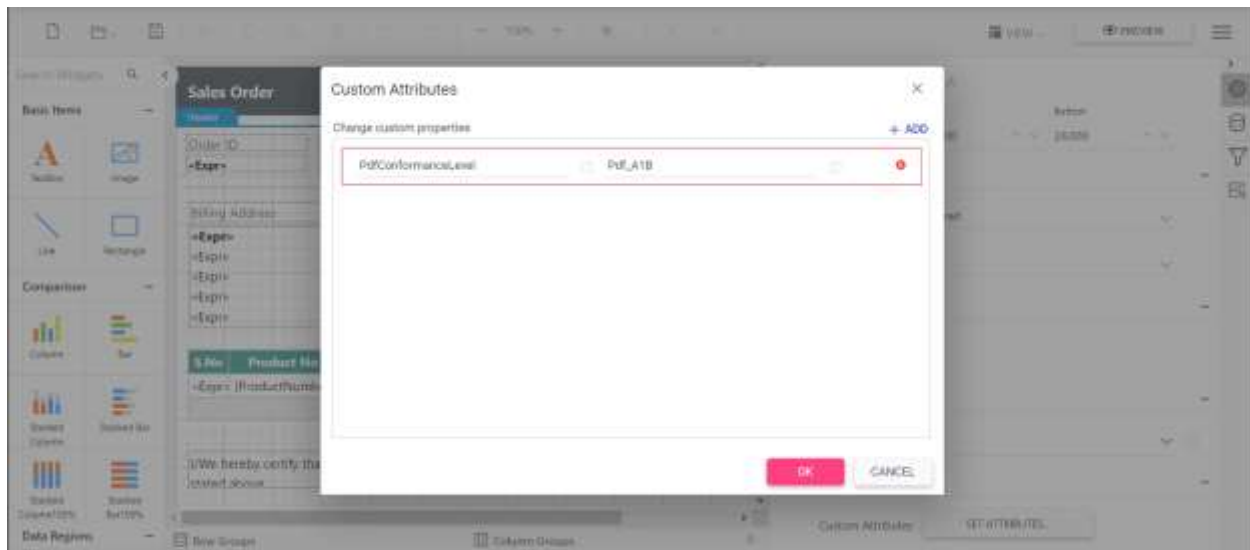
Set pdf conformance level

The `PdfConformanceLevel` allows you set PDF document versions.

You can set anyone of the following PDF conformance option.

- **PdfA1B** - You can create a PdfA1B document by specifying the conformance level as Pdf_A1B through PdfConformanceLevel Enum when creating the new PDF document.
- **PdfA2B** - You can create a PdfA2B document by specifying the conformance level as Pdf_A2B through PdfConformanceLevel Enum when creating the new PDF document
- **PdfA3B** - The PDF/A3B conformance supports the external files as attachment to the PDF document, so you can attach any document format such as Excel, Word, HTML, CAD, or XML files.
- **PdfA1A** - This conformance includes all PdfA1B requirements in addition to the features intended to improve a document's accessibility. PDF/A-1a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2A** - This conformance includes all PDF/A2B requirements in addition to the features intended to improve a document's accessibility. PDF/A-2a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2U** - This conformance includes all PdfA2U requirements, and additionally Unicode mapping for all text in the document.
- **PdfX1A2001** - You can create a PDF/X-1a document by specifying the conformance level as PdfX1A2001 through PdfConformanceLevel Enum when creating the new PDF document.

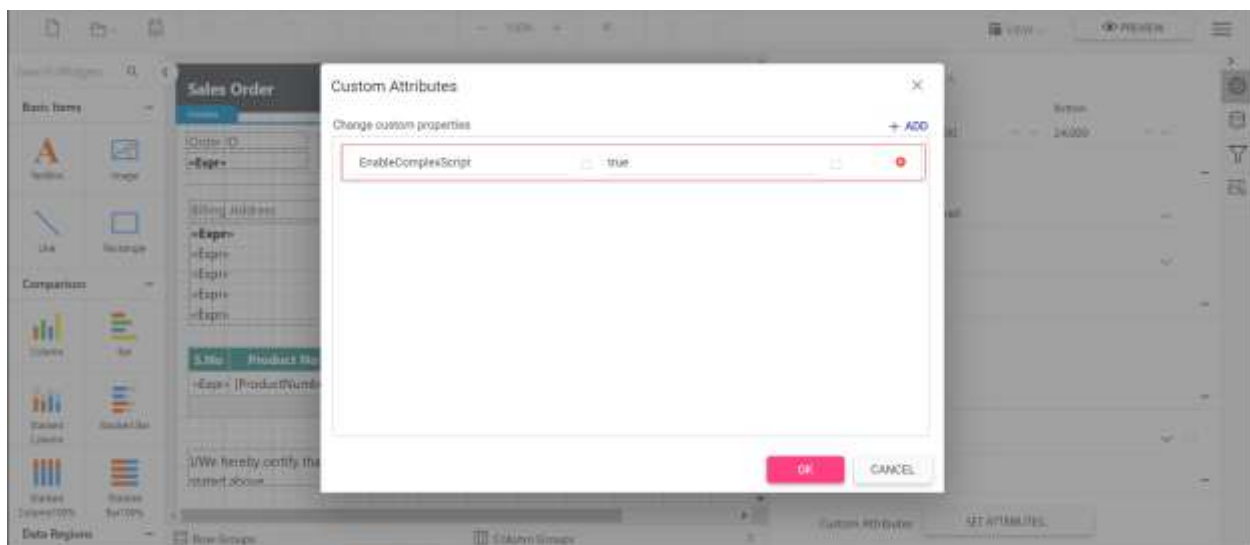
You can set the `PdfConformanceLevel` custom property as shown below,



Export pdf document with complex script

The `EnableComplexScript` custom property allows you to export pdf documents with complex script language texts.

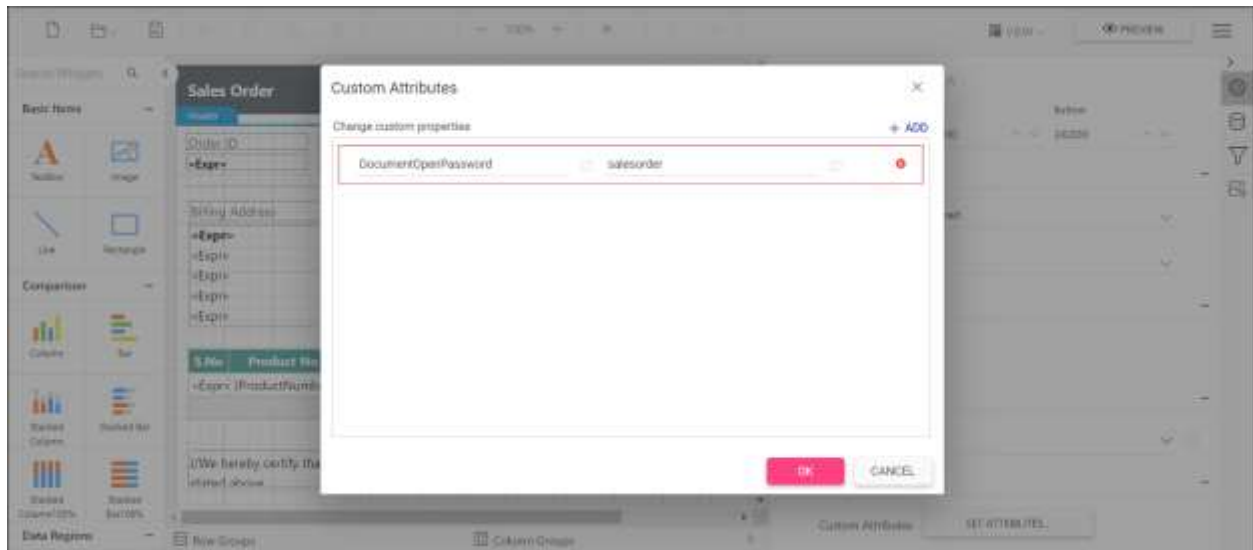
You can set the property value, as shown below,



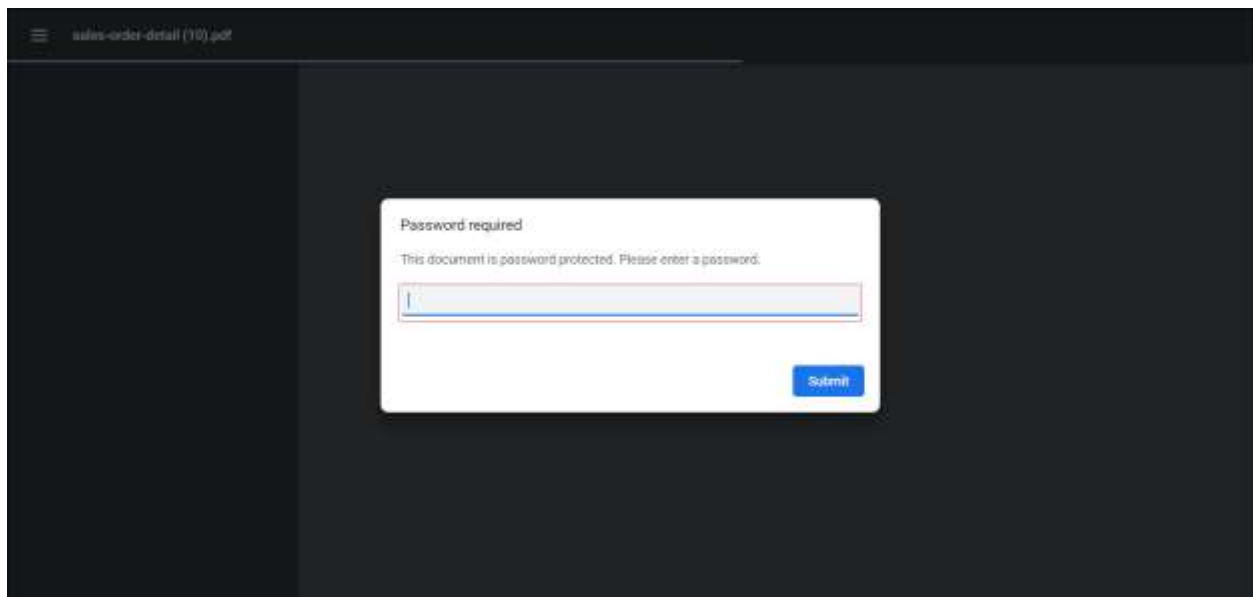
Encrypt and secure export documents

The custom property `DocumentOpenPassword` allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password.

You can set the property value, as shown below,



open the pdf document and see the below output.

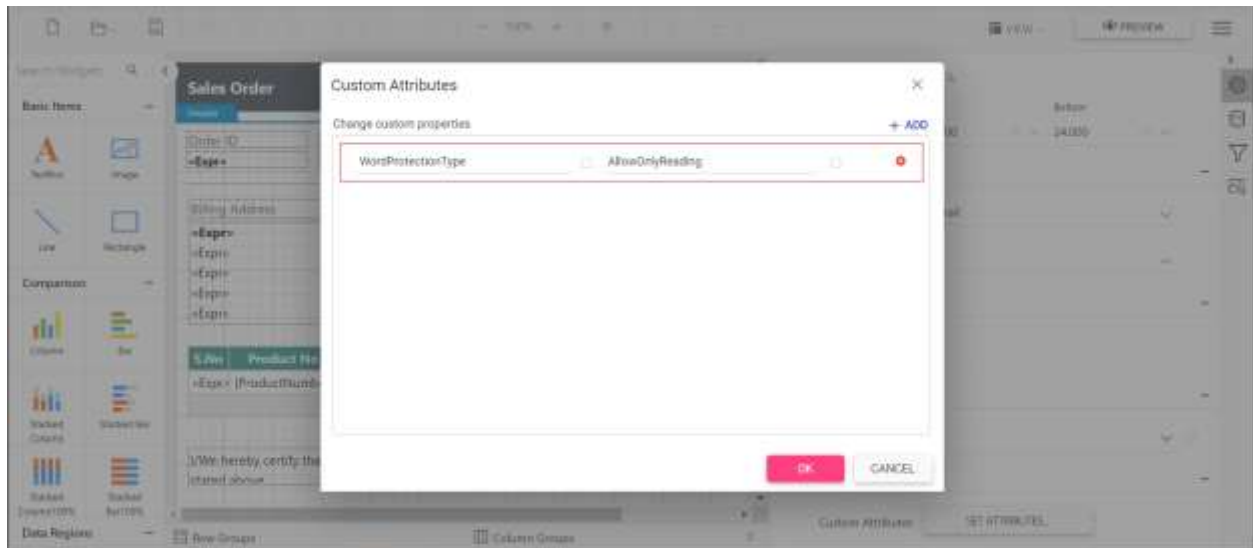


Protecting Word document from editing

The custom property `WordProtectionType` allows you to restrict a Word document from editing either by providing a password or without a password by using following types of protection.

- `AllowOnlyComments`- You can add or modify only the comments in the Word document.
- `AllowOnlyFormFields`- You can modify the form field values in the Word document.
- `AllowOnlyRevisions`- You can accept or reject the revisions in the Word document.
- `AllowOnlyReading`- You can only view the content in the Word document.
- `NoProtection`- You can access or edit the Word document contents as normally.

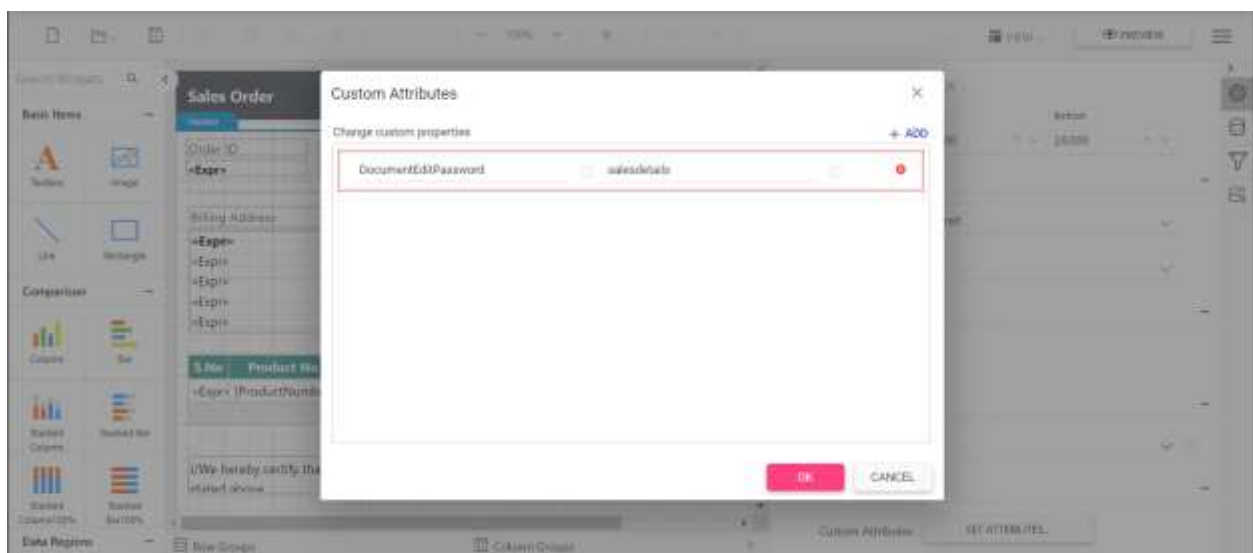
You can set the `WordProtectionType` custom property as shown below,



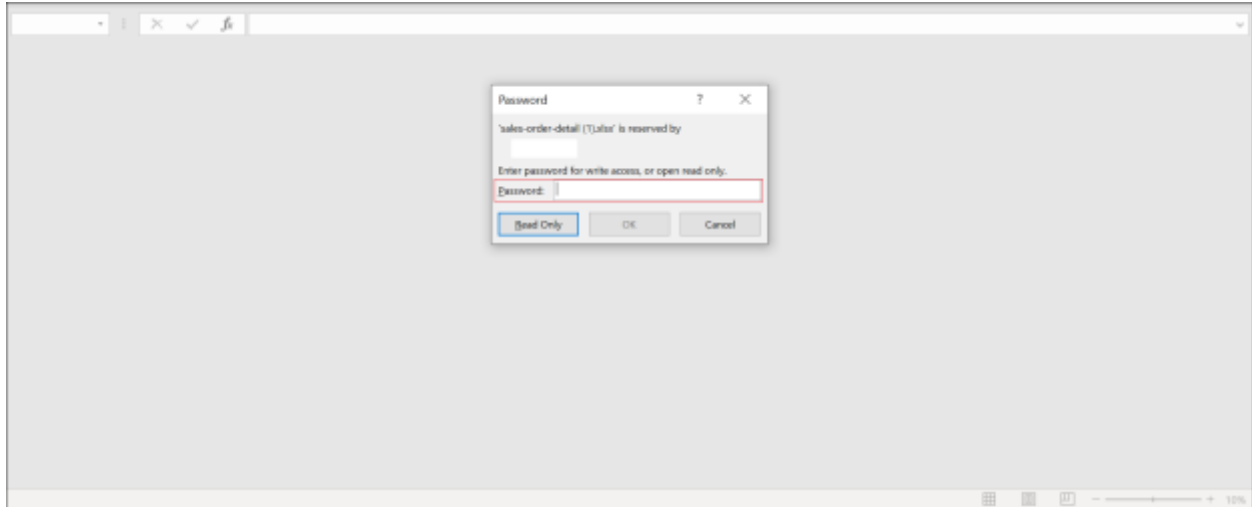
Set excel document edit password

The custom property `DocumentEditPassword` helps to allow specific users permission to modify the workbook data and save changes to the file in the excel document.

You can set the property value, as shown below,



open the excel document and see the below output.

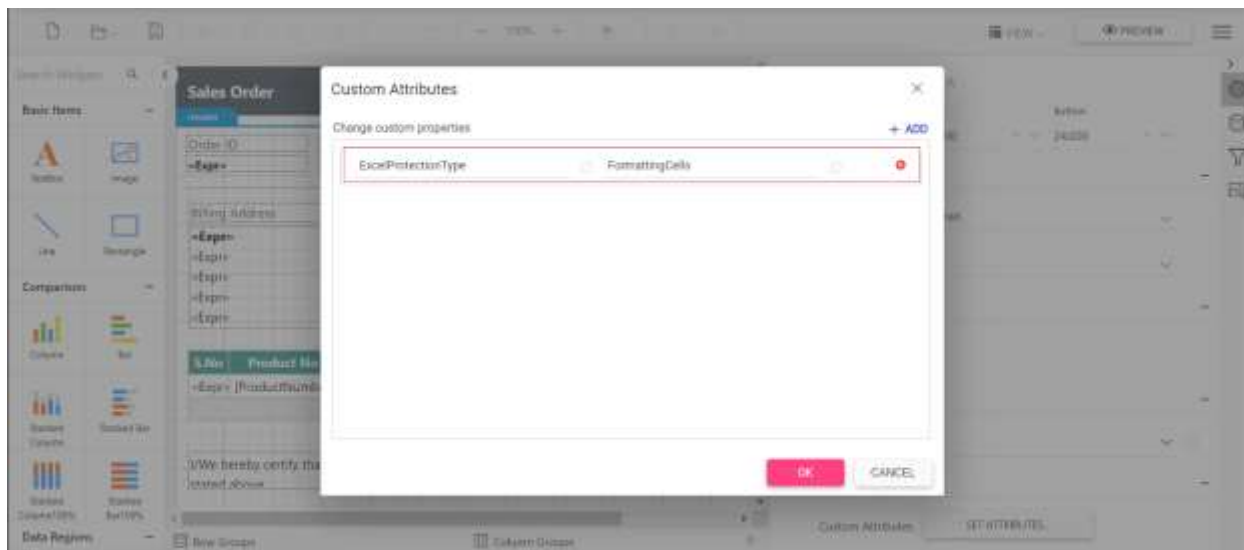


Set excel document protection

The custom property `ExcelProtectionType` allows you to protect the worksheet of excel document either by providing a password or without a password by using following types of protection.

- `None` - Represents no protection in excel sheet
- `Objects` - allows to protect shapes in excel sheet
- `Scenarios` - allows to protect scenarios.
- `FormattingCells` - allows the user to format any cell on a protected worksheet.
- `FormattingColumns` - allows the user to format any column on a protected worksheet.
- `FormattingRows` - allows the user to format any rows on a protected worksheet.
- `InsertingColumns` - allows the user to insert columns on the protected worksheet.
- `InsertingRows` - allows the user to insert rows on the protected worksheet.
- `InsertingHyperlinks` - allows the user to insert hyperlinks on the worksheet.
- `DeletingColumns` - allows the user to delete columns on the protected worksheet, where every cell in the column to be deleted is unlocked.
- `DeletingRows` - allows the user to delete rows on the protected worksheet, where every cell in the row to be deleted is unlocked.
- `LockedCells` - allows to protect locked cells.
- `Sorting` - allows the user to sort on the protected worksheet
- `Filtering` - allows the user to set filters on the protected worksheet. Users can change filter criteria but can not enable or disable an auto filter.
- `UsingPivotTables` - allows the user to use pivot table reports on the protected worksheet.
- `UnLockedCells` - allows to protect the user interface, but not macros.
- `Content` - allows to protect the contents in the excel sheet.
- `All` - allows to protect all type of protection.

You can set the `ExcelProtectionType` custom property as shown below,



Responsive layout rendering of React Report Viewer

Report Viewer will adaptively render itself with optimal user interfaces for phone, tablet, or desktop form factors. This helps your application to scale elegantly on all form factors with ease. You can enable responsive layout rendering in Report Viewer by setting `isResponsive` property to true.

```
`javascript
```

```
var isResponsive = true;
```

```
function App() {
```

```
  return (<div id="viewer" style={viewerStyle}>
```

```
    <BoldReportViewerComponent id="reportviewer-container"
```

```
    reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
```

```
    reportPath = { 'GroupingAgg.rdl' }
```

```
    isResponsive = { isResponsive }
```

```
  </BoldReportViewerComponent>
```

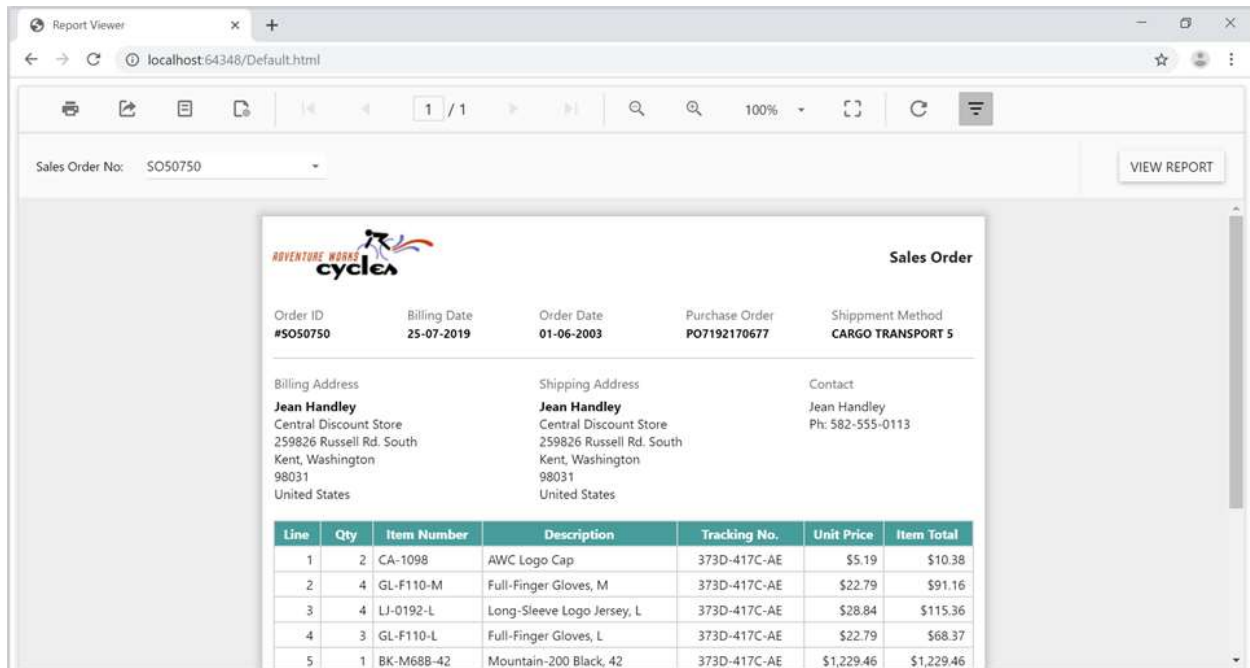
```
</div>);
```

```
}
```

```
,
```

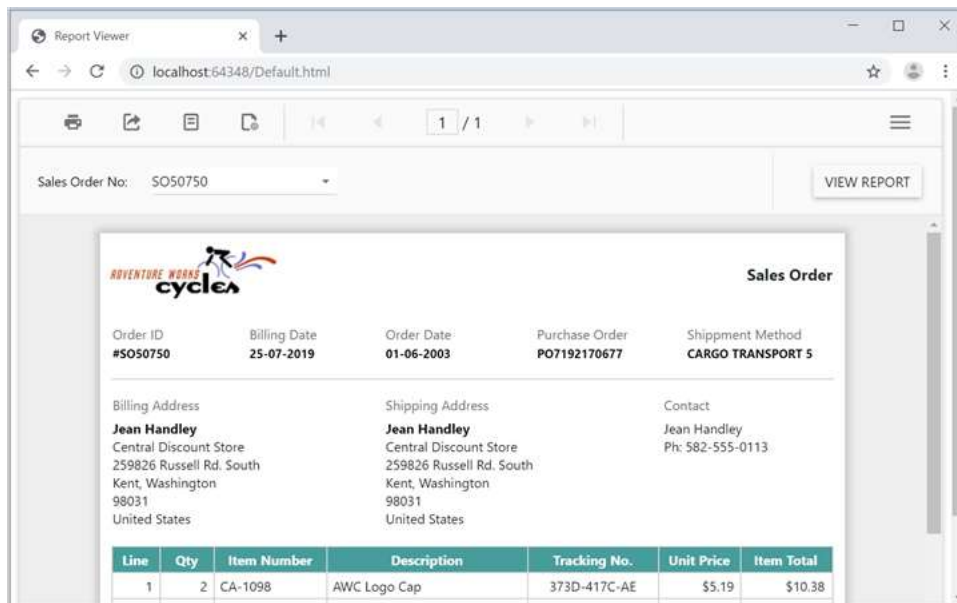
Normal layout

The following output shows the normal layout rendering of Report Viewer tool bar items.



Responsive layout

The following output shows the responsive layout rendering of Report Viewer tool bar items.



Localization of Bold Reports React Report Viewer

Localization of React Report Viewer allows you to localize the static text such as tooltip, parameter block, and dialog text based on a specific culture. To render the static text with specific culture, refer to the following corresponding culture script files and set culture name to the `locale` property of the Report Viewer.

- `ej.localetexts.fr-FR.min.js`
- `ej.culture.fr-FR.min.js`

Refer this [CDN links for Localization and Culture](#) to get the Localization and Culture scripts for available Culture Code.

- Run the below command, to install the `@boldreports/global` package.

```
`javascript
npm install @boldreports/global --save
`
```

- Refer to the `ej.localetexts.fr-FR.min.js` and `ej.culture.fr-FR.min.js` script files from `node_modules` in the `App.js` file.

```
`javascript
import React from 'react';
import './App.css';
//Report Viewer source
import '@boldreports/javascript-reporting-controls/Scripts/bold.report-viewer.min';
import '@boldreports/javascript-reporting-controls/Content/material/bold.reports.all.min.css';
//Data-Visualization
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.bulletgraph.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej.chart.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej2-circulargauge.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej2-lineargauge.min';
import '@boldreports/javascript-reporting-controls/Scripts/data-visualization/ej2-maps.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
import '@boldreports/global/l10n/ej.localetexts.fr-FR.min.js';
import '@boldreports/global/i18n/ej.culture.fr-FR.min.js';
...
...
`
```

Initialize the `locale` property in the `App.js` file as shown like in below code snippet.

```
`javascript
var locale = 'fr-FR';
function App() {
return (<div id="viewer" style={viewerStyle}>
```



```

<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'GroupingAgg.rdl' }
locale = {locale}

</BoldReportViewerComponent>
</div>);
}
`

```

React Report Viewer Reporting Service

The React Report Viewer requires a Web API service to process the report files. The following topics explains how to create reporting Web API service to preview the reports.

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

IReportController

The **IReportController** interface has the declaration of action methods that is defined in the Web API Controller for processing the RDL, RDLC, and SSRS report and handling the request from the Report Viewer control. The IReportController has the following action methods declaration.

Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

`csharp

```

public class ReportsController: ApiController,IReportController
{
    /// <summary>
    /// Action (HttpGet) method for getting resource for report.
    /// </summary>
    /// <param name="key">The unique key to get the required resource.</param>
    /// <param name="resourceType">The type of the requested resource.</param>
    /// <param name="isPrinting">If set to <see langword="true"/>, then the resource is generated for
    printing.</param>
    /// <returns>The object data.</returns>
    public object GetResource(string key, string resourceType, bool isPrinting)
    {

```

```

//Returns the report resource for the requested key.
return ReportHelper.GetResource(key, resourceType, isPrinting);
}
/// <summary>
/// Report Initialization method that is triggered when report begin processed.
/// </summary>
/// <param name="reportOptions">The ReportViewer options.</param>
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOptions)
{
//You can update report options here
}
/// <summary>
/// Report loaded method that is triggered when report and sub report begins to be loaded.
/// </summary>
/// <param name="reportOptions">The ReportViewer options.</param>
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOptions)
{
//You can update report options here
}
/// <summary>
/// Action (HttpPost) method for posting the request for report process.
/// </summary>
/// <param name="jsonData">The JSON data posted for processing report.</param>
/// <returns>The object data.</returns>
public object PostReportAction(Dictionary < string, object > jsonData)
{
//Processes the report request and returns the result.
return ReportHelper.ProcessReport(jsonData, this);
}
}
,

```

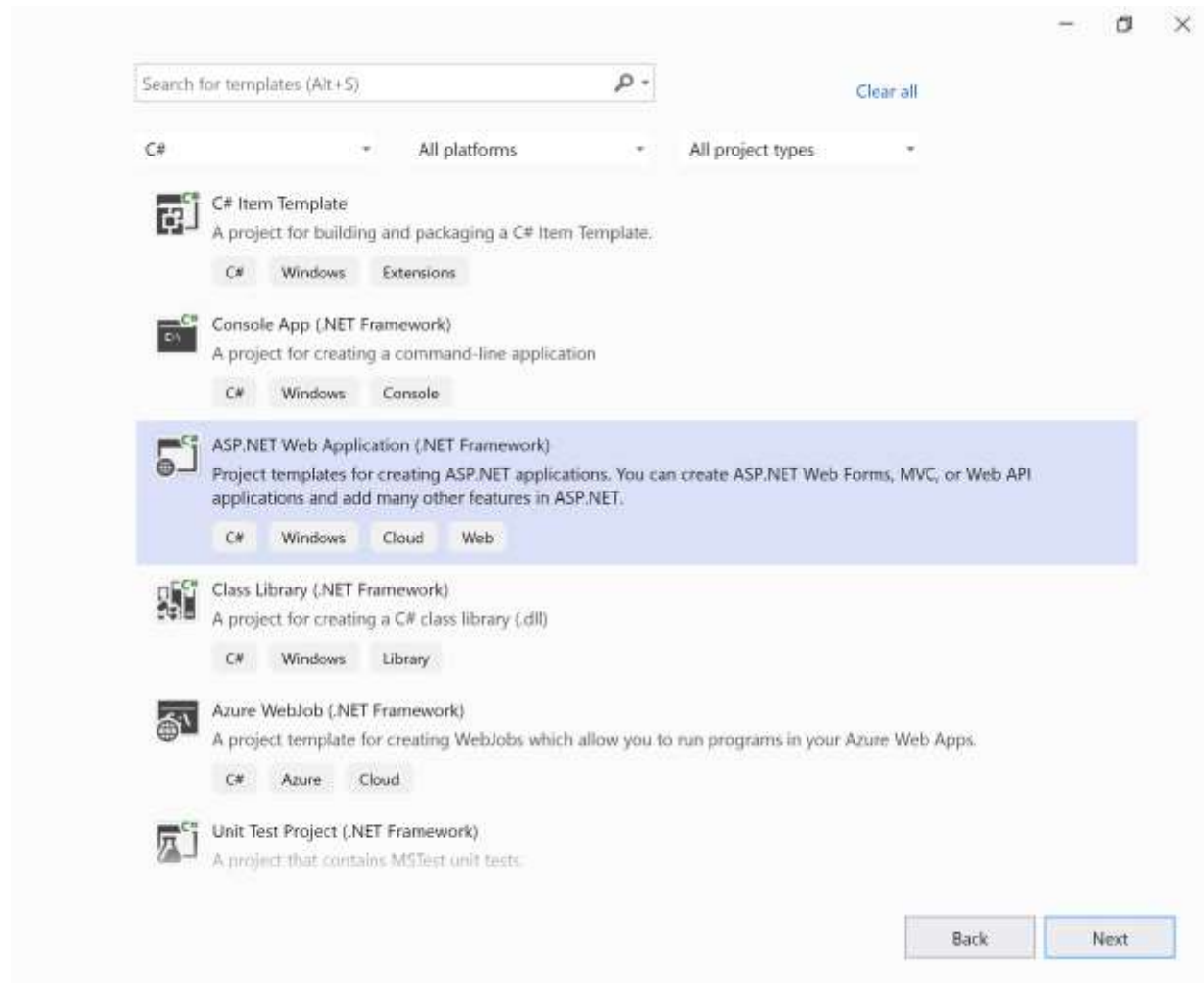
Create ASP.NET Web API service

In this section, you will learn how to create a Web API Service for Report Viewer using the new ASP.NET Empty Web Application template.

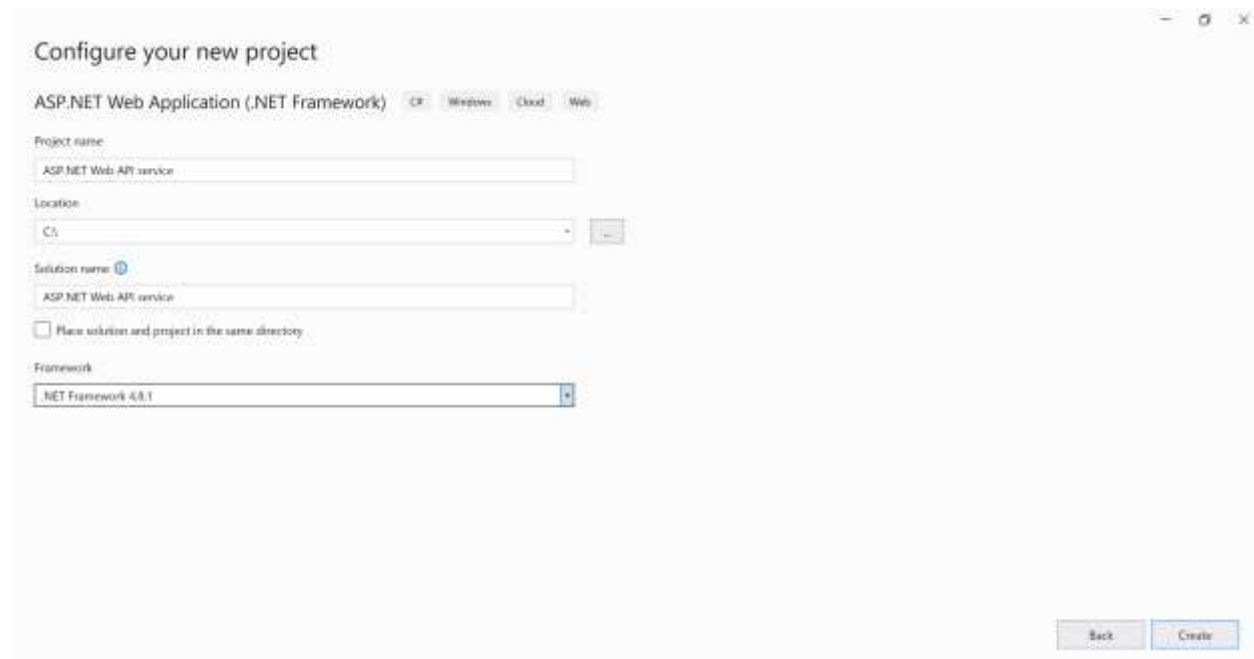
To get start quickly with Web API Service for Report Viewer, you can check on this video:

youtube: https://youtu.be/Qbho_8cnhJ8

1. Start Visual Studio 2022 and click **Create new project**.
2. Select **ASP.NET Web Application (.NET Framework)**.



3. Change the application name, and then select the required **.NET Framework** in the drop-down.



Configure your new project

ASP.NET Web Application (.NET Framework) **CR** Windows Cloud Web

Project name
ASP.NET Web API service

Location
C:\

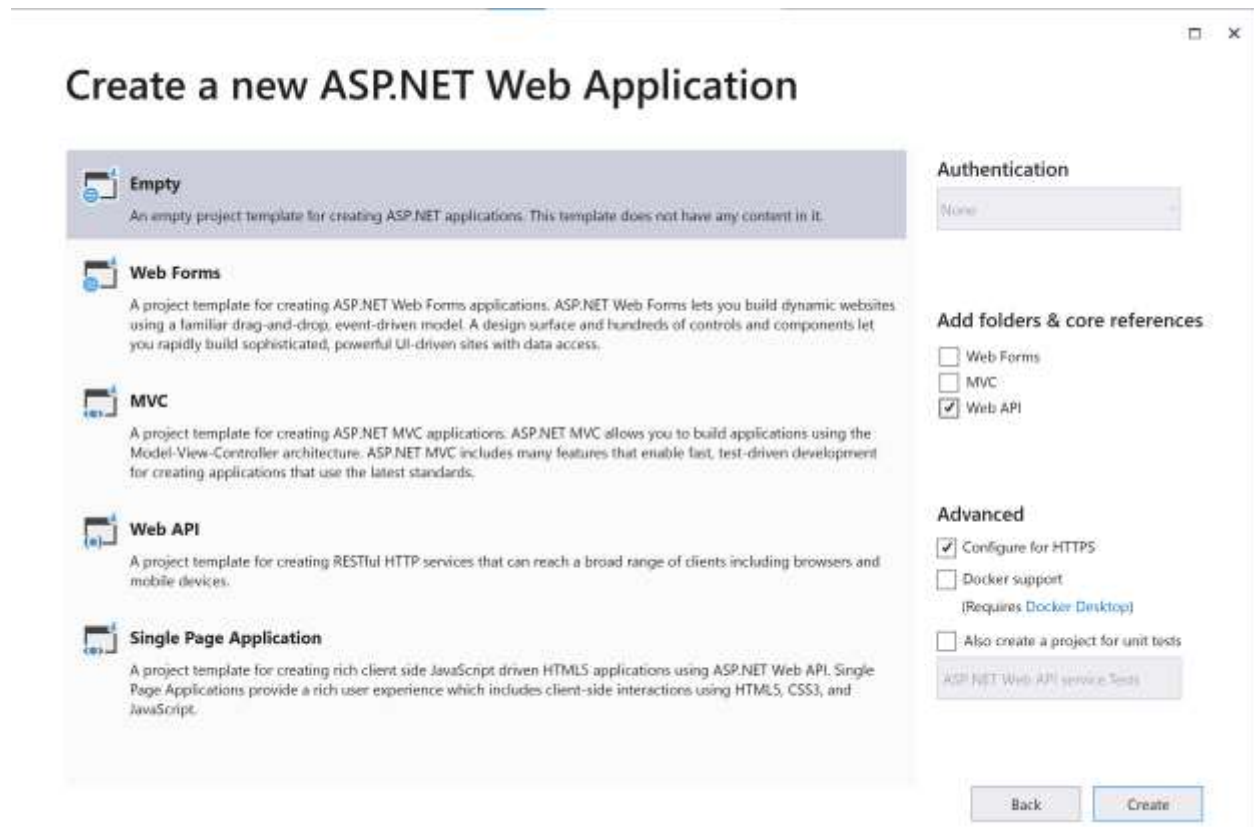
Solution name
ASP.NET Web API service

☐ Place solution and project in the same directory

Framework
.NET Framework 4.5.1

Back Create

4. Choose **Empty**, **Web API** and then click **OK**. Now, the Web application project is created with default ASP.NET Web template.



Create a new ASP.NET Web Application

Empty
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Web Forms
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

MVC
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Web API
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Single Page Application
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
None

Add folders & core references

- ☐ Web Forms
- ☐ MVC
- ☒ Web API

Advanced

- ☒ Configure for HTTPS
- ☐ Docker support
(Requires [Docker Desktop](#))
- ☐ Also create a project for unit tests

ASP.NET Web API service Tests

Back Create

Configure Report Viewer Web API

1. Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

2. Search for **BoldReports.Web** NuGet packages, and install them in your Web application.

Package | Purpose

PostReportAction | Action (HttpPost) method for posting the request in report process.

OnInitReportOptions | Report initialization method that occurs when the report is about to be processed.

OnReportLoaded | Report loaded method that occurs when the report and sub report start loading.

GetResource | Action (HttpGet) method to get resource for the report.

ReportHelper

The class **ReportHelper** contains helper methods that help to process a Post or Get request from the Report Viewer control and return the response to the Report Viewer control. It has the following methods:

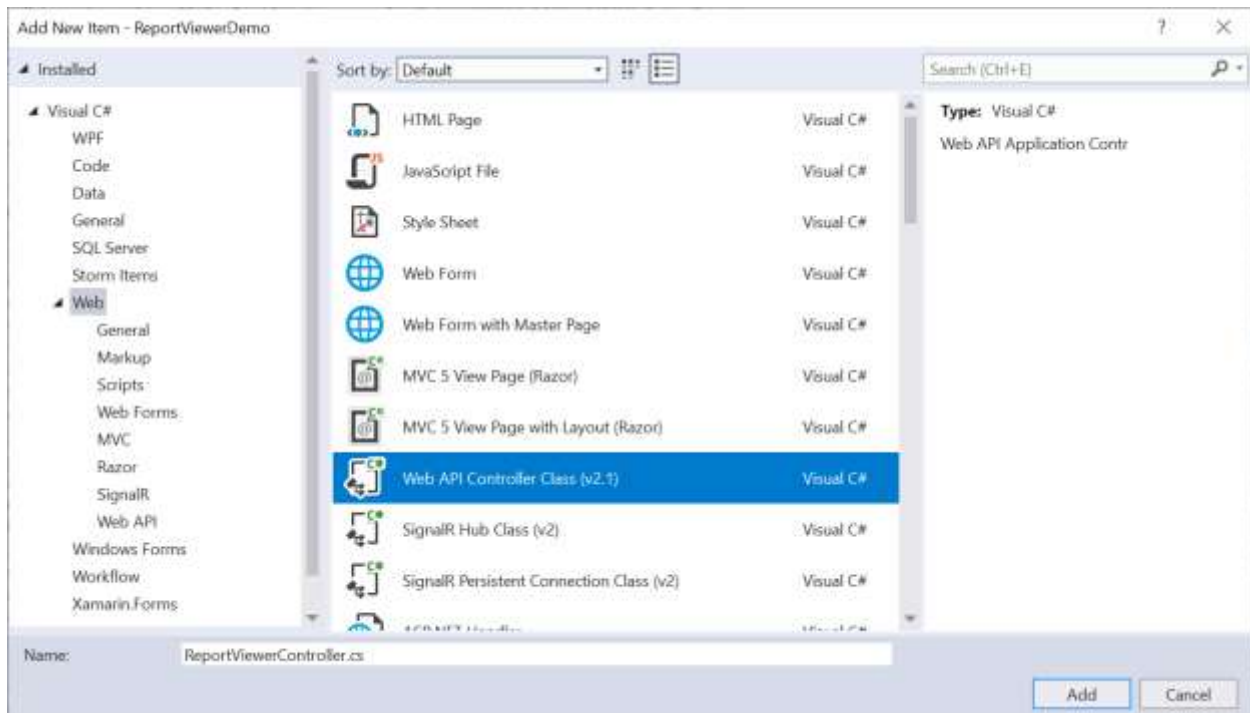
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click **Controller** folder in your project and select **Add > New Item** from the context menu.
2. Select **Web API Controller Class** from the listed templates and name it as **ReportViewerController.cs**



3. Click **Add**.

While adding Web API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and implement its methods (replace the following code in newly created Web API controller).

```
`csharp
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
}
```

```

[System.Web.Http.ActionName("GetResource")]
[AcceptVerbs("GET")]
public object GetResource(string key, string resourcetype, bool isPrint)
{
    return ReportHelper.GetResource(key, resourcetype, isPrint);
}
// Method that will be called when initialize the report options before start processing the report
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    // You can update report options here
}
// Method that will be called when reported is loaded
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    // You can update report options here
}
}
`

```

Add routing information

1. To configure routing to include an action name in the URI, open the **WebApiConfig.cs** file and change the **routeTemplate** in the **Register** method as follows,

```

`csharp
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services
        // Web API routes
        config.MapHttpAttributeRoutes();
        config.Routes.MapHttpRoute(

```

```

name: "DefaultApi",
routeTemplate: "api/{controller}/{action}/{id}",
defaults: new { id = RouteParameter.Optional }
);
}
}
,

```

2. Compile and run the Web API service application.

Enable Cross-Origin Requests

Browser security prevents Report Viewer from making requests to your Web API Service when both runs in a different domain. To allow access to your Web API service from a different domain, you must enable cross-origin requests.

1. Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, go to **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for `Microsoft.AspNet.WebApi.Cors` NuGet packages, and install them in your Web API application.
3. Call `EnableCors` in `WebApiConfig` to add CORS services to the **Register** method. Replace the following code to allow any origin requests.

```

`csharp
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Add Enable Cors
        config.EnableCors();

        // Web API configuration and services

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{action}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}

```



```

}
}
`

```

4. To specify the CORS policy for API controller, add the `[EnableCors]` attribute to the controller class. Specify the policy name.

```

`csharp
[EnableCors(origins: "", headers: "", methods: "*")]
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
    [System.Web.Http.ActionName("GetResource")]
    [AcceptVerbs("GET")]
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }
    // Method that will be called when initialize the report options before start processing the report
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
    // Method that will be called when reported is loaded
    [NonAction]
    public void OnReportLoaded(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
}

```

```
}  
,
```

Create ASP.NET Core Web API Service

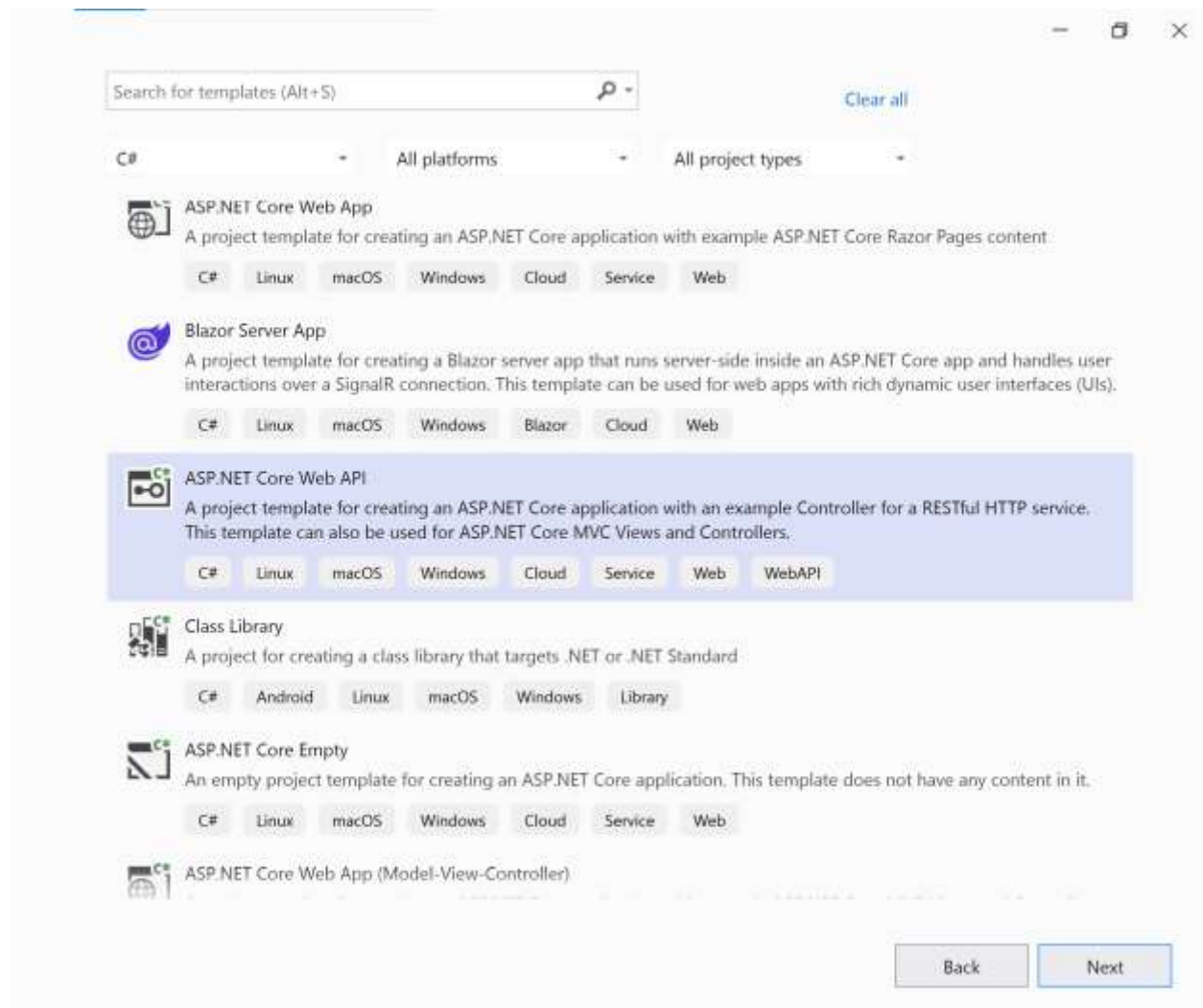
To create an ASP.NET Core Web API for Report Viewer using a new ASP.NET Core Web Application template, follow these steps:

Bold Reports ASP.NET Core supports from **.NET Core 2.1** only. So, choose the .NET Core version **ASP.NET Core 2.1** or higher versions for Viewer API creation.

To get start quickly with ASP.NET Core Web API for Report Viewer, you can check on this video:

youtube: <https://youtu.be/b1yZGAeWoHQ>

1. Start Visual Studio 2022 and click **Create new project**.
2. Choose **ASP.NET Core Web API**, and then click **Next**.



3. Change the project name, and then click **Next**.

4. In the dropdown for the **ASP.NET Core version**, choose **ASP.NET Core 6.0**, then click **Create**.

The screenshot shows the 'Additional information' configuration page for an ASP.NET Core Web API service. At the top, there are tabs for 'ASP.NET Core Web API', 'CR', 'Linux', 'macOS', 'Windows', 'Cloud', 'Service', 'Web', and 'WebAPI'. The 'Framework' dropdown is set to '.NET 6.0 (Long Term Support)'. The 'Authentication type' is set to 'None'. There are checkboxes for 'Configure for HTTPS' (checked), 'Enable Docker' (unchecked), and 'Use controllers (unchecked to use minimal APIs)' (checked). There are also checkboxes for 'Enable OpenAPI support' (checked) and 'Do not use top-level statements' (unchecked). At the bottom right, there are 'Back' and 'Create' buttons.

If you need to use Bold Reports with ASP.NET Core on Linux or macOS, then refer to this [Can Bold Reports be used with ASP.NET Core on Linux and macOS](#) section.

List of dependency Libraries

The Web API service configuration requires the following reporting server-side packages. Right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages** and then search for **BoldReports.Net.Core** package, and install to the application. The following provides detail of the packages and its usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Supports for exporting the report to PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the packages **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core** and **Syncfusion.XlsIO.Net.Core**.

Syncfusion.Pdf.Net.Core | Supports for exporting the report to a PDF.

Syncfusion.DocIO.Net.Core | Supports for exporting the report to a Word.

Syncfusion.XlsIO.Net.Core | Supports for exporting the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serialize and deserialize the data for report viewer. It is a mandatory package for the report viewer, and the package version should be higher of 10.0.1 for NET Core 2.0 and others should be higher of 9.0.1.

System.Data.SqlClient | This is an optional package for the report viewer. It should be referred in project when renders the RDL report and which contains the SQL Server and SQL Azure data source. Also, the package version should be higher of 4.1.0.

Configure Web API

The interface **IReportController** has declaration of action methods that are defined in the Web API Controller for processing the RDL, RDLC, and SSRS reports and for handling request from the Report Viewer control. The IReportController has the following action methods declaration:

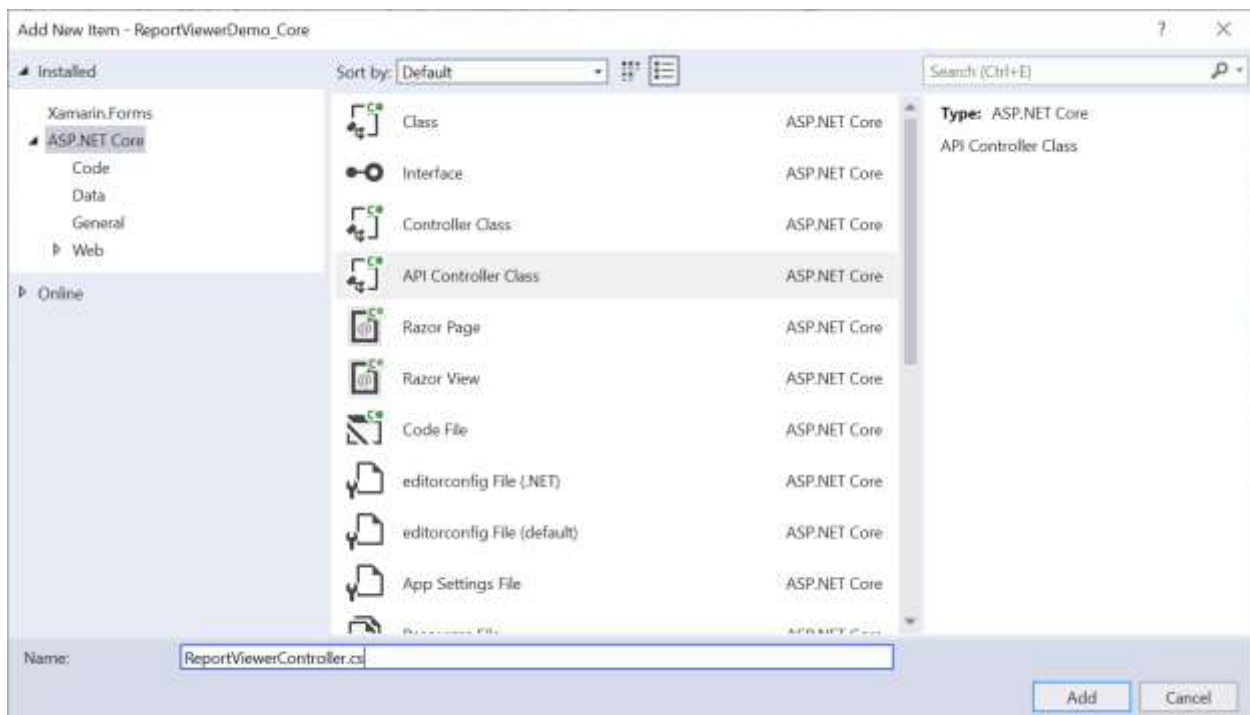
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller** class and name it as **ReportViewerController.cs**



3. Click **Add**.

While adding API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the `IReportController` interface, and then implement its methods.
6. Create local references for the interfaces given in following table.

Interface | Purpose

`IMemoryCache` | Report Viewer requires a memory cache to store the information of consecutive client request and have the rendered report viewer information in server.

`IWebHostEnvironment` | `IWebHostEnvironment` used to get the report stream from application `wwwroot\Resources` folder.

7. Next, add the `[EnableCors]` attribute to the `ReportViewerController` class and specify the policy name which given in `Program.cs`.

```
`csharp
[Route("api/[controller]/[action]")]
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
}
`
```

8. You cannot load the application report with path information in ASP.NET Core service. So, you should load the report as stream in `OnInitReportOptions`.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    System.IO.FileStream reportStream = new System.IO.FileStream(basePath + @"\Resources\sales-order-
    detail.rdl", System.IO.FileMode.Open, System.IO.FileAccess.Read);
    reportOption.ReportModel.Stream = reportStream;
}
`
```

9. You can replace the template code with the following code.

```
`csharp
```

```
[Route("api/[controller]/[action]")]
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered report viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;

    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }

    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
    public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
    {
        return ReportHelper.ProcessReport(jsonArray, this, this._cache);
    }

    // Method will be called to initialize the report information to load the report with ReportHelper for
    // processing.
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sales-order-detail.rdl report from application the folder
        // wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
        System.IO.FileStream reportStream = new System.IO.FileStream(basePath + @"\Resources\sales-order-
        detail.rdl", System.IO.FileMode.Open, System.IO.FileAccess.Read);
        reportOption.ReportModel.Stream = reportStream;
    }
}
```

```

}
// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}
//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
return ReportHelper.GetResource(resource, this, _cache);
}
[HttpPost]
public object PostFormReportAction()
{
return ReportHelper.ProcessReport(null, this, _cache);
}
}
`

```

The `sales-order-detail.rdl` report can be downloaded from [here](#). Also, you can add the report from Bold Reports installation location. For more information on installed sample location, see [Samples and demos](#).

10. Run the application and use the API URL in the Report Viewer `reportServiceUrl` property.

Enable Cross-Origin requests

Browser security prevents Report Viewer from making requests to your Web API Service when both runs in a different domain. To allow access to your Web API service from a different domain, you must enable cross-origin requests.

Call `AddCors` in `Program.cs` to add the CORS services to the app's service container. Replace the following code to allow any origin requests.

```

`csharp
builder.Services.AddCors(o => o.AddPolicy("AllowAllOrigins", builder =>

```

```
{
builder.AllowAnyOrigin()
.AllowAnyMethod()
.AllowAnyHeader();
});
.....
.....
app.UseHttpsRedirection();
app.UseCors();
app.UseAuthorization();
`
```

For more information about CORS, see [Configure CORS for API](#)

How to queries for Bold Reports ReportViewer

This section helps to get the answer for the frequently asked how to queries in Bold Reports React ReportViewer.

- [How to pass multiple values using custom data?](#)
- [How to clear cache when closing the Report Viewer?](#)

How to pass multiple values using custom data

Pass the multiple data values as JSON in `ajaxBeforeLoad` event using the 'data' property, which needs to be serialized in the server side API using known class type and `JsonConvert.DeserializeObject`.

1. Map the [ajaxBeforeLoad](#) event with `onAjaxRequest` function in the script to pass custom data to the server.

```
`js
function App() {
return (<div id="viewer" style={viewerStyle}>
<BoldReportViewerComponent id="reportviewer-container"
reportServiceUrl = {'https://demos.boldreports.com/services/api/ReportViewer'}
reportPath = { 'Sales Order Detail.rdl' }
ajaxBeforeLoad = {onAjaxRequest}

</BoldReportViewerComponent>
</div>);
```



```

}
function onAjaxRequest(args) {
//Passing custom data to server
}
`

```

2. You need to pass the multiple custom data values as JSON and use the `data` property to send custom data to the server in the Ajax request.

```

`js
function onAjaxRequest(args) {
//Passing custom data to server
var jsonData = {
customerID: "CI0021",
productID: "PO0022"
};
args.data = jsonData;
}
`

```

3. The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates deserialization of the JSON string and change the data source connection strings based on Customer ID and Product ID in the `OnInitReportOptions` method.

```

`csharp
public SampleData customDatas = new SampleData();
public class SampleData
{
public string customerID { get; set; }
public string productID { get; set; }
}
[HttpPost]
public object PostReportAction([FromBody]Dictionary<string, object> jsonResult)
{
if (jsonResult.ContainsKey("customData"))
{

```

```
//Get client side JSON custom data, desirialize it and store in local variable.
customDatas = JsonConvert.DeserializeObject<SampleData>(jsonResult["customData"].ToString());
}
return ReportHelper.ProcessReport(jsonResult, this, this._cache);
}
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
if (customDatas != null)
{
if (customDatas.customerID == "CI0021" && customDatas.productID == "PO0022") {
//If you are changing the connection string based on customer id then could you please change the
connection string as below.
//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));
reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>;"));
}
}
}
}
```

How to clear cache when closing the Report Viewer

By using the `clearReportCache` and `destroy` method, you can clear the server side cache. You have to use this method when closing report viewer and switching to another report within your application.

JavaScript

Add the following code in `index.js` file.

```
`js
class Header extends React.Component {
constructor(props) {
}
componentDidMount() {
$(document.body).bind('submit', $.proxy(window.addEventListener, window));
}
}
```

```

var isSubmit = true;
$(document.body).bind('submit', $.proxy(window.addEventListener, window));
function addEventListener()
{
isSubmit = false;
}
window.onbeforeunload = function () {
if (isSubmit) {
var reportviewerObj = $("#reportviewer-container").data("boldReportViewer");
reportviewerObj.clearReportCache();
reportviewerObj.destroy();
}
isSubmit = true;
};
ReactDOM.render(<App {...isSubmit} />, document.getElementById('root'));
`

```

Web API

In the `PostReportAction` method, you have to collect the GC with `ClearCache` as shown in the following code sample.

```

`csharp
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
{
bool isclearcache = false;
if (jsonArray != null && jsonArray.ContainsKey("reportAction") && jsonArray["reportAction"].ToString()
== "ClearCache")
{
isclearcache = true;
}
var reportresult = ReportHelper.ProcessReport(jsonArray, this, this._cache);
if (isclearcache)
{
GC.Collect(); isclearcache = false;
}
return reportresult;
}

```

```
}
,
```

Migrate to Report Viewer v2.0 component

This section provides simple step-by-step instructions to update your existing Report Viewer application to our latest modern v2.0 scripts, styles and components.

Replacing Report Viewer component

1. Open the `App.js` file.
2. Remove the Report Viewer component old style and scripts
 - o `bold.report-viewer.min`
 - o `bold.reports.all.min.css`
 - o `ej.bulletgraph.min`
 - o `ej.chart.min`
3. Add v2.0 scripts
 - o `\v2.0\tailwind-light\bold.report-viewer.min.css`
 - o `\v2.0\common\bold.reports.common.min`
 - o `\v2.0\common\bold.reports.widgets.min`
 - o `\v2.0\bold.report-viewer.min.css`
4. The final updated `App.js` file.

```
`typescript
/ eslint-disable /
import React from 'react';
import './App.css';
import '@boldreports/javascript-reporting-controls/Content/v2.0/tailwind-light/bold.report-viewer.min.css';
import '@boldreports/javascript-reporting-controls/Scripts/v2.0/common/bold.reports.common.min';
import '@boldreports/javascript-reporting-controls/Scripts/v2.0/common/bold.reports.widgets.min';
import '@boldreports/javascript-reporting-controls/Scripts/v2.0/bold.report-viewer.min';
//Reports react base
import '@boldreports/react-reporting-components/Scripts/bold.reports.react.min';
var viewerStyle = {'height': '700px', 'width': '100%'};
function App() {
  return (
    <div style={viewerStyle}>
    <BoldReportViewerComponent
    id="reportviewer-container">
```

```

</BoldReportViewerComponent>
</div>
);
}
export default App;
`

```

Frequently asked questions

This section helps to get the answer for the frequently asked questions in Bold Reports React Report Viewer.

1. [Is it possible to hide report parameters?](#)
2. [Is it possible to hide the export options in Report Viewer?](#)

Is it possible to hide the parameters in Report Viewer

Yes, it is possible to hide the parameters in Report Viewer. On hiding the parameters, the users will not be able to have the parameter block in the Report Viewer. Refer to this [Hide parameter block and toolbar items](#) section.

Is it possible to hide the export options in Report Viewer

Yes, it is possible to hide the export options in Report Viewer. The Report Viewer provides the `exportOptions` property to show or hide the default export types available in the component. Refer to this [Decide or Hide the export options](#) section.

How to section for React Bold Reporting Tools

This section helps to get the answer for the frequently asked how to queries in Bold Reporting Tools.

- [How to resolve the Javascript memory heap out issue that occurs while building the React application?](#)

How to resolve the Javascript memory heap out issue that occurs while building the React application

The memory heap out issue occurs when the heap size is not sufficient to run the application. To resolve this issue, open the `package.json` file, which can be found in the root folder of React application and use `--max-old-space-size=4096` as like in the below code snippet.

```

`typescript
...
...
"scripts": {
  "start": "react-scripts --max-old-space-size=4096 start",

```

```
"build": "react-scripts --maxOldSpace_size=4096 build",  
"test": "react-scripts test",  
"eject": "react-scripts eject"  
}  
...  
,
```

If you are referring Bold Reports scripts from npm packages using import, then you will need larger heap size to compile the application. Instead of using the Bold Reports scripts from npm packages, when using the online [CDN](#) scripts for references, you need not increase the heap size of the application as like above.

See also

[Bold Reports CDN links](#)

Reporting tools for ASP.NET Core

Enterprise-class reporting tools for ASP.NET Core development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your core applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

Reporting tools for ASP.NET Core

Enterprise-class reporting tools for ASP.NET Core development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your core applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

System Requirements

This topic describes the software and hardware requirements for setting up the development environment of Bold Reports ASP.NET Core.

Supported Operating Systems

- Windows 7+, 8+
- Windows Server 2008 R2+

Framework

The below tool is required for Bold Reports ASP.NET Core.

.NET Core 2.1 or higher

Hardware Environments

The following hardware environments are necessary for setting up the Bold Reports ASP.NET Core.

- 1 GHZ or faster, 32 bit or 64 bit processor.
- 1 GB RAM for 32 bit or 2 GB RAM for 64 bit.

Development Environments

By using the following IDEs, you can develop the Bold Reports ASP.NET Core.

- [Microsoft Visual Studio 2017](#) or [later](#)
- Internet Information Services (IIS) 7.0+

Browser Compatibility

- IE 9+
- Microsoft Edge
- Mozilla Firefox 22+
- Chrome 17+
- Opera 12+
- Safari 5+

See Also

- [Licensing procedure for deployment](#)

Overview

The Report Viewer is a visualization control used to display SSRS, RDL, RDLC, and Bold Report Server reports within web applications. It allows you to view RDL/RDLC reports with or without using SSRS or Bold Report Server. You can bind data sources, parameters, and render reports with all major

Display SSRS RDL report in Bold Reports ASP.NET Core Report Viewer **Create** an ASP.NET Core application

capabilities of RDL reporting and export the report to PDF, Microsoft Excel, CSV, Microsoft Word, and HTML formats. Some of the key features are,

- Renders interactive reports with drill down, drill through, hyperlinks, and interactive sorting.
- Easily customize each element of Report Viewer and provide events for report processing customization.
- Supports jQuery, Angular, React, Blazor, ASP.NET Core, ASP.NET MVC, ASP.NET WebForms, WPF, and UWP.

Introducing the newly redesigned Report Viewer! We have given it a refreshing new look that embraces sleek and modern design, aligning perfectly with the latest web design trends. It allows seamless integration into your application. For detailed guidance on the migration process, we recommend you to refer our [Report Viewer v2.0 migration document](#).

Display SSRS RDL report in Bold Reports ASP.NET Core Report Viewer

Create your first ASP.NET Core reporting Web application in the .NET 6.0 framework to display an already created SSRS RDL report in the Bold Reports ASP.NET Core Report Viewer without using a Report Server using this step-by-step instructions.

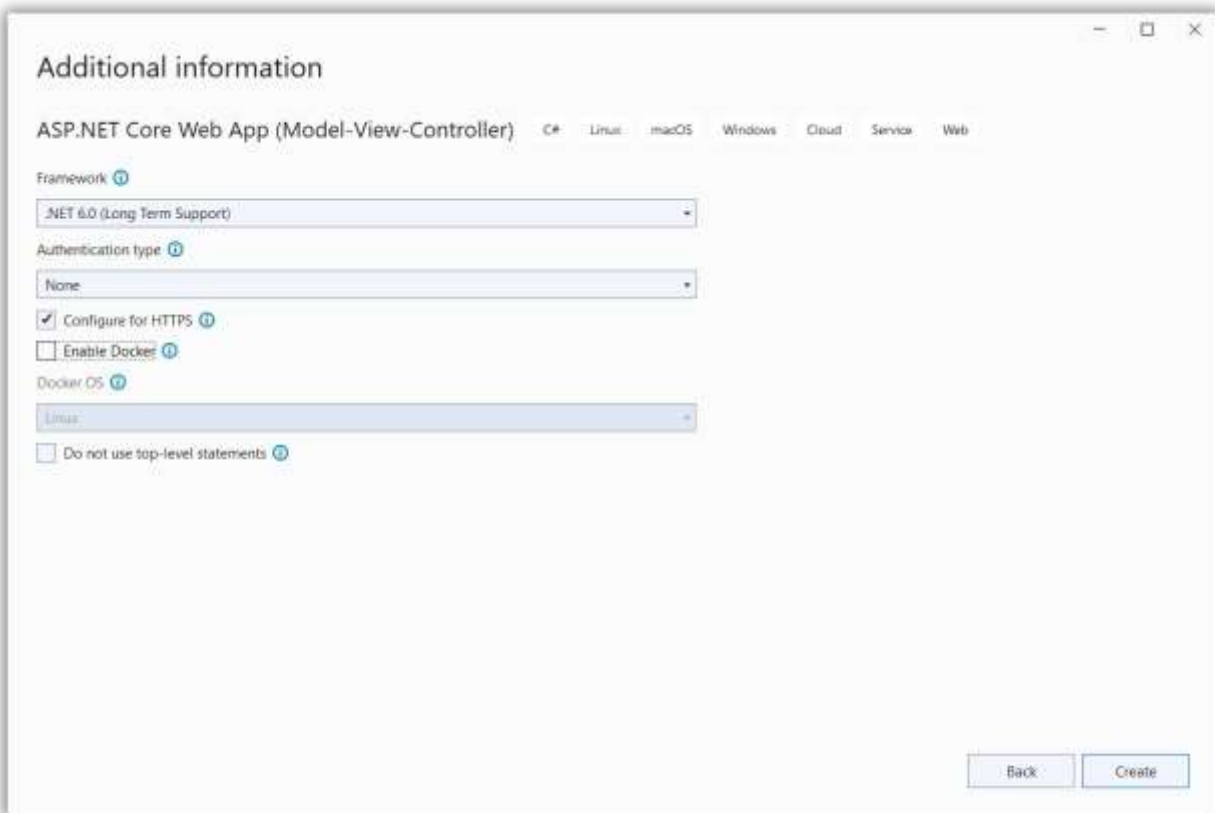
To create your first application on the other .NET Core frameworks, refer to the documentation for [ASP .NET Core 2.1](#), [ASP.NET Core 3.1](#) or [.NET 5.0](#)

To get start quickly with Report Viewer, you can check on this video:

youtube: <https://youtu.be/-zt8p6RnOPg>

Create an ASP.NET Core application

1. Start Visual Studio 2022 and click **Create new project**.
2. Choose **ASP.NET Core Web App (Model-View-Controller)**, and then click **Next**.
3. Change the project name, and then click **Next**.
4. In the dropdown for the **ASP.NET Core version**, choose **ASP.NET Core 6.0**, then click **Create**.



If you need to use Bold Reports with ASP.NET Core on Linux or macOS, then refer to the [Can Bold Reports be used with ASP.NET Core on Linux and macOS](#) section.

List of dependency libraries

1. In the Solution Explorer tab, right-click the project or solution, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.AspNet.Core**, **BoldReports.Net.Core**, and **System.Data.SqlClient** packages, and install them in your Core application. The following table provides details about the packages and their usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Exports the report to a PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core**, and **Syncfusion.XlsIO.Net.Core** packages.

Syncfusion.Pdf.Net.Core | Exports the report to a PDF.

Syncfusion.DocIO.Net.Core | Exports the report to a Word.

Syncfusion.XlsIO.Net.Core | Exports the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serializes and deserialize data for the Report Viewer. It is a mandatory package for the Report Viewer, and the package version should be 10.0.1 or higher.

Refer scripts and CSS

Directly refer all the required scripts and style sheets from the [CDN](#) links.

- The following scripts and style sheets are mandatorily required to use the Report Viewer.
 - bold.reports.all.min.css**
 - jquery.min.js**
 - ej2-maps.min.js** - Renders the map item. Add this script only if your report contains the map report item.
 - ej2-base.min.js**, **ej2-data.min.js**, **ej2-pdf-export.min.js**, **ej2-svg-base.min.js**, **ej2-lineargauge.min.js** and **ej2-circulargauge.min.js** - Render the gauge item. Add these scripts only if your report contains the gauge report item.
 - bold.reports.common.min.js**
 - bold.reports.widgets.min.js**
 - ej.chart.min.js** - Renders the chart item. Add this script only if your report contains the chart report item.
 - bold.report-viewer.min.js**
- Open the `\Views\Shared_Layout.cshtml` page.
- Replace the following code in your `\Views\Shared_Layout.cshtml` page tag.

```
`html
```

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
```

```
<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
```

To learn more about rendering a report with data visualization report items, refer to the [how to render data visualization report items](#) section.

The Report Viewer scripts and styles can be added to your application by installing the **BoldReports.JavaScript** online nuget package.

Tag helper

It is necessary to define the following tag helper within the `_ViewImports.cshtml` page to initialize the Report Viewer component with the tag helper support.

```
`js
@using BoldReports.TagHelpers
@addTagHelper *, BoldReports.AspNet.Core
```

Configure Script Manager

Open the `~/Views/Shared/_Layout.cshtml` page and add the reporting Script Manager at the end of the `<body>` element as in the following code sample.

```
`html
<body>
<div style="min-height: 600px;width: 100%;">
@RenderBody()
</div>
@RenderSection("Scripts", required: false)
<!-- Bold Reports script manager -->
<bold-script-manager></bold-script-manager>
</body>
```

Initialize Report Viewer

1. Open the `Index.cshtml` page.
2. Remove the existing codes and add the following code.

```
`html
<bold-report-viewer id="viewer"></bold-report-viewer>
```

Add already created reports

1. Create a folder `Resources` in the `wwwroot` folder in your application to store the RDL reports.
2. Download the `sales-order-detail.rdl` from [here](#). For more sample reports, refer to the [samples and demos](#) section.
3. Extract the compressed file and paste the `sales-order-detail.rdl` to the `Resources` folder.

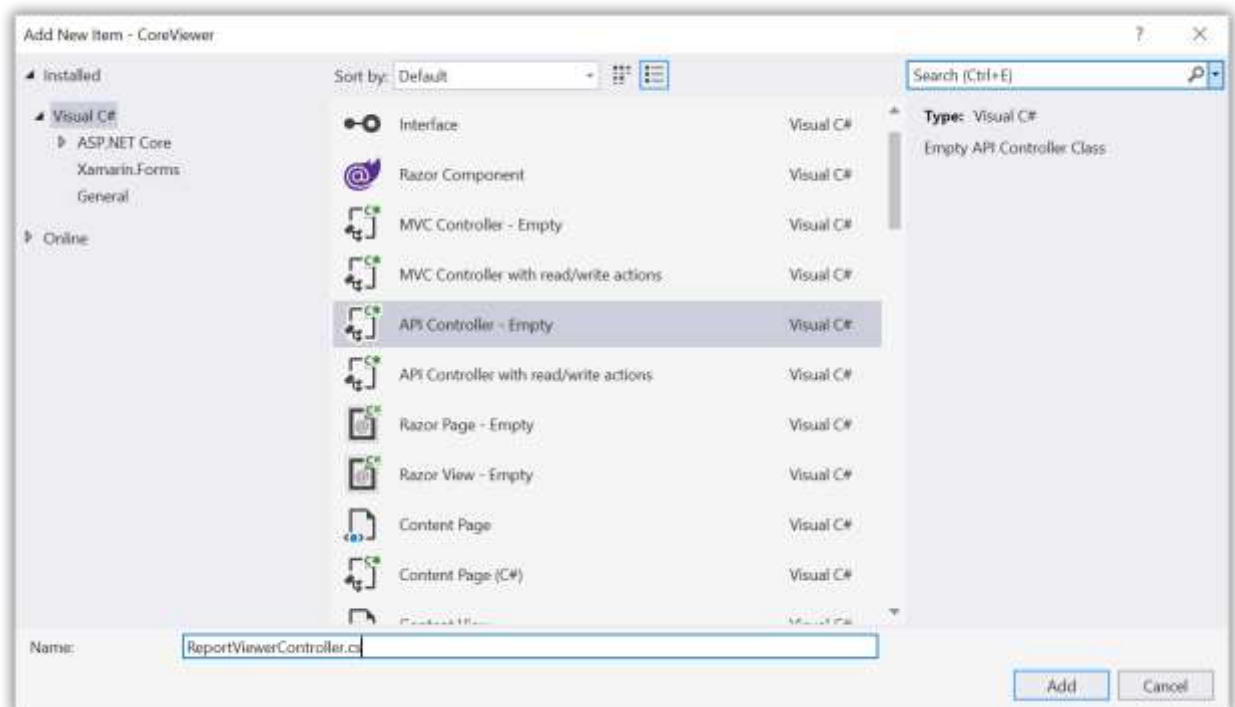
The Report Viewer is only for rendering reports. Refer to the [create RDL report](#) section to create a report.

Configure Web API

The ASP.NET Core Report Viewer requires a Web API service to process the RDL, RDLC, and SSRS report files.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller Empty** class and name it as `ReportViewerController.cs`.



3. Click **Add**.

While adding the API Controller class, naming it with the suffix `Controller` that is mandatory.

4. Open the `ReportViewerController` and add the following using statement.

```
`csharp
using System.IO;
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the `IReportController` interface, and then implement its methods.

It is required for processing the reports and for handling requests from the Report Viewer.

6. Create local variables inside the `ReportViewerController` class.

```
`csharp
//Report Viewer requires a memory cache to store the information of consecutive client request and
have the rendered report viewer information in server
private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
// IWebHostEnvironment used to get the report stream from application wwwroot\Resources folder.
private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
`
```

7. Load the report as a stream in the `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
    reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}
`
```

You cannot load the report stored in the application with path information from an ASP.NET Core service.

8. Set the `Route` attribute for `ReportViewerController`.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    ...
}
```

9. You can replace the template code with the following code.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered Report Viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
```

```
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
{
    //Contains helper methods that help to process a Post or Get request from the Report Viewer control
    and return the response to the Report Viewer control
    return ReportHelper.ProcessReport(jsonArray, this, this._cache);
}

// Method will be called to initialize the report information to load the report with ReportHelper for
processing.
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
    reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}

// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}

//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
    return ReportHelper.GetResource(resource, this, _cache);
}
```

```
}  
[HttpPost]  
public object PostFormReportAction()  
{  
    return ReportHelper.ProcessReport(null, this, _cache);  
}  
}
```

Set report path and service URL

To render the reports available in the application, of the Report Viewer. You can replace the following code on your Report Viewer page.

1. Open the `Index.cshtml` page.
2. Set the `report-path` and `report-service-url` properties as shown below.

```
`html  
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-  
url="/api/ReportViewer"></bold-report-viewer>
```

The report path property is set to the RDL report that is added to the project `Resources` folder.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

1 / 1 75%

Sales Order No: 5050750 View Report

Adventure Works cyclex

Sales Order

Order ID #5050750	Billing Date 22-04-2019	Order Date 01-06-2003	Purchase Order PO7192170677	Shipment Method CARGO TRANSPORT 5
-----------------------------	-----------------------------------	---------------------------------	---------------------------------------	---

Billing Address Jean Handley Central Discount Store 259826 Russell Rd. South Kerr, Washington 98031 United States	Shipping Address Jean Handley Central Discount Store 259826 Russell Rd. South Kerr, Washington 98031 United States	Contact Jean Handley Ph: 562-555-0113
--	---	--

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0162-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.84	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	B6-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	B6-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

We hereby certify that the information on this invoice is true and correct and that the contents of this shipment are as stated above.

Signature of Authorized Person

Note: You can refer to our feature tour page for the [ASP.NET Core Report Viewer](#) to see its innovative features. Additionally, you can view our [ASP.NET Core Report Viewer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

See Also

[Create your first app in ASP.Net Core 2.1 version](#)

[Create your first app in ASP.Net Core 3.1 version](#)

[Render report with data visualization report items](#)

[Create RDLC report](#)

[Render RDLC reports](#)

[Preview report in print mode](#)

[Set data source credential for shared data sources](#)

[Change data source connection string](#)

[Create your first app in visual studio 2017](#)

[List of SSRS server versions are supported in Bold Reports](#)

Display SSRS RDL report in an ASP.NET Core Razor Pages

Create your first ASP.NET Core Razor Pages application to display an already created SSRS RDL report in the Bold Reports ASP.NET Core Report Viewer without using a Report Server using this step-by-step instruction.

Bold Reports ASP.NET Core supports from **.NET Core 2.1** only. So, choose the .NET Core version **ASP.NET Core 2.1** or higher versions for Viewer API creation.

The source code for this ASP.NET Core Razor Pages application is available on [GitHub](#).

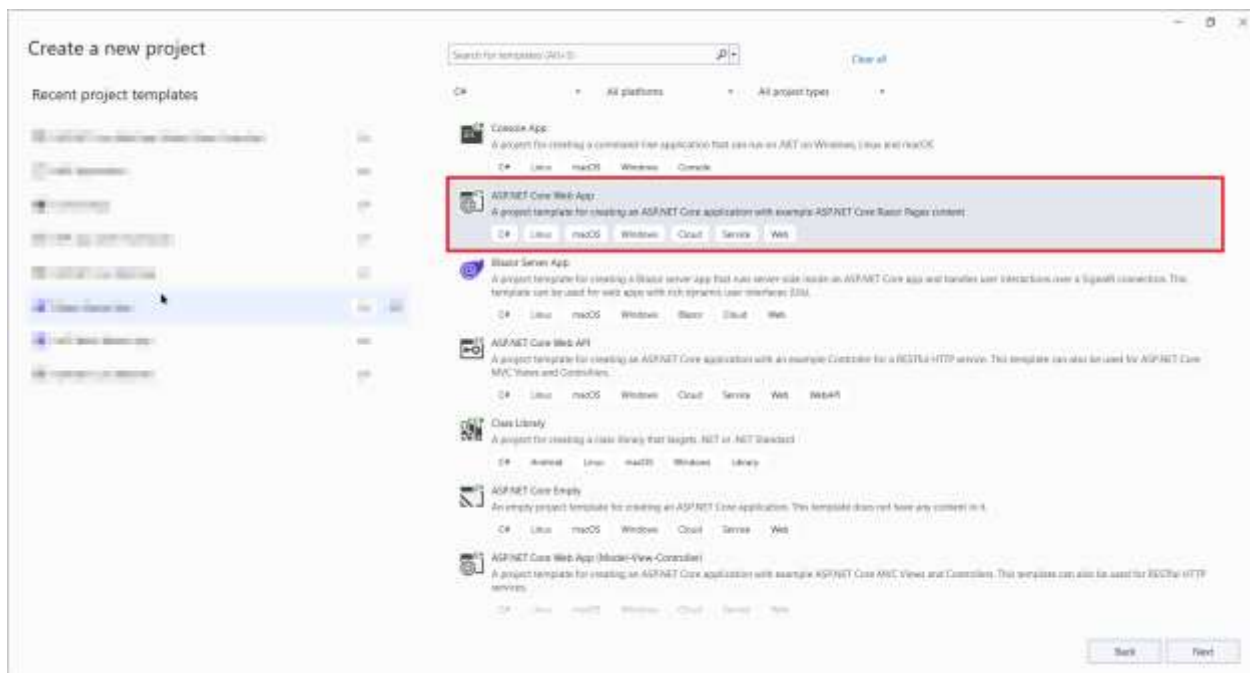
Prerequisites

Before getting started with a bold web report viewer, make sure your development environment includes the following requirements.

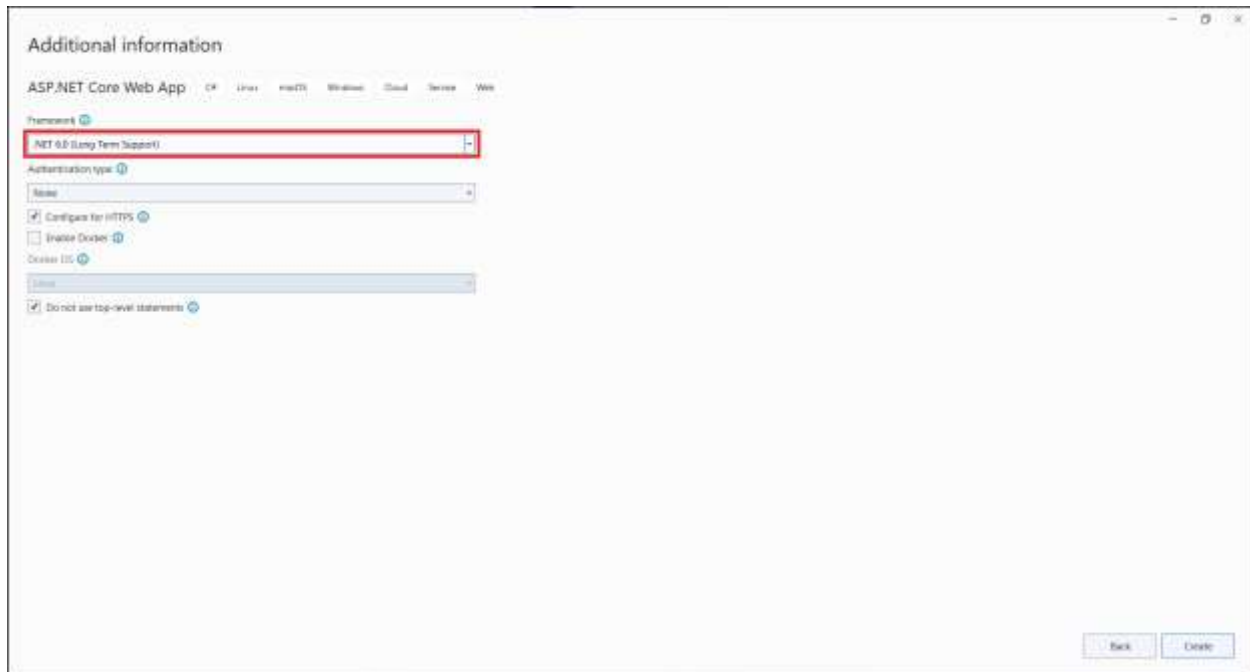
- [Visual Studio 2022](#) with ASP.NET and Web Development workloads.
- [.NET Core 6.0](#) Framework.

Create an ASP.NET Core application

1. Start Visual Studio 2022 and click **Create new project**.
2. Choose **ASP.NET Core Web App**, and then click **Next**.



3. Change the project name, and then click **Next**.
4. In the dropdown for the **Framework**, choose **.NET 6.0**, then click **Create**.



List of dependency libraries

1. In the Solution Explorer tab, right-click the project or solution, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.AspNetCore**, **BoldReports.Net.Core**, and **System.Data.SqlClient** packages, and install them into your Core application. The following table provides details about the packages and their usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Exports the report to a PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core**, and **Syncfusion.XlsIO.Net.Core** packages.

Syncfusion.Pdf.Net.Core | Exports the report to a PDF.

Syncfusion.DocIO.Net.Core | Exports the report to a Word.

Syncfusion.XlsIO.Net.Core | Exports the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serializes and deserializes data for the Report Viewer. It is a mandatory package for the Report Viewer, and the package version should be 10.0.1 or higher.

Refer scripts and CSS

Directly refer all the required scripts and style sheets from the [CDN](#) links.

1. The following scripts and style sheets are mandatorily required to use the Report Viewer.
 - o **bold.reports.all.min.css**

- jquery-1.10.2.min.js
 - ej2-base.min.js, ej2-data.min.js, ej2-pdf-export.min.js, ej2-svg-base.min.js, ej2-lineargauge.min.js and ej2-circulargauge.min.js - Render the gauge item. Add these scripts only if your report contains the gauge report item.
 - ej2-maps.min.js - Renders the map item. Add this script only if your report contains the map report item.
 - bold.reports.common.min.js
 - bold.reports.widgets.min.js
 - ej.chart.min.js - Renders the chart item. Add this script only if your report contains the chart report item.
 - bold.report-viewer.min.js
2. Open the \Pages\Shared_Layout.cshtml page.
 3. Replace the following code in your \Pages\Shared_Layout.cshtml page tag.

`html

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />

<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>

<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>

<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>

<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>

<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>

<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
```

To learn more about rendering a report with data visualization report items, refer to the [how to render data visualization report items](#) section.

The Report Viewer scripts and styles can be added into your application by installing the **BoldReports.JavaScript** online nuget package.

Tag helper

It is necessary to define the following tag helper within the `_ViewImports.cshtml` page to initialize the Report Viewer component with the tag helper support.

```
`js
@using BoldReports.TagHelpers
@addTagHelper *, BoldReports.AspNet.Core
`
```

Configure Script Manager

Open the `~/Pages/Shared/_Layout.cshtml` page and add the reporting Script Manager at the end of the `<body>` element as in the following code sample.

```
`html
<body>
<div>
@RenderBody()
</div>
@RenderSection("Scripts", required: false)
<!-- Bold Reports script manager -->
<bold-script-manager></bold-script-manager>
</body>
`
```

Initialize Report Viewer

1. Open the `Index.cshtml` page.
2. Remove the existing codes and add the following code.

```
`html
<div style="min-height: 950px;width: 100%;">
<bold-report-viewer id="viewer"></bold-report-viewer>
</div>
`
```

Add already created reports

1. Create a folder **Resources** into the **wwwroot** folder in your application to store the RDL reports.
2. Download the **sales-order-detail.rdl** from [here](#). For more sample reports, refer to the [samples and demos](#) section.
3. Extract the compressed file and paste the **sales-order-detail.rdl** to the **Resources** folder.

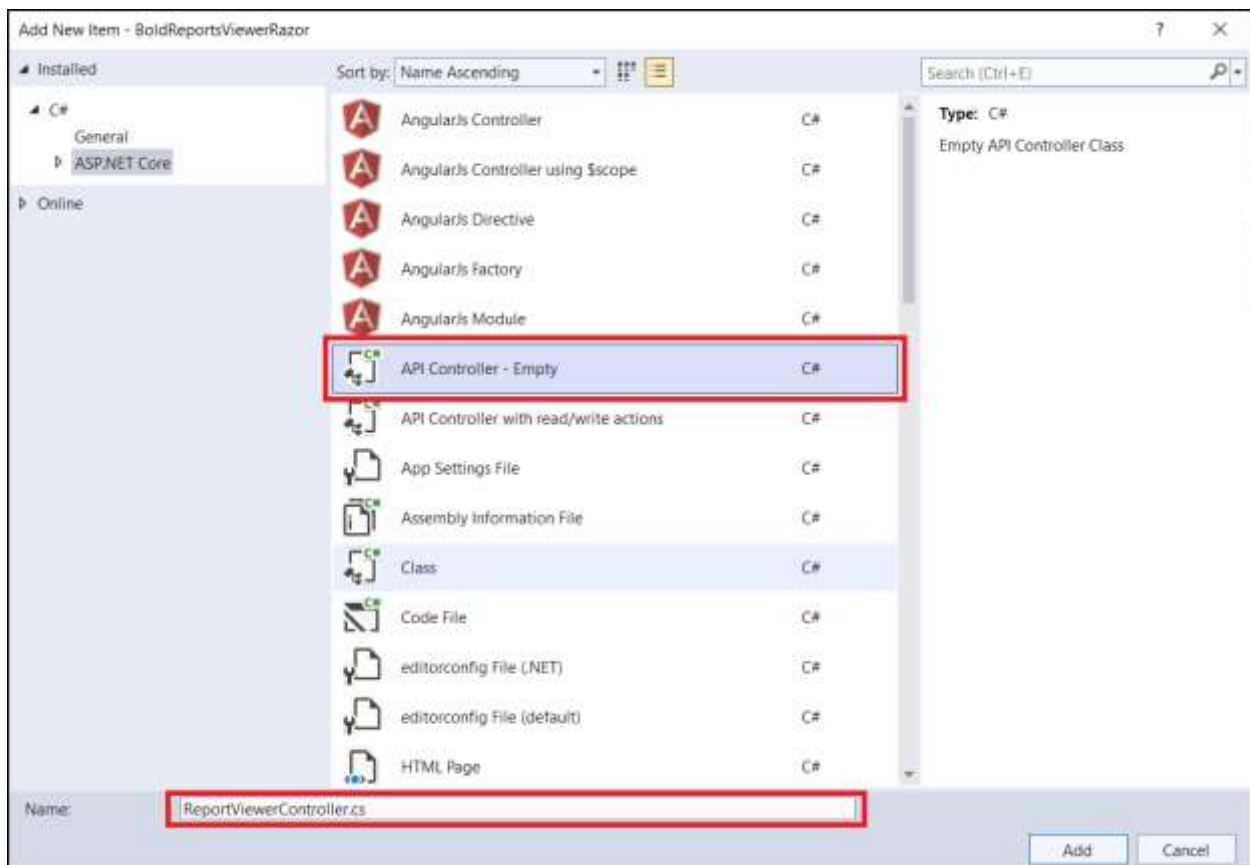
The Report Viewer is only for rendering reports. Refer to the [create RDL report](#) section to create a report.

Configure Web API

The ASP.NET Core Report Viewer requires a Web API service to process the RDL, RDLC, and SSRS report files.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller - Empty** class and name it as **ReportViewerController.cs**.



3. Click **Add**.

While adding the API Controller class, naming it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using System.IO;
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and then implement its methods.

It is required for processing the reports and for handling requests from the Report Viewer.

6. Create local variables inside the **ReportViewerController** class.

```
`csharp
//Report Viewer requires a memory cache to store the information of consecutive client request and
have the rendered report viewer information in server
private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
// IWebHostEnvironment used to get the report stream from application wwwroot\Resources folder.
private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
`
```

7. Load the report as a stream in the **OnInitReportOptions** method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
    reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
}
```

```
reportOption.ReportModel.Stream = reportStream;
}
`
```

You cannot load the report stored in the application with path information in an ASP.NET Core service.

8. Set the `Route` attribute for the `ReportViewerController`.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    ...
}
`
```

9. You can replace the template code with the following code.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered Report Viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
}
```



```
[HttpPost]
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
{
    //Contains helper methods that help to process a Post or Get request from the Report Viewer control
    and return the response to the Report Viewer control
    return ReportHelper.ProcessReport(jsonArray, this, this._cache);
}

// Method will be called to initialize the report information to load the report with ReportHelper for
processing.
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
    reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}

// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}

//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
}
```

```

return ReportHelper.GetResource(resource, this, _cache);
}
[HttpPost]
public object PostFormReportAction()
{
return ReportHelper.ProcessReport(null, this, _cache);
}
}
`

```

Web API with ASP.NET Core Web Application

Default ASP.NET Core Razor Pages template (Web Application) does not have MapControllers configuration to use the Web API. So, we need to add MapControllers options with App using the below steps,

1. Open Program.cs file.
2. Call MapControllers with app in next of app.UseRouting() as in the below code.

```

`csharp
....
....
app.UseRouting();
app.UseMvc();
....
....
`

```

Finally, your Program.cs file looks like below,

```

`csharp
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddRazorPages();
var app = builder.Build();
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
app.UseExceptionHandler("/Error");
}

```

```
app.UseHsts();  
}  
app.UseHttpsRedirection();  
app.UseStaticFiles();  
app.UseRouting();  
app.MapControllers();  
app.UseAuthorization();  
app.MapRazorPages();  
app.Run();  
`
```

If you are looking to load the report directly from the SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#) section.

Set report path and service URL

To render the reports available in the application, of the Report Viewer. You can replace the following code on your Report Viewer page.

1. Open the `Index.cshtml` page.
2. Set the `report-path` and `report-service-url` properties as in the below code.

```
`html  
<div style="min-height: 950px;width: 100%;">  
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-  
url="/api/ReportViewer"></bold-report-viewer>  
</div>  
`
```

The report path property is set to the RDL report that is added to the project `Resources` folder.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

1 / 1 75%

Sales Order No: 5050750 View Report

Adventure Works cycle

Sales Order

Order ID #5050750	Billing Date 22-04-2019	Order Date 01-06-2003	Purchase Order PO7192170677	Shipment Method CARGO TRANSPORT 5
-----------------------------	-----------------------------------	---------------------------------	---------------------------------------	---

Billing Address Jean Handley Central Discount Store 259626 Russell Rd. South Kerr, Washington 98031 United States	Shipping Address Jean Handley Central Discount Store 259626 Russell Rd. South Kerr, Washington 98031 United States	Contact Jean Handley Ph: 562-555-0113
---	--	---

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.84	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	B6-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	B6-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

We hereby certify that the information on this invoice is true and correct and that the contents of this shipment are as stated above.

Signature of Authorized Person

See Also

[Render report with data visualization report items](#)[Create RDLC report](#)[Render RDLC reports](#)[Preview report in print mode](#)[Set data source credential for shared data sources](#)[Change data source connection string](#)[Create your first app in visual studio 2017](#)[List of SSRS server versions are supported in Bold Reports](#)

Create your first ASP.NET Core reporting Web application to display already created SSRS RDL report in Bold Reports ASP.NET Core Report Viewer using the [Visual Studio for Mac](#).

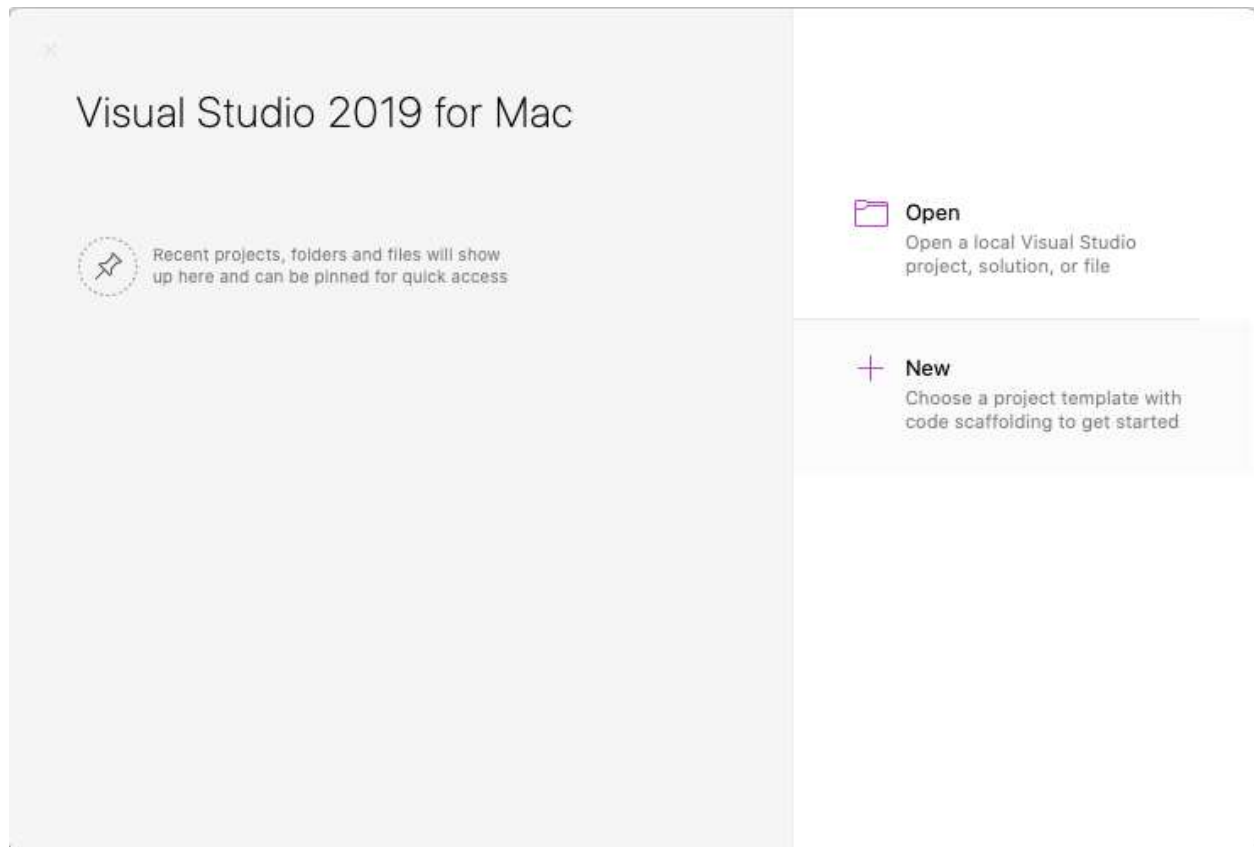
Getting started for MAC system

Prerequisites

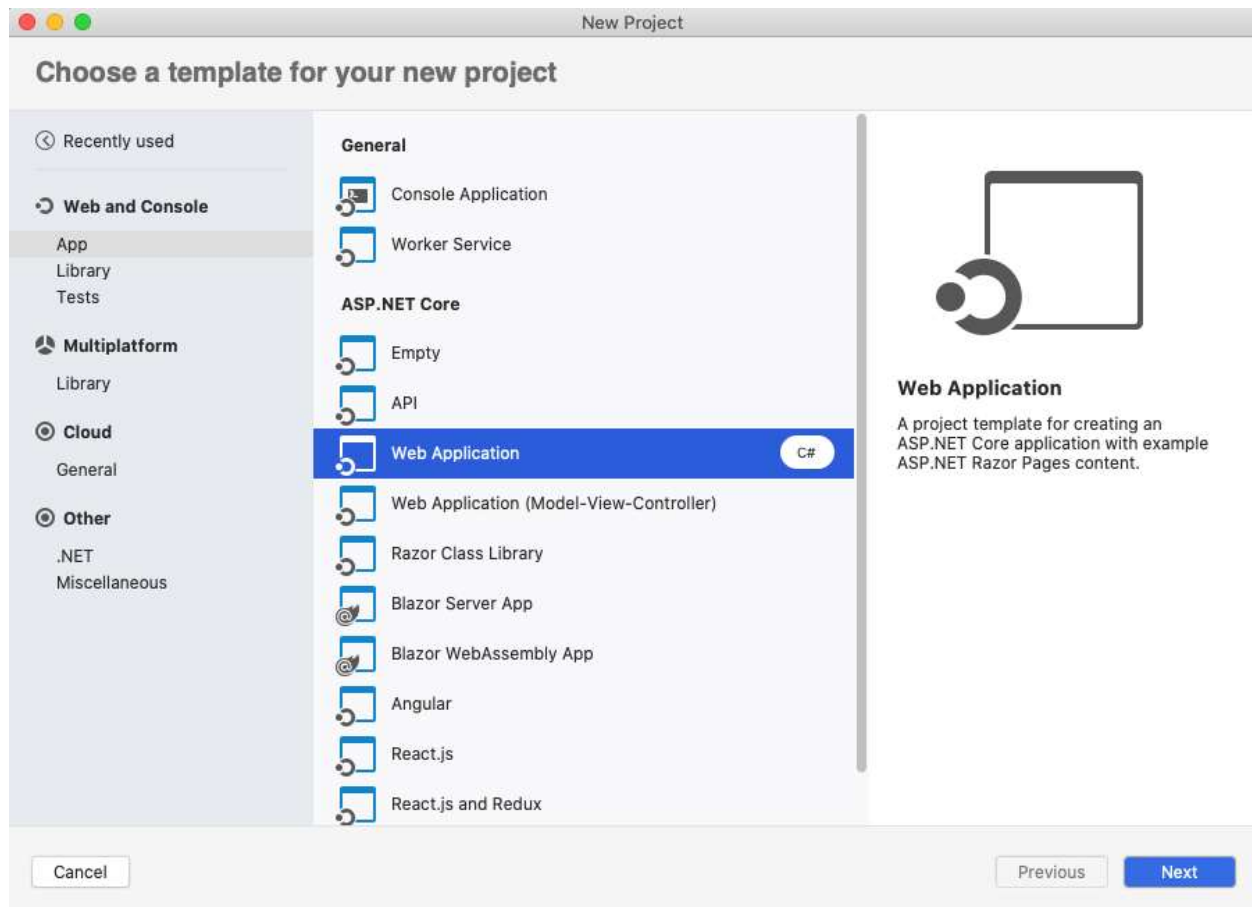
The official prerequisites to create and run an ASP.NET Core 3.x application on Mac environment are described in the [.NET Core documentation website](#).

Create ASP.NET core web application

1. Choose New from Visual Studio for Mac dashboard.



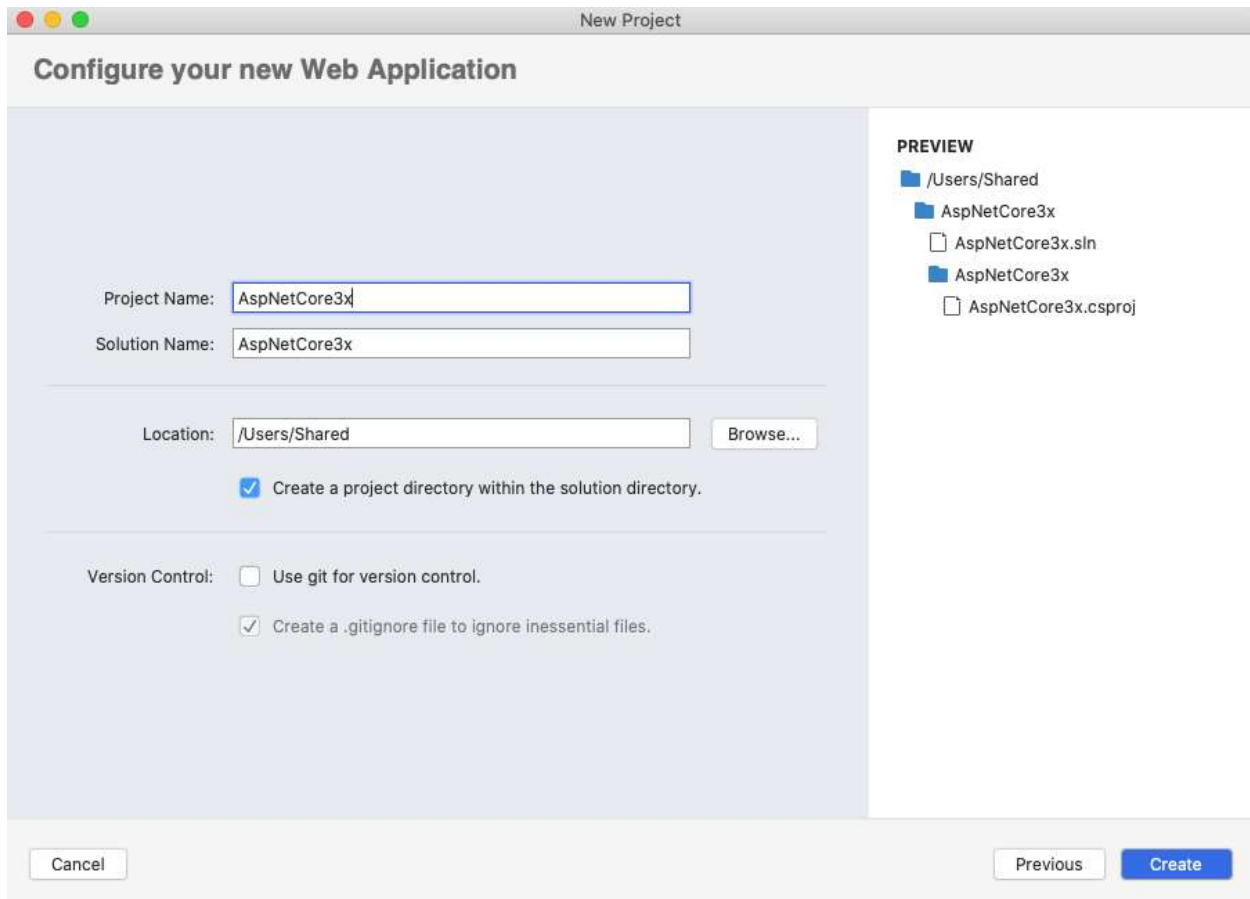
2. Select Web Application from the template in App under Web and Console section and click the Next button.



3. Select .NET Core 3.1 in Target Framework and then click Next button.

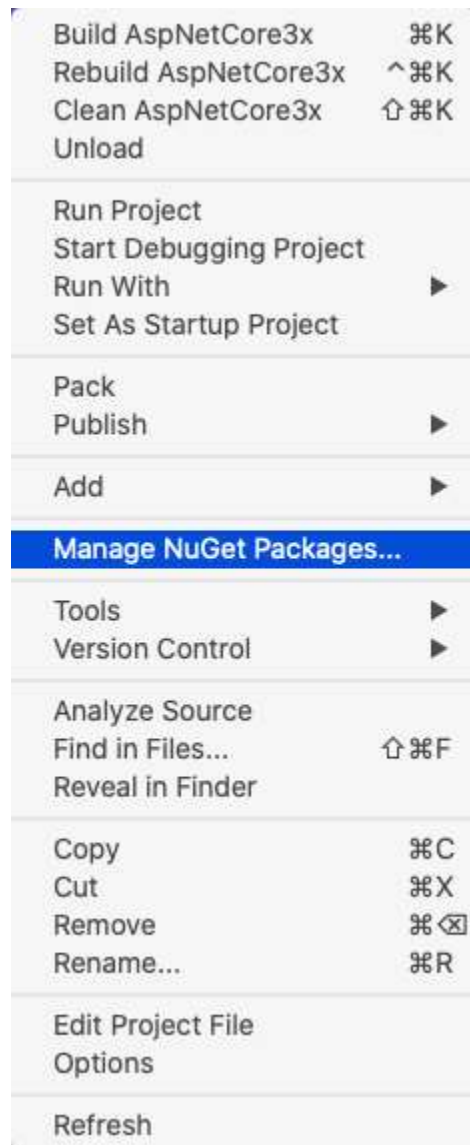


4. Now in the project configuration window, fill the Project Name and click the Create button.



List of dependency libraries

1. In the Solution Explorer tab right-click the project or solution , and choose **Manage NuGet Packages**. Alternatively, go to **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.



2. Search for **BoldReports.AspNet.Core**, **BoldReports.Net.Core** and **System.Data.SqlClient** packages, and install them in your Core application. The following table provides details about the packages and their usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Exports the report to PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core**, and **Syncfusion.XlsIO.Net.Core** packages.

Syncfusion.Pdf.Net.Core | Exports the report to a PDF.

Syncfusion.DocIO.Net.Core | Exports the report to a Word.

Syncfusion.XlsIO.Net.Core | Exports the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serializes and deserialize data for the Report Viewer. It is a mandatory package for Report Viewer, and the package version should be 10.0.1 or higher.

Refer scripts and CSS

Directly refer all the required scripts and style sheets from [CDN](#) links.

- The following scripts and style sheets are mandatorily required to use the Report Viewer.
 - bold.reports.all.min.css**
 - jquery-1.10.2.min.js**
 - ej2-maps.min.js** - Renders the map item. Add this script only if your report contains the map report item.
 - ej2-base.min.js**, **ej2-data.min.js**, **ej2-pdf-export.min.js**, **ej2-svg-base.min.js**, **ej2-lineargauge.min.js** and **ej2-circulargauge.min.js** - Render the gauge item. Add these scripts only if your report contains the gauge report item.
 - bold.reports.common.min.js**
 - bold.reports.widgets.min.js**
 - ej.chart.min.js** - Renders the chart item. Add this script, only if your report contains the chart report item.
 - bold.report-viewer.min.js**
- Open the `\Views\Shared_Layout.cshtml` page.
- Replace the following code in your `\Views\Shared_Layout.cshtml` page tag.

`html

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />

<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>

<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>

<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>

<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
```

```

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
,

```

To learn more about rendering a report with data visualization report items, refer to the [how to render data visualization report items](#) section.

The Report Viewer scripts and styles can be added into your application by installing the **BoldReports.JavaScript** online nuget package.

Tag helper

It is necessary to define the following tag helper within the `_ViewImports.cshtml` page to initialize the Report Viewer component with the tag helper support.

```

`js
@using BoldReports.TagHelpers
@addTagHelper *, BoldReports.AspNet.Core
,

```

Configure Script Manager

Open the `~/Views/Shared/_Layout.cshtml` page and add the reporting Script Manager at the end of `<body>` element as in the following code sample.

```

`html
<body>
<div style="min-height: 600px;width: 100%;">
@RenderBody()
</div>
@RenderSection("Scripts", required: false)
<!-- Bold Reports script manager -->
<bold-script-manager></bold-script-manager>
</body>
,

```

Initialize Report Viewer

1. Open the `Index.cshtml` page.
2. Remove the existing codes and add the following code.

```
`html
<bold-report-viewer id="viewer"></bold-report-viewer>
`
```

Add already created reports

1. Create a folder **Resources** into the **wwwroot** folder in your application to store the RDL reports.
2. Download the **sales-order-detail.rdl** from [here](#). For more sample reports, refer to [samples and demos](#) section.
3. Extract the compressed file and paste the **sales-order-detail.rdl** to **Resources** folder.

The Report Viewer is only for rendering the reports. Refer to the [create RDL report](#) section to create a report.

Configure Web API

The ASP.NET Core Report Viewer requires a Web API service to process the RDL, RDLC, and SSRS report files.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller** class and name it as **ReportViewerController.cs**
3. Click **Add**.

While adding API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using System.IO;
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and then implement its methods.

It is required for processing the reports and for handling request from the Report Viewer.

6. Create local variables inside the **ReportViewerController** class.

```
`csharp
//Report Viewer requires a memory cache to store the information of consecutive client request and
have the rendered report viewer information in server
```

```
private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
// IWebHostEnvironment used to get the report stream from application wwwroot\Resources folder..
private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
`
```

7. Load the report as stream in `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
    reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}
`
```

You cannot load the report stored in application with path information in ASP.NET Core service.

8. Set the `Route` attribute for `ReportViewerController`.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    ...
}
`
```

9. You can replace the template code with the following code.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered Report Viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
    public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
    {
        //Contains helper methods that help to process a Post or Get request from the Report Viewer control
        //and return the response to the Report Viewer control
        return ReportHelper.ProcessReport(jsonArray, this, this._cache);
    }
    // Method will be called to initialize the report information to load the report with ReportHelper for
    // processing.
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sales-order-detail.rdl report from application the folder
        // wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    }
}
```

```

FileStream inputStream = new FileStream(basePath + @"\Resources\" +
reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
MemoryStream reportStream = new MemoryStream();
inputStream.CopyTo(reportStream);
reportStream.Position = 0;
inputStream.Close();
reportOption.ReportModel.Stream = reportStream;
}

// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}

//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
return ReportHelper.GetResource(resource, this, _cache);
}

[HttpPost]
public object PostFormReportAction()
{
return ReportHelper.ProcessReport(null, this, _cache);
}
}
,

```

Enable cross-origin requests

Browser security prevents the Report Viewer from making requests to your Web API Service when both server-side and client-side requests run in different domains. To allow access to your Web API service from a different domain, enable the cross-origin requests.

1. Open **Startup.cs** file.

2. Call `AddCors` in `Startup.ConfigureServices` to add CORS services to the app's service container as in below code.

```
`csharp
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddCors(o => o.AddPolicy("AllowAllOrigins", builder =>
    {
        builder.AllowAnyOrigin()
        .AllowAnyMethod()
        .AllowAnyHeader();
    }));
}
```

To specify the CORS policy for `ReportViewerController` add the `[EnableCors]` attribute.

```
`csharp
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    ....
    ....
}
```

If you are looking to load the report directly from SQL Server Reporting Services (SSRS), then you can skip the following steps and move to [SSRS Report](#) section.

Set report path and service URL

To render the reports available in the application, of the Report Viewer. You can replace the following code in your Report Viewer page.

1. Open the `Index.cshtml` page.
2. Set the `report-path` and `report-service-url` properties as in below.

```
`html
```



```
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-url="/api/ReportViewer"></bold-report-viewer>
```

The report path property is set to the RDL report that is added to the project **Resources** folder.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

Sales Order No: View Report

REVENUE cyclex

Sales Order

Order ID	Billing Date	Order Date	Purchase Order	Shipment Method
#SO50750	22-04-2019	01-06-2003	PO7192170677	CARGO TRANSPORT 5

Billing Address		Shipping Address		Contact
Jean Handley Central Discount Store 259628 Russell Rd. South Kerr, Washington 98031 United States		Jean Handley Central Discount Store 259628 Russell Rd. South Kerr, Washington 98031 United States		Jean Handley Ph: 562-555-0113

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-7098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0162-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.04	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	BK-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	BK-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

We hereby certify that the information on this invoice is true and correct and that the contents of this shipment are as stated above.

Signature of Authorized Person

See Also

[Render report with data visualization report items](#)

[Create RDLC report](#)

[Render RDLC reports](#)

[Preview report in print mode](#)

[Set data source credential for shared data sources](#)

[Change data source connection string](#)

[List of SSRS server versions are supported in Bold Reports](#)

Load SSRS Report Server reports

Report Viewer has support to load RDL reports from SSRS Report Server. To render SSRS Reports, set the `reportServerUrl`, `reportPath`, and `reportServiceUrl` properties as shown in the following steps.

1. To create your first ASP.NET Core reporting application, refer to the [Getting-started](#) section.

If you need to know about the difference between `reportServiceUrl` and `reportServerUrl`, then refer to [Difference between Report Service URL and Report Server URL](#).

2. Set the `reportServerUrl` API on Bold Report Viewer with `WebServiceURL`. Open the `Index.cshtml` and replace the following code example.

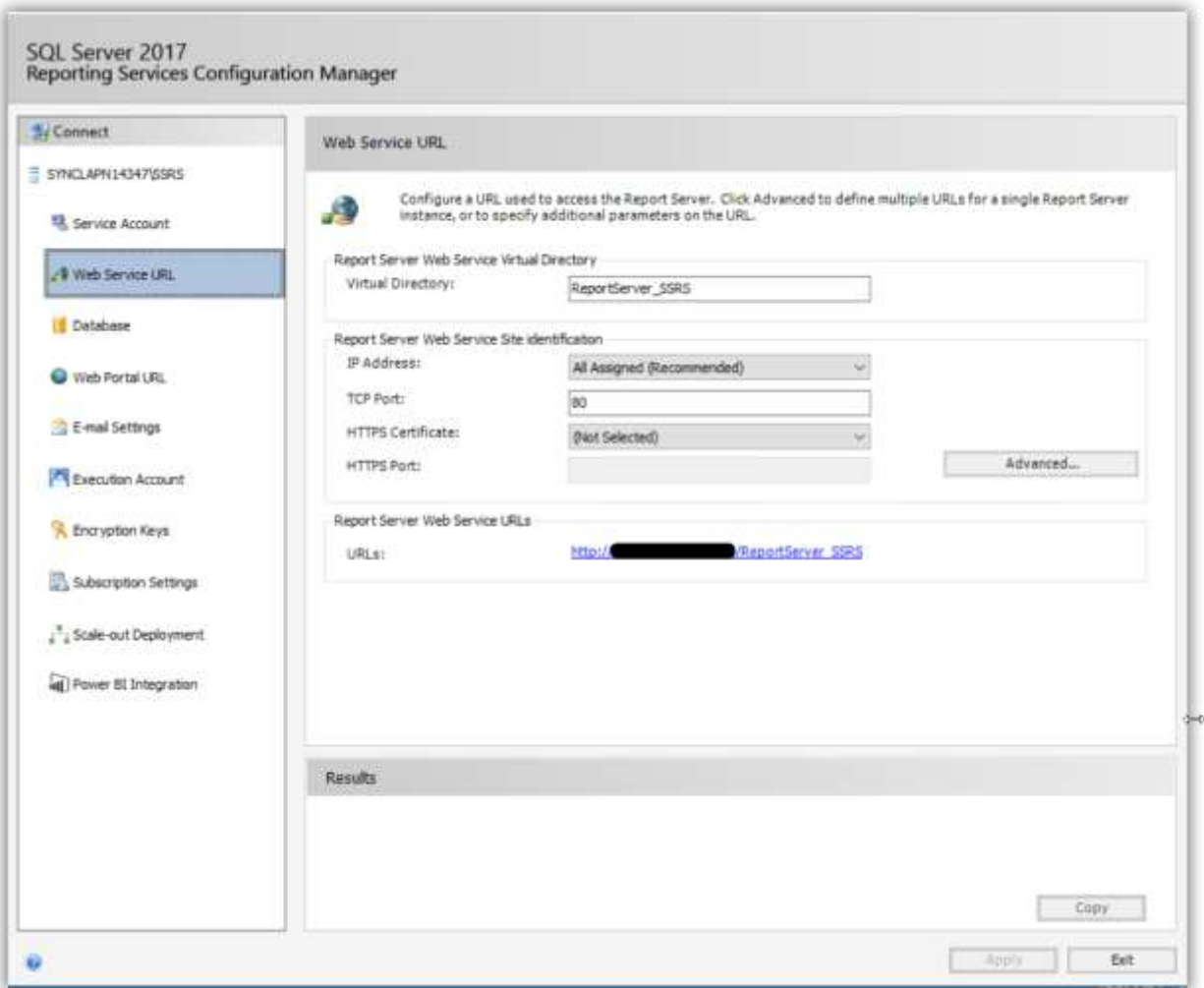
```
`html
```

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-server-url="http://<servername>/Reports_SSRS">
```

```
</bold-report-viewer>
```

```
,
```

The Web Service URL should be set as `reportServerUrl` in the report viewer configuration. The Web Service URL can be found from the Reporting Services Configuration manager under the `Web Service URL` section, as shown in the following image.

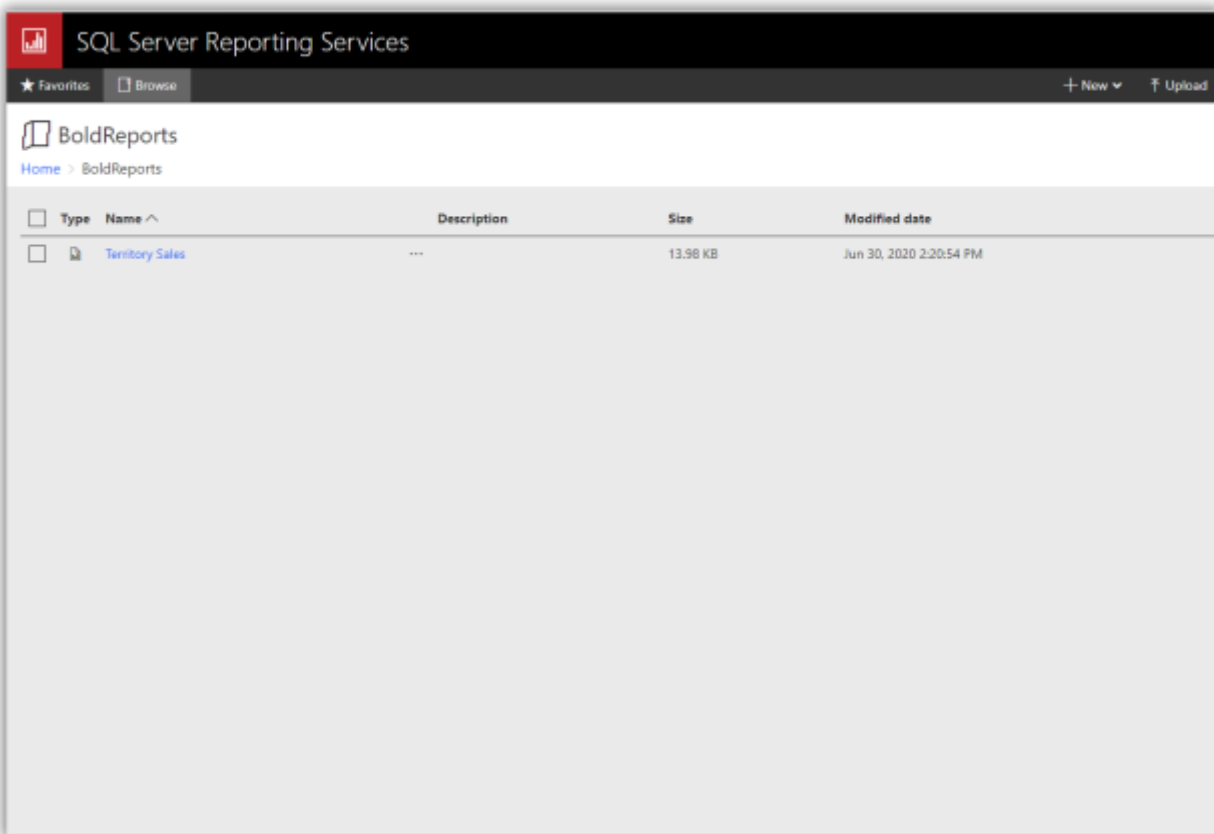


- Set the report path for loading the reports from the SSRS Report Server. The report path should be in the format of `/folder name/report name`. Open the `Index.cshtml` file and replace the following code example.

```
`html
```

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-server-url="http://<servername>/Reports_SSRS" report-path="/SSRSSamples2/Territory Sales">
</bold-report-viewer>
```

The report path can be found from the SSRS Report Server by navigating to the path of the report to be loaded, as shown in the following image.



Network credentials for SSRS

The network credentials are required to load specified SSRS report from the specified SSRS Report Server using the Report Viewer. Specify the `ReportServerCredential` property in the Web API Controller `OnInitReportOptions` method.

```
`csharp
```

```
[NonAction]
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
```

```
//Add SSRS Report Server credential
```

```
reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",  
"RDLReport1");  
}  
`
```

If you are facing problem to access the SSRS Report server reports, you can refer [How to provide the permission for user to access the SSRS Report Server reports](#).

Set data source credential to shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS Server. If the report has any data source that uses credentials to connect with the database, then you should specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp  
[NonAction]  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
    //Add SSRS Report Server and data source credentials  
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",  
"RDLReport1");  
    reportOption.ReportModel.DataSourceCredentials.Add(new  
    BoldReports.Web.DataSourceCredentials("<database>", "<username>", "<password>"));  
}  
`
```

Data source credentials should be added to the shared data sources that do not have credentials in the connection strings.

Change data source connection string

You can change the connection string of a report data source before it is loaded in the Report Viewer. The `DataSourceCredentials` class provides the option to set and update the modified connection string as in the following code snippet.

```
`csharp  
[NonAction]  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
    reportOption.ReportModel.DataSourceCredentials.Add(new BoldReports.Web.DataSourceCredentials()  
    { Name = "<database>", ConnectionString = "Data Source=<instancename>;Initial Catalog=<database>;"  
    , UserId = "<username>" , Password = "<password>"});  
}  
`
```

,

The previous code shows an option to change the connection string only, but the class provides multiple options to change data source information. To learn more about this, refer to this [DataSourceCredentials](#) class.

See also

[Does Bold Report Viewer use SSRS Report processing?](#)

Load SharePoint Server reports

To render the SharePoint server reports, set the `report-server-url`, `report-path`, and `report-service-url` properties as shown in the following code snippet.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-server-url="http://<servername>/reportserver$instanceName" report-
path="http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl">
</bold-report-viewer>
```

,

In SharePoint integrated mode, the `report-server-url` will be same as your site URL. The `report-path` is relative to the Report Server URL with the file extension.

Forms credential for SharePoint Server

The forms credentials are required to load the SharePoint integrated SSRS report from the specified SharePoint integrated SSRS Report Server using the Report Viewer. Specify the `ReportServerFormsCredential` property in the Web API Controller `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add ReportServerFormsCredential for server
    reportOption.ReportModel.ReportServerFormsCredential = new
    BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");
}
```

,

Set data source credential to shared data sources

The shared data source credentials can be added to the `DataSourceCredentials` property to connect with the database.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```

{
//Add ReportServerFormsCredential and data source credentials
reportOption.ReportModel.ReportServerFormsCredential = new
BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");
reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
}
`

```

Data source credentials should be added to shared data sources that do not have credentials in the connection strings.

Render RDLC report

The data binding support allows you to view the RDLC reports that exist on the local file system with JSON array and custom business object data collection. The following steps demonstrates how to render a RDLC report with JSON array and custom business object data collection.

Add the RDLC report **Product List.rdlc** from Bold Reports installation location to your application **wwwroot/Resources** folder. For more information, see [Samples and demos](#).

Bind data source in Web API controller

The following steps help you to configure the Web API to render the RDLC report with business object data collection.

1. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```

`csharp
public class ProductList
{
public string ProductName { get; set; }
public string OrderId { get; set; }
public double Price { get; set; }
public string Category { get; set; }
public string Ingredients { get; set; }
public string ProductImage { get; set; }
public static IList GetData()
{
List<ProductList> datas = new List<ProductList>();
ProductList data = null;

```

```
data = new ProductList()
{
    ProductName = "Baked Chicken and Cheese",
    OrderId = "323B60",
    Price = 55,
    Category = "Non-Veg",
    Ingredients = "grilled chicken, corn and olives.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Delite",
    OrderId = "323B61",
    Price = 100,
    Category = "Non-Veg",
    Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
```

2. Set the value of the `ProcessingMode` property to `ProcessingMode.Local` in the RDLC report location.
3. To load a report as a stream, create a report stream using the `FileStream` class, and assign the report stream to the `Stream` property.
4. Bind the business object data values collection by adding a new item to the `DataSources` as in the following code snippet.

```
`csharp
private IMemoryCache _cache;
private IWebHostEnvironment _hostingEnvironment;
public ReportViewerController(IMemoryCache memoryCache, IWebHostEnvironment
hostingEnvironment)
{
    _cache = memoryCache;
    _hostingEnvironment = hostingEnvironment;
}
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    reportOption.ReportModel.ProcessingMode = ProcessingMode.Local;
    FileStream inputStream = new FileStream(basePath + @"\Resources\Product List.rdlc", FileMode.Open,
    FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
    Value = ProductList.GetData() });
}
```

Here, the `Name` is case sensitive and it should be same as in the data source name in the report definition.

The `Value` accepts `ICollection`, `DataSet`, and `DataTable` inputs.

In the previous code, the `Product List.rdlc` report is loaded from the `wwwroot/Resources` folder location.

Bind data source with razor view

The following steps help you to provide the data source for a Report Viewer in razor view using the ViewBag.

1. Create a class and methods that returns business object data collection. Use the following code in your application.

```
`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = null;
        data = new ProductList()
        {
            ProductName = "Baked Chicken and Cheese",
            OrderId = "323B60",
            Price = 55,
            Category = "Non-Veg",
            Ingredients = "grilled chicken, corn and olives.",
            ProductImage = ""
        };
        datas.Add(data);
        data = new ProductList()
        {
            ProductName = "Chicken Delite",
```

```

OrderId = "323B61",
Price = 100,
Category = "Non-Veg",
Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
`

```

2. Create **BoldReports.Models.ReportViewer.ReportDataSource** collection with business object collection as follows and assign to the ViewBag to provide the data for report dataset.

```

`csharp
public IActionResult Index()
{
    BoldReports.Models.ReportViewer.ReportDataSource reportDataSource = new BoldReports.
    Models.ReportViewer.ReportDataSource();
    reportDataSource.Name = "list";
    reportDataSource.Value = ProductList.GetData();
    ViewBag.dataSources = new List<BoldReports.Models.ReportViewer.ReportDataSource> {
    reportDataSource };
    return View();
}

```

```
}
,
```

3. Pass the ViewBag value to the **dataSources** property of Report Viewer.

```
`html
<bold-report-viewer id="viewer"
report-service-url="/api/ReportViewer"
dataSources="ViewBag.dataSources"
processing-mode="Local" >
</bold-report-viewer>
,
```

View report click

You can get the user selected parameter details when users click the **ViewReport** button in the parameter block. The **view-report-click** event allows you to handle the **ViewReport** button click at client-side as shown in the following code.

```
`html
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-
url="/api/ReportViewer" view-report-click="onViewReportClick" processing-mode="Remote"></bold-
report-viewer>
<script type="text/javascript">
function onViewReportClick(args) {
args[0] = ({ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'], nullable: false });
}
</script>
,
```

The model property in the event argument has the details of current processing report model.

Render subreport

You can display another report inside the body of main report using the Report Viewer. The following steps help you to customize the subreport properties such as data source, report path, and parameters.

1. Add the sub report and main reports to the application **wwwroot/Resources** folder. In this tutorial, the already created reports are used. Refer to the [Create RDL Report section](#) or [Create RDLC Report section](#) section.

Download the **SideBySideMainReport.rdl**, **SideBySideSubReport.rdl** reports from [here](#). You can add a report from the Bold Reports installation location. For more information, refer to [Samples and demos](#).

The reports used from installed location requires `NorthwindIO_Reports.sdf` database to run, so add the database to your application.

- Set the `report-path`, `processing-mode`, and `report-service-url` properties of the Report Viewer as in the following code snippet.

The following code example demonstrates how to load a subreport in the Report Viewer at client side.

```
`html
```

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" report-
path="SideBySideMainReport.rdl" processing-mode="Remote"></bold-report-viewer>
```

```
,
```

The following code example demonstrates how to load a subreport in the Report Viewer at server side.

```
`csharp
```

```
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered report viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
    public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
    {
        return ReportHelper.ProcessReport(jsonArray, this, this._cache);
    }
}
```

// Method will be called to initialize the report information to load the report with ReportHelper for processing.

[NonAction]

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
    string basePath = _hostingEnvironment.WebRootPath;
```

```
    // Here, we have loaded the SideBySideSubReport.rdl report from application the folder  
    wwwroot\Resources and loads the sub report stream.
```

```
    if (reportOption.SubReportModel != null)
```

```
    {
```

```
        FileStream inputSubStream = new FileStream(basePath + @"\Resources\" +  
        reportOption.SubReportModel.ReportPath, FileMode.Open, FileAccess.Read);
```

```
        MemoryStream SubStream = new MemoryStream();
```

```
        inputSubStream.CopyTo(SubStream);
```

```
        SubStream.Position = 0;
```

```
        inputSubStream.Close();
```

```
        reportOption.SubReportModel.Stream = SubStream;
```

```
    }
```

```
    else
```

```
    {
```

```
        FileStream inputStream = new FileStream(basePath + @"\Resources\" +  
        reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
```

```
        MemoryStream reportStream = new MemoryStream();
```

```
        inputStream.CopyTo(reportStream);
```

```
        reportStream.Position = 0;
```

```
        inputStream.Close();
```

```
        reportOption.ReportModel.Stream = reportStream;
```

```
    }
```

```
}
```

// Method will be called when reported is loaded with internally to start to layout process with ReportHelper.

[NonAction]

```
public void OnReportLoaded(ReportViewerOptions reportOption)
```

```
{
```

```
}
```

```
//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
    return ReportHelper.GetResource(resource, this, _cache);
}
[HttpPost]
public object PostFormReportAction()
{
    return ReportHelper.ProcessReport(null, this, _cache);
}
}
```

Change subreport path

To change the subreport file path, set the **Stream** property of SubReportModel in the **OnInitReportOptions** method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        FileStream inputSubStream = new FileStream(basePath + @"\Resources\" +
            reportOption.SubReportModel.ReportPath, FileMode.Open, FileAccess.Read);
        MemoryStream SubStream = new MemoryStream();
        inputSubStream.CopyTo(SubStream);
        SubStream.Position = 0;
        inputSubStream.Close();
        reportOption.SubReportModel.Stream = SubStream;
    }
}
}
```

Set subreport parameter

You can change the parameter default values of a subreport in the `OnReportLoaded` method of the Web API Controller as given in the following code snippet.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.Parameters = new BoldReports.Web.ReportParameterInfoCollection();
        reportOption.SubReportModel.Parameters.Add(new BoldReports.Web.ReportParameterInfo()
        {
            Name = "SalesPersonID",
            Values = new List<string>() { "2" }
        });
    }
}
```

Modify subreport data source connection string

You can change the credential and connection information of the data sources used in the subreport using the `SubReportModel` in the `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSourceCredentials = new
        List<BoldReports.Web.DataSourceCredentials>();

        BoldReports.Web.DataSourceCredentials dataSourceCredentials = new
        BoldReports.Web.DataSourceCredentials();

        dataSourceCredentials.Name = "<database>";

        dataSourceCredentials.ConnectionString = "Data Source=<instancename>;Initial
        Catalog=<database>;user id=<username>;password=<password>";

        reportOption.SubReportModel.DataSourceCredentials.Add(dataSourceCredentials);
    }
}
```

```
}
}
,
```

Set subreport data source

You can bind local business object data source collection only for RDLC reports. To specify data source of a RDLC subreport, set the `ReportDataSource` property in the `OnReportLoaded` method.

The RDL report has the connection information in report definition itself, so no need to bind data source.

1. Add the RDLC sub report and main reports to your application `wwwroot/Resources` folder. You can download it from [here](#).
2. Set the `report-path`, `processing-mode`, and `report-service-url` properties of the Report Viewer as shown in following code snippet.

```
`js
```

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" report-path="Product List Main.rdlc" processing-mode="Local"></bold-report-viewer>
```

```
,
```

3. Create a class and methods that returns business object data collection. Use the following code in the application Web API Service.

```
`csharp
```

```
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = null;
        data = new ProductList()
        {
            ProductName = "Baked Chicken and Cheese",
```



```

OrderId = "323B60",
Price = 55,
Category = "Non-Veg",
Ingredients = "grilled chicken, corn and olives.",
ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Delite",
    OrderId = "323B61",
    Price = 100,
    Category = "Non-Veg",
    Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
`

```

4. Bind the business object data values collection to the subreport by adding a new item to the **DataSources** as shown in the following code snippet.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'Product List.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
"list", Value = ProductList.GetData() });
    }
}
`
```

The data source name is case sensitive, it should be same as in the report definition.

Load subreport stream

To load subreport as stream, set the **Stream** property in the **OnInitReportOptions** method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    if (reportOption.SubReportModel != null)
    {
        // Here, we have loaded the Product List.rdlc report from application the folder wwwroot\Resources
        and loads the sub report stream.

        FileStream inputSubStream = new FileStream(basePath + @"\Resources\" +
reportOption.SubReportModel.ReportPath, FileMode.Open, FileAccess.Read);
        MemoryStream SubStream = new MemoryStream();
        inputSubStream.CopyTo(SubStream);
        SubStream.Position = 0;
        inputSubStream.Close();
        reportOption.SubReportModel.Stream = SubStream;
    }
    else

```

```

{
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}
}
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'Product List.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
"list", Value = ProductList.GetData() });
    }
}
`

```

Report parameters

Provides property options to pass or set report parameters default values at run time using the **parameters** property. You can set the report parameters while creating the Report Viewer control in a script or in the Web API Controller.

In this tutorial, the **sales-order-detail.rdl** report is used, and it can be downloaded from [here](#).

Set a parameter with razor view

Report Viewer **parameter** property does not have support to create parameter collection with razor view. So, we have to create and assign the parameter with help of **ViewBag** as follows.

```

`csharp
public ActionResult Index()
{
    List<BoldReports.Models.ReportViewer.ReportParameter> parameters = new
List<BoldReports.Models.ReportViewer.ReportParameter>();
}

```

```

parameters.Add(new BoldReports.Models.ReportViewer.ReportParameter()
{
    Name = "SalesOrderNumber",
    Values = new List<string>() { "SO50756" }
});
ViewBag.parameters = parameters;
return View();
}
`html
<bold-report-viewer id="reportviewer1" report-service-url="../ReportApi" processing-mode="Remote"
parameters="ViewBag.parameters" />
`

```

Set parameters in Web API Controller

To set parameter default value in Web API Controller, use the following code in the **OnReportLoaded** method.

```

`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    List<BoldReports.Web.ReportParameter> userParameters = new
    List<BoldReports.Web.ReportParameter>();
    userParameters.Add(new BoldReports.Web.ReportParameter()
    {
        Name = "SalesOrderNumber",
        Values = new List<string>() { "SO50756" }
    });
    reportOption.ReportModel.Parameters = userParameters;
}
`

```

The Report Parameters name should be case sensitive

Get report parameter

The **ReportHelper** class provides methods to get the report parameters used in the report. The following helper methods used to get parameter with or without values.

Methods | Description

MaxDateTime | Specify minimum range value of a date parameter

MinDateTime | Specify maximum range value of a date parameter

Refer to the following code sample to set data range using **parameterSettings** in Report Viewer controller.

```
`csharp
public ActionResult Index()
{
    ViewBag.parameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
    ViewBag.parameterSettings.MaxDateTime = new DateTime(2003, 8, 22);
    ViewBag.parameterSettings.MinDateTime = new DateTime(2003, 4, 22);
    return View();
}
```

Refer to the following code sample to set data range to the report viewer initialization.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
parameter-settings="ViewBag.parameterSettings"> </bold-report-viewer>
```

The above code sets date range for all the date parameters used in the report.

To set different date range for each date parameter used in the report, register the event **beforeParameterAdd** and specify range based on parameter name as in below code sample.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" report-path="product-line-
sales.rdl" before-parameter-add="beforeParameterAdd"> </bold-report-viewer>
<script type="text/javascript">
function beforeParameterAdd(args) {
    if (args.parameterModel.Name === "StartDate") {
        args.parameterSettings.minDateTime = new Date("4/5/2003 5:00:00 AM");
        args.parameterSettings.maxDateTime = new Date("4/15/2003 5:00:00 AM");
    }
    if (args.parameterModel.Name === "EndDate") {
        args.parameterSettings.minDateTime = new Date("5/10/2003 5:00:00 AM");
        args.parameterSettings.maxDateTime = new Date("5/20/2003 5:00:00 AM");
    }
}
```

```

}
</script>
`

```

Set date time display format for date parameter

The properties `dateTimeFormat` and `timeDisplayFormat` in the `parameterSettings` are used to set date and time format to be displayed in the `DateTimePicker` control in a report.

Format | Display in `DateTimePicker`

Short Date and Time - `d/M/yy h:mm tt` | 9/12/2014 2:04 PM

Medium Date - `d MMM yy h:mm tt` | 12 Sep 14 2:04: PM

Full Date and short time - `dddd, MMMM dd, yyyy HH:mm tt` | Friday, September 12,2014 2:04 PM

Full Date and Long Time - `dddd, MMMM dd, yyyy HH:mm:ss tt` | Friday, September 12,2014 2:04:00 PM

UTC - `yyyy-MM-dThh:mm:ssz` | 2014-09-12T2:04:00+5

Refer to the following code sample to set date and time format to be displayed, using **parameterSettings** in Report Viewer controller.

```

`csharp
public ActionResult Index()
{
    ViewBag.parameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
    ViewBag.parameterSettings.DateTimeFormat = "d/M/yyyy h:mm tt";
    ViewBag.parameterSettings.TimeDisplayFormat = "HH:mm";
    ViewBag.parameterSettings.TimeInterval = 60;
    return View();
}
`

```

Refer to the following code sample to set date and time format to be displayed in the report viewer initialization.

```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
parameter-settings="ViewBag.parameterSettings"> </bold-report-viewer>
`

```

The above code sets date and time value to be display for all the date parameters used in the report.

To set different date and time value to be display for each date parameter used in the report, register the event `beforeParameterAdd` and specify date and time value based on parameter name as in below code sample.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" report-path="product-line-
sales.rdl" before-parameter-add="beforeParameterAdd"> </bold-report-viewer>

<script type="text/javascript">
function beforeParameterAdd(args) {
if (args.parameterModel.Name === "StartDate") {
args.parameterSettings.dateTimeFormat = "d/M/yyyy h:mm tt";
args.parameterSettings.timeDisplayFormat = "HH:mm";
args.parameterSettings.timeInterval = 60;
}
if (args.parameterModel.Name === "EndDate") {
args.parameterSettings.dateTimeFormat = "d/M/yyyy h:mm tt";
args.parameterSettings.timeDisplayFormat = "HH:mm";
args.parameterSettings.timeInterval = 60;}
}
</script>
`
```

Change the Parameter drop-down height and width

The `parameterSettings` helps you to change the height and width of the parameter available in parameter panel.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
parameter-settings="ViewBag.parameterSettings"> </bold-report-viewer>
`

`csharp
public ActionResult Index()
{
    ViewBag.parameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
    ViewBag.parameterSettings.PopupHeight = "200px";
    ViewBag.parameterSettings.PopupWidth = "100px";
    return View();
}
```

,

Hide a Parameter scroller

The `enableparameterblockscroller` helps you to hide the scrollbar in parameter panel.

`html

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
enable-parameter-block-scroller="true"> </bold-report-viewer>
```

,

Hide a Parameter Pane on load

The `parameterSettings` helps you to hide and show the parameter block.

`html

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
parameter-settings="ViewBag.parameterSettings"> </bold-report-viewer>
```

,

`csharp

```
public ActionResult Index()
```

```
{
```

```
    ViewBag.parameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
```

```
    ViewBag.parameterSettings.HideParameterBlock = true;
```

```
    return View();
```

```
}
```

,

Change the Parameter Item Width and Label Width

The `parameterSettings` helps you to change the parameter Item width and label width.

`html

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
parameter-settings="ViewBag.parameterSettings"> </bold-report-viewer>
```

,

`csharp

```
public ActionResult Index()
```

```
{
```

```
    ViewBag.parameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
```

```
    ViewBag.parameterSettings.ItemWidth = "250px";
```

```
    ViewBag.parameterSettings.LabelWidth = "auto";
```

```
    return View();
```



```
}
,
```

Access the hidden or internal parameter information

The `accessInternalValue` property in the `parameterSettings` helps you to expose the `hidden` or `internal` report parameter information used in report to the user.

```
`html
```

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
parameter-settings="ViewBag.parameterSettings"> </bold-report-viewer>
```

```
,
```

```
`csharp
```

```
public ActionResult Index()
```

```
{
```

```
    ViewBag.parameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
```

```
    ViewBag.parameterSettings.AccessInternalValue = true;
```

```
    return View();
```

```
}
```

```
,
```

Set the report parameter visibility in Web API Controller

The `Hidden` property of `ReportParameter` allows you to show or hide the parameter at the top of the report viewer panel. The following code example shows hiding a report parameter in the Web API controller's `OnReportLoaded` method.

```
`csharp
```

```
[NonAction]
```

```
public void OnReportLoaded(ReportViewerOptions reportOption)
```

```
{
```

```
    var reportParameters = ReportHelper.GetParameters(jsonArray, this, _cache);
```

```
    List<BoldReports.Web.ReportParameter> modifiedParameters = new
    List<BoldReports.Web.ReportParameter>();
```

```
    if (reportParameters != null)
```

```
{
```

```
        foreach (var rptParameter in reportParameters)
```

```
{
```

```
            modifiedParameters.Add(new BoldReports.Web.ReportParameter()
```

```
{
```

```
                Name = rptParameter.Name,
```

```

Hidden = true
});
}
reportOption.ReportModel.Parameters = modifiedParameters;
}
}
,

```

IReportController

The **IReportController** interface has the declaration of action methods that is defined in Web API Controller to process the RDL, RDLC, SSRS report and handling request from Report Viewer control. The IReportController has the following action methods declaration.

Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

`csharp

```

public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered report viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
    public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)

```

```
{
return ReportHelper.ProcessReport(jsonArray, this, this._cache);
}

// Method will be called to initialize the report information to load the report with ReportHelper for
processing.
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
string basePath = _hostingEnvironment.WebRootPath;
// Here, we have loaded the sample report from application the folder wwwroot. Sample.rdl should be
there in wwwroot application folder.
FileStream inputStream = new FileStream(basePath + @"\Resources\" +
reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
MemoryStream reportStream = new MemoryStream();
inputStream.CopyTo(reportStream);
reportStream.Position = 0;
inputStream.Close();
reportOption.ReportModel.Stream = reportStream;
}

// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}

//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
return ReportHelper.GetResource(resource, this, _cache);
}

[HttpPost]
```

```
public object PostFormReportAction()
{
    return ReportHelper.ProcessReport(null, this, _cache);
}
}
```

Extensions

See Also

- [Custom Data Extension](#)

Custom Data Extension

This section explains the steps required to create and load data extensions in Web Report Viewer ASP.NET core application

See Also

- [Configure Data Extension](#)

Configure a new data source extension in ASP.NET Core Report Viewer

The ASP.NET Core Report Viewer provides SQL, ODBC and OLEDB data sources as built in support data sources. Other data source like Web API, JSON, XML, and OData are provided as extension data sources.

This documentation provides step by step procedure to register and connect with new extension data sources in Report Viewer Application.

Create ASP.NET Core Report Viewer Application

Refer [Getting Started](#) and create a ASP.NET Core Report Viewer Application.

Install data source extension from NuGet

Based on the required data connector install the respective NuGet package to the application. The NuGet packages name for each data connectors are provided in below table,

Data source	Package Name	Assembly Name
-----	-----	-----
Web data sources(WebAPI, JSON, XML, and OData)	BoldReports.Data.WebData	BoldReports.Data.WebData.dll
PostgreSQL data sources	BoldReports.Data.PostgreSQL	BoldReports.Data.PostgreSQL.dll
CSV data sources	BoldReports.Data.Csv	BoldReports.Data.Csv.dll
Excel data sources	BoldReports.Data.Excel	BoldReports.Data.Excel.dll
MySQL data sources	BoldReports.Data.MySQL	BoldReports.Data.MySQL.dll
Oracle data sources	BoldReports.Data.Oracle	BoldReports.Data.Oracle.dll

Configure a new data source extension in ASP.NET Core Report Viewer from NuGet

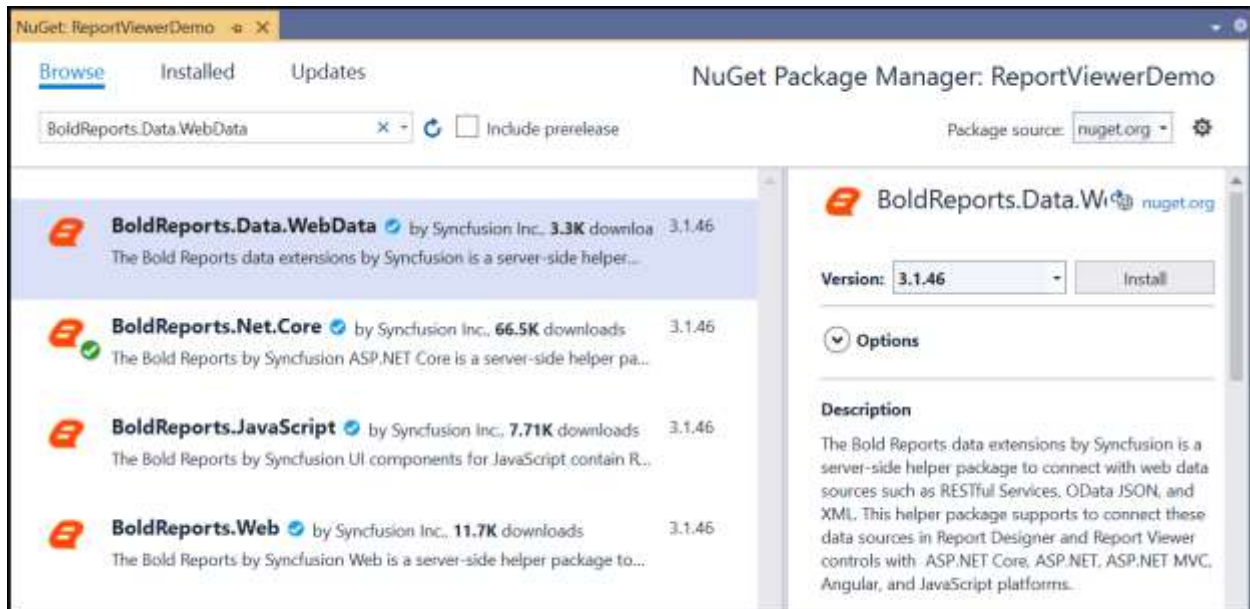
Install data source extension

For example, to register and load web data sources in the application install *BoldReports.Data.WebData* package.

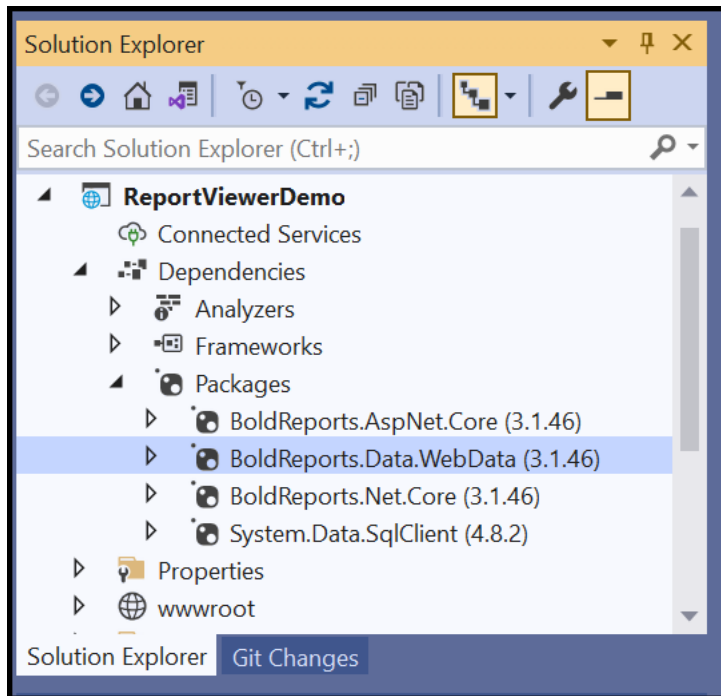
Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring NuGet packages.

Search for **BoldReports.Data.WebData** NuGet package, and install it in your application.



BoldReports.Data.WebData will install into your application. Click OK. Now, the assembly will be added in the respective project references.



Register data source extension

1. Open the code-behind file `Startup.cs` and add the following using statement.

```
`csharp
using BoldReports.Web;
`
```

2. Then add the following code to register extension assembly in `Startup` method.

```
`csharp
public Startup(IConfiguration configuration, IWebHostEnvironment _hostingEnvironment)
{
    //Use the below code to register extensions assembly into report viewer
    ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {
        "BoldReports.Data.WebData" });

    //To register multiple data extensions, provide the assembly name's as list of strings. For example:
    "ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {
        "BoldReports.Data.WebData", "BoldReports.Data.Excel"};

    //Incase the data source extensions fails to register or any error occurs replace the code as below,
    "ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string>
    {System.IO.Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory) +
    "BoldReports.Data.WebData.dll" });"
```



```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Local"
report-path="Product List.rdlc" report-loaded="onReportLoaded">
</bold-report-viewer>
<script type="text/javascript">
function onReportLoaded(args) {
var dataSource = [
{
ProductName: "Baked Chicken and Cheese", OrderId: "323B60", Price: 55, Category: "Non-Veg",
Ingredients: "Grilled chicken, Corn and Olives.", ProductImage: ""
},
{
ProductName: "Chicken Delite", OrderId: "323B61", Price: 100, Category: "Non-Veg", Ingredients:
"Cheese, Chicken chunks, Onions & Pineapple chunks.", ProductImage: ""
},
{
ProductName: "Chicken Tikka", OrderId: "323B62", Price: 64, Category: "Non-Veg", Ingredients: "Onions,
Grilled chicken, Chicken salami & Tomatoes.", ProductImage: ""
}
];
var reportObj = $('#viewer').data("boldReportViewer");
reportObj.model.dataSources = [{
value: ej.DataManager(dataSource).executeLocal(ej.Query()),
name: "list"
}
];
}
</script>
`

```

Report error

When an error occurs in the report processing, it raises the **report-error** event. You can handle the event and show the details in your custom dialog instead of component default error detail interface.

```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Local"
report-path="Product List.rdlc" report-error="onReportError">
</bold-report-viewer>
<script type="text/javascript">

```



```
function onReportError(args) {
  alert(args.errmsg);
  args.cancel = true;
}
</script>
`
```

To suppress the default error dialog, set the cancel argument to true.

Show error

The **show-error** event is invoked whenever users click a report item that contains an error in processing. It provides detailed information about the cause of error. You can hide the default dialog and show your customized dialog.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Local"
report-path="Product List.rdlc" show-error="onShowError">
</bold-report-viewer>
<script type="text/javascript">
function onShowError(args) {
  alert("Error code : " + args.errorCode + "\n" +
"Error Detail : " + args.errorDetail + "\n" +
"Error Message : " + args.errorMessage);
  args.cancel = true;
}
</script>
`
```

Drill through

When a drill through item is selected in a report, it invokes the **drill-through** event. You can change the drill through arguments such as report parameter and data sources. The following sample code can be used to change the drill through report name and set the parameter value before the drill through report is rendered.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="SalesPersonDetails.rdl" drill-through="onDrillThrough">
</bold-report-viewer>
<script type="text/javascript">
function onDrillThrough(args) {
  args.actionInfo.ReportName = "PersonalDetails";
}
```

```
args.actionInfo.Parameters = [{ name: 'EmployeeID', value: ['3'] }];
}
</script>
`
```

Hyperlink

The **hyperlink** event occurs when users click a hyperlink in a report, before the hyperlink is followed. The following sample code redirects to a new custom URL and cancels the component default action.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="Customer Support Analysis (Random data).rdl" hyperlink="onHyperlink">
</bold-report-viewer>
<script type="text/javascript">
function onHyperlink(args) {
args.cancel = true;
//You can modify the URL here
window.open(args.actionInfo.Hyperlink);
}
</script>
`
```

Handle post actions

Report processing actions are sent in an Ajax request to exchange data with the Web API service. You can handle post actions event to customize the Ajax requests.

- **AjaxBeforeLoad**
- **AjaxSuccess**
- **AjaxError**

AjaxBeforeLoad

This event can be triggered before an Ajax request is sent to the Report Viewer Web API service. It allows you to set additional headers and custom data in the Ajax request. The following code sample demonstrates adding custom authorization header and passing default parameter values to service.

Add custom header to Ajax request

Initialize the **ajaxBeforeLoad** event in the script and add the authorization token to the **headers** property.

```
`js
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="sales-order-detail.rdl" ajax-before-load="onAjaxRequest"></bold-report-viewer>
```

```

<script type="text/javascript">
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
}
</script>
`

```

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded from [here](#).

Get the custom header value from the `HttpContext` header collection using the key name specified at client side.

```

`csharp
private Microsoft.Extensions.Primitives.StringValues authenticationHeader;
[HttpPost]
public object PostReportAction([FromBody] Dictionary<string, object> jsonResult)
{
//jsonArray = jsonResult;
if (jsonResult != null)
{
//Get client side custom ajax header and store in local variable
HttpContext.Request.Headers.TryGetValue("Authorization", out authenticationHeader);
//Perform your custom validation here
if (authenticationHeader == "")
{
return new Exception("Authentication failed!!!");
}
else
{
return ReportHelper.ProcessReport(jsonResult, this, _cache);
}
}
return ReportHelper.ProcessReport(jsonResult, this, this._cache);
}
`

```

Perform your own action to validate the header values.

Pass custom data in Ajax request

Use the `data` property to set custom data to the server in the Ajax request. In the following code sample, parameter values are passed to the server side.

```
`js
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="sales-order-detail.rdl" ajax-before-load="onAjaxRequest"></bold-report-viewer>
<script type="text/javascript">
function onAjaxRequest(args) {
//Passing custom data to server
var customerID = "CI0021";
args.data = customerID;
}
</script>
`
```

The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates to change the datasource connection strings based on Customer ID in the `OnInitReportOptions` method.

```
`csharp
string customerID = null;
[HttpPost]
public object PostReportAction([FromBody] Dictionary<string, object> jsonResult)
{
if (jsonResult != null)
{
if (jsonResult.ContainsKey("customData"))
{
//Get client side custom data and store in local variable.
customerID = jsonResult["customData"].ToString();
}
}
return ReportHelper.ProcessReport(jsonResult, this, this._cache);
}
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```

{
if (customerID != null)
{
if(customerID.Contains("CI0021"))
{
//If you are changing the connection string based on customer id then could you please change the
connection string as below.

//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>;"));
}
else if(customerID.Contains("CI0022"))
{
//If you are changing the connection string based on customer id then could you please change the
connection string as below.

//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>;"));
}
}
}
}

```

AjaxSuccess

To perform custom action or show user defined message, use the `ajaxSuccess` event on the successful Ajax request.

```

`js
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="sales-order-detail.rdl" ajax-before-load="onAjaxRequest" ajax-
success="onAjaxSuccess"></bold-report-viewer>
<script type="text/javascript">
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
//Passing custom parameter data to server

```

```

args.data = [{ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'] }];
}
function onAjaxSuccess(args) {
//Perform your custom success message here
alert("Ajax request success!!!");
}
</script>
`

```

AjaxError

The `ajaxError` event is called, if an error occurred with the request, you can display the customized error detail in the event method.

```

`js
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="sales-order-detail.rdl" ajax-before-load="onAjaxRequest" ajax-
error="onAjaxFailure"></bold-report-viewer>

<script type="text/javascript">
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
//Passing custom parameter data to server
args.data = [{ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'] }];
}
function onAjaxFailure(args) {
alert("Status: " + args.status + "\n" +
"Error: " + args.responseText);
}
</script>
`

```

You can never have both an error and a success callback with a request.

Change Report Viewer default WEB API action with custom endpoint

The `actionName` event argument allows to change the methods of `IReportController` to custom WEB API action endpoints.

Change report processing endpoint

Create a new Web Api action method in Report Viewer API controller as in the following code snippet,

```

`csharp
[HttpPost]

```

```
public object PostReportCustomAction([FromBody] Dictionary<string, object> jsonArray)
{
    return ReportHelper.ProcessReport(jsonArray, this, this._cache);
}
`
```

The custom method must have the `Dictionary<TKey, TValue>` argument and add code `ReportHelper.ProcessReport` for processing the report.

Register the event `ajaxBeforeLoad` in your html page and set the newly created name to `actionName` property as in the below code snippet.

```
`js
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="sales-order-detail.rdl" ajax-before-load="onAjaxRequest"></bold-report-viewer>
<script type="text/javascript">
function onAjaxRequest(args) {
    args.actionName = "PostReportCustomAction";
}
</script>
`
```

Change export action endpoint

Create a new Web Api action method in Report Viewer API controller as in the following code snippet,

```
`csharp
[HttpPost]
public object ExportReportCustomAction()
{
    return ReportHelper.ProcessReport(null, this, _cache);
}
`
```

The custom method must have the code `ReportHelper.ProcessReport` for exporting the report.

Register the event `onExportProgressChanged` in your html page and set the newly created name to `actionName` property as in the below code snippet.

```
`js
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="sales-order-detail.rdl" export-progress-changed="onExportProgressChanged"></bold-
report-viewer>
<script type="text/javascript">
```

Error logging in ASP.NET Core Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

```
function onExportProgressChanged(args) {  
if(args.stage === "exportStarted"){  
args.actionName = "ExportReportCustomAction";  
}  
}  
</script>  
`
```

Error logging in ASP.NET Core Report Viewer

If an error occurred in report processing, ASP.NET Core Report Viewer displays short messages about the error in view. You need to configure the **Controller** to save all logs, stack trace, and error information into a physical file location.

This section explains how to log the detailed error information to your ASP.NET Core application.

This section requires a ASP.NET Core Report Viewer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

1. In Solution Explorer, open the Report Viewer Controller file.
2. Inherit the **IReportLogger** interface and implement the interface methods.

```
`csharp  
public class ReportViewerController : Controller, IReportController, IReportLogger  
{  
    public void LogError(string message, Exception exception, MethodBase methodType, ErrorType  
        errorType)  
    {  
        throw new NotImplementedException();  
    }  
    public void LogError(string errorCode, string message, Exception exception, string errorDetail, string  
        methodName, string className)  
    {  
        throw new NotImplementedException();  
    }  
}  
`
```

3. Create a method in **ReportViewerController** to write the error text into application folder.

```
`csharp
```


Error logging in ASP.NET Core Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

```
internal void WriteLogs(string errorMessage)
{
    string filePath = Path.Combine(_hostingEnvironment.WebRootPath, "ErrorDetails.txt");
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
        writer.WriteLine(errorMessage);
    }
}
```

4. Invoke the newly created function in **LogError** as below.

```
`csharp
public void LogError(string message, Exception exception, MethodBase methodType, ErrorType
errorType)
{
    WriteLogs(string.Format("Error Message: {0} \n Stack Trace: {1}", message, exception.StackTrace));
}

public void LogError(string errorCode, string message, Exception exception, string errorDetail, string
methodName, string className)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2} \n Stack Trace:
{3}", className, methodName, errorDetail, exception.StackTrace));
}
`
```

In cases of any issues faced in the report rendering, share the log file to our technical support team to get assistance on that.

5. The final controller given as follows, you can replace it in your application.

```
`csharp
public class ReportViewerController : Controller, IReportController, IReportLogger
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {

```

Error logging in ASP.NET Core Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

```
return ReportHelper.ProcessReport(jsonResult, this);
}

// Get action for getting resources from the report
[System.Web.Http.ActionName("GetResource")]
[AcceptVerbs("GET")]
public object GetResource(string key, string resourcetype, bool isPrint)
{
    return ReportHelper.GetResource(key, resourcetype, isPrint);
}

// Method that will be called when initialize the report options before start processing the report
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    // You can update report options here
}

// Method that will be called when reported is loaded
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    // You can update report options here
}

public void LogError(string message, Exception exception, MethodBase methodType, ErrorType
errorType)
{
    WriteLogs(string.Format("Error Message: {0} \n Stack Trace: {1}", message, exception.StackTrace));
}

public void LogError(string errorCode, string message, Exception exception, string errorDetail, string
methodName, string className)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2} \n Stack Trace:
{3}", className, methodName, errorDetail, exception.StackTrace));
}

internal void WriteLogs(string errorMessage)
{

```

```
// Error details text file path location
string filePath = Path.Combine(_hostingEnvironment.WebRootPath, "ErrorDetails.txt");
using (StreamWriter writer = new StreamWriter(filePath, true))
{
    writer.AutoFlush = true;
    writer.WriteLine(errorMessage);
}
}
}
,
```

Print report

The Report Viewer provides print report option in the toolbar to print a copy of the report. The page setup dialog allows you to set the paper size or other page setup properties. To see print margins, click **Print Layout** on the toolbar.

You can set values in the Page Setup dialog box for current session only. When you close the report and reopen it, it will have the default values again. The default values of the Page Setup dialog is based on the report properties set in the design view.

View report in print mode

Print margins are displayed in the print layout only. To view report in print mode by default, set the `print-mode` property to true.

```
`js
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-
url="/api/ReportViewer" print-mode="true" processing-mode="Remote"></bold-report-viewer>
,
```

By default, the Report Viewer renders report in normal layout in which the print margins are not displayed.

Print in new page

To open the print in a new tab of the current browser, set the `printOption` property to `NewTab`. By default, it shows the print dialog in the same page.

```
`js
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-
url="/api/ReportViewer" print-mode="true" print-option="NewTab" processing-
mode="Remote"></bold-report-viewer>
,
```

The pop-up blocker should be enabled for the page to open the print view in new tab.

Set page orientation and paper size

You can specify the print page paper size and orientation at client-side to change the page setup properties by setting the `page-settings` property.

The following code example illustrates how to set page orientation and paper size in the Report Viewer for client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
page-settings="ViewBag.pageSettings">
</bold-report-viewer>
`
```

The following code example illustrates how to set page orientation and paper size in the Report Viewer for server side.

```
`csharp
public ActionResult Index()
{
    ViewBag.pageSettings = new BoldReports.Models.ReportViewer.PageSettings();
    ViewBag.pageSettings.Orientation = BoldReports.ReportViewerEnums.Orientation.Landscape;
    ViewBag.pageSettings.PaperSize = BoldReports.ReportViewerEnums.PaperSize.Letter;
    return View();
}
`
```

Set report margin

To set margin values to the report page setup, use the `Margins` property and specify the value to top, right, bottom, and left.

The following code example illustrates how to set report margin in the Report Viewer for client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
page-settings="ViewBag.pageSettings">
</bold-report-viewer>
`
```

The following code example illustrates how to set report margin in the Report Viewer for server side.

```
`csharp
public ActionResult Index()
{
    ViewBag.pageSettings = new BoldReports.Models.ReportViewer.PageSettings();

```

```

ViewBag.pageSettings.Margins = new BoldReports.Models.ReportViewer.Margins();
ViewBag.pageSettings.Margins.Top = 0.5;
ViewBag.pageSettings.Margins.Left = 0.5;
ViewBag.pageSettings.Margins.Bottom = 0.5;
ViewBag.pageSettings.Margins.Right = 0.5;
return View();
}
`

```

The values set in the margin property is considered as inches input.

Set page height and width

To set height and width values to the report page setup, use the **Height** and **Width** properties.

The following code example illustrates how to set page height and width in the Report Viewer for client side.

```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
page-settings="ViewBag.pageSettings">
</bold-report-viewer>
`

```

The following code example illustrates how to set page height and width in Report Viewer for server side.

```

`csharp
public ActionResult Index()
{
    ViewBag.pageSettings = new BoldReports.Models.ReportViewer.PageSettings();
    ViewBag.pageSettings.Height = 11.69;
    ViewBag.pageSettings.Width = 8.27;
    return View();
}
`

```

The values set in the height and width properties are considered as inches input.

Print report with images

When the report has more images, the browser will send the report stream to the print dialog before the images are completely loaded. To load the print report stream with complete images, you should set the **EmbedImageData** property to true in **OnInitReportOptions** as shown in the following code.

```

`csharp

```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.EmbedImageData = true;
}
`js
```

Replace the following code sample in the client-side HTML file.

```
`js
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="Product Details.rdl" page-settings="ViewBag.pageSettings">
</bold-report-viewer>
`js
```

In this tutorial, the `Product Details.rdl` report is used, and it can be downloaded from [here](#).

External styles in report printing

While printing report, the external styles are used in the application overrides printable page style and prints output with incorrect alignments. To avoid the external script overriding, set the `isStyleLoad` property to false, which will print the page using only the Report Viewer styles.

```
`js
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="Product Details.rdl" report-print="onReportPrint">
</bold-report-viewer>
<script type="text/javascript">
function onReportPrint(args) {
    args.isStyleLoad = false;
}
</script>
`js
```

Show print progress

Report Viewer provides events to show the progress information when the printing takes long time to complete.

To show print progress, follow these steps:

1. Set the `print-progress-changed` in Report Viewer initialization.
2. Implement the function and add code samples to show a custom message based on the print progress status as shown in the following code snippet.

```
`js
```

```

<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
report-path="Product Details.rdl" print-progress-changed="onPrintProgressChanged">
</bold-report-viewer>
<script type="text/javascript">
function onPrintProgressChanged(args) {
if (args.stage == "beginPrint") {
$('#viewer').ejWaitingPopup({ showOnInit: true, cssClass: "customStyle", text: "Preparing print data..
Please wait..." });
}
if (args.stage == "printStarted") {
var popupObj = $('#viewer').data('ejWaitingPopup');
popupObj.hide();
}
else if (args.stage == "preparation") {
console.log(args.stage);
if (args.preparationStage == "dataPreparation") {
console.log(args.preparationStage);
console.log(args.totalPages);
console.log(args.currentPage);
if (args.totalPages > 1 && args.currentPage > 1) {
var progressPercentage = Math.floor((args.currentPage / args.totalPages) * 100);
if (progressPercentage > 0) {
var popupObj = $('#viewer').data('ejWaitingPopup');
popupObj.setModel({ text: "Preparing print data.." + progressPercentage + " % completed.. Please
wait..." });
}
}
}
}
args.handled = true;
}
</script>

```

Remove empty spaces in printing

The extra blank page is created when the body of your report is too wide for your page. To make the report appear on a single page, all the content within the report body must fit on the physical page, and the body width should be as the following formula.

Body Width <= Page Width - (Left Margin + Right Margin)

For more details about removing the empty pages in the report while designing, refer to the knowledge base article of [report page sizing](#).

Export report

The Report Viewer provides events and properties to control and customize the report exporting functionality.

Export event handling

You can show the progress information, when the exporting process takes long time to complete using the `export-progress-changed` event.

1. Set the `export-progress-changed` event in Report Viewer initialization.
2. Implement the function and replace the following code samples to show a custom message based on the progress stage.

The following code example demonstrates how to export event handling in the Report Viewer at client side.

```
`html
```

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
export-progress-changed="onExportProgressChanged">
```

```
</bold-report-viewer>
```

```
<script type="text/javascript">
```

```
function onExportProgressChanged(args) {
```

```
if (args.stage === "beginExport") {
```

```
console.log(args.stage);
```

```
args.format =
```

```
$('#viewer').ejWaitingPopup({ showOnInit: true, cssClass: "customStyle", text: "Preparing exporting
document.. Please wait..." });
```

```
}
```

```
else if (args.stage === "exportStarted") {
```

```
console.log(args.stage);
```

```
var popupObj1 = $('#viewer').data('ejWaitingPopup');
```

```
popupObj1.hide();
```

```
}
```



```

else if (args.stage === "preparation") {
  console.log(args.stage);
  console.log(args.format);
  console.log(args.preparationStage);
  if (args.format === "PDF" && args.preparationStage === "documentPreparation") {
    console.log(args.totalPages);
    console.log(args.currentPage);
    if (args.totalPages > 1 && args.currentPage > 1) {
      var progressPercentage = Math.floor((args.currentPage / args.totalPages) * 100);
      if (progressPercentage > 0) {
        var popupObj2 = $('#viewer').data('ejWaitingPopup');
        popupObj2.setModel({ text: "Preparing exporting document.." + progressPercentage + " % completed.. Please wait..." });
      }
    }
  }
  args.handled = true;
}
</script>
`

```

Change Excel and Word export format

Allows you change the default file format to any other file format using the `ExcelFormat` and `WordFormat` properties.

The following code example demonstrates how to change Excel and Word export format in the Report Viewer at client side.

```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
export-settings="ViewBag.exportSettings">
</bold-report-viewer>
`

```

The following code example demonstrates how to change Excel and Word export format in the Report Viewer at server side.

```

`csharp
public ActionResult Index()

```

```
{
    ViewBag.exportSettings = new BoldReports.Models.ReportViewer.ExportSettings();
    ViewBag.exportSettings.ExcelFormat = BoldReports.ReportViewerEnums.ExcelFormats.Excel2013;
    ViewBag.exportSettings.WordFormat = BoldReports.ReportViewerEnums.WordFormats.Word2013;
    return View();
}
```

Hide specific export type for report

Show or hide the default export types available in the component using the **ExportOptions** property.

The following code example demonstrates how to hide specific export type to the report in the Report Viewer at client side.

```
`html

<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
export-settings="ViewBag.exportSettings">

</bold-report-viewer>
```

The following code example demonstrates how to hide specific export type to the report in the Report Viewer at server side.

```
`csharp

public ActionResult Index()
{
    ViewBag.exportSettings = new BoldReports.Models.ReportViewer.ExportSettings();
    ViewBag.exportSettings.ExportOptions = BoldReports.ReportViewerEnums.ExportOptions.All
    & ~BoldReports.ReportViewerEnums.ExportOptions.Pdf;
    return View();
}
```

PDF export options

The **PDFOptions** provides properties to manage PDF export behaviors. You should set the properties in the **OnInitReportOptions** method of the Web API service.

Export with complex scripts

To export reports with the complex scripts, set the **ComplexScript** property of **PDFOptions** instance to true.

```
`csharp

public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
{
EnableComplexScript = true
};
}
```

PDF conformance

You can export the report as a PDF/A-1b document by specifying the PdfConformanceLevel.Pdf_A1B conformance level in the PdfConformanceLevel property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
{
PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B
};
}
```

Add custom fonts to PDF document

This section explains the steps to add custom fonts to the PDF exported document. The Report Viewer provides Fonts property in PDFOptions to add the new fonts.

Any fonts used in the report that is not installed or not available in the local system, then you must add the font stream to Fonts property.

The following points should be followed while adding new fonts:

1. Key should be same as in FontFamily element of the report item.
2. Key is case sensitive, so casing is must match with FontFamily.
3. To add streams for different font style, weight, style consider the below,
 - o Key should be in the format of {fontname} {weight} {style}. For example Roboto Light Italic.
 - o Use only single white space between font name, weight, and style.
 - o When FontStyle is set to Normal then use style as Regular. For example Roboto Light Regular.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```

{
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
{
//Load missing font stream
Fonts = new Dictionary<string, System.IO.Stream>
{
{ "Roboto", new FileStream(basePath + @"\fonts\roboto\Roboto.ttf", FileMode.Open, FileAccess.Read)
},
{ "Roboto Bold", new FileStream(basePath + @"\fonts\roboto\Roboto-Bold.ttf", FileMode.Open,
FileAccess.Read) },
{ "Roboto Bold Regular", new FileStream(basePath + @"\fonts\roboto\Roboto-Bold.ttf",
FileMode.Open, FileAccess.Read) },
{ "Roboto Light Italic", new FileStream(basePath + @"\fonts\roboto\Roboto-Light-Italic.ttf",
FileMode.Open, FileAccess.Read) },
{ "Roboto Thin", new FileStream(basePath + @"\fonts\roboto\Roboto-Thin.ttf", FileMode.Open,
FileAccess.Read) }
}
};
}
,

```

In the above code, loaded the `ttf` streams for `Roboto` font with different style and weight combinations.

Word export options

The `WordOptions` provides properties to manage Word document export behaviors.

Word document type

You can save the report to the required document version by setting the `FormatType` property.

```

`csharp
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
FormatType = BoldReports.Writer.WordFormatType.Docx
};
,

```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```

`csharp

```

```
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
    LayoutOption = BoldReports.Writer.WordLayoutOptions.TopLevel,
    ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
    {
        Bottom = 0.5f,
        Top = 0.5f
    }
};
`
```

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior.

The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, add an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without password.

The following are the types of protection:

- **AllowOnlyComments**: Adds or modifies only the comments in the Word document.
- **AllowOnlyFormFields**: Modifies the form field values in the Word document.
- **AllowOnlyRevisions**: Accepts or rejects the revisions in the Word document.
- **AllowOnlyReading**: Views the content only in the Word document.
- **NoProtection**: Accesses or edits the Word document contents as normally.

```
`csharp
```

```
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
    ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading
};
`
```

Excel export options

The **ExcelOptions** provides properties to manage Excel document export behaviors.

Excel document type

You can save the report to the required excel version by setting the **ExcelSaveType** property.

```
`csharp
```

```
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
```

```
{
ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013
};
`
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` to `IgnoreCellMerge`.

```
`csharp
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
};
`
```

Protecting Excel document from editing

You can restrict the Excel document from editing either by providing the `ExcelSheetProtection` or enabling the `ReadOnlyRecommended` properties.

```
`csharp
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
ReadOnlyRecommended = true,
ExcelSheetProtection = Syncfusion.XlsIO.ExcelSheetProtection.DeletingColumns
};
`
```

CSV export options

The `CsvOptions` allows you to change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document.

```
`csharp
reportOption.ReportModel.CsvOptions = new BoldReports.Writer.CsvOptions()
{
Encoding = System.Text.Encoding.Default,
FieldDelimiter = ",",
UseFormattedValues = false,
Qualifier = "#",
RecordDelimiter = "@",
SuppressLineBreaks = true,
}
```

```
FileExtension = ".txt"
```

```
};
```

```
,
```

Password protect exported document

Allows you protect the exported document such as PDF, Word, Excel, and PowerPoint from unauthorized users by encrypting the document using encryption password. The following code snippet illustrates how to encrypt the exported document with user-defined password.

```
`csharp
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
//PDF encryption
```

```
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions();
```

```
reportOption.ReportModel.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity()
```

```
{
```

```
UserPassword = "password"
```

```
};
```

```
//Word encryption
```

```
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
```

```
{
```

```
EncryptionPassword = "password"
```

```
};
```

```
//Excel encryption
```

```
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
```

```
{
```

```
PasswordToModify = "password",
```

```
PasswordToOpen = "password"
```

```
};
```

```
}
```

```
,
```

Password protection is not supported for HTML export format.

Change file name in export

You can change the file name of report in export using the `FileName` property.

```
`csharp
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
reportOption.ReportModel.ExportSettings = new BoldReports.Writer.ExportSettings()
{
FileName = "Invoice"
};
}
```

Change image quality in export

You can change image quality of data visualization items in report export using the `ImageQuality` property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.ExportSettings = new BoldReports.Writer.ExportSettings()
{
ImageQuality = 4
};
}
```

Custom Actions

Add user defined buttons to the toolbar and invoke custom actions using the `Report Viewer` property. You can create a custom email button with the rendered report to users.

Add email button

1. Create an email button in the toolbar using the `CustomItems` property with the values such as `GroupIndex`, `Index`, `Type`, `CssClass`, and `Tooltip`. The `tool-bar-item-click` event triggers when you click the email button.
2. Access the Report Viewer model and create a JSON array for sending requests to the Web API Server.
3. You can use the following codes to add email button from controller and passing the data to view using `ViewBag`.

```
`csharp
public ActionResult Index()
{
ToolbarSettings toolbarSettings = new ToolbarSettings();
```



```

toolbarSettings.CustomItems = new List<CustomItem>();
var customItem = new CustomItem()
{
    GroupIndex = 1,
    Index = 1,
    CssClass = "e-icon e-mail e-reportviewer-icon",
    Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
    Id = "E-Mail",
    Tooltip = new ToolTip() { Header = "E-Mail", Content = "Send rendered report as mail attachment" }
};
toolbarSettings.CustomItems.Add(customItem);
ViewBag.toolbarSettings = toolbarSettings;
return View();
}

```

4. You can use the following codes to set an **toolbar-settings** property at client side.

```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings">
</bold-report-viewer>

```

5. You can use the following codes to create an **tool-bar-item-click** event at client side.

```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
tool-bar-item-click="onToolBarItemClick">
</bold-report-viewer>
<script type="text/javascript">
function onToolBarItemClick(args) {
    alert('Action Triggered');
}
</script>

```

6. You can use the following codes to get `toolbar-settings` properties on a dynamic object using `ViewBag.toolbarSettings` and invoke custom actions.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings" tool-bar-item-click="onToolBarItemClick">
</bold-report-viewer>
<script type="text/javascript">
function onToolBarItemClick(args) {
if (args.value == "E-Mail") {
alert('Action Triggered');
}
}
</script>
`
```

Create custom email action

1. Create a new action method `SendEmail` in the Web API service.
2. Export the report to the required type using the `ReportHelper.GetReport` method to send a report stream as an attachment.
3. The following code sample exports the report to stream and send it as an attachment to a specified mail address. In the code, the `SmtpClient` method is used to send the report as an email attachment.

```
`csharp
public object SendEmail([FromBody] Dictionary<string, object> jsonResult)
{
string _token = jsonResult["reportViewerToken"].ToString();
var stream = ReportHelper.GetReport(token, jsonResult["exportType"].ToString(), this, cache);
stream.Position = 0;
if (!ComposeEmail(stream, jsonResult["reportName"].ToString()))
{
return "Mail not sent !!!";
}
return "Mail Sent !!!";
}

public bool ComposeEmail(Stream stream, string reportName)
```

```
{
try
{
    MailMessage mail = new MailMessage();
    SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
    mail.IsBodyHtml = true;
    mail.From = new MailAddress("xx@gmail.com");
    mail.To.Add("xx@gmail.com");
    mail.Subject = "Report Name : " + reportName;
    stream.Position = 0;
    if (stream != null)
    {
        ContentType ct = new ContentType();
        ct.Name = reportName + DateTime.Now.ToString() + ".pdf";
        System.Net.Mail.Attachment attachment = new System.Net.Mail.Attachment(stream, ct);
        mail.Attachments.Add(attachment);
    }
    SmtpServer.Port = 587;
    SmtpServer.Credentials = new System.Net.NetworkCredential("xx@gmail.com", "xx");
    SmtpServer.EnableSsl = true;
    SmtpServer.Send(mail);
    return true;
}
catch (Exception ex)
{
    throw ex;
}
return false;
}
```

In the above code sample, the report is exported to PDF format and send to users using the `SmtpClient` method.

Custom properties

The custom properties helps you to include additional features that are not natively supported in the RDL reporting. This topic explains the list of custom properties supported in ASP. NET Core Report Viewer.

See Also

- [Textbox Custom Properties](#)
- [Table Custom Properties](#)
- [Image Custom Properties](#)
- [Chart Custom Properties](#)
- [Report Custom Properties](#)
- [Parameter Custom Properties](#)
- [Export Custom Properties](#)

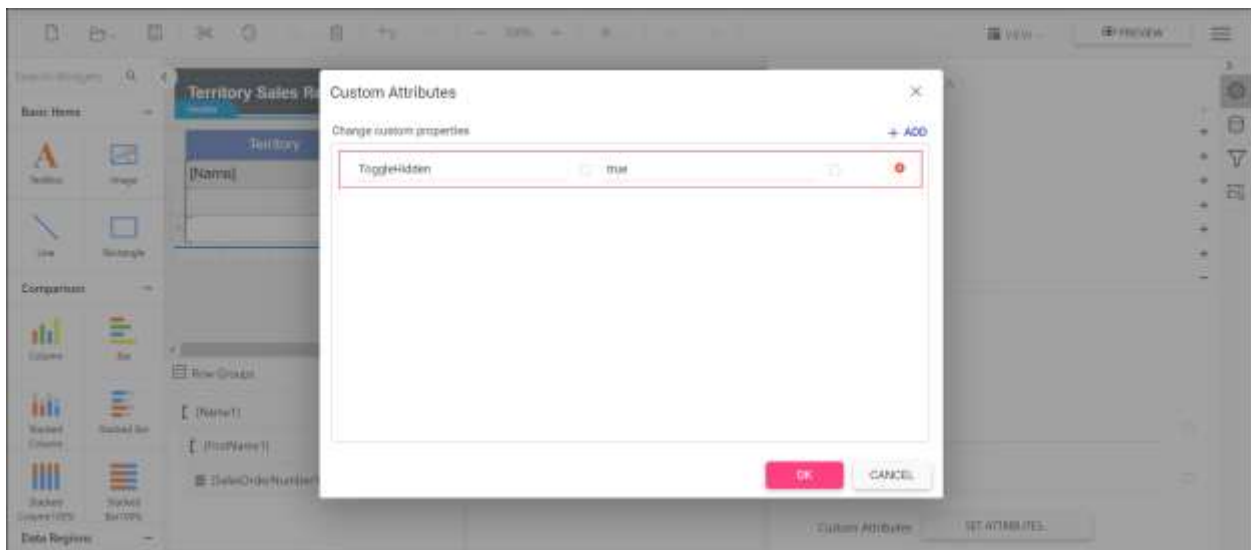
Textbox custom properties

This topic explains about the list of textbox report item custom properties that are supported to render in ASP.NET Core Report Viewer.

Show or hide toggle icon in text box report item

The `ToggleHidden` custom property is used to show or hide the toggle icon in the textbox.

You can set the `ToggleHidden` property value, as shown in the below.



Before setting the toggle hidden property, the default value will be displayed as below.



Territory	Sales Person	Order Number	Total Sales
Australia			\$1,943,016
	Lynn Tsotlias		\$1,943,016
Canada			\$12,868,458
	Garrett Vargas		\$4,840,689
	José Saraiva		\$7,967,769
Central			\$13,434,510
	Jillian Carson		\$13,434,510
France			\$6,083,691
	Ranjit Varkey Chudukatil		\$6,083,691
Germany			\$2,476,530
	Rachel Valdez		\$2,476,530
Northeast			\$12,433,503
	Michael Blythe		\$12,433,503
Northwest			\$6,305,407

Preview the report and the see the toggle icon is hidden in the report.



Territory	Sales Person	Order Number	Total Sales
Australia			\$1,943,016
	Lynn Tsotlias		\$1,943,016
Canada			\$12,868,458
	Garrett Vargas		\$4,840,689
	José Saraiva		\$7,967,769
Central			\$13,434,510
	Jillian Carson		\$13,434,510
France			\$6,083,691
	Ranjit Varkey Chudukatil		\$6,083,691
Germany			\$2,476,530
	Rachel Valdez		\$2,476,530
Northeast			\$12,433,503
	Michael Blythe		\$12,433,503
Northwest			\$6,305,407

Table custom properties

This topic explains about the list of table report item custom properties that are supported to render in ASP.NET Core Report Viewer.

Limit number of table records on each page

The **RowsPerPage** custom property is used to specify the number of table records to display on each page. It supports integer data value greater than zero.

This property is ignored when table rows heights higher than current page size. Increase the report page height or reduce **RowsPerPage** count that fits within the page.

You can set the **RowsPerPage** property value, as shown in the below.

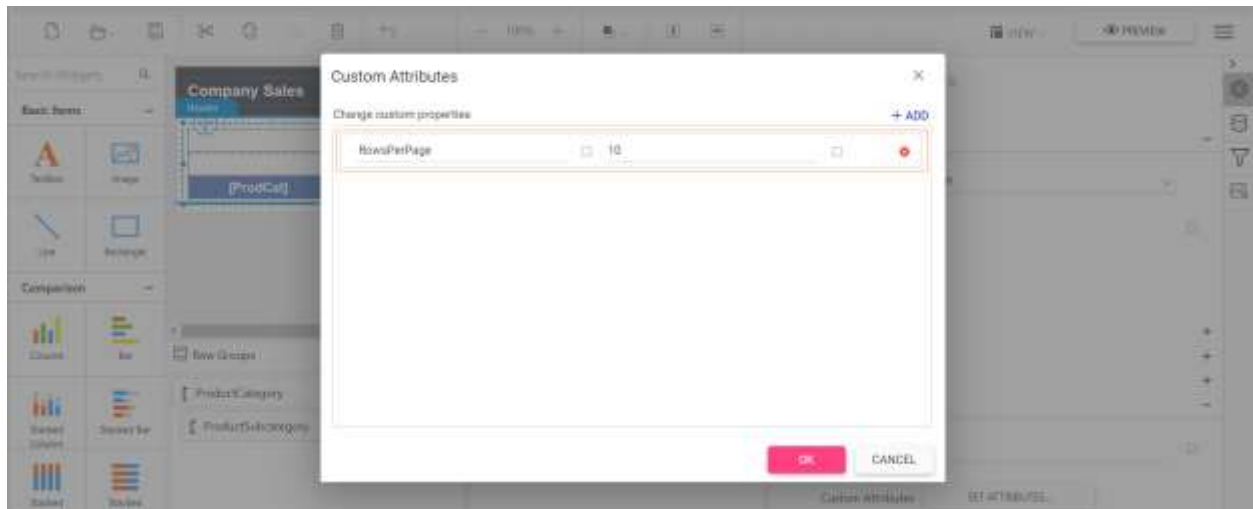


Image custom properties

This topic explains about the list of image custom properties that are supported to render in the ASP.NET Core Report Viewer.

Angle

Set **Angle** custom property to image report item to rotate the image on a specified angle. It supports the angle values 0, 90, 180, and 270. You can set the property value, as shown in the below.

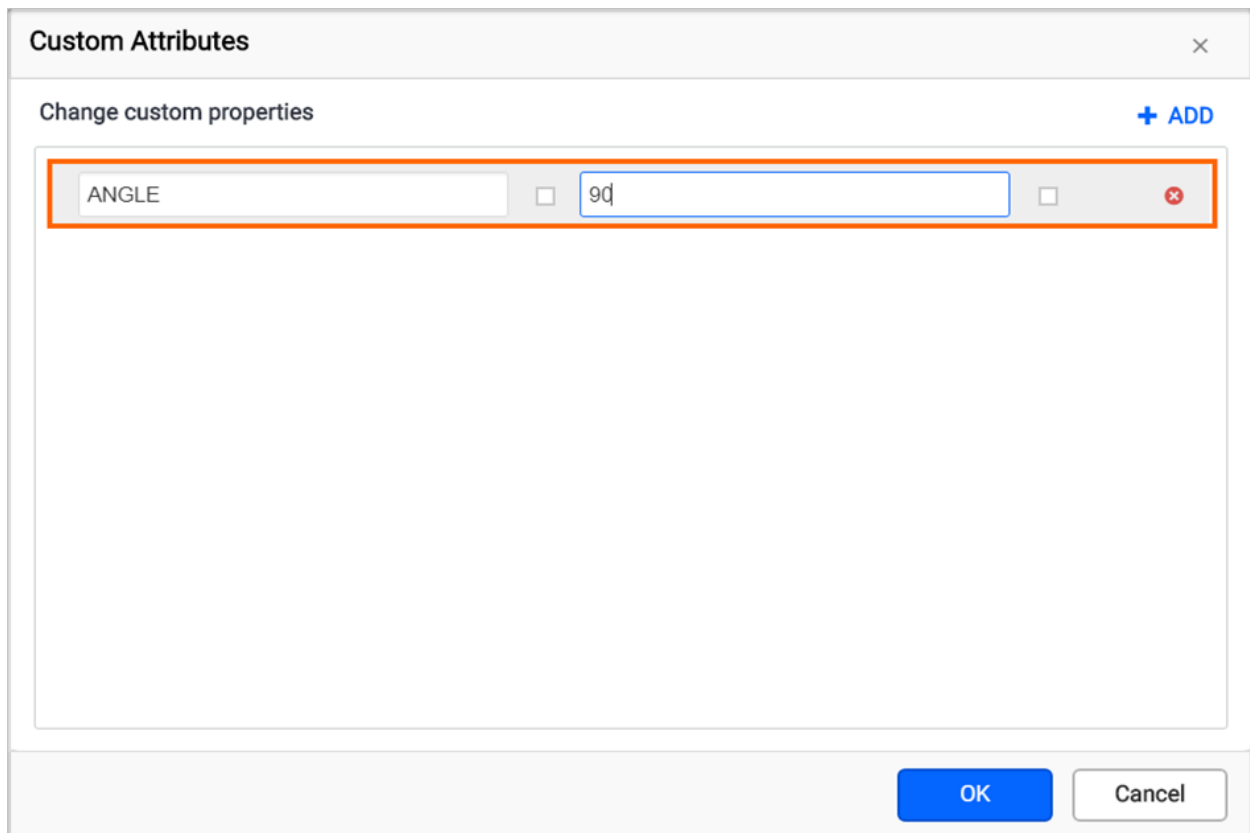
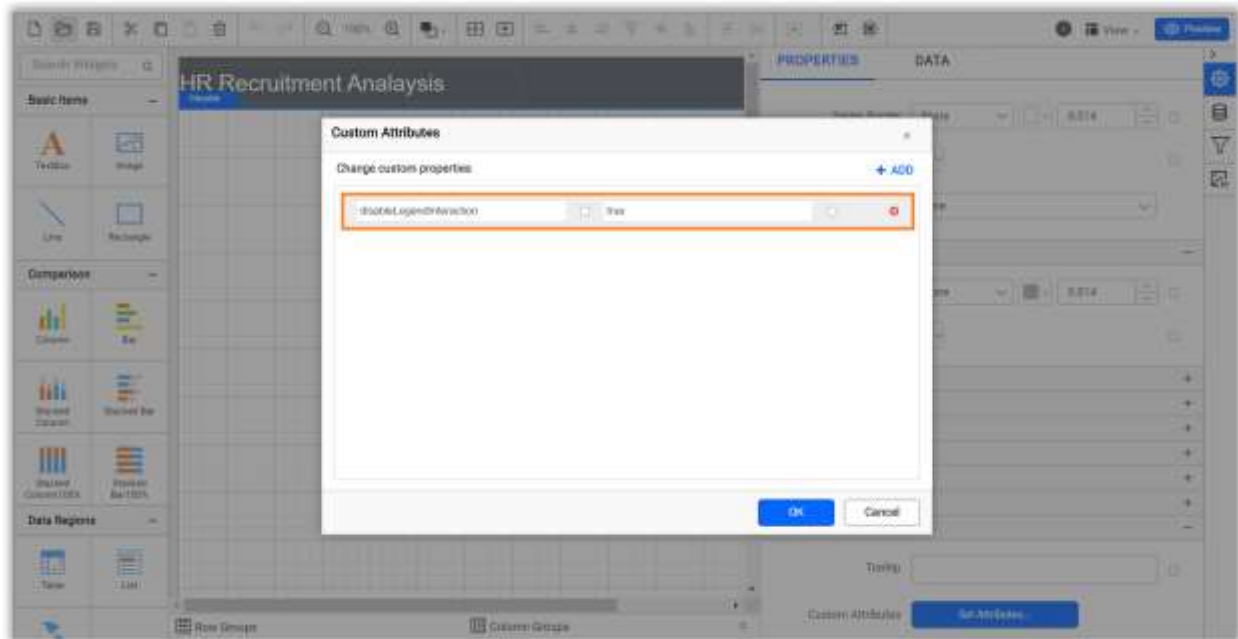


Chart custom properties

This topic explains about the list of chart custom properties that are supported to render in the ASP.NET Core Report Viewer.

Disable legend item interaction

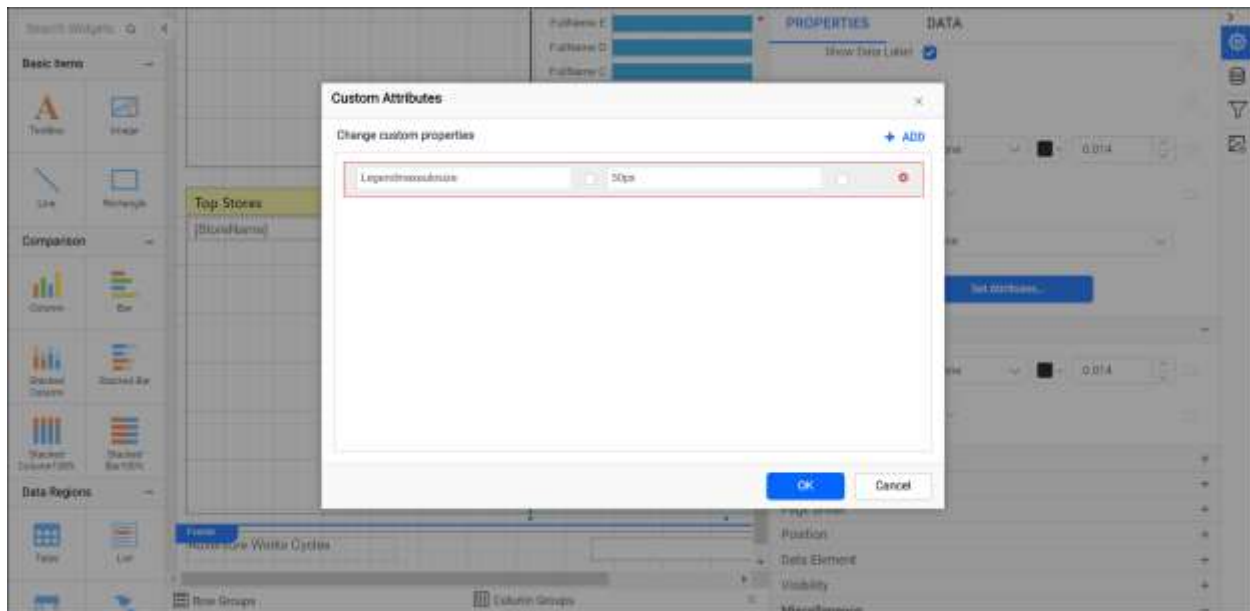
Set `DisableLegendInteraction` custom property value as `true` to stop the legend item interaction. The property value should be boolean. You can set the property value, as shown in the below.



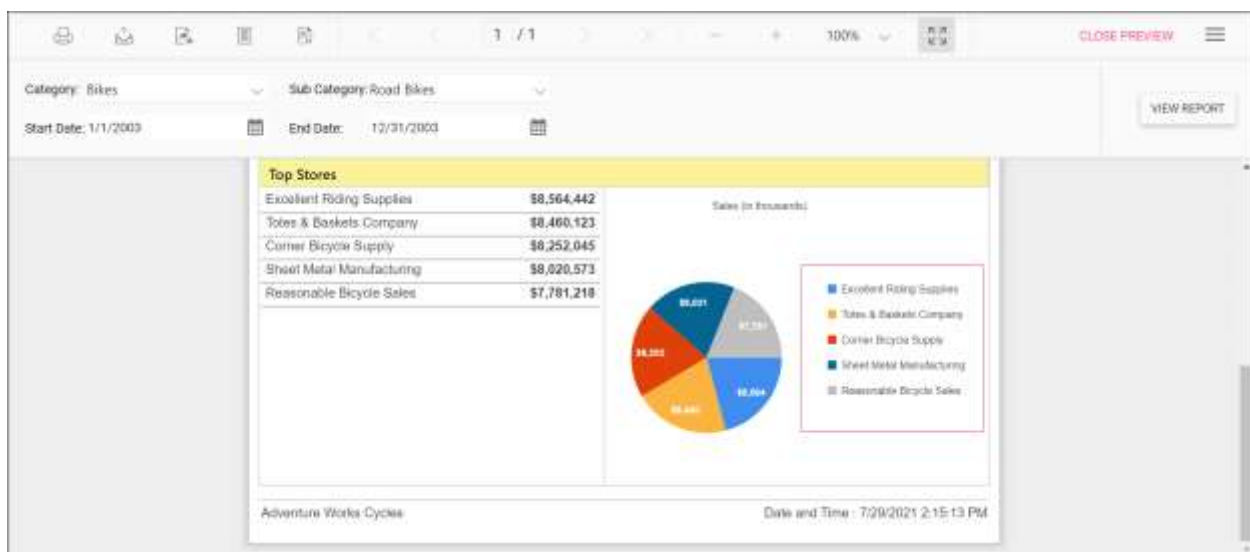
Set maximum size for chart legend

The `LegendMaxAutoSize` custom property specifies the maximum size of the legend container in the report.

You can set the `LegendMaxAutoSize` property value, as shown in the below.



Preview the report and see legend container size in chart report.

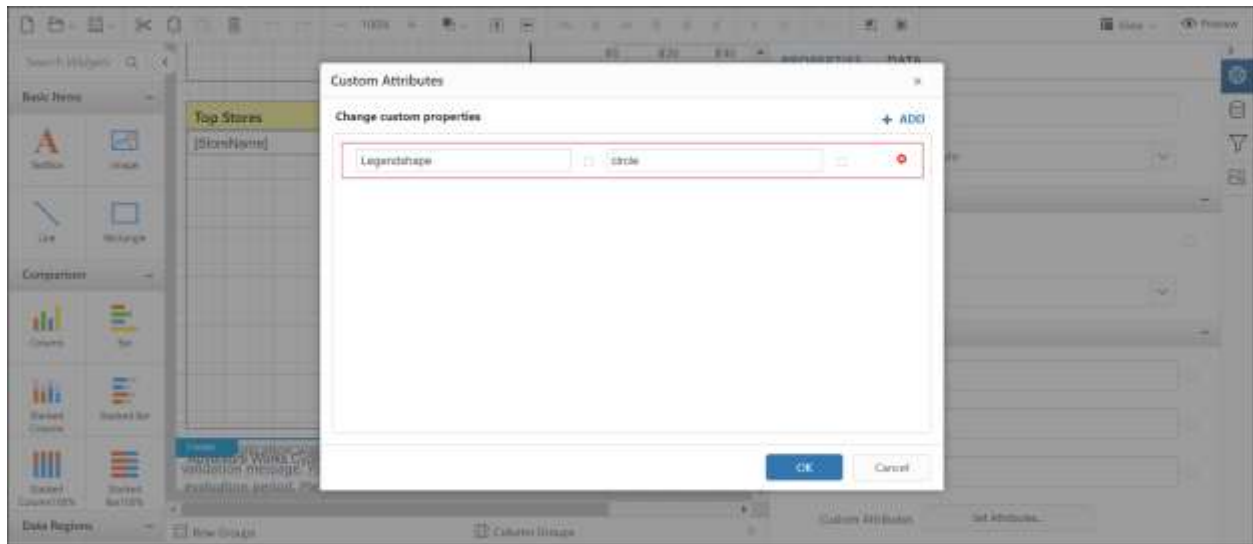


Change chart legend shape

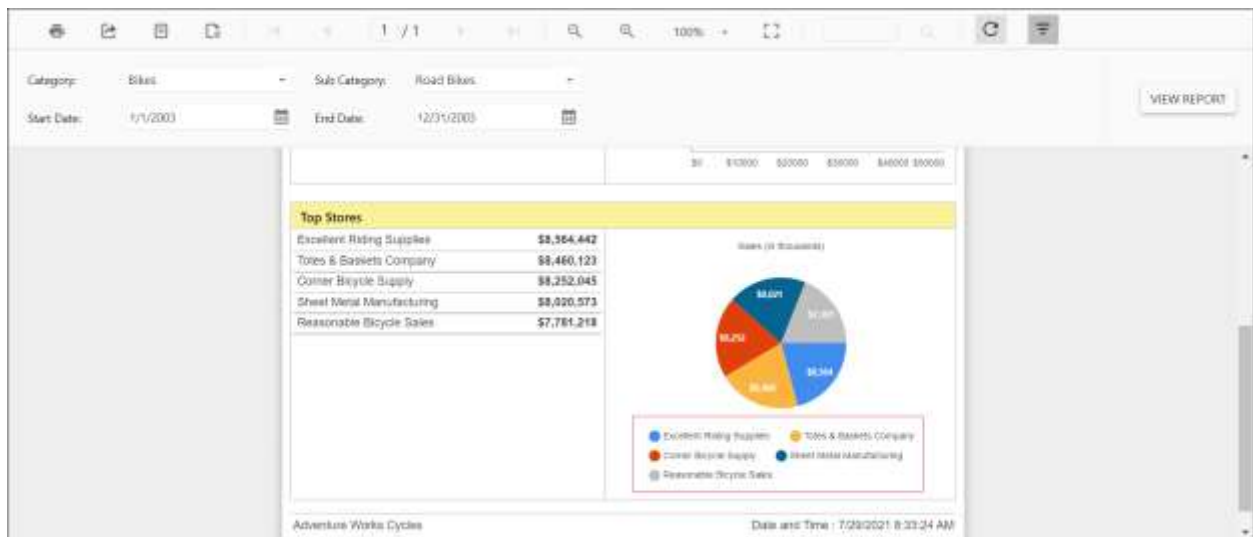
The `LegendShape` custom property allows changing the shape of the legend in the chart report item. By default, the `LegendShape` value is `rectangle`.

- Rectangle - legend shapes displayed in `rectangle`.
- Circle - legends are displayed in `circle`.
- Thumbnail - legends are displayed in `seriestype`.

You can set the `LegendShape` property value, as shown in the below.



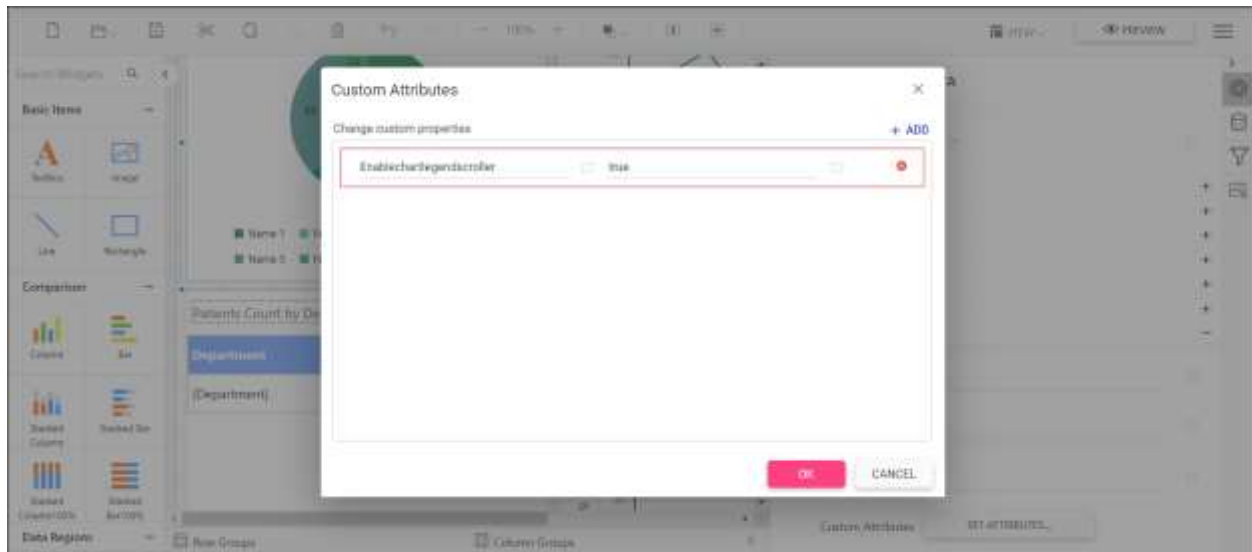
Preview the report and the see legend shape in chart report.



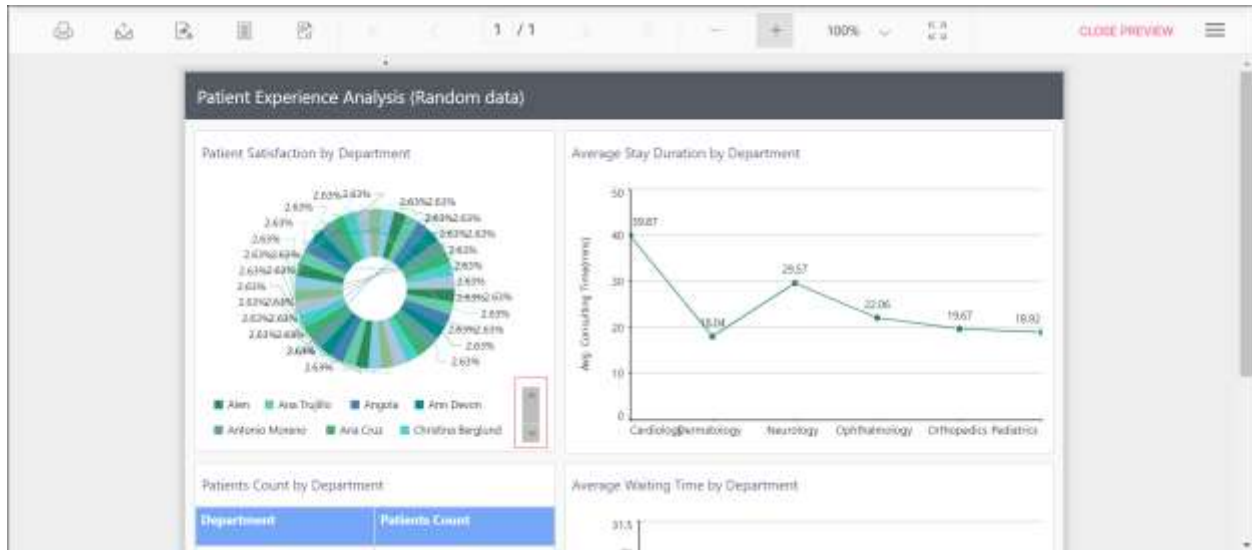
Show or hide chart legend scroller

The `EnableChartLegendScroller` custom property controls whether legend has to use scrollbar or not. The scrollbar appears depending upon size and position properties of legend. By default, the `EnableChartLegendScroller` value is `false`.

You can set the `EnableChartLegendScroller` property value, as shown in the below.



Preview the report and the see the legend scrollbar in chart report.



Set range padding for X-axis and Y-axis

Padding can be applied to the minimum and maximum extremes of the axis range by using the `XAxisRangePadding` and `YAxisRangePadding` property. The default value is `none`.

Numeric axis supports the following types of padding.

Name | Description

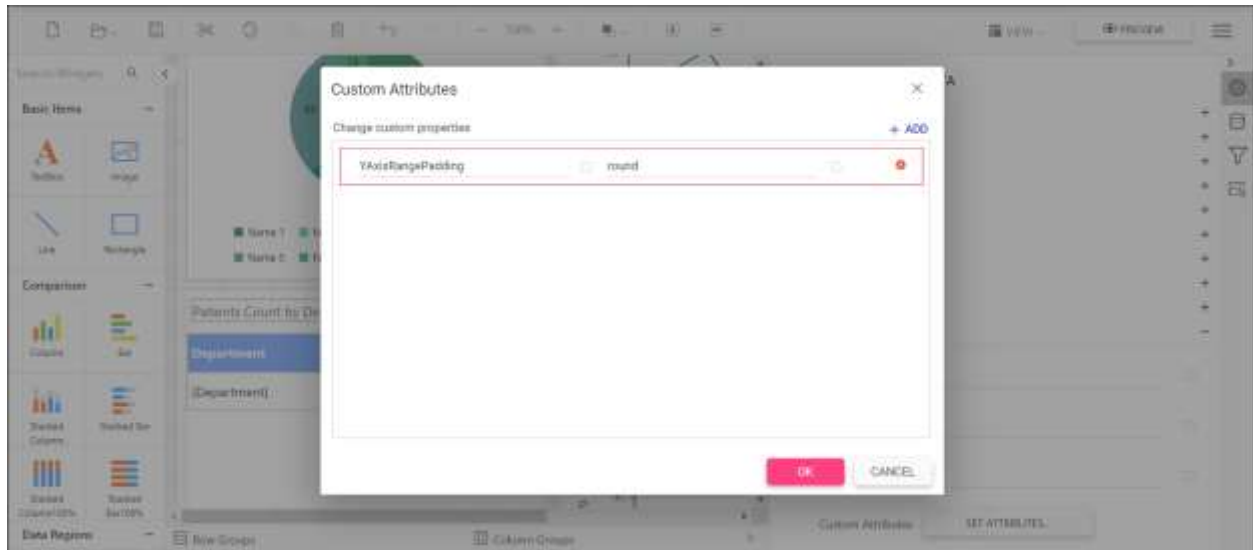
Additional | Interval of the axis is added as padding to the minimum and maximum values of the range

Normal | Padding is applied to the axis based on the range calculation

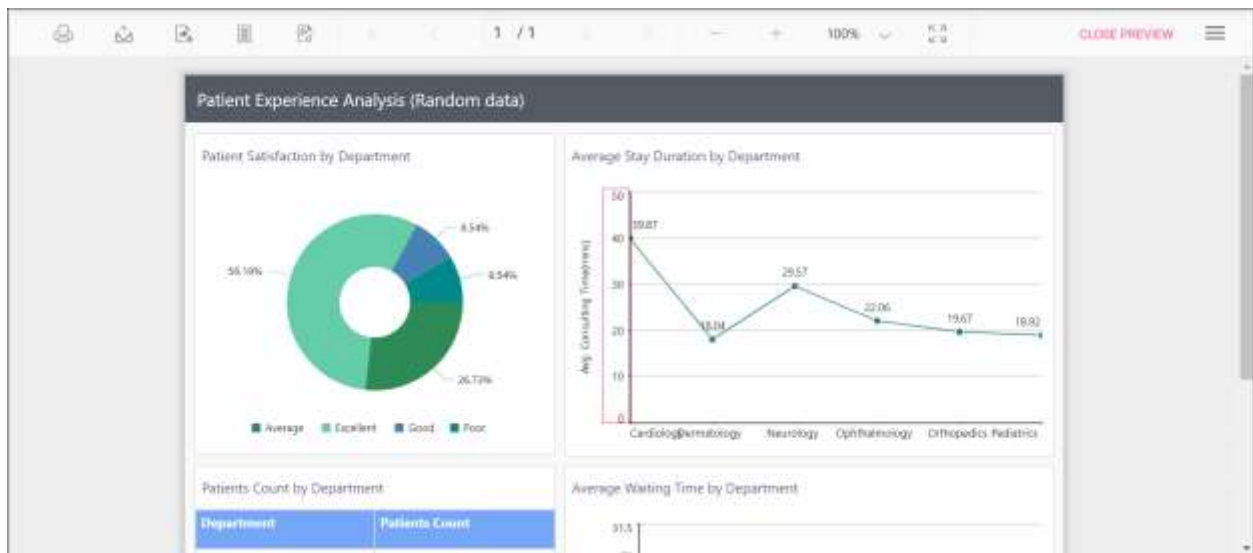
None | Padding cannot be applied to the axis

Round | Axis range is rounded to the nearest possible value divided by the interval

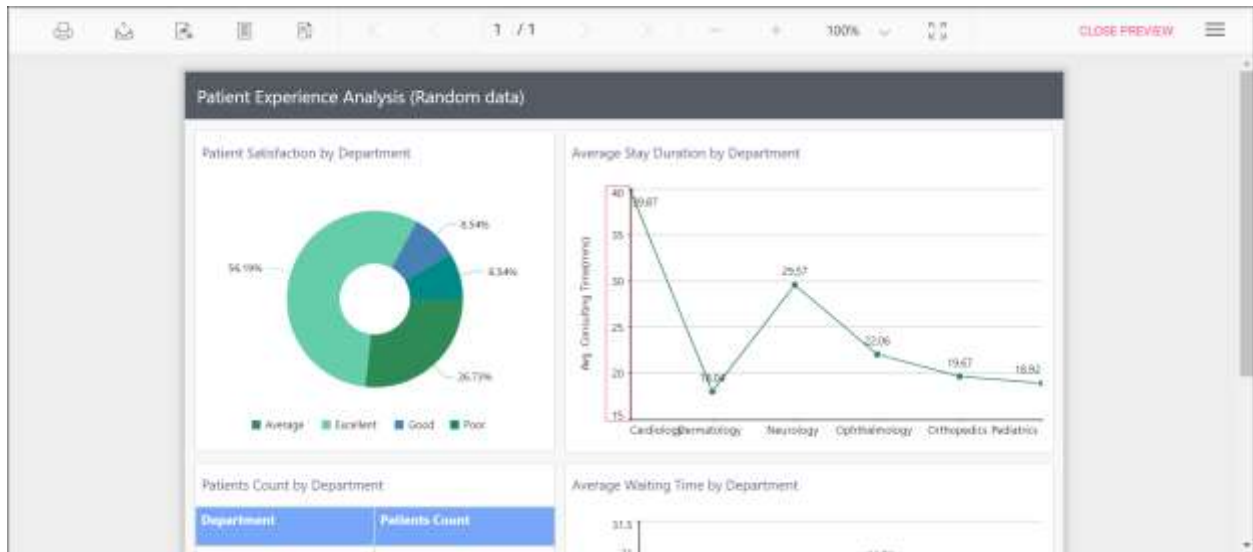
You can set the `XAxisRangePadding` and `YAxisRangePadding` property value, as shown in the below.



Before setting the range padding, the default value will be displayed as below.



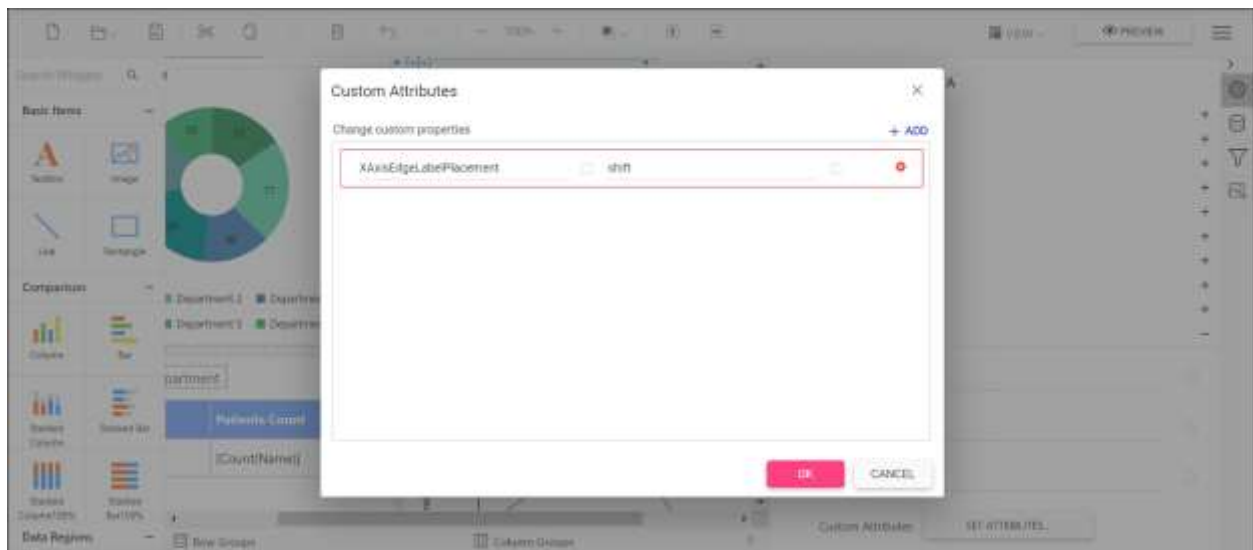
Set the padding range and see the changes in chart report as below.



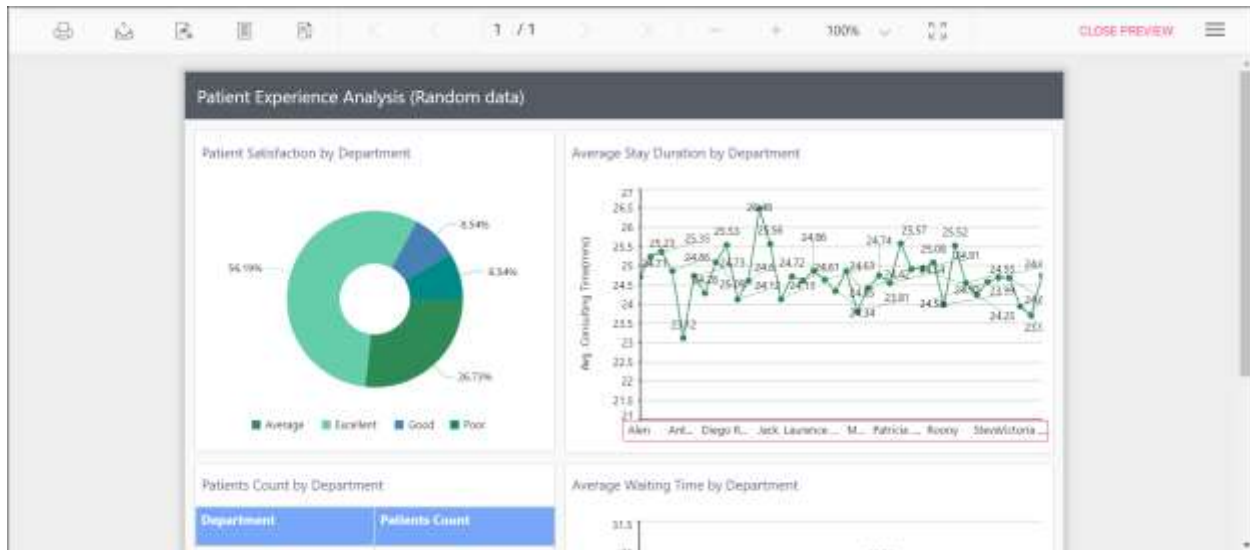
Control edge label placement in chart axis

Labels with long text at the edges of an axis may appear partially outside the chart. The `XAxisEdgeLabelPlacement` and `YAxisEdgeLabelPlacement` custom property can be used to avoid the partial appearance of the labels at the corners. The default value is `none`.

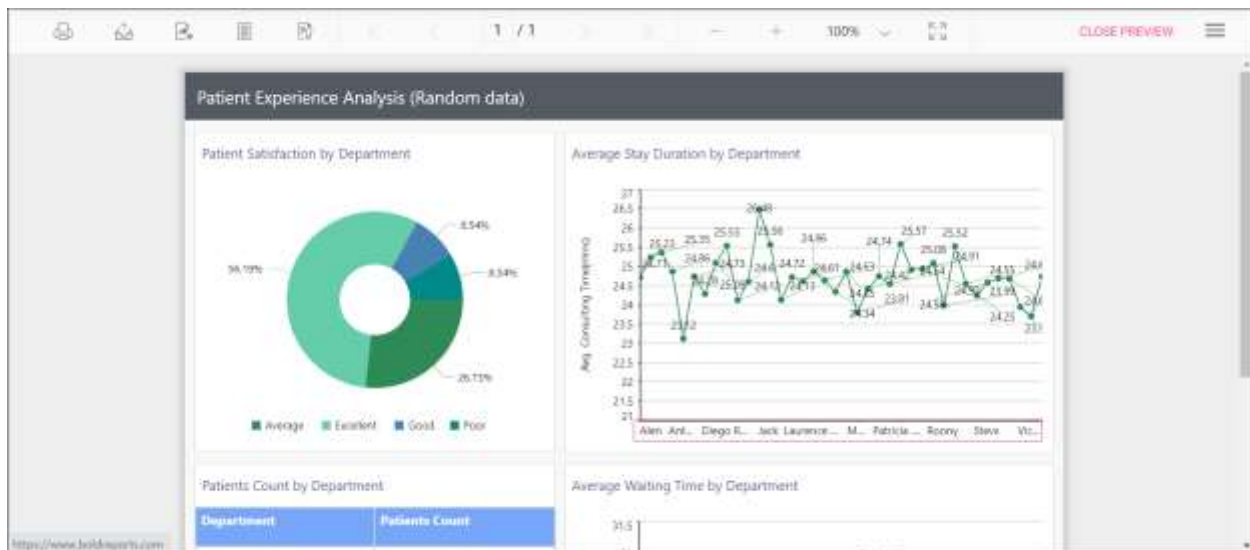
You can set the `XAxisEdgeLabelPlacement` property value, as shown in the below.



Before setting the edge label placement, the default value will be displayed as below.



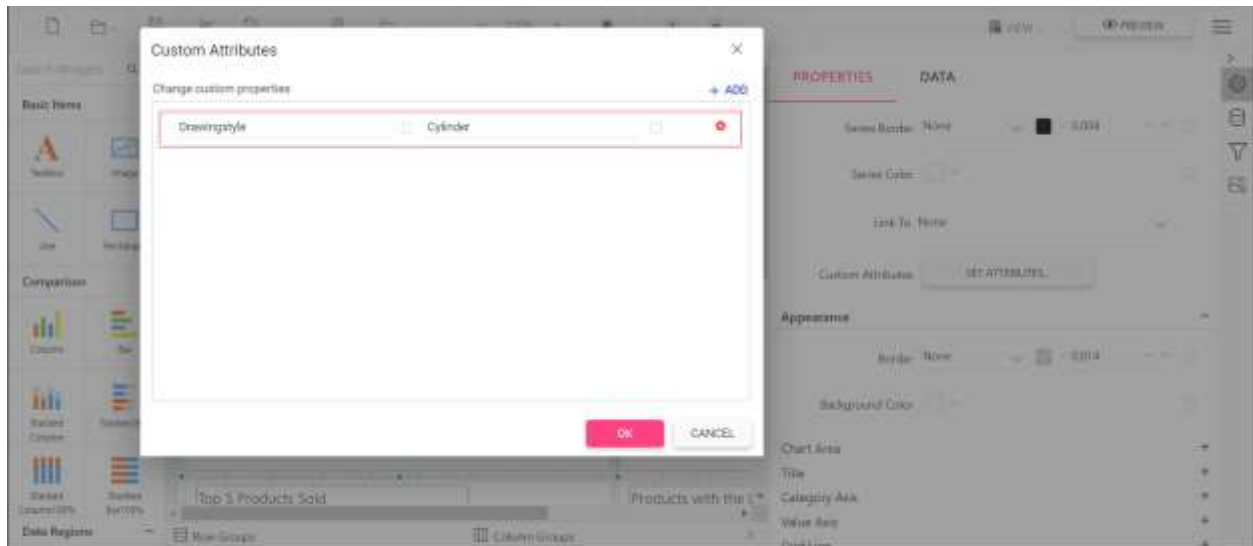
Set the edge label placement and see the changes in chart report as below.



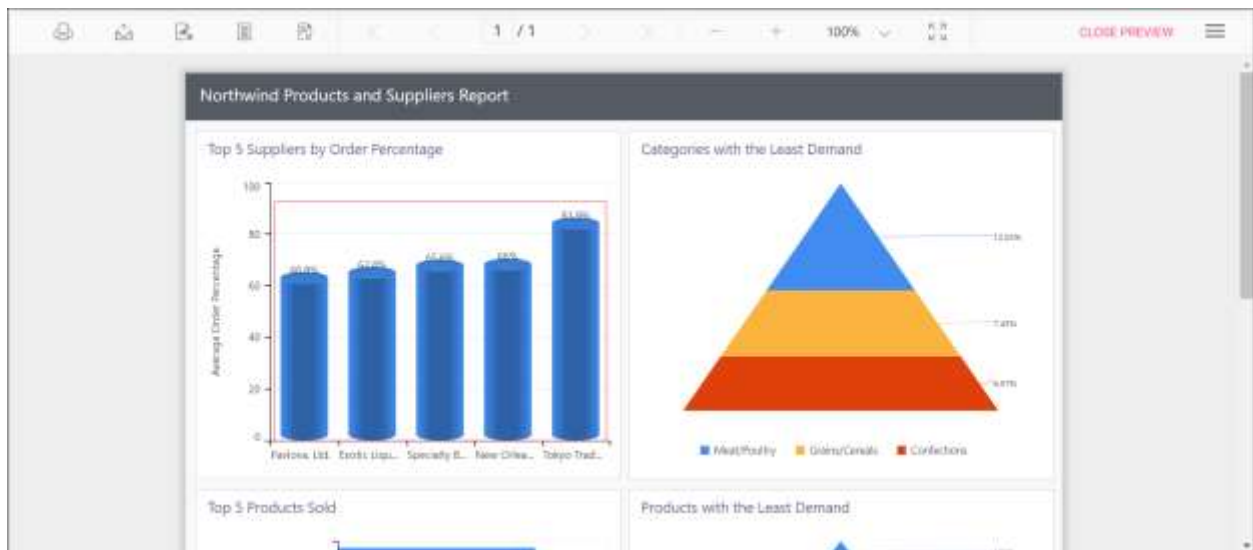
Change the drawing style of a chart column or bar series

The shape of the chart column or bar can be changed using this **Drawingstyle** custom property. By default, the **Drawingstyle** property value is **rectangle**.

You can set the **Drawingstyle** property value, as shown in the below.



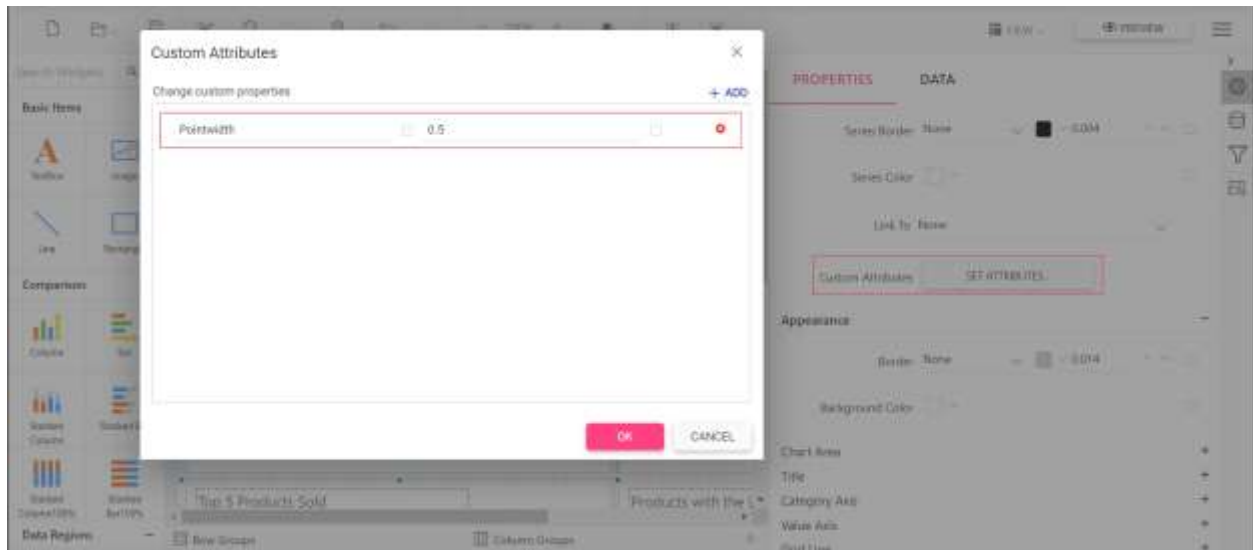
Preview the report and the see the column or bar shape in chart report.



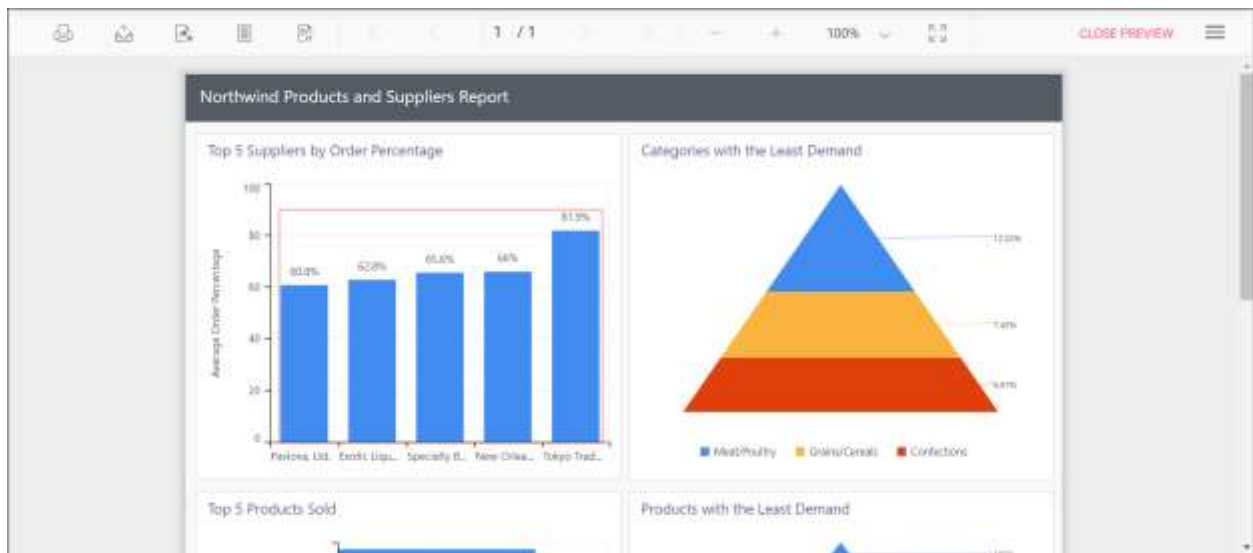
Change width of the column type series in chart

Width of the column type series can be customized by using the **Pointwidth** property. Default value of **Pointwidth** is 0.7. Value ranges from 0 to 1. Here 1 corresponds to 100% of available width and 0 corresponds to 0% of available width.

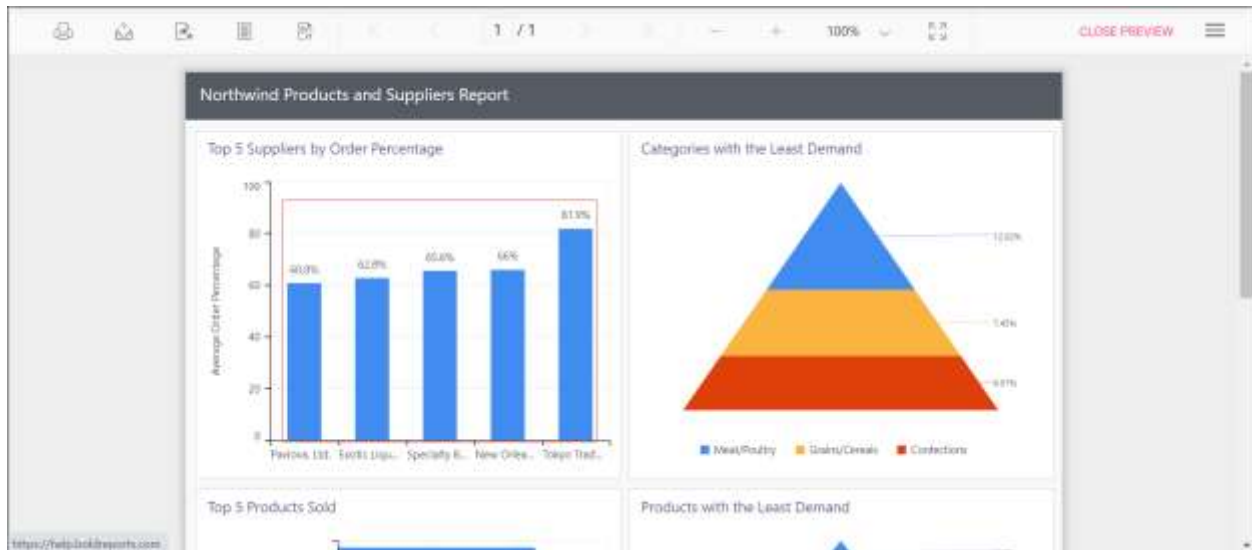
You can set the **Pointwidth** property value, as shown in the below.



Before setting the point width, the default value will be displayed as below.



Preview the report and the see the column width in chart report.



report custom properties

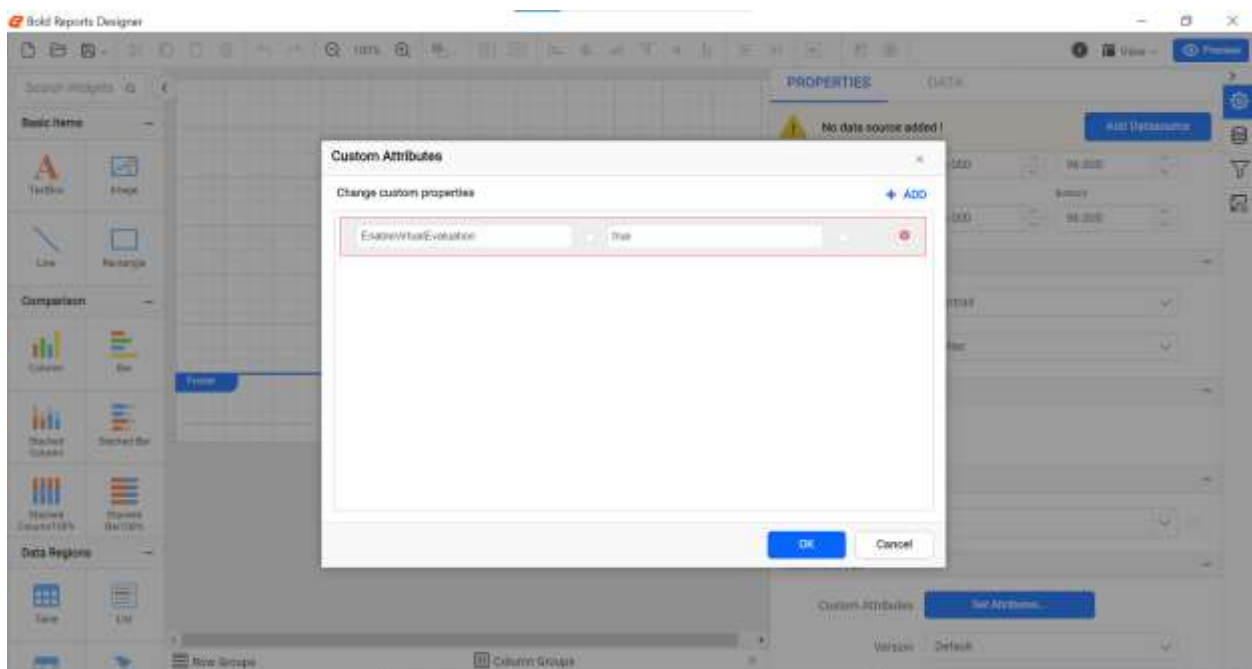
This topic explains about the list of report custom properties that are supported to render in the ASP.NET Core Report Viewer.

Improve performance and handle large amount of data

Set the `EnableVirtualEvaluation` and `DisablePageSplitting` custom properties in a report to improve performance and handle the larger amount of data with less memory footprint.

Render large data report faster

The `EnableVirtualEvaluation` custom property is used to render the large data report faster. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.

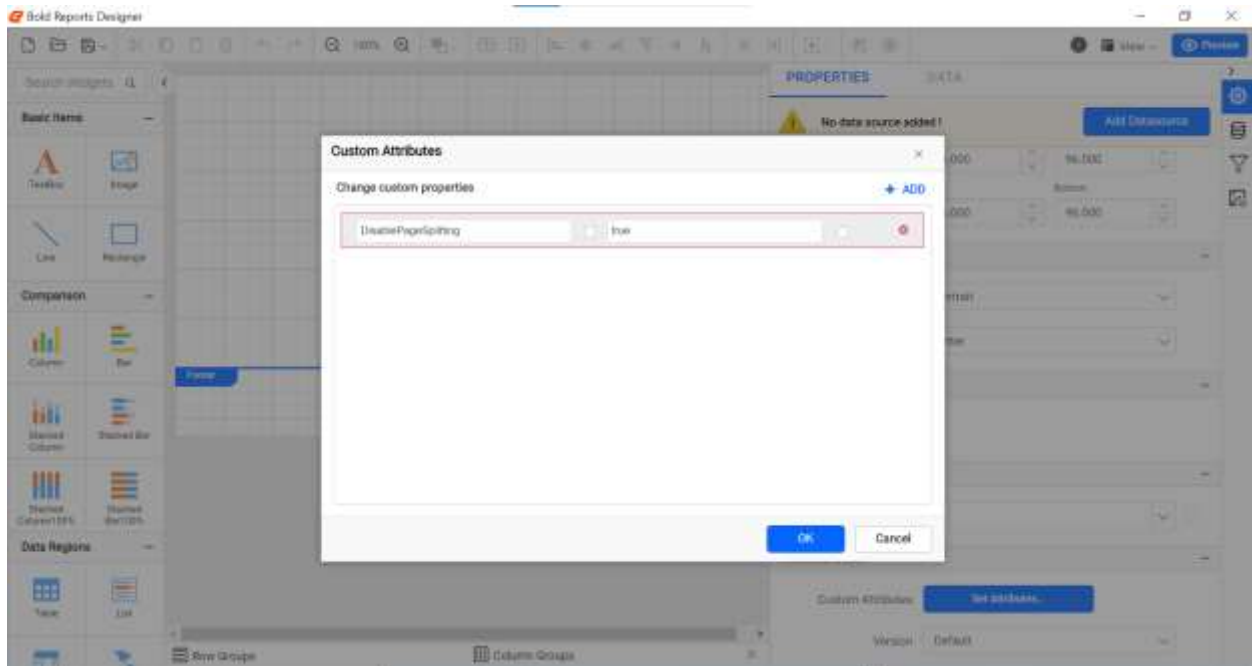


report custom properties

Improve report items layout and avoid extra blank pages

Reduce memory footprint for large data report

The `DisablePageSplitting` custom property is used to reduce the memory footprint for large data report. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.

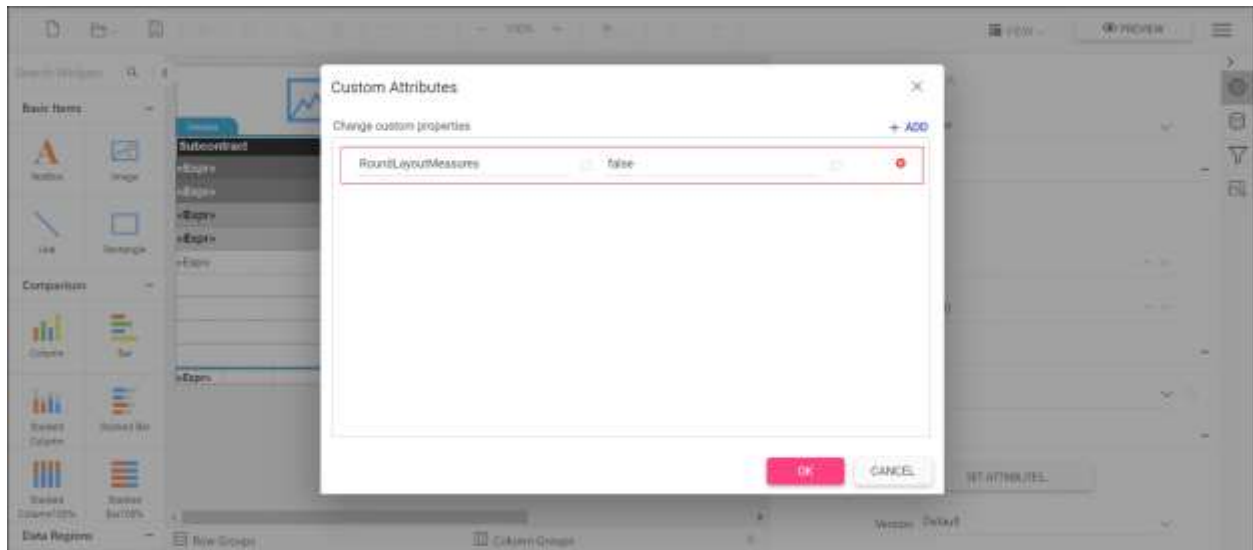


Improve report items layout and avoid extra blank pages

When the `RoundLayoutMeasures` property is false, all non-integral values that are calculated during the report processing are rounded to whole pixel values. It provides following improvements,

- Report text rendered without cuts.
- Eliminates extra blank pages.
- Removes item and text overlaps.
- Eliminates the blur semi-transparent edges that are produced by anti-aliasing.
- Produces identical look in report view and export output.

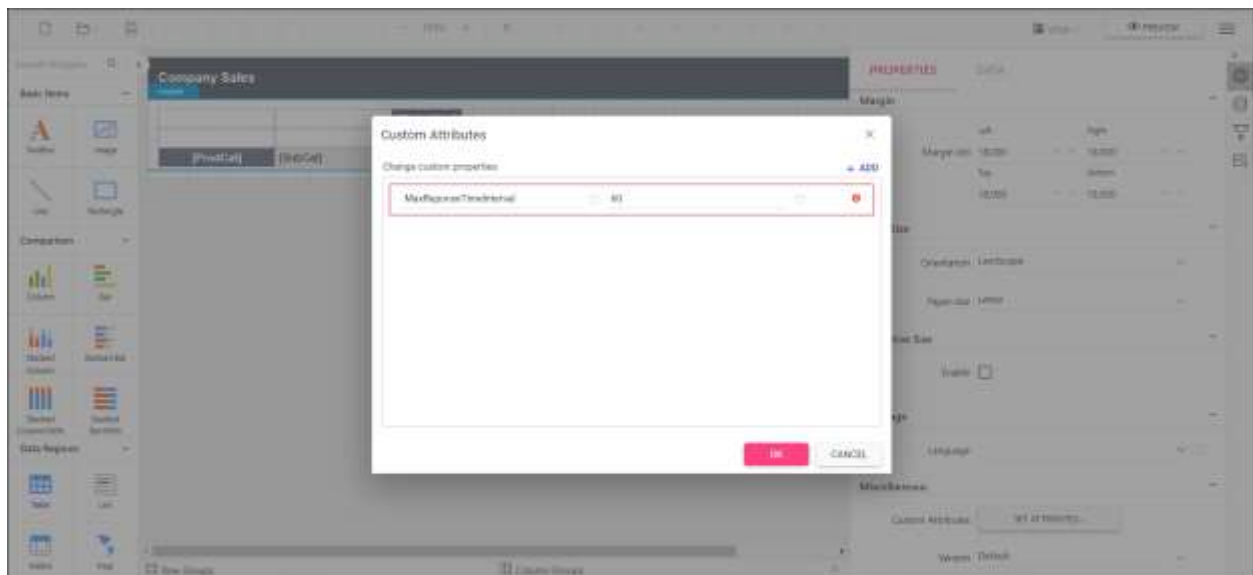
You can set the property value, as shown in the below.



Handling failure in long-running HTTP requests, report processes, and timeouts

When a report process for a long time due to huge records or a HTTP request takes too long to respond then it results in Gateway Timeout or report rendering errors. The `MaxResponseTimeInterval` allows to specify the seconds to process an HTTP request and respond back. It helps to keep the client and server connection live by avoiding the timeout.

You can set the property value as shown in the below.



Parameter custom properties

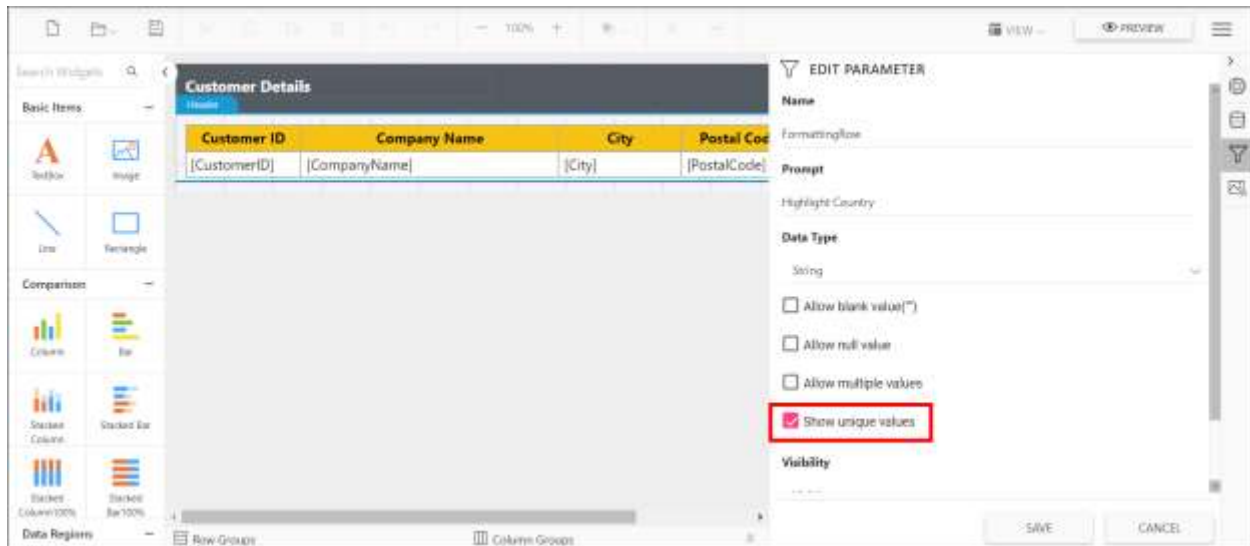
This topic explains about the list of parameter custom properties that are supported to customize the reports preview in Report Viewer.

Show only distinct values in parameter

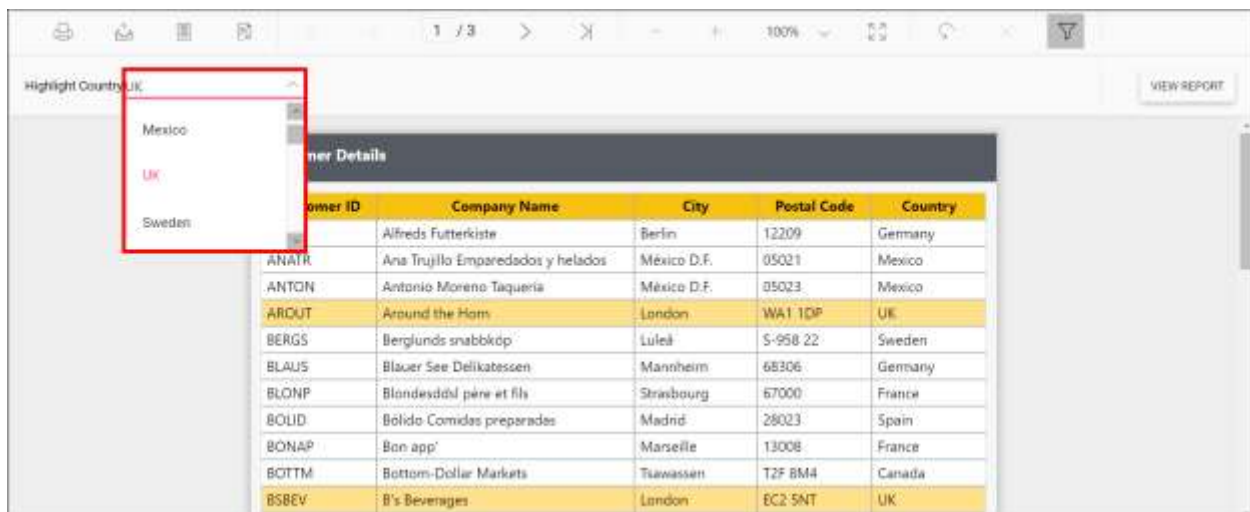
A parameter created with data set query values may contain duplicate values. The Report Viewer supports `UniqueValueParameters` custom property that helps show only unique values in the

parameter drop-down while viewing the report, without creating new a data set. Refer to the below image for property setting option.

You can also provide the parameter names with comma (,) separator to set for multiple parameters.



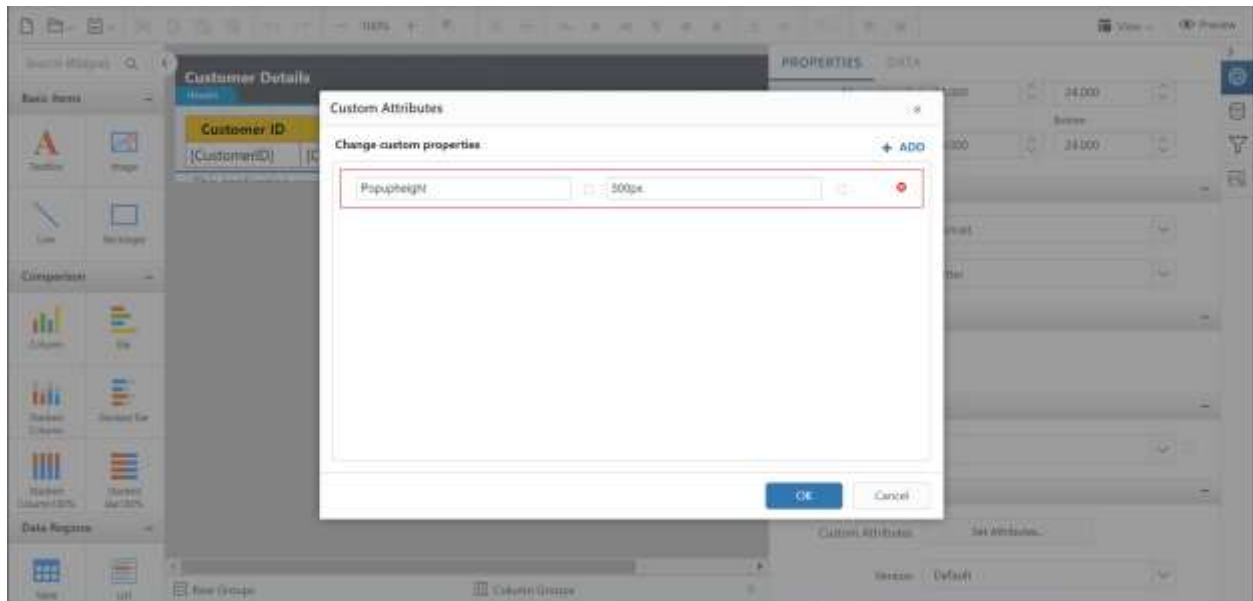
Preview the report and the unique values showed in the parameter drop down.



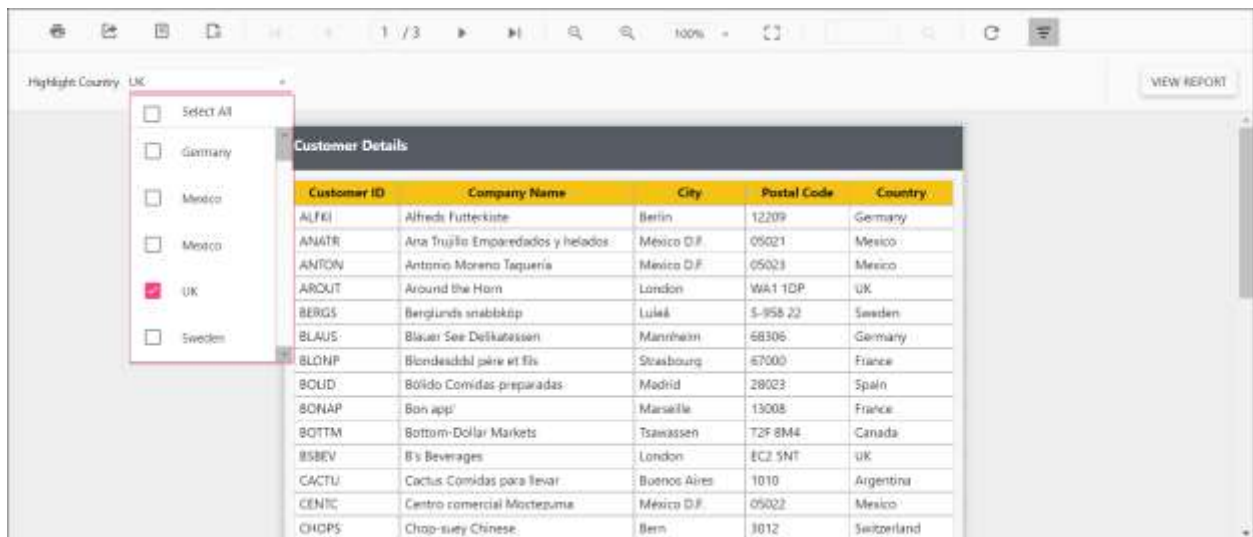
Change the dropdown parameter pop up height value

The **PopupHeight** custom property specifies the height of the parameter combobox popup list in the report. By default, the **PopupHeight** value is **152px**.

You can set the **PopupHeight** property value, as shown in the below.



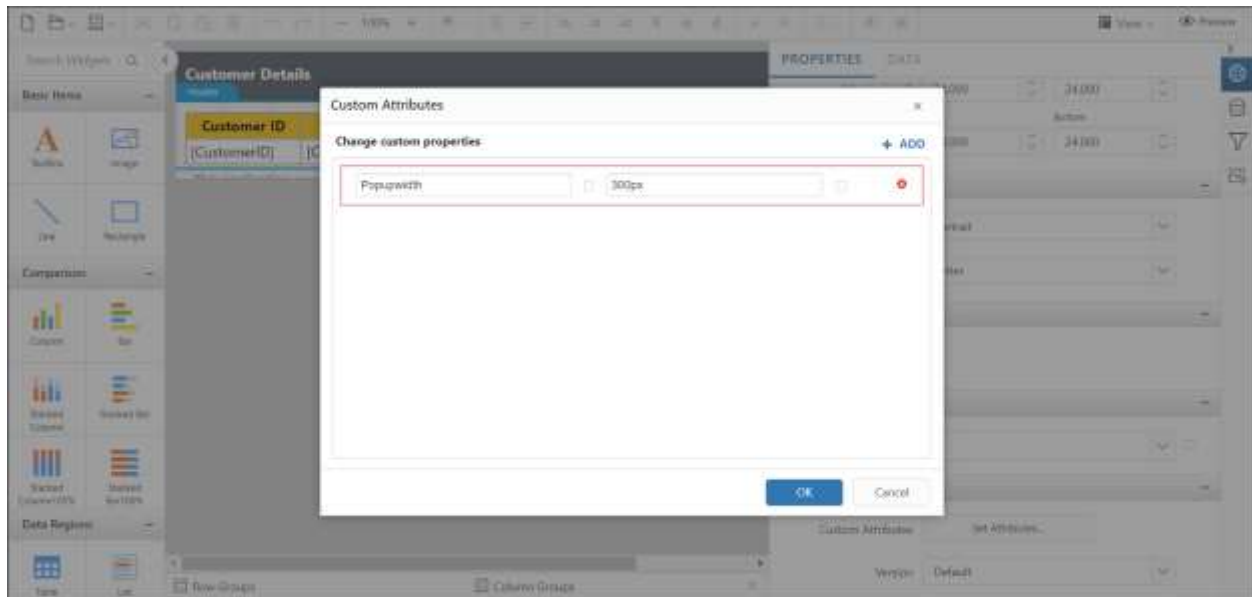
Preview the report and the pop up height showed in the parameter drop down.



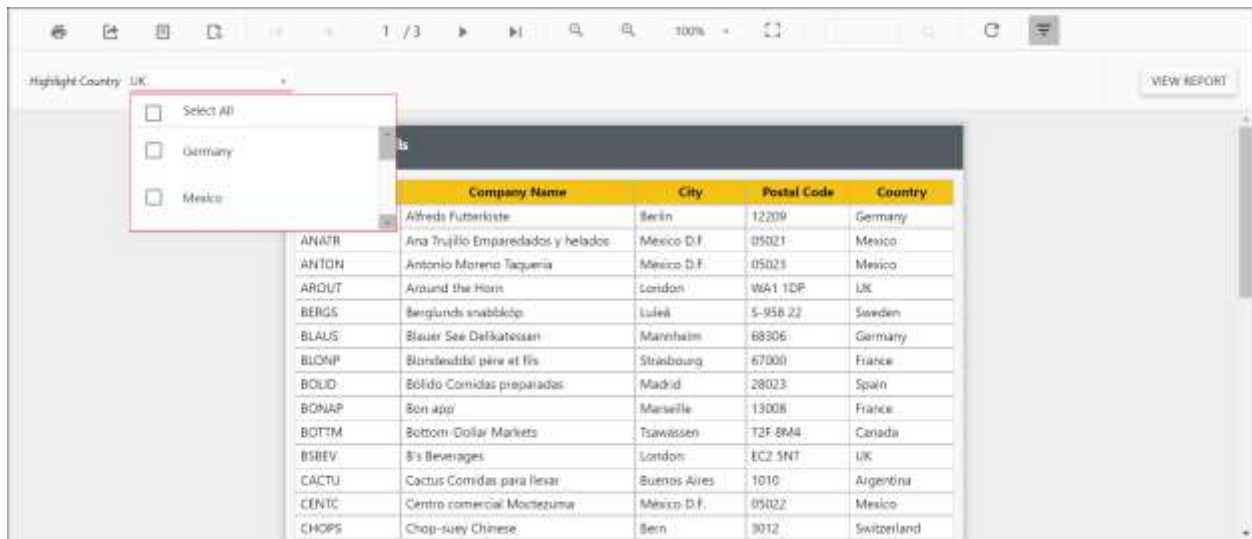
Change the dropdown parameter pop up width value

The **PopupWidth** custom property specifies the width of the parameter combobox popup list in the report. By default, the popup width sets based on the width of the component.

You can set the **PopupWidth** property value, as shown in the below.



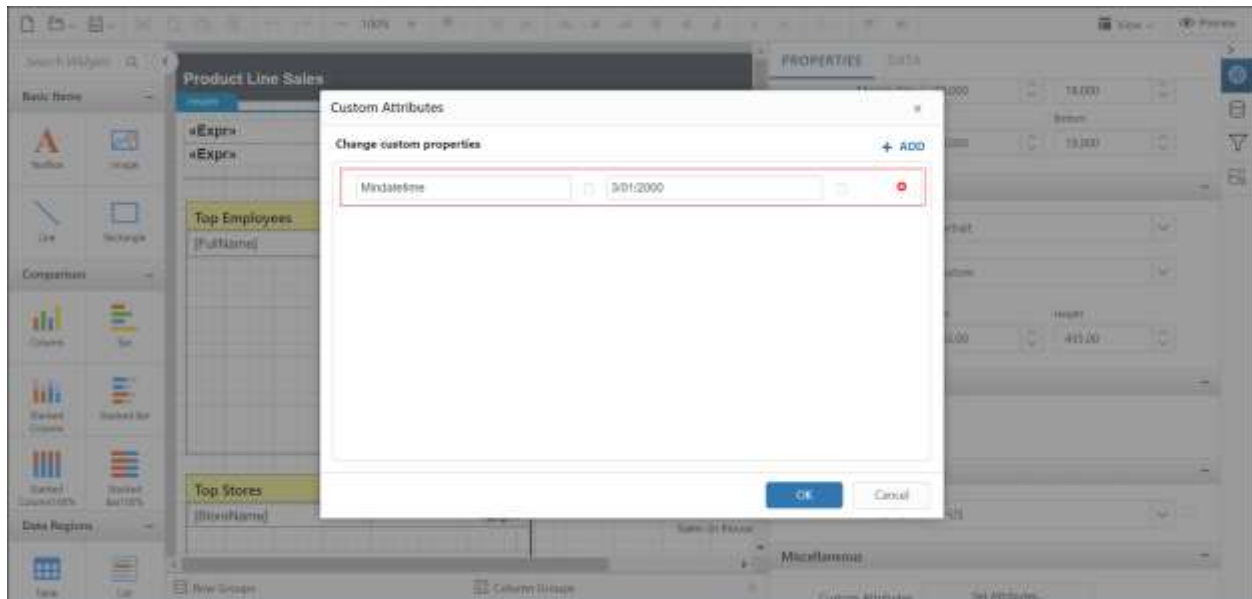
Preview the report and the pop up width showed in the parameter drop down.



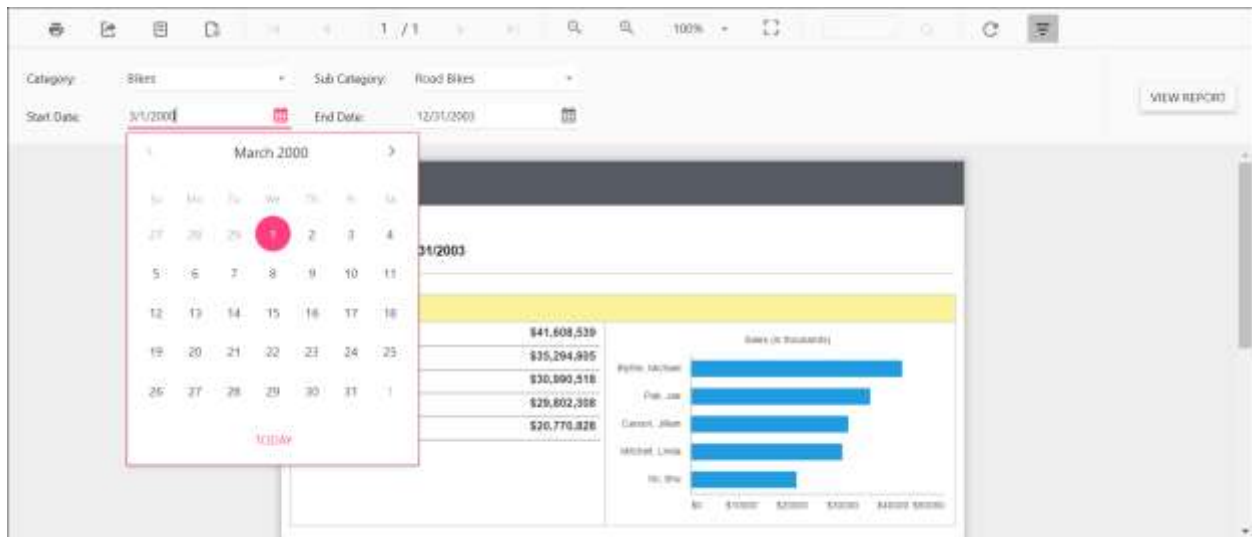
Set the minimum date range for the date report parameter

The **MinDateTime** custom property specifies the minimum date in the datetime parameter item. By default, the **MinDateTime** value is set as **null**.

You can set the **MinDateTime** property value, as shown in the below.



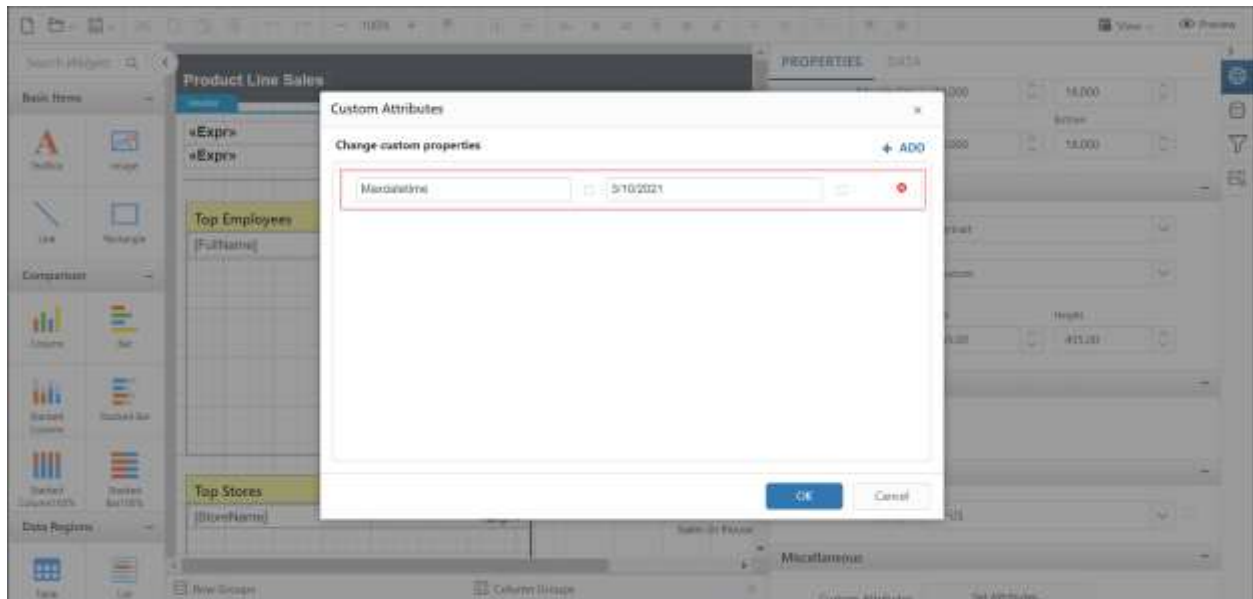
Preview the report and the minimum date showed in the datetime parameter drop down.



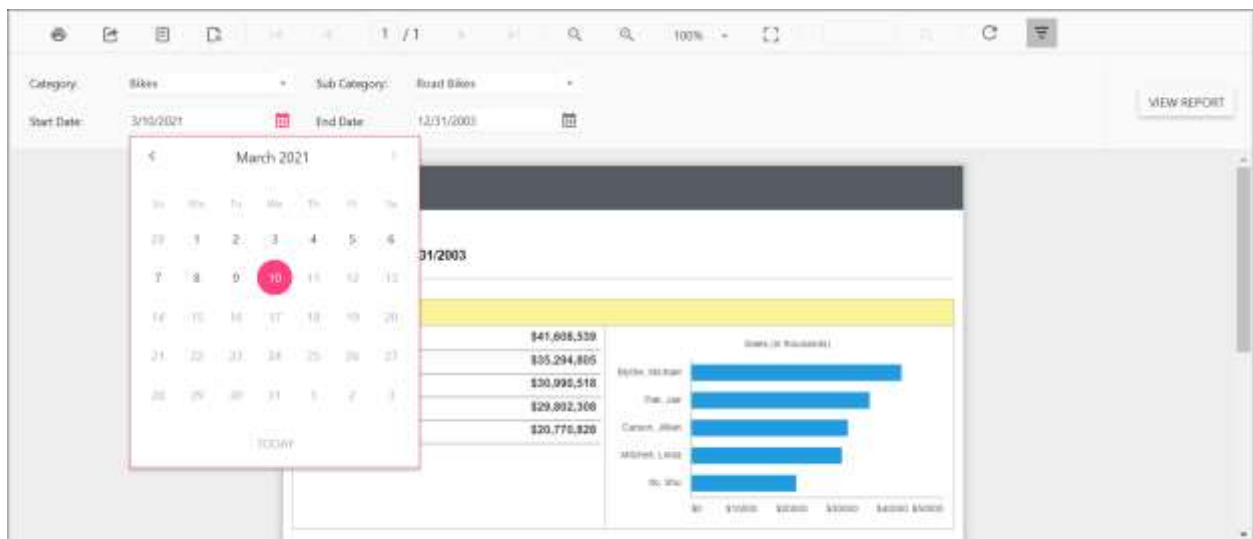
Set the maximum date range for date report parameter

The `MaxDateTime` custom property specifies the maximum date in the datetime parameter item. By default, the `MaxDateTime` value is set as `null`.

You can set the `MaxDateTime` property value, as shown in the below.



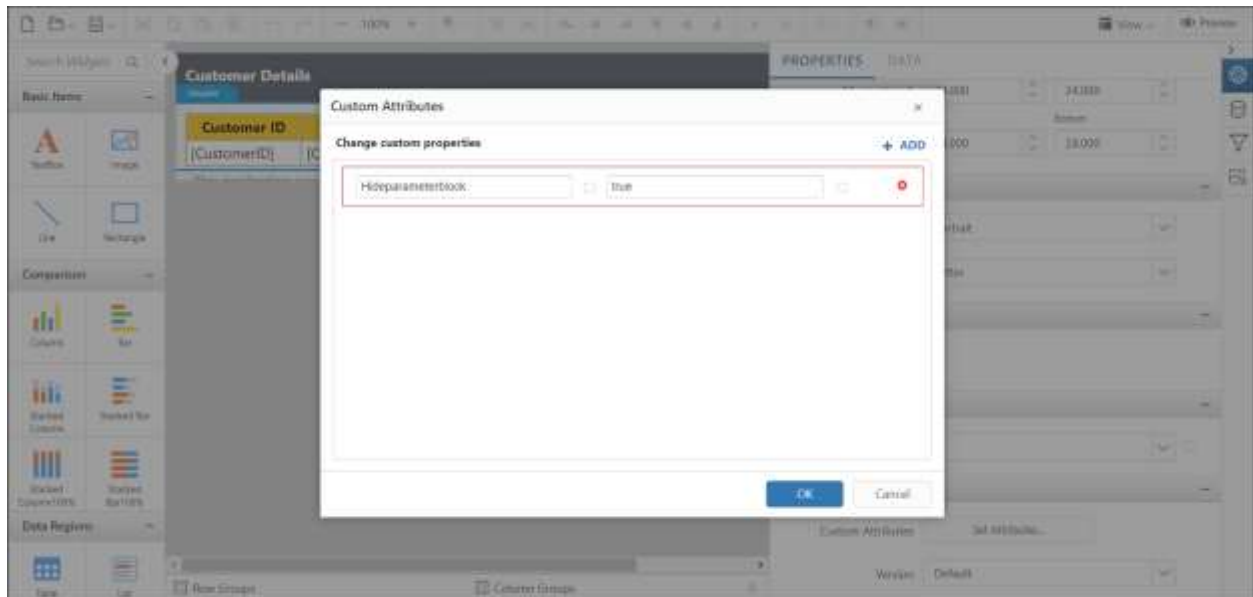
Preview the report and the maximum date showed in the datetime parameter drop down.



Hide the report parameter block

The `HideParameterBlock` custom property is used to hide the parameter block on report initial rendering. The property value should be boolean. By default, the `HideParameterBlock` value is `false`.

You can set the `HideParameterBlock` property value, as shown in the below.



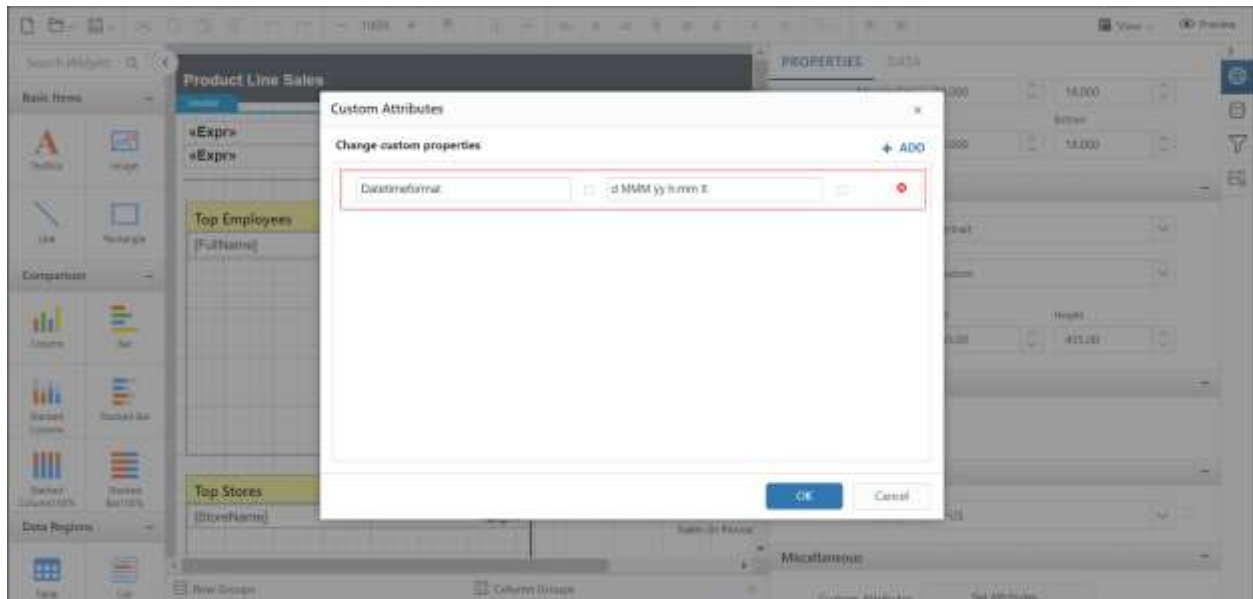
Preview the report and see the parameter block is hidden.

Customer ID	Company Name	City	Postal Code	Country
ALFKI	Alfreds Futterkiste	Berlin	12209	Germany
ANATR	Ana Trujillo Emparedados y helados	México D.F.	05021	Mexico
ANTON	Antonio Moreno Taquería	México D.F.	05023	Mexico
AROUT	Around the Horn	London	WA1 1DP	UK
BERGS	Berglunds snabbköp	Luleå	S-958 22	Sweden
BLAUS	Blaauw Sea Delicatessen	Mannheim	68306	Germany
BONAP	Bonaparte's pates en file	Strasbourg	67000	France
BOLID	Bolido Comidas preparadas	Madrid	28023	Spain
BONAP	Bonaparte	Marseille	13008	France
BOITM	Bottom-Dollar Markets	Toronto	T2F 8M4	Canada
BSBEV	B's Beverages	London	EC2 5NT	UK
CACTU	Cactus Comidas para llevar	Buenos Aires	1010	Argentina
CENTC	Centro comercial Motezuma	México D.F.	05022	Mexico
CHOPS	Chop-icey Chinese	Bern	3012	Switzerland
COMM1	Comércio Mineiro	São Paulo	05432-043	Brazil
CONSH	Consolidated Holdings	London	WX1 6LT	UK
DRACD	Drachendorff & Schmitt	Dresden	81068	Germany

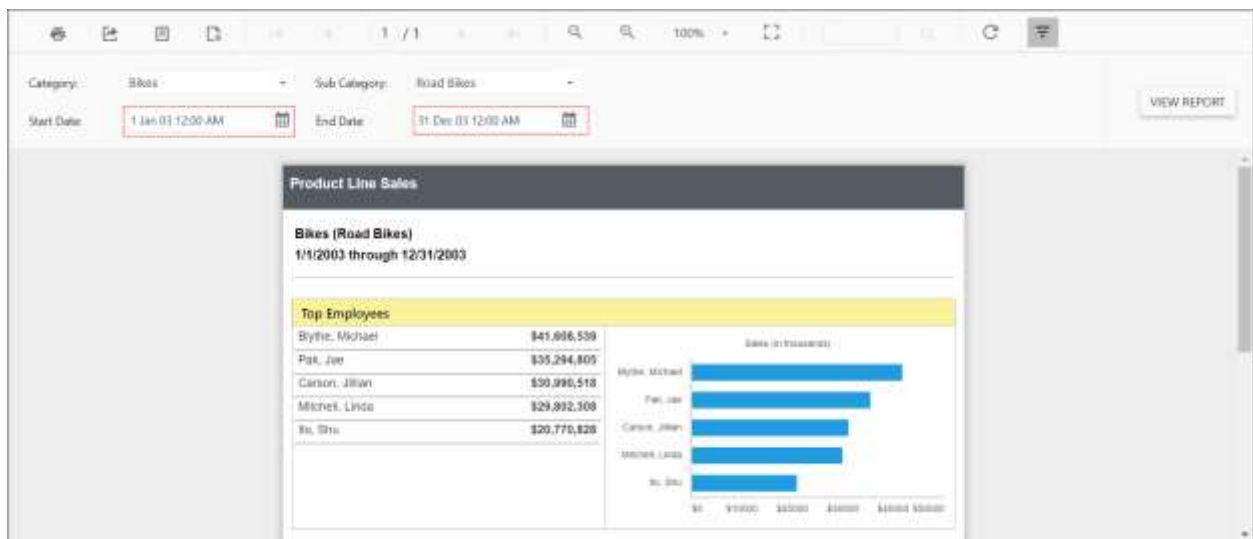
Set date and time format for parameter

The `DateTimeFormat` custom property defines the date time format to be displayed in the `DateTimePicker` popup. By default, the `DateTimeFormat` value is set as `empty`.

You can set the `DateTimeFormat` property value, as shown in the below.



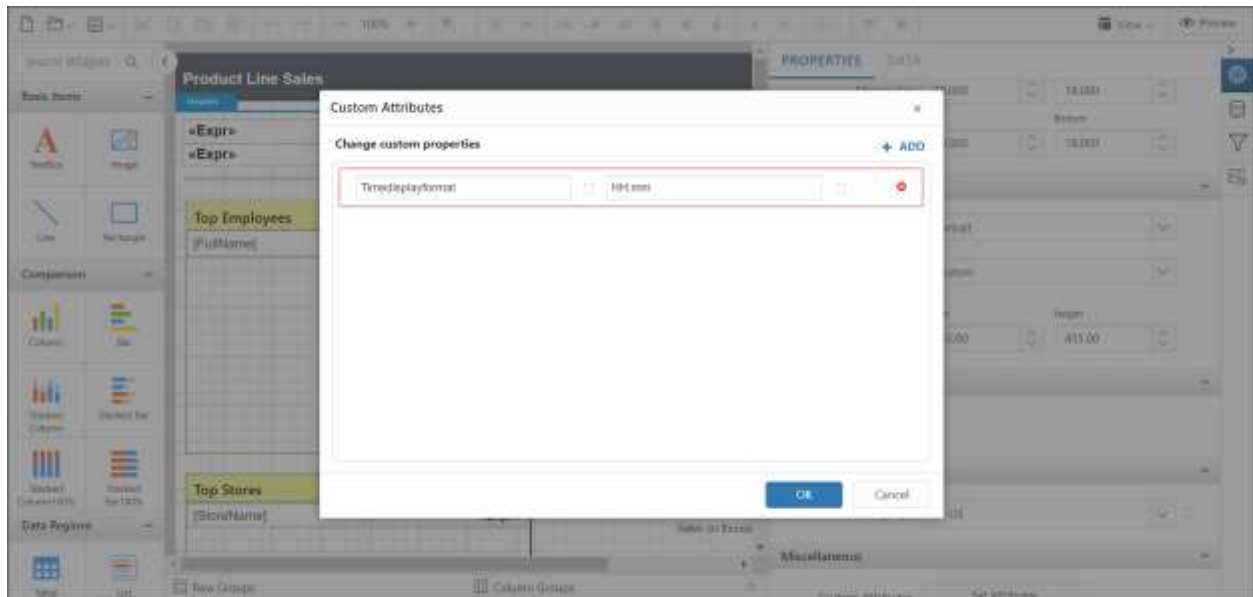
Preview the report and the see date time format in datetime parameter.



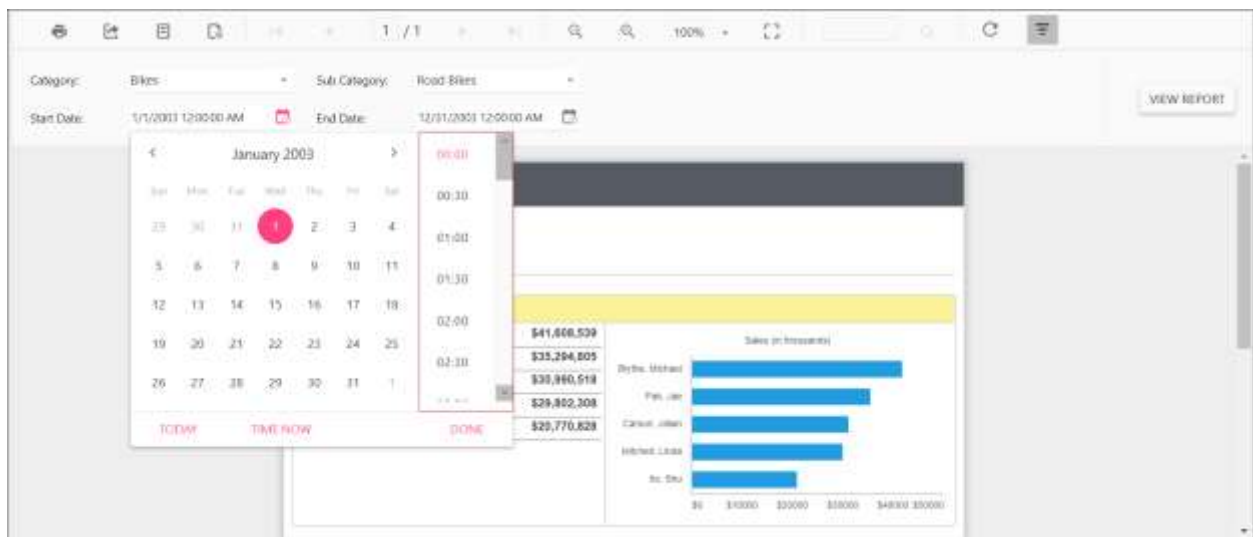
Change time display format for parameter

The `TimeDisplayFormat` custom property defines the time format to be displayed in the time dropdown inside the `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeDisplayFormat`. By default, the `TimeDisplayFormat` value is set as empty.

You can set the `TimeDisplayFormat` property value, as shown in the below.



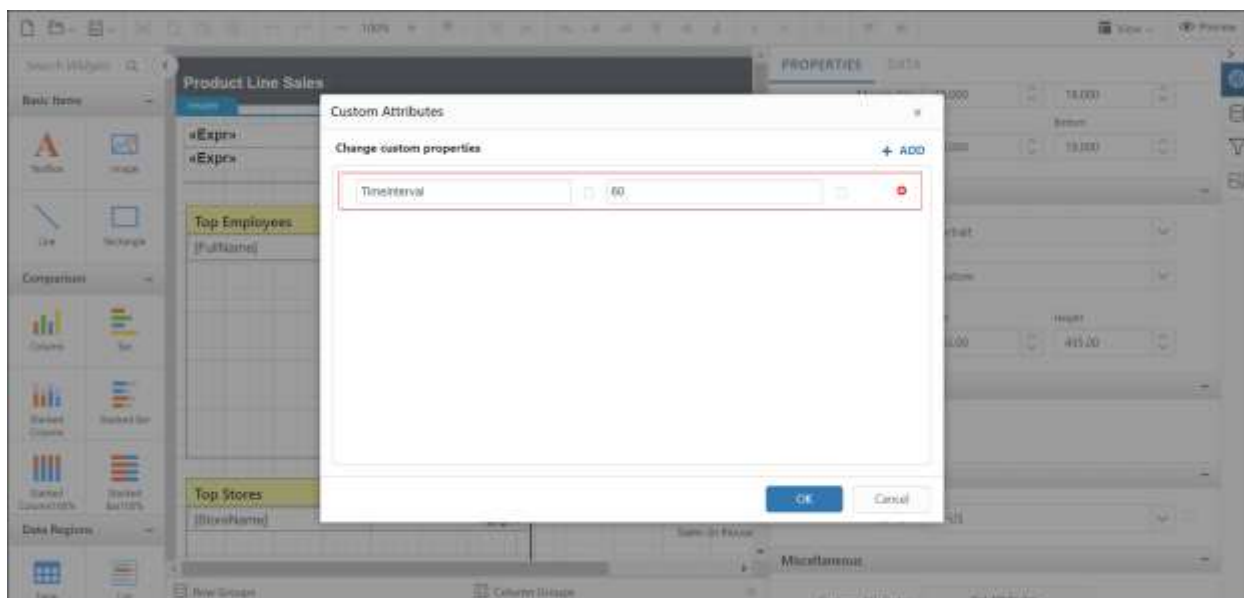
Preview the report and the see time format in datetime parameter.



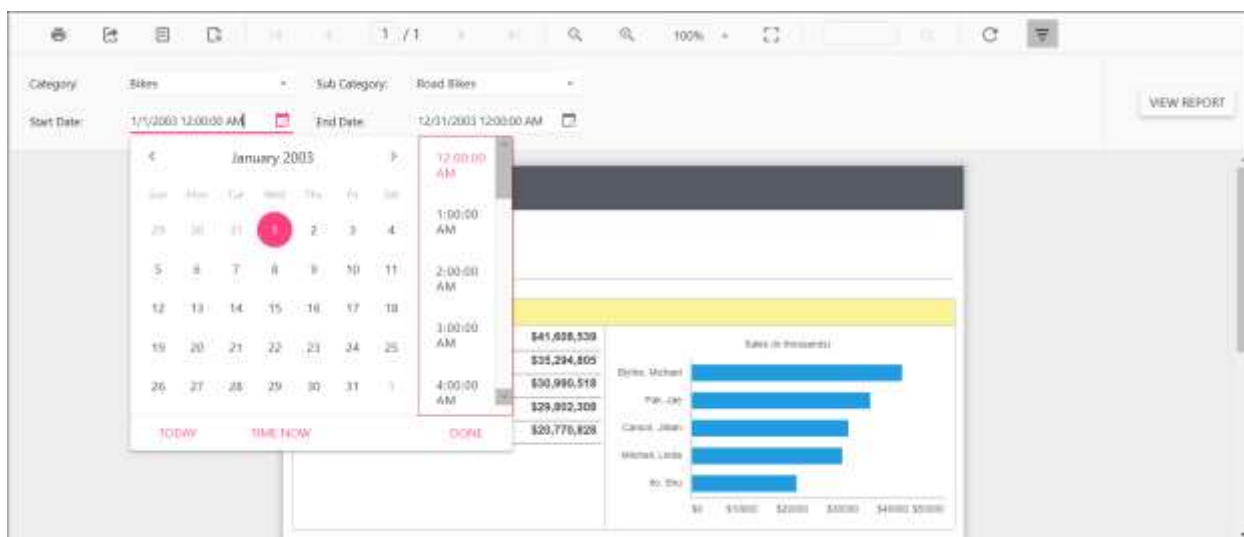
Set time interval in datetime parameter

The `TimeInterval` custom property is used to set the interval between the two adjacent time values in `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeInterval`. By default, the `TimeInterval` value is set as 30.

You can set the `TimeInterval` property value, as shown in the below.



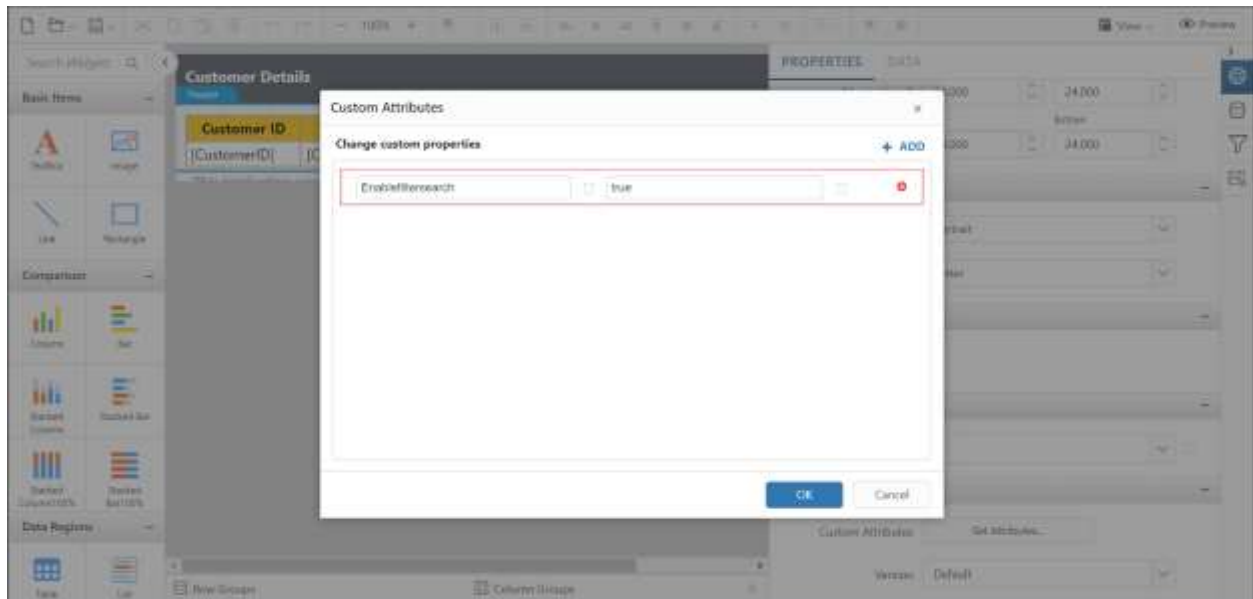
Preview the report and the see time interval in datetime parameter.



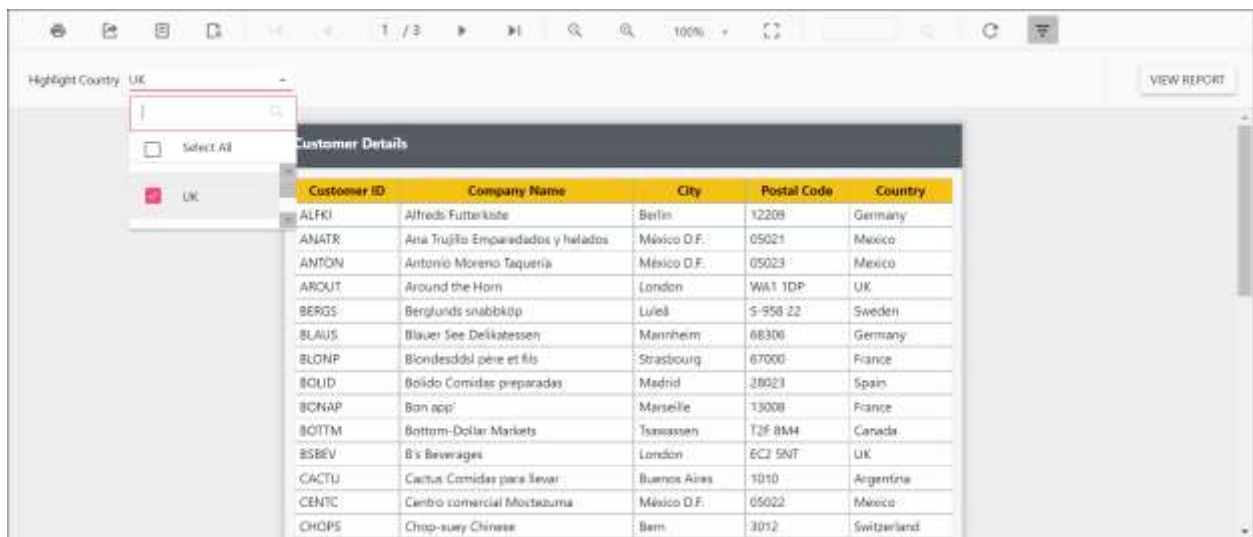
Enable filtering and searching in drop-down report parameter

Setting `EnableFilterSearch` custom property enables search and filtering option in dropdown parameter to easily find the values. The property value should be boolean. By default, the `EnableFilterSearch` value is `false`.

You can set the `EnableFilterSearch` property value, as shown in the below.



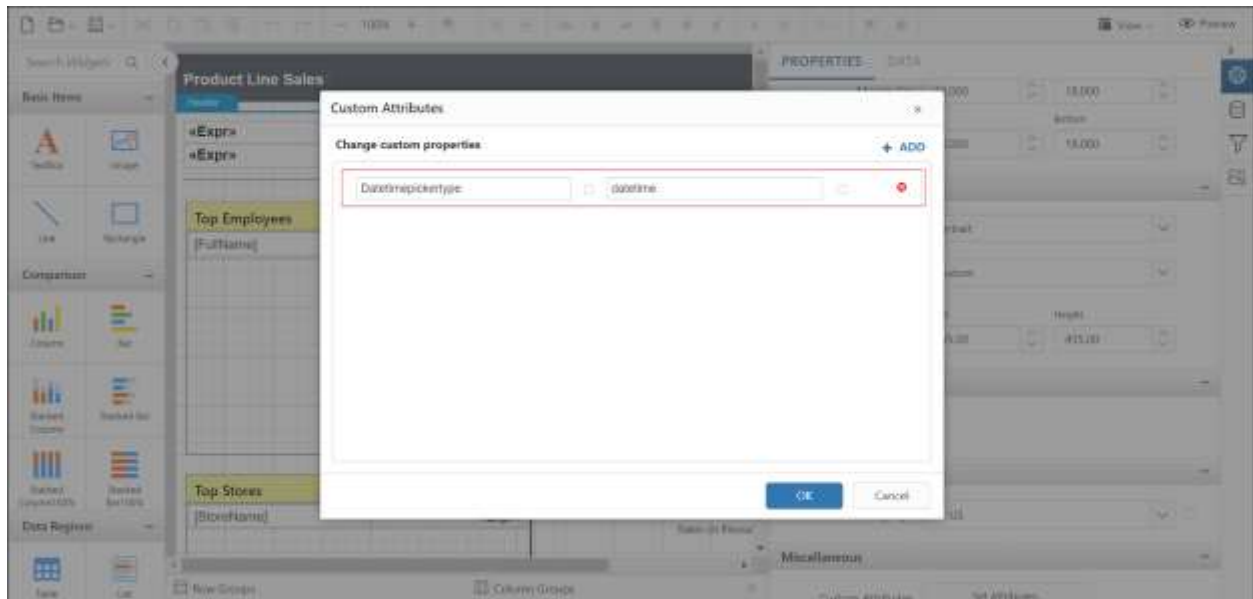
Preview the report and the see search filter in parameter.



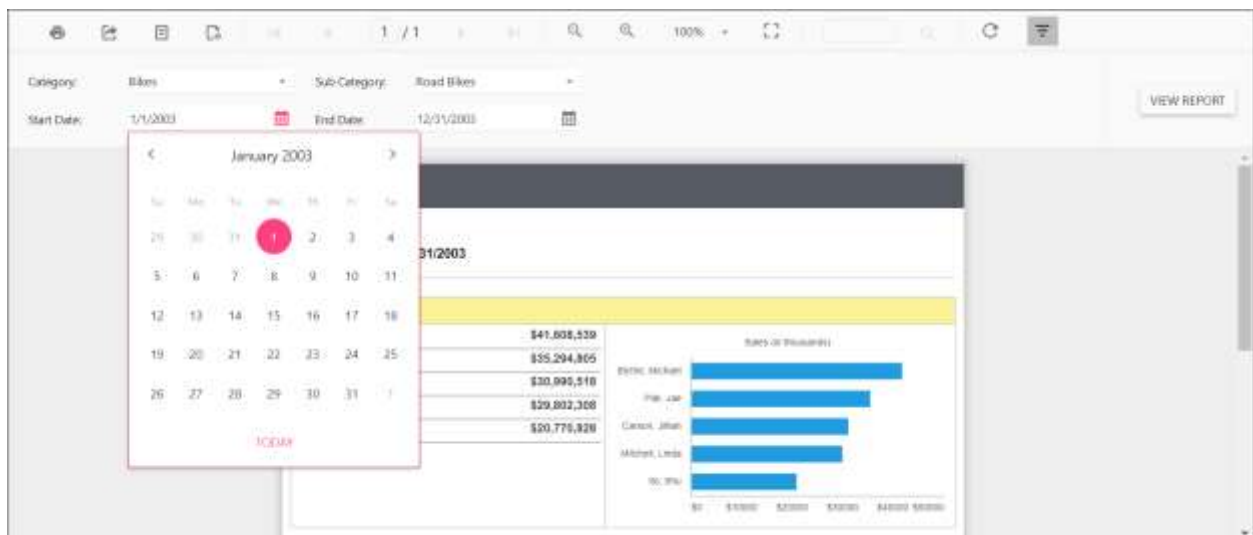
Change date report parameter to display date time picker

You can set `DateTimePickerType` custom property value as `DateTime` to change date parameter item to display `DateTimePicker` for value selection as shown the below.

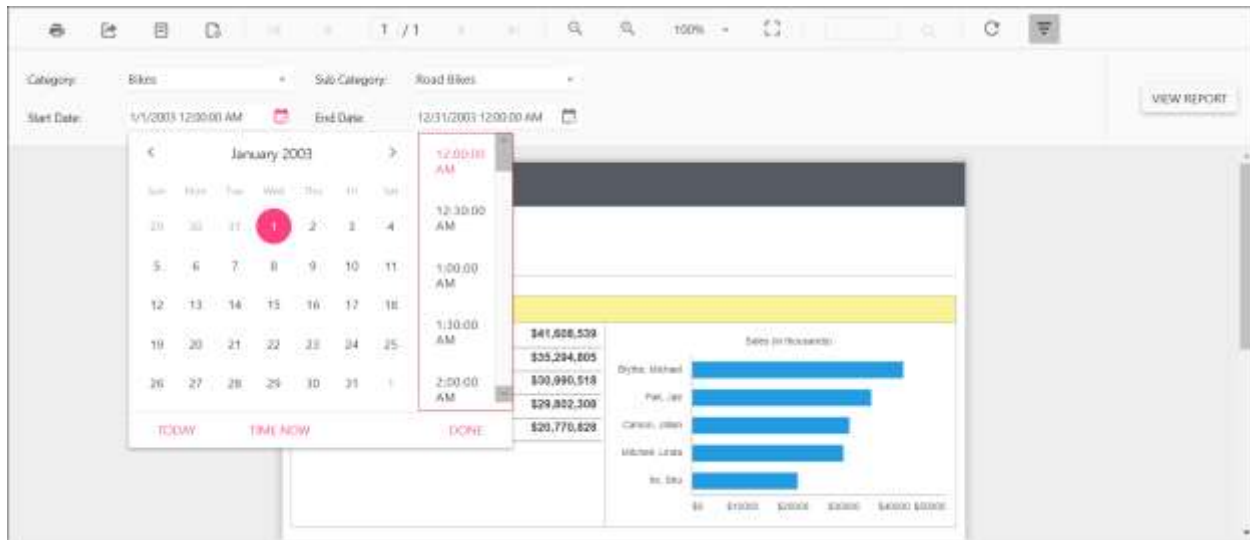
You can set the `DateTimePickerType` property value, as shown in the below.



Before enabling date time picker type, the default value will be displayed as below..



Enable date time picker type and see the time in `DateTimePicker` as in below output.



Export custom properties

This topic explains the list of custom properties that are supported at the report level to control the export behaviour in ASP.NET Core Report Viewer.

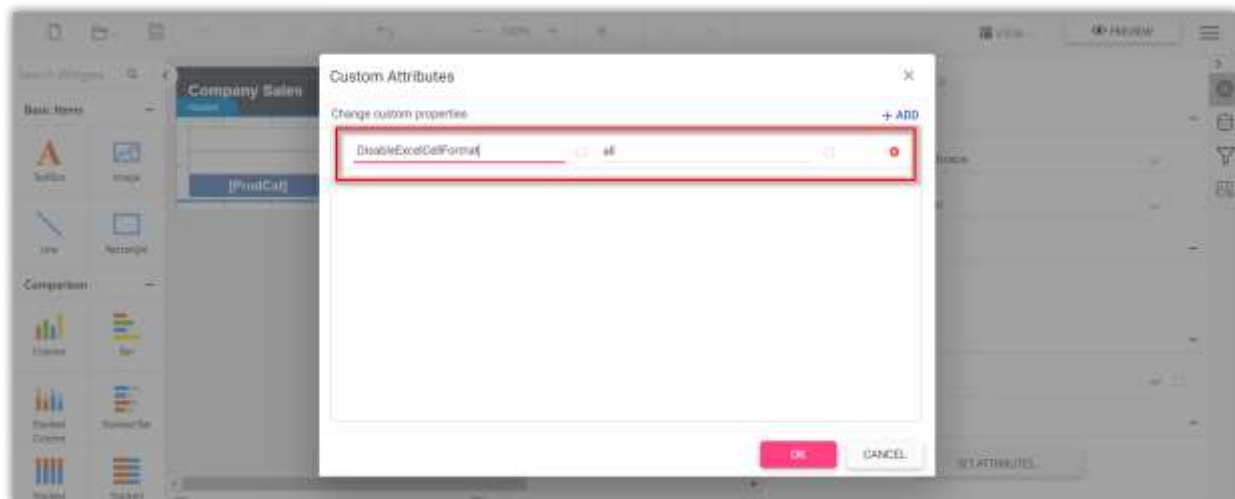
Improve excel export performance

The custom property `DisableExcelCellFormat` helps to improve the excel export performance by ignoring the rendering of report item styles. It allows property value from anyone of the following values.

- **Style** - Disables rendering of the cell styles like padding, background, color, and text style
- **Border** - Disables rendering of the cell border
- **All** - Disables rendering of the cell border, padding, background, color, and text style

Set the property value as **All** to improve maximum excel export performance.

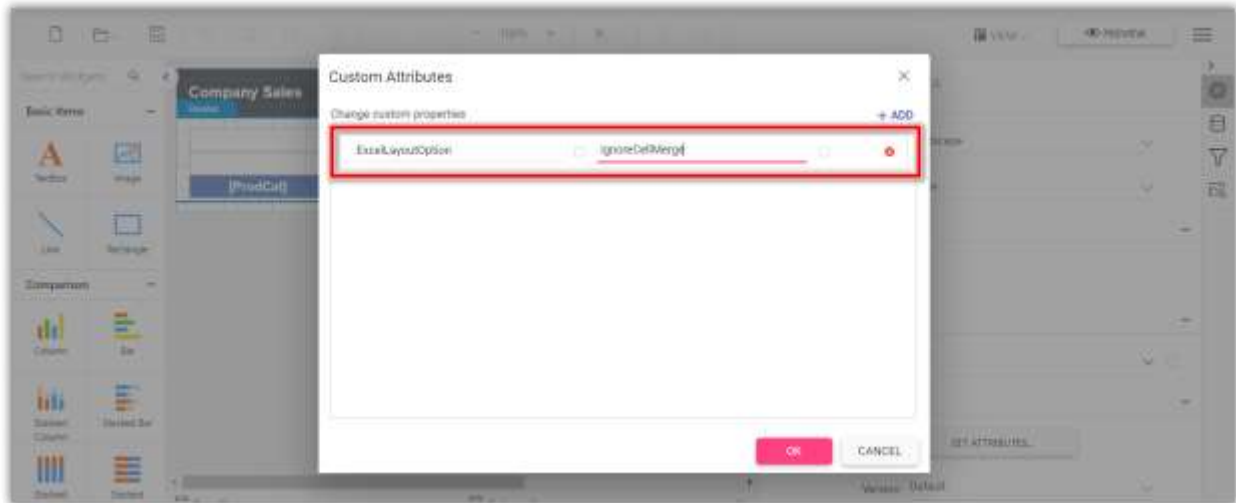
You can set the `DisableExcelCellFormat` custom property as shown below,



The `DisableExcelCellFormat` property must be added to report properties.

Improve excel export readability

Set `ExcelLayoutOption` custom property value as `IgnoreCellMerge` to improve the readability of the excel document by eliminating the tiny columns, rows, and merged cells. You can set the property value, as shown below,



The `ExcelLayoutOption` property must be added to report properties.

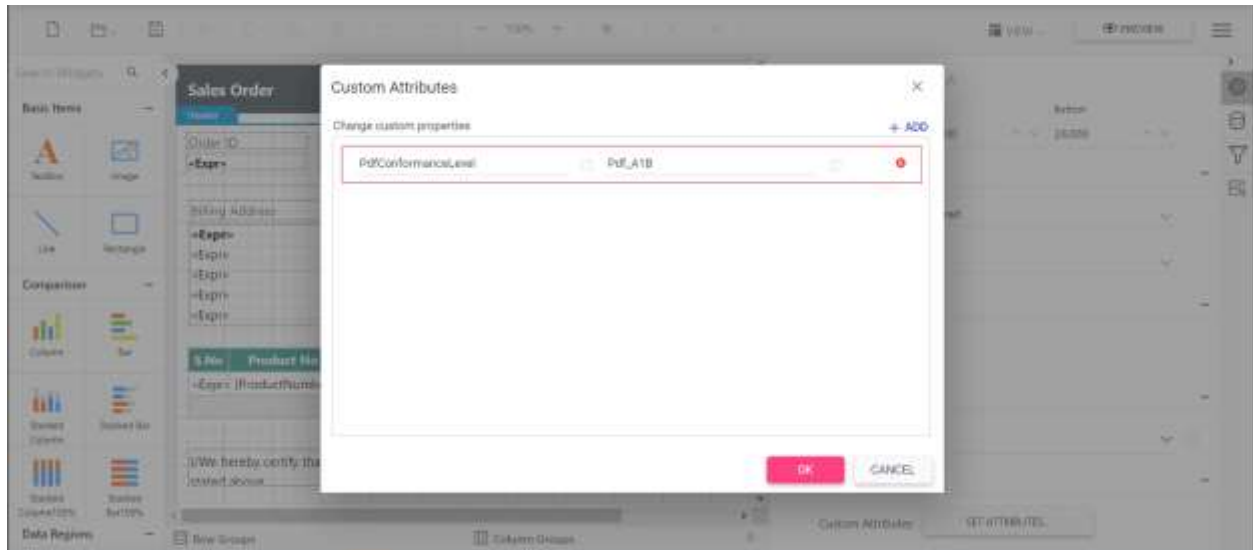
Set pdf conformance level

The `PdfConformanceLevel` allows you set PDF document versions.

You can set anyone of the following PDF conformance option.

- **PdfA1B** - You can create a PdfA1B document by specifying the conformance level as Pdf_A1B through PdfConformanceLevel Enum when creating the new PDF document.
- **PdfA2B** - You can create a PdfA2B document by specifying the conformance level as Pdf_A2B through PdfConformanceLevel Enum when creating the new PDF document
- **PdfA3B** - The PDF/A3B conformance supports the external files as attachment to the PDF document, so you can attach any document format such as Excel, Word, HTML, CAD, or XML files.
- **PdfA1A** - This conformance includes all PdfA1B requirements in addition to the features intended to improve a document's accessibility. PDF/A-1a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2A** - This conformance includes all PDF/A2B requirements in addition to the features intended to improve a document's accessibility. PDF/A-2a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2U** - This conformance includes all PdfA2U requirements, and additionally Unicode mapping for all text in the document.
- **PdfX1A2001** - You can create a PDF/X-1a document by specifying the conformance level as PdfX1A2001 through PdfConformanceLevel Enum when creating the new PDF document.

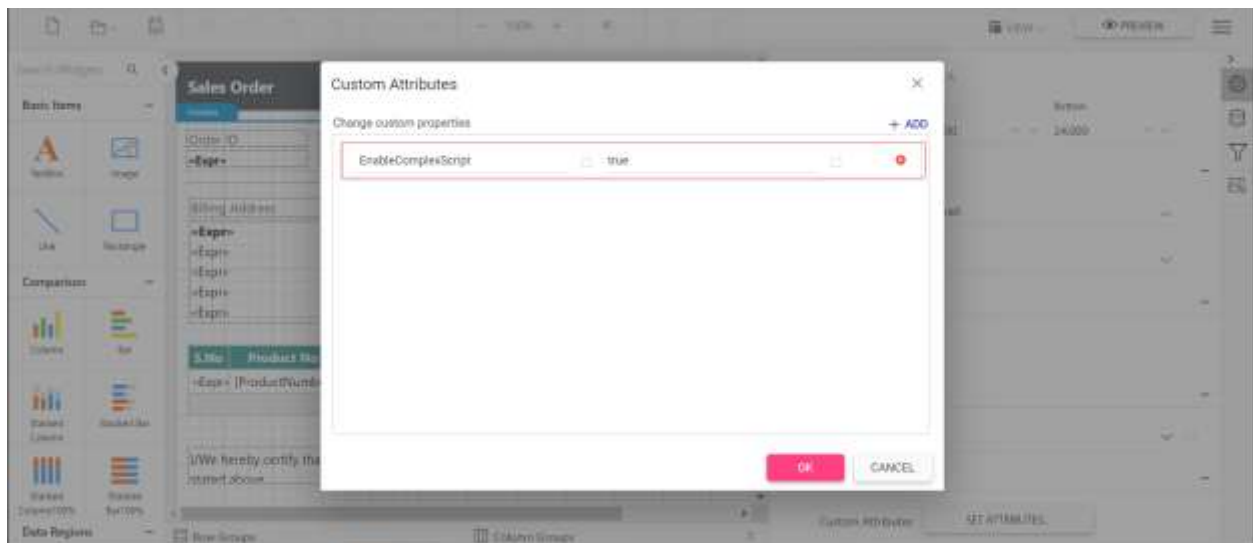
You can set the `PdfConformanceLevel` custom property as shown below,



Export pdf document with complex script

The `EnableComplexScript` custom property allows you to export pdf documents with complex script language texts.

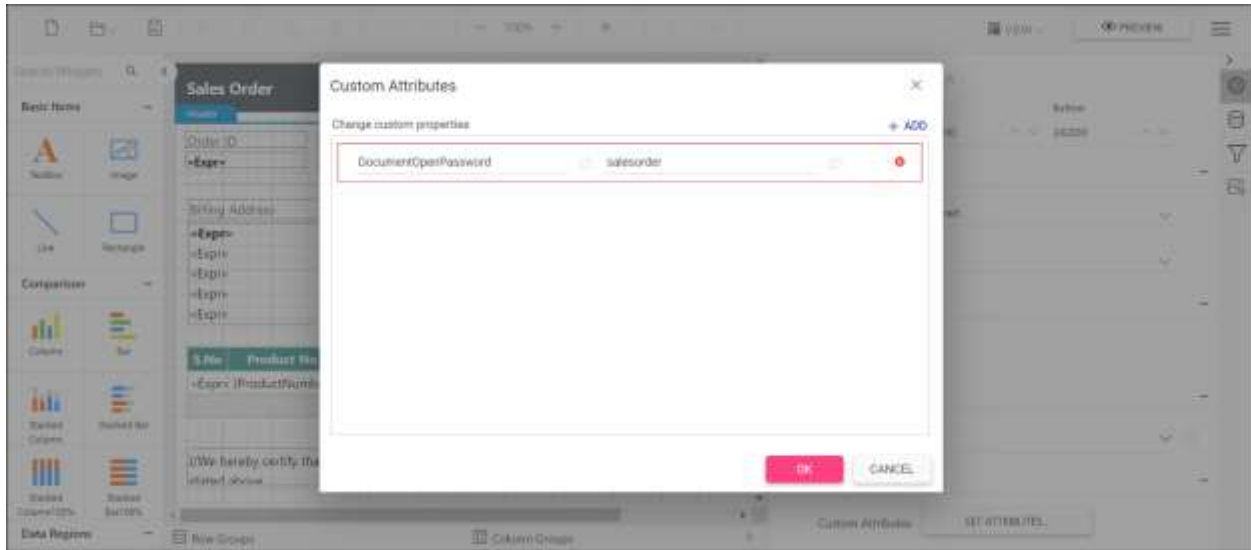
You can set the property value, as shown below,



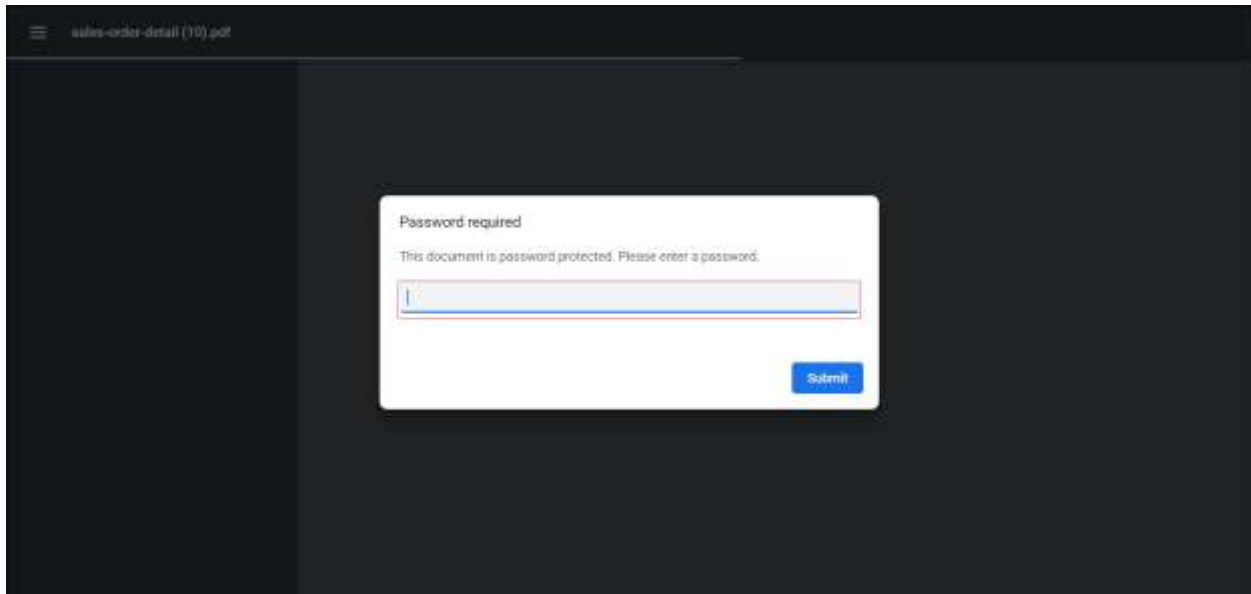
Encrypt and secure export documents

The custom property `DocumentOpenPassword` allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password.

You can set the property value, as shown below,



open the pdf document and see the below output.

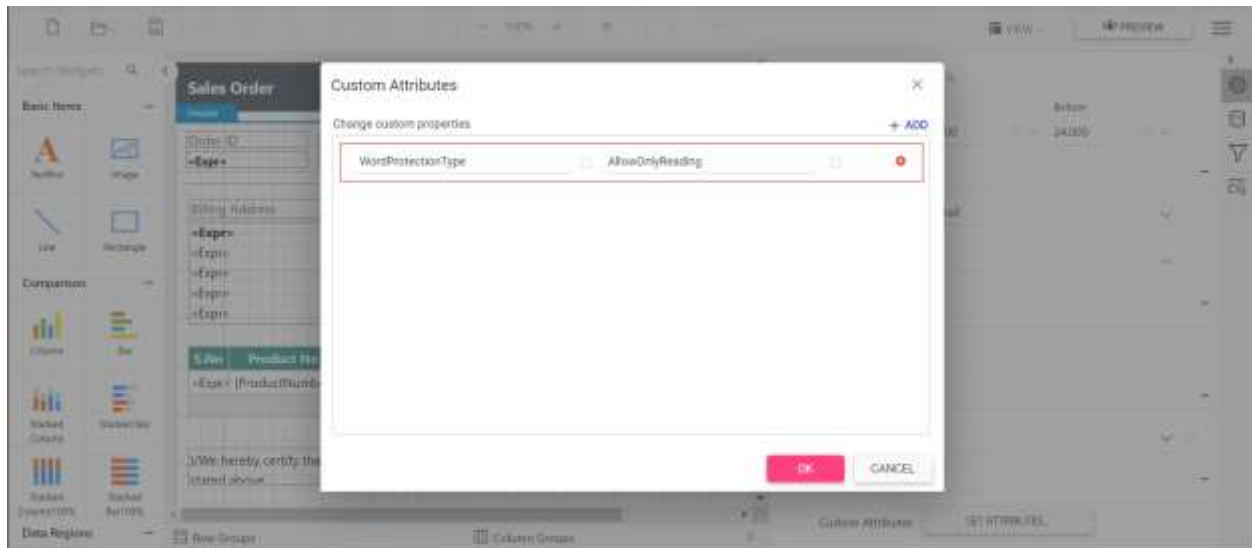


Protecting Word document from editing

The custom property `WordProtectionType` allows you to restrict a Word document from editing either by providing a password or without a password by using following types of protection.

- `AllowOnlyComments`- You can add or modify only the comments in the Word document.
- `AllowOnlyFormFields`- You can modify the form field values in the Word document.
- `AllowOnlyRevisions`- You can accept or reject the revisions in the Word document.
- `AllowOnlyReading`- You can only view the content in the Word document.
- `NoProtection`- You can access or edit the Word document contents as normally.

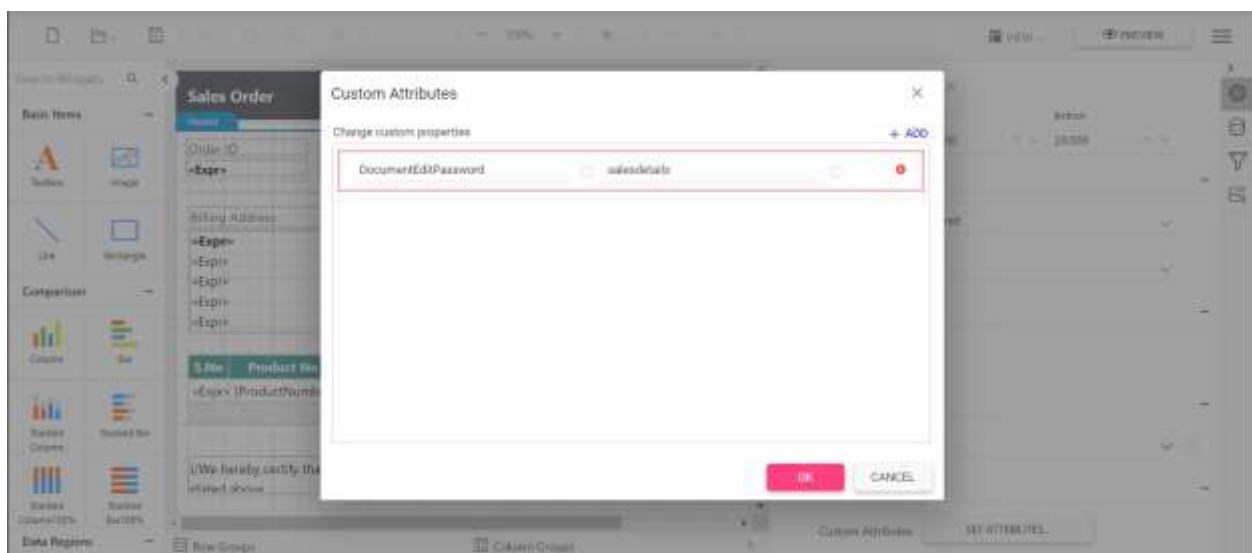
You can set the `WordProtectionType` custom property as shown below,



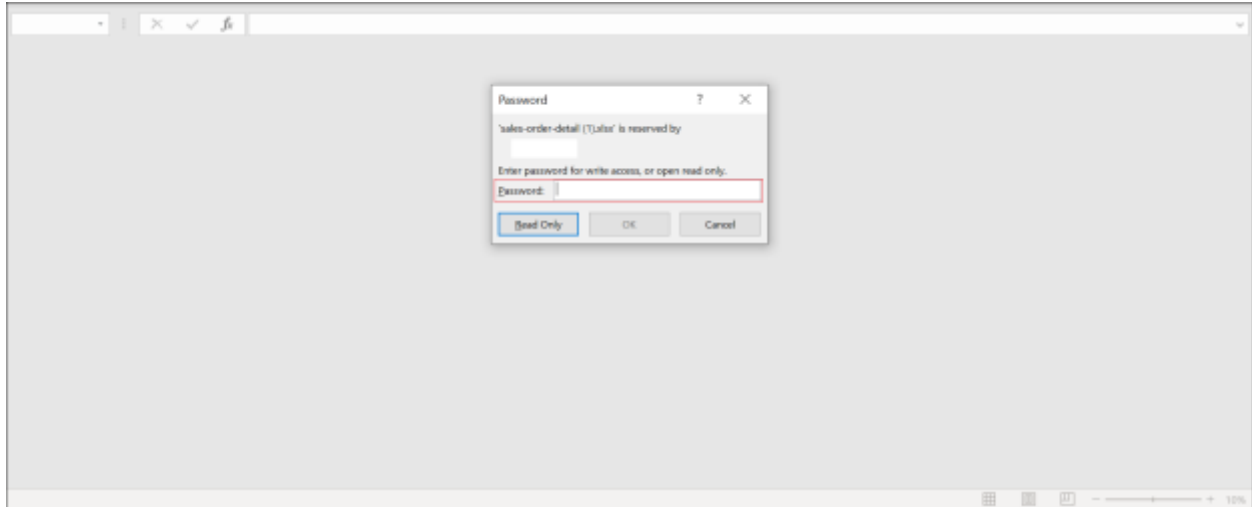
Set excel document edit password

The custom property `DocumentEditPassword` helps to allow specific users permission to modify the workbook data and save changes to the file in the excel document.

You can set the property value, as shown below,



open the excel document and see the below output.

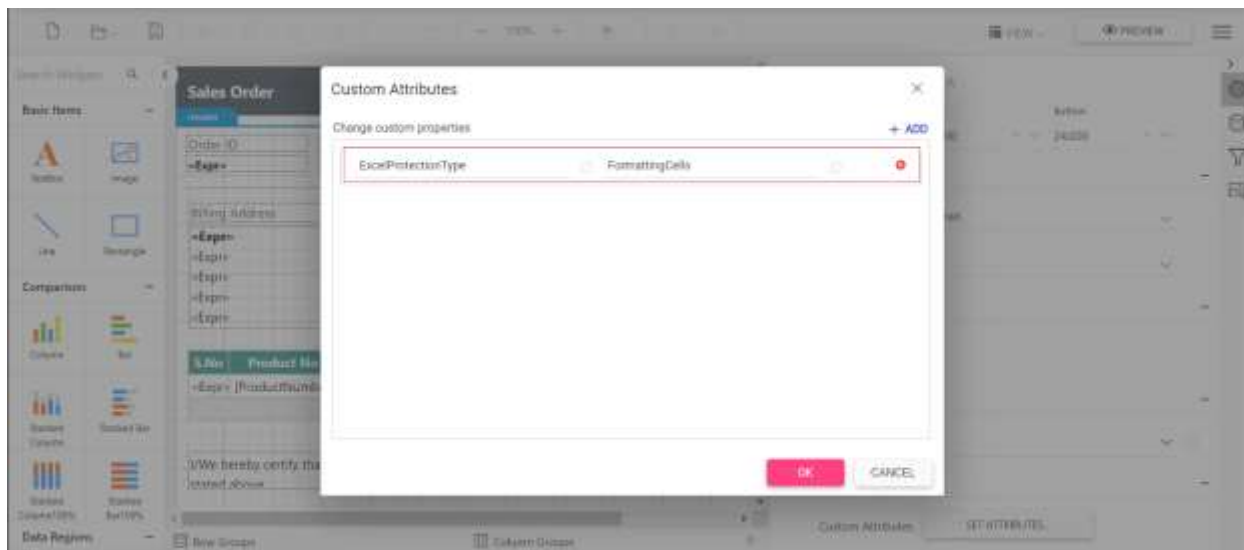


Set excel document protection

The custom property `ExcelProtectionType` allows you to protect the worksheet of excel document either by providing a password or without a password by using following types of protection.

- `None` - Represents no protection in excel sheet
- `Objects` - allows to protect shapes in excel sheet
- `Scenarios` - allows to protect scenarios.
- `FormattingCells` - allows the user to format any cell on a protected worksheet.
- `FormattingColumns` - allows the user to format any column on a protected worksheet.
- `FormattingRows` - allows the user to format any rows on a protected worksheet.
- `InsertingColumns` - allows the user to insert columns on the protected worksheet.
- `InsertingRows` - allows the user to insert rows on the protected worksheet.
- `InsertingHyperlinks` - allows the user to insert hyperlinks on the worksheet.
- `DeletingColumns` - allows the user to delete columns on the protected worksheet, where every cell in the column to be deleted is unlocked.
- `DeletingRows` - allows the user to delete rows on the protected worksheet, where every cell in the row to be deleted is unlocked.
- `LockedCells` - allows to protect locked cells.
- `Sorting` - allows the user to sort on the protected worksheet
- `Filtering` - allows the user to set filters on the protected worksheet. Users can change filter criteria but can not enable or disable an auto filter.
- `UsingPivotTables` - allows the user to use pivot table reports on the protected worksheet.
- `UnLockedCells` - allows to protect the user interface, but not macros.
- `Content` - allows to protect the contents in the excel sheet.
- `All` - allows to protect all type of protection.

You can set the `ExcelProtectionType` custom property as shown below,



Toolbar customization

You can hide the component toolbar to show customized user interface or to customize the toolbar icons and element's appearances using the templates and Report Viewer toolbar customization properties.

In this tutorial, the `sales-order-detail.rdl` report is used and it can be downloaded from [here](#). You can add the reports from the Bold Reports installation location. For more information, refer to [Samples and demos](#).

Hide toolbar items

To hide toolbar items, set the `toolbar-settings` property. The following code can be used to remove the parameter option from the toolbar and hide the parameter block.

Similarly, you can show or hide all other toolbar options with the help of [toolbarSettings.items](#) enum.

The following code example demonstrates how to hide the parameter block in the Report Viewer at client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings">
</bold-report-viewer>
`
```

The following code example demonstrates how to hide the parameter block in the Report Viewer at server side.

```
`csharp
public ActionResult Index()
{
    ViewBag.toolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
}
```

```
ViewBag.toolbarSettings.Items = BoldReports.ReportViewerEnums.ToolbarItems.All
& ~BoldReports.ReportViewerEnums.ToolbarItems.Parameters;
return View();
}
`
```

Enable stop option in toolbar

To enable stop option in toolbar, set the `toolbarSettings.Items` property to `BoldReports.ReportViewerEnums.ToolbarItems.All`. The following code can be used to enable stop option in toolbar.

The following code example demonstrates how to enable stop option in the Report Viewer toolbar at client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings">
</bold-report-viewer>
`
```

The following code example demonstrates how to enable stop option in the Report Viewer toolbar at server side.

```
`csharp
public ActionResult Index()
{
    ViewBag.toolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
    ViewBag.toolbarSettings.Items = BoldReports.ReportViewerEnums.ToolbarItems.All;
    return View();
}
`
```

Enable export setup option in toolbar

To enable export setup option in toolbar, set the `toolbarSettings.Items` property to `BoldReports.ReportViewerEnums.ToolbarItems.All`. The following code can be used to enable export setup option in toolbar.

The following code example demonstrates how to enable export setup option in the Report Viewer toolbar at client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings">
</bold-report-viewer>
`
```

,

The following code example demonstrates how to enable export setup option in the Report Viewer toolbar at server side.

```
`csharp
public ActionResult Index()
{
    ViewBag.toolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
    ViewBag.toolbarSettings.Items = BoldReports.ReportViewerEnums.ToolbarItems.All;
    return View();
}
```

,

Enable search text option in toolbar

To enable search text option in toolbar, set the `toolbarSettings.Items` property to `BoldReports.ReportViewerEnums.ToolbarItems.All`. The following code can be used to enable search text option in toolbar.

The following code example demonstrates how to enable search text option in the Report Viewer toolbar at client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings">
</bold-report-viewer>
```

,

The following code example demonstrates how to enable search text option in the Report Viewer toolbar at server side.

```
`csharp
public ActionResult Index()
{
    ViewBag.toolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
    ViewBag.toolbarSettings.Items = BoldReports.ReportViewerEnums.ToolbarItems.All;
    return View();
}
```

,

Hide toolbar

To hide the Report Viewer toolbar, set the `ShowToolbar` property to false.

The following code example demonstrates how to hide the toolbar in the Report Viewer at client side.

```
`html
```

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings">
</bold-report-viewer>
```

The following code example demonstrates how to hide the toolbar in the Report Viewer at server side.

```
`csharp
```

```
public ActionResult Index()
{
    ViewBag.toolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
    ViewBag.toolbarSettings.ShowToolbar = false;
    return View();
}
```

Decide or hide the export option

The Report Viewer provides the **ExportOptions** property to show or hide the default export types available in the component. The following code hides the HTML export type from the default export options.

The following code example demonstrates how to decide or hide the export option in the Report Viewer at client side.

```
`html
```

```
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
export-settings="ViewBag.exportSettings">
</bold-report-viewer>
```

The following code example demonstrates how to decide or hide the export option in the Report Viewer at server side.

```
`csharp
```

```
public ActionResult Index()
{
    ViewBag.exportSettings = new BoldReports.Models.ReportViewer.ExportSettings();
    ViewBag.exportSettings.ExportOptions = BoldReports.ReportViewerEnums.ExportOptions.All
    & ~BoldReports.ReportViewerEnums.ExportOptions.Html;
    return View();
}
```

,

Add custom items to the export drop-down

To add custom items to the export drop-down available in the Report Viewer toolbar, use the property **CustomItems** and provide the JSON array of collection input with the **Index**, **CssClass** name, and **Value** properties.

You can use the following codes to add custom items to the export drop-down list from controller and passing the data to view using **ViewBag**.

```
`csharp
public ActionResult Index()
{
    ExportSettings exportSettings = new ExportSettings();
    exportSettings.CustomItems = new List<CustomExportItem>();
    var exportItem1 = new CustomExportItem() { Index = 2, CssClass = "", Value = "Text File" };
    var exportItem2 = new CustomExportItem() { Index = 4, CssClass = "", Value = "DOT" };
    exportSettings.CustomItems.Add(exportItem1);
    exportSettings.CustomItems.Add(exportItem2);
    ViewBag.exportSettings = exportSettings;
    return View();
}
```

,

You can use the following codes to set an **export-settings** property at client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
export-settings="ViewBag.exportSettings">
</bold-report-viewer>
```

,

You can use the following codes to create an **export-item-click** event at client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
export-item-click="onExportItemClick">
</bold-report-viewer>
<script type="text/javascript">
//Export click event handler
function onExportItemClick(args) {
    alert('Action Triggered');
```



```
}
</script>
`
```

You can use the following codes to get `export-settings` properties on a dynamic object using `ViewBag.exportSettings` and invoke custom actions.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
export-settings="ViewBag.exportSettings" export-item-click="onExportItemClick">
</bold-report-viewer>
<script type="text/javascript">
//Export click event handler
function onExportItemClick(args) {
if (args.value === "Text File") {
//Implement the code to export report as Text
alert("Text File export option clicked");
} else if (args.value === "DOT") {
//Implement the code to export report as DOT
alert("DOT export option clicked");
}
}
</script>
`
```

Add custom toolbar item

You can add custom items to Report Viewer toolbar using the `toolbar-settings` property. You must register the `tool-bar-item-click` event to handle the newly added custom items action.

You can use the following codes to add custom toolbar item from controller and passing the data to view using `ViewBag`.

```
`csharp
public ActionResult Index()
{
    ToolbarSettings toolbarSettings = new ToolbarSettings();
    toolbarSettings.CustomItems = new List<CustomItem>();
    var customItem = new CustomItem()
    {
```

```

GroupIndex = 1,
Index = 1,
CssClass = "e-icon e-mail e-reportviewer-icon",
Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
Id = "E-Mail",
Tooltip = new ToolTip() { Header = "E-Mail", Content = "Send rendered report as mail attachment" }
};
toolbarSettings.CustomItems.Add(customItem);
ViewBag.toolbarSettings = toolbarSettings;
return View();
}
`

```

You can use the following codes to set an `toolbar-settings` property at client side.

```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings">
</bold-report-viewer>
`

```

You can use the following codes to create an `tool-bar-item-click` event at client side.

```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings" tool-bar-item-click="onToolBarItemClick">
</bold-report-viewer>
<script type="text/javascript">
function onToolBarItemClick(args) {
alert('Action Triggered');
}
</script>
`

```

Add custom item to exiting toolbar group

To add a custom item to existing toolbar group use the property `CustomGroups` in `ToolbarSettings` and provide the JSON array of collection input with the `GroupIndex`, `Items`, `Type`, `CssClass` name, and `Tooltip` properties as given in following code snippet.

You can use the following codes to add custom item to exiting toolbar group from controller and passing the data to view using `ViewBag`.

```
`csharp
public ActionResult Index()
{
    ToolbarSettings toolbarSettings = new ToolbarSettings();
    var groupItem = new List<CustomItem>();
    groupItem.Add(new CustomItem()
    {
        CssClass = "e-icon e-mail e-reportviewer-icon CustomGroup",
        Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
        Id = "CustomGroup",
        Tooltip = new ToolTip() { Header = "CustomGroup", Content = "toolbargroups" }
    });
    groupItem.Add(new CustomItem()
    {
        CssClass = "e-icon e-mail e-reportviewer-icon subCustomGroup",
        Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
        Id = "subCustomGroup",
        Tooltip = new ToolTip() { Header = "subCustomGroup", Content = "subtoolbargroups" }
    });
    toolbarSettings.CustomGroups.Add(new CustomGroup() { Items = groupItem, GroupIndex = 4 });
    ViewBag.toolbarSettings = toolbarSettings;
    return View();
}
```

You can use the following codes to set an **toolbar-settings** property at client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings">
</bold-report-viewer>
```

You can use the following codes to create an **tool-bar-item-click** event at client side.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
tool-bar-item-click="onToolBarItemClick">
```

```

</bold-report-viewer>
<script type="text/javascript">
function onToolBarItemClick(args) {
alert('Action Triggered');
}
</script>
`

```

You can use the following codes to get `toolbar-settings` properties on a dynamic object using `ViewBag.toolbarSettings` and invoke custom actions.

```

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings" tool-bar-item-click="onToolBarItemClick">
</bold-report-viewer>
<script type="text/javascript">
function onToolBarItemClick(args) {
if (args.value === "CustomGroup") {
//Implement the code to CustomGroup toolbar option
alert("CustomGroup toolbar option clicked");
}
if (args.value === "subCustomGroup") {
//Implement the code to subCustomGroup toolbar option
alert("SubCustomGroup toolbar option clicked");
}
}
</script>
`

```

Add new toolbar group

To add new toolbar group and custom items to it, use the property `CustomItems` in `ToolbarSettings` and provide the JSON array of collection input with the `GroupIndex`, `Index` properties. The `CustomItem` must have the properties `Type`, `CssClass` and `Tooltip` as given in following code snippet.

You can use the following codes to add email button from controller and passing the data to view using `ViewBag`.

```

`csharp
public ActionResult Index()
{

```

```

ToolbarSettings toolbarSettings = new ToolbarSettings();
toolbarSettings.CustomItems = new List<CustomItem>();
var customItem = new CustomItem()
{
    GroupIndex = 1,
    Index = 1,
    CssClass = "e-icon e-mail e-reportviewer-icon",
    Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
    Id = "E-Mail",
    Tooltip = new ToolTip() { Header = "E-Mail", Content = "Send rendered report as mail attachment" }
};
toolbarSettings.CustomItems.Add(customItem);
ViewBag.toolbarSettings = toolbarSettings;
return View();
}
`

```

You can use the following codes to set an `toolbar-settings` property on a dynamic object using `ViewBag.toolbarSettings` at client side.

```

`html

<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" processing-mode="Remote"
toolbar-settings="ViewBag.toolbarSettings">

</bold-report-viewer>
`

```

Localization of Bold Reports ASP.NET Core Report Viewer

Localization of ASP.NET Core Report Viewer allows you to localize the static text such as tooltip, parameter block, and dialog text based on a specific culture. To render the static text with specific culture, refer to the following corresponding culture script files and set culture name to the `locale` property of the Report Viewer.

- `ej.localtexts.fr-FR.min.js`
- `ej.culture.fr-FR.min.js`

Refer this [CDN links for Localization and Culture](#) to get the Localization and Culture scripts for available Culture Code.

1. Refer to the `ej.localtexts.fr-FR.min.js` script file from the CDN link using the following code.

```
`html
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localtexts.fr-FR.min.js"></script>
`
```

2. Refer to the `ej.localtexts.fr-FR.min.js` script file from the CDN link using the following code.

```
`html
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.fr-FR.min.js"></script>
`
```

3. To render the localization Report Viewer, use the following code snippet.
 - The following code example illustrates how to localize Report Viewer in the index page.

```
`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" locale="fr-FR"></bold-report-viewer>
`
```

- The following code example illustrates how to localize Report Viewer in the layout page.

```
`html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Render Report Viewer in French localization</title>
<link
href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css" rel="stylesheet"
/>
<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localtexts.fr-FR.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.fr-FR.min.js"></script>
`
```

```

</head>
<body style="height:100%;width:100%;padding:0;">
<div class="container body-content" style="height:100%;width:100%;">
@RenderBody()
</div>
@RenderSection("scripts", required: false)
</body>
</html>

```

Responsive layout rendering of ASP.NET Core Report Viewer

Report Viewer will adaptively render itself with optimal user interfaces for phone, tablet, or desktop form factors. This helps your application to scale elegantly on all form factors with ease. You can enable responsive layout rendering in Report Viewer by setting `isResponsive` property to true.

```

`html
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-
url="/api/ReportViewer" is-responsive="true">
</bold-report-viewer>

```

Normal layout

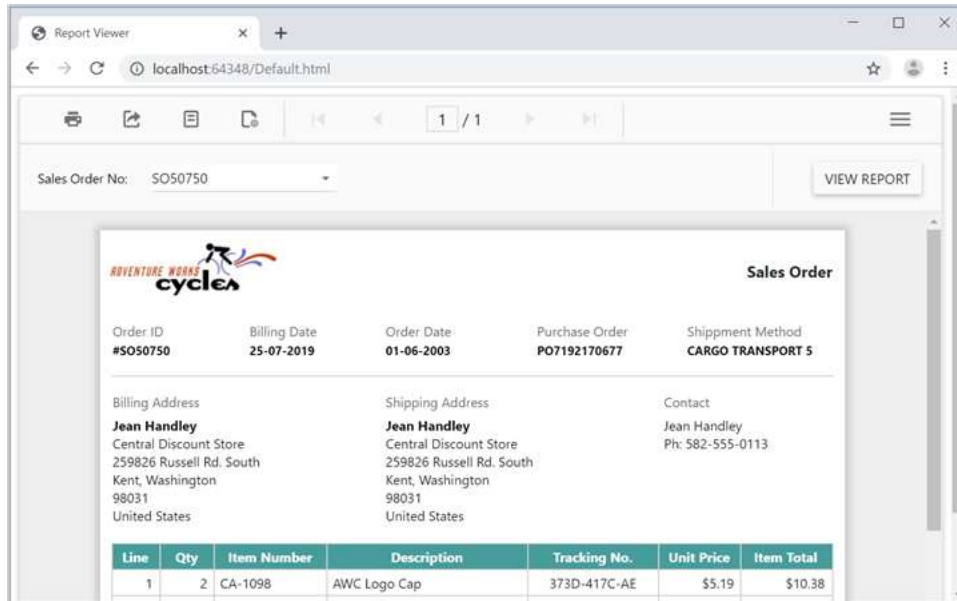
The following output shows the normal layout rendering of Report Viewer tool bar items.

The screenshot shows the Report Viewer interface in a desktop browser window. The report is titled 'Sales Order' and is for 'Adventure Works Cycles'. It includes a table of order details and a table of items.

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.84	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	BK-M688-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46

Responsive layout

The following output shows the responsive layout rendering of Report Viewer tool bar items.



Limitations

RDL specification

The Report Viewer control does not support RDL specification for SQL Server 2000 and SQL Server 2005.

Report layout

- In the Tablix cell split layout process, the entire cell moves to the next page to display the complete cell items, when the table cell width value exceeds the page width.

Expressions

The object function and VB function do not have complete support.

SSRS

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the server. If the report has any data source that uses credentials to connect with the database, then you should specify the data source credentials for each report data source to establish database connection.

Image format

PNG and **JPEG** format images only supports in the ASP.NET Core Report Viewer for PDF export. This limitation is from our **Syncfusion.Pdf.Net.Core** supports.

Samples and Demos

Browse and explore the ready-to-use RDL, RDLC reports, samples, online, and offline demos.

Locally installed reports

You can obtain the sample RDL and RDLC files from the Bold Reports installed location

%localappdata%\Bold Reports\Embedded Reporting\Samples\Common\Data\ReportTemplate.

Offline demos

The offline samples are provided in the Bold Reporting Tools setup. For more details, refer to the [Bold Reporting Tools sample deployment](#).

Online demos

You can view the ASP.NET Core Report Viewer online demo samples from [here](#).

GitHub demo samples

Click [here](#) to view the GitHub Report Viewer demo samples.

Migrate Report Viewer application

In our Bold Reports new assemblies are introduced for both client and server-side to resolve the compatibility problem between Essential Studio Report Viewer versions. It has changes in both Web API service and client-side scripts.

This section provides step-by-step instructions for migrating Report Viewer from Syncfusion Essential Studio release version to Bold Reports version of ASP.NET Core Report Viewer application:

Client-side migration

1. To uninstall NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search the **Syncfusion.EJ.ASPNET.Core** NuGet Package and uninstall from your application.
2. Search for **BoldReports.AspNet.Core** NuGet package, and install to your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Scripts and CSS references

1. Remove the following scripts and CSS references from the Report Viewer **\Views\Shared_Layout.cshtml** page:
 - o ej.web.all.min.css
 - o ej.web.all.min.js
2. Add the listed references in the same order given in above list. You can replace the following code in your **\Views\Shared_Layout.cshtml** page tag.

```
`html
```

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
```

```
<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
```

```
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
```

```
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
`
```

3. The component prefix has been changed from `ej` to `sf`.
4. Open the `\Views\Shared_Layout.cshtml` page.
5. Remove the following code in your `_Layout.cshtml` page.
6. Replace the following code in your `_Layout.cshtml` page.

```
`html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>@ViewData["Title"] - ReportViewerDemo</title>
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
</head>
<body>
<div style="min-height: 600px; width: 100%;">
@RenderBody()
</div>
@RenderSection("Scripts", required: false)
<!-- Bold Reports script manager -->
<bold-script-manager></bold-script-manager>
</body>
</html>
`
```

Adding data visualization scripts

To render the report with data visualization components such as chart, gauge, and map items, add scripts of the visualization element. The following table shows the script reference that needs to be added in Report Viewer page for data visualization elements.

Visualization item | Script file

Gauge | ej2-base.min.js, ej2-data.min.js, ej2-pdf-export.min.js, ej2-svg-base.min.js, ej2-lineargauge.min.js, ej2-circulargauge.min.js|

Map | ej2-maps.min.js

Chart | ej.chart.min.js

To render the chart report item, add chart control script `ej.chart.min.js` before the `bold.report-viewer.min.js` reference in the `\Views\Shared_Layout.cshtml` page as demonstrated in the following code sample.

```
`html
<link
href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css" rel="stylesheet"
/>

<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>

<!-- Report Viewer component script-->

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>

<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>

<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
`
```

The following code can be used to render the chart, gauge, and map report items in Report Viewer.

```
`html
<link
href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css" rel="stylesheet"
/>

<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>

<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->

<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>

<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>

<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
`
```

```

<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>

```

Tag helper

1. Remove following tag helper within the `_ViewImports.cshtml` page.

```

`js
@using Syncfusion.JavaScript
@addTagHelper *, Syncfusion.EJ

```

2. Add following tag helper within the `_ViewImports.cshtml` page.

```

`js
@using BoldReports.TagHelpers
@addTagHelper *, BoldReports.AspNet.Core

```

Control initialization

1. The component prefix has been changed from `ej-report-viewer` to `bold-report-viewer`.
2. Open the Report Viewer CSHTML page.
3. Replace the component tag `ej-report-viewer` to `bold-report-viewer`.

```

`js

```

```
<bold-report-viewer id="viewer"></bold-report-viewer>
```

```
,
```

If your application contains Report Viewer initialization in multiple pages then replace in all the pages.

Server-side migration

1. To uninstall NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search the **Syncfusion.EJ.ReportViewer.ASPNET.Core** NuGet Package and uninstall from your application.
2. Search for **BoldReports.Net.Core** NuGet package, and install to your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Web API Controller

1. The **IReportController** interface is moved to **BoldReports.Web.ReportViewer**. Open the Report Viewer Web API Controller file and remove the following using statement.

```
`csharp
using Syncfusion.EJ.ReportViewer;
,
```

2. Add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
,
```

3. Your application is successfully upgraded to the latest version of Report Viewer, and you can run the application with new assemblies.

NuGet Packages for ASP.NET Core

Refer to the following steps to configure Bold Reporting NuGet packages for ASP.NET Core application.

Configure NuGet feed URL

Online NuGet feed URL

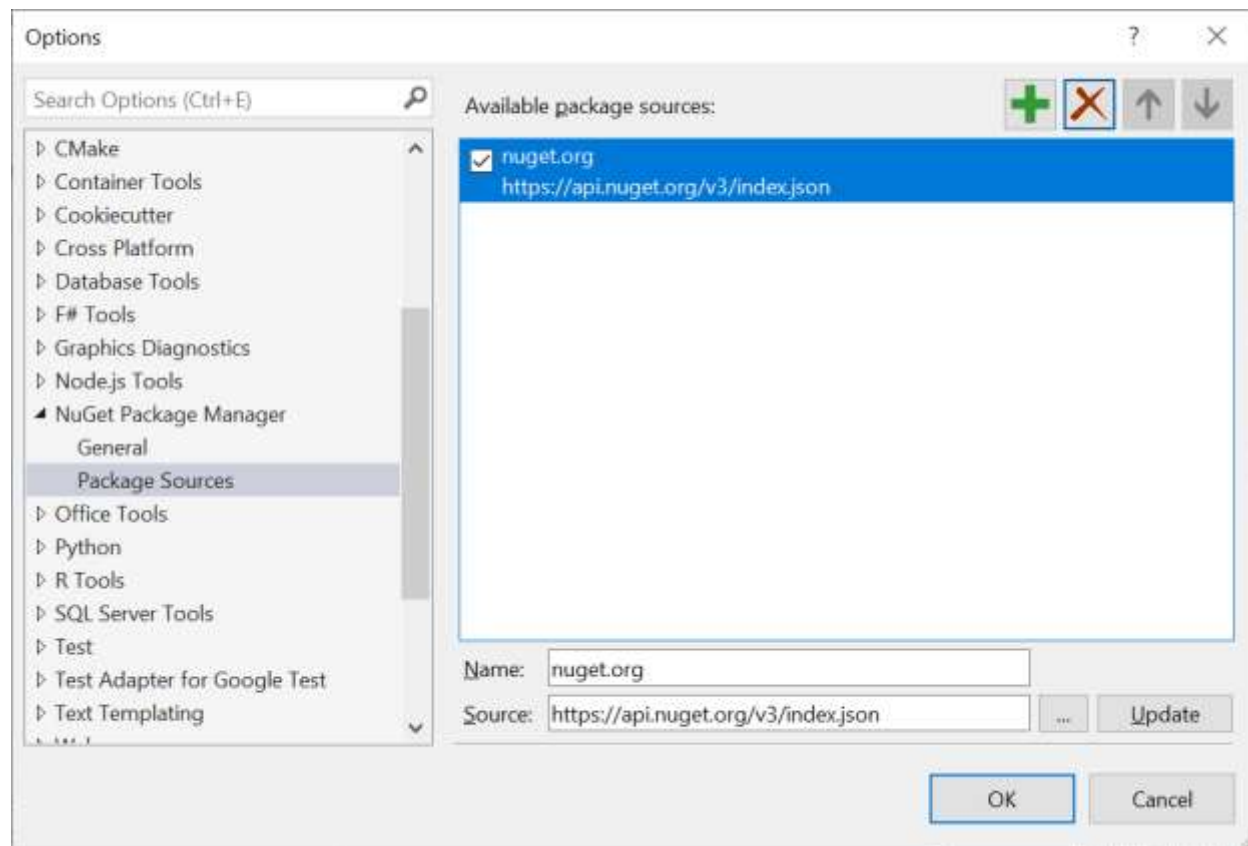
The Bold Reporting NuGet packages are published in **Nuget.org**. To configure the online packages, use the following steps:

1. Open Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager** and select **Package Sources**.

- Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: NuGet.org

Source: <https://api.nuget.org/v3/index.json>



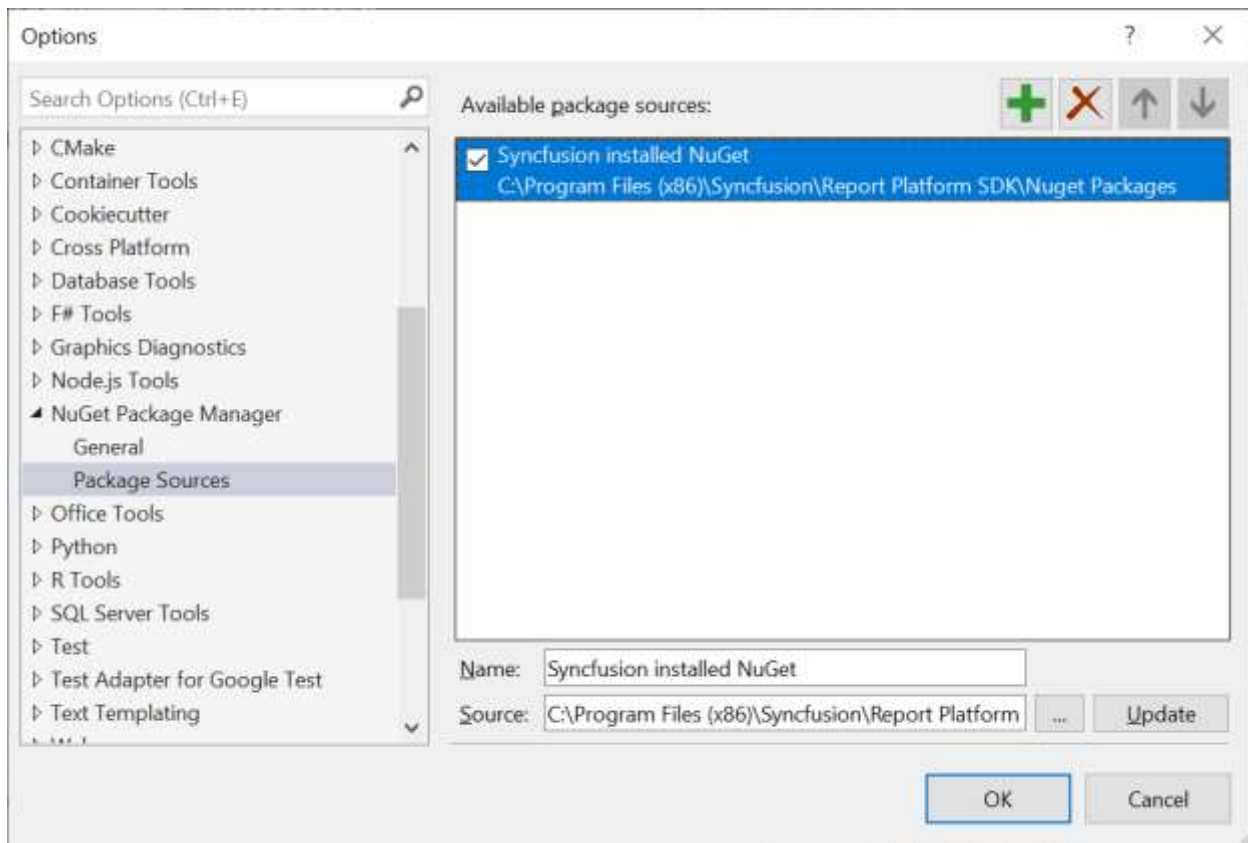
Offline NuGet feed URL

Bold Reporting NuGet packages are shipped into our Bold Reporting Tools build. To configure the packages from Bold Reports installed location, use the following steps:

- Open your Visual Studio application.
- On the **Tools** menu, select **Options**.
- Expand the **NuGet Package Manager**, and then select **Package Sources**.
- Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: Bold Reports installed NuGet

Source: {System Drive}:\Program Files (x86)\Bold Reports\Reporting Tools\Nuget Packages.



The system drive varies based on the installed location in your machine.

Installing NuGet packages

Install using NuGet Package Manager

The NuGet Package Manager can be used to search and install NuGet packages in Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution**. Alternatively, right-click the project/solution in Solution Explorer tab, and choose **Manage NuGet Packages**.
2. By default, the **NuGet.org** package is selected in the **Package source** drop-down. Select package source, search for the packages (**BoldReports.Net.Core** or **BoldReports.AspNet.Core**), and then click **Install** button.

Install using Package Manager Console

To install the Reporting component using the Package Manager Console as NuGet packages,

1. On the **Tools** menu, select **NuGet Package Manager**, and then click **Package Manager Console**.
2. Run the following NuGet installation commands:

```
`cmd
```

install specified package in default project

Install-Package <Package Name>

install specified package in default project with specified package source

Install-Package <Package Name> -Source <Source Location>

install specified package in specified project

Install-Package <Package Name> -ProjectName <Project Name>

,

For example:

`cmd

install specified package in default project

Install-Package BoldReports.Net.Core

install specified package in default project with specified Package Source

Install-Package BoldReports.Net.Core -Source "https://api.nuget.org/v3/index.json"

install specified package in specified project

Install-Package BoldReports.Net.Core -ProjectName BoldReportsApplication

,

Upgrading NuGet packages

Upgrading using NuGet Package Manager

NuGet packages can be updated to their specific version or latest version available in the Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution....** Alternatively, right-click the project/solution in the Solution Explorer tab, and then choose **Manage NuGet Packages**.
2. Select the **Updates** tab to see the packages available for update from the desired package sources, select the required packages and specific version from the drop-down, and then click the **Update** button.

Upgrading using Package Manger Console

To update the installed Bold Reporting NuGet packages using the Package Manager Console:

1. On the **Tools** menu, select **NuGet Package Manager**, and then select **Package Manager Console**.
2. Run the following NuGet installation commands:

`cmd

Update specific NuGet package in default project

Update-Package <Package Name>

Update all the packages in default project

Update-Package

Update specified package in default project with specified package source

Update-Package <Package Name> -Source <Source Location>

Update specified package in specified project

Update-Package <Package Name> - ProjectName <Project Name>

,

For example:

`cmd

Update specified Bold Reporting NuGet package

Update-Package BoldReports.Net.Core

Update specified package in default project with specified Package Source

Update-Package BoldReports.Net.Core -Source "https://api.nuget.org/v3/index.json"

Update specified package in specified project

Update-Package BoldReports.Net.Core -ProjectName BoldReportsApplication

,

Upgrading using NuGet CLI

Using the NuGet CLI, all the NuGet packages in the project can be updated to the available latest version:

1. Download the latest [NuGet CLI](#).

To update the existing `nuget.exe` to latest version use the following command:

`cmd

nuget update -self

,

2. Open the downloaded executable location in the command window. Run the following "update commands" to update the Bold Reporting NuGet packages.

```
`cmd
```

update all NuGet packages from config file

```
nuget update <configPath> [options]
```

update all NuGet packages from specified Packages Source

```
nuget update -Source <Source Location> [optional]
```

```
,
```

`configPath` is optional. It identifies the `package.config` or solutions file lists the packages utilized in the project.

For example:

```
`cmd
```

Update all NuGet packages from config file

```
nuget update "C:\Users\BoldReportsApplication\package.config"
```

Update all NuGet packages from specified Packages Source

```
nuget update -Source "https://api.nuget.org/v3/index.json"
```

```
,
```

The Update command is not working as expected in Mono (Mac and Linux) and projects using PackageReference format.

Report page layouting and rendering

This section describes about the report page layouting and rendering options in Bold Reports ASP.NET Core Report Viewer.

- [Measurement options](#)

Report measurement options

Report measurement options used to improvise the appearance of report in rendering and export outputs.

Improve text rendering in view and export

ASP.NET Core Report Viewer provides the property `MeasureTextOption` that helps to include following improvements in text rendering,

- Produces identical text sizes in view and export output.
- Improves rendering performance.
- Eliminates text overlaps.
- Allows to measure text with user-defined or custom fonts.

To enable the above benefits, set the `MeasureTextOption` property value to `PdfFont` in `OnInitReportOptions` of Report Viewer Web API controller as shown in below code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.MeasureTextOption = MeasureTextOptions.PdfFont;
}
`
```

The above measurement option uses the custom fonts added in the `reportOption.ReportModel.PDFOptions.Fonts` collection.

Improve report items layouting and pagination

When the `RoundLayoutMeasures` property is false, all non-integral values that are calculated during the report processing are rounded to whole pixel values. It provides following improvements,

- Report text rendered without cuts.
- Eliminates extra blank pages.
- Removes item and text overlaps.
- Eliminates the blur semi-transparent edges that are produced by anti-aliasing.
- Produces identical look in report view and export output.

To enable the above benefits, set the `RoundLayoutMeasures` property value to `false` in `OnInitReportOptions` of Report Viewer Web API controller as shown in below code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.RoundLayoutMeasures = false;
}
`
```

How to Create RDL/RDLC Report

The following sections explain about how to create a new RDL/RDLC report using Bold Report Designer, Microsoft Report Builder and Visual Studio Report Server project template.

- [Create a RDL report](#)
- [Create a RDLC report](#)
- [Change the exporting document file name based on the parameter](#)
- [Change the connection string datasource dynamically](#)
- [Disable the vertical scrollbar in parameter panel](#)
- [Generate RDL and RDLC reports programmatically](#)
- [How to pass multiple values using custom data?](#)

- [How to resolve InvalidOperationException when creating a new application?](#)
- [How to load the report from database?](#)
- [How to clear cache when closing the Report Viewer?](#)

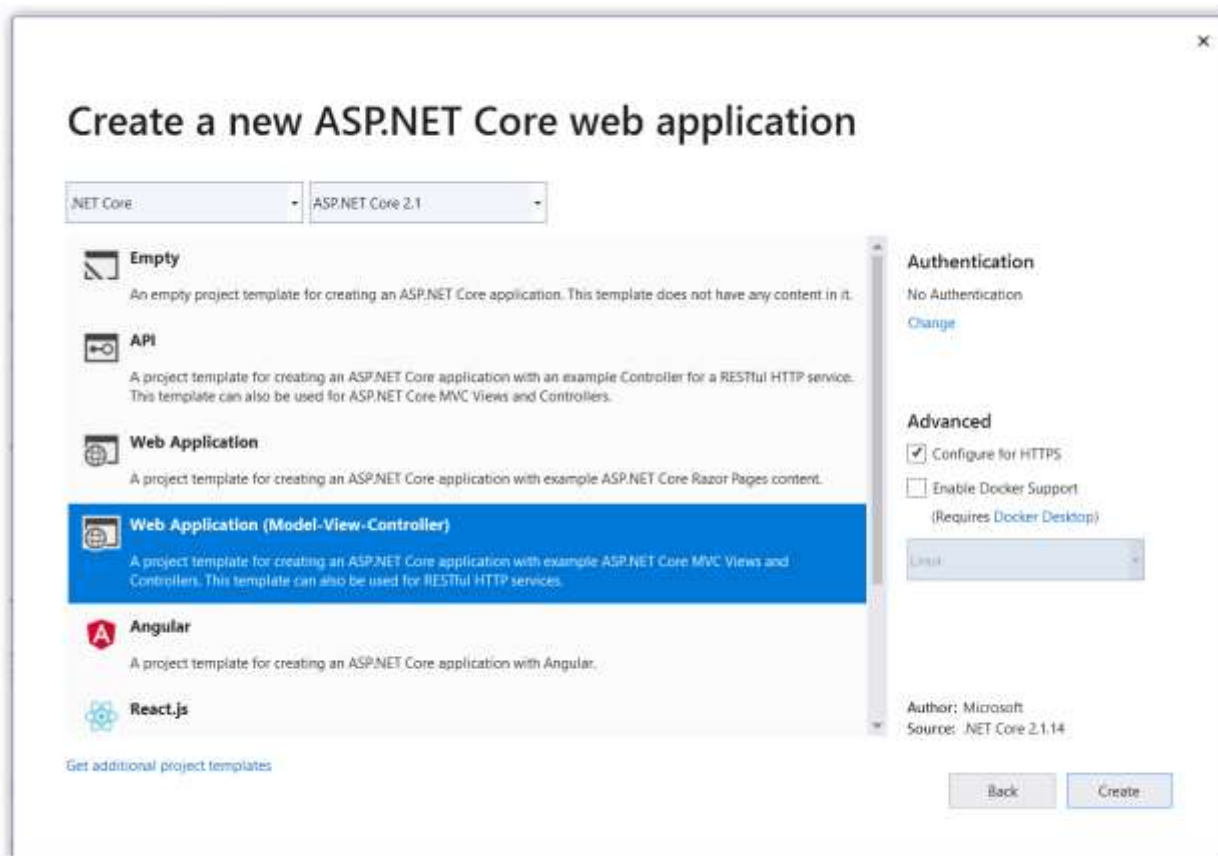
Display ssrs rdl report in ASP.NET Core 2.1 application using Report Viewer

Create your first ASP.NET Core reporting Web application to display already created SSRS RDL report in Bold Reports ASP.NET Core Report Viewer without using a Report Server using this step-by-step instructions.

To create your first application in other .NET Core frameworks, refer to the documentation [ASP .NET Core 3.1](#), [.NET 5.0](#) or [.NET 6.0](#)

Create ASP.NET Core application

1. Start Visual Studio 2019 and click **Create new project**.
2. Choose **ASP.NET Core Web Application**, and then click **Next**.
3. Change the project name, and then click **Create**.
4. In the dropdown with the **ASP.NET Core version**, choose **ASP.NET Core 2.1**.
5. Select the **Web Application (Model-View-Controller)** template, then click **Create**.



If you need to use Bold Reports with ASP.NET Core on Linux or macOS, then refer to this [Can Bold Reports be used with ASP.NET Core on Linux and macOS](#) section.

List of dependency libraries

1. In the Solution Explorer tab right-click the project or solution , and choose **Manage NuGet Packages**. Alternatively, go to **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.AspNet.Core**, **BoldReports.Net.Core** and **System.Data.SqlClient** packages, and install them in your Core application. The following table provides details about the packages and their usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Exports the report to PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the **Syncfusion.Pdf.Net.Core** , **Syncfusion.DocIO.Net.Core**, and **Syncfusion.XlsIO.Net.Core** packages.

Syncfusion.Pdf.Net.Core | Exports the report to a PDF.

Syncfusion.DocIO.Net.Core | Exports the report to a Word.

Syncfusion.XlsIO.Net.Core | Exports the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serializes and deserialize data for the Report Viewer. It is a mandatory package for Report Viewer, and the package version should be 10.0.1 or higher.

Refer scripts and CSS

Directly refer all the required scripts and style sheets from [CDN](#) links.

1. The following scripts and style sheets are mandatorily required to use the Report Viewer.
 - o **bold.reports.all.min.css**
 - o **jquery-1.10.2.min.js**
 - o **ej2-maps.min.js** - Renders the map item. Add this script only if your report contains the map report item.
 - o **ej2-base.min.js**, **ej2-data.min.js**, **ej2-pdf-export.min.js**, **ej2-svg-base.min.js**, **ej2-lineargauge.min.js** and **ej2-circulargauge.min.js** - Render the gauge item. Add these scripts only if your report contains the gauge report item.
 - o **bold.reports.common.min.js**
 - o **bold.reports.widgets.min.js**
 - o **ej.chart.min.js** - Renders the chart item. Add this script, only if your report contains the chart report item.
 - o **bold.report-viewer.min.js**
2. Open the **\Views\Shared_Layout.cshtml** page.
3. Replace the following code in your **\Views\Shared_Layout.cshtml** page tag.

```
`html
```

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
```

```

<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
`

```

To learn more about rendering a report with data visualization report items, refer to the [how to render data visualization report items](#) section.

The Report Viewer scripts and styles can be added into your application by installing the **BoldReports.JavaScript** online nuget package.

Tag helper

It is necessary to define the following tag helper within the `_ViewImports.cshtml` page to initialize the Report Viewer component with the tag helper support.

```

`js
@using BoldReports.TagHelpers
@addTagHelper *, BoldReports.AspNet.Core
`

```

Configure Script Manager

Open the `~/Views/Shared/_Layout.cshtml` page and add the reporting Script Manager at the end of `<body>` element as in the following code sample.

```
`html
<body>
<div style="min-height: 600px;width: 100%;">
@RenderBody()
</div>
@RenderSection("Scripts", required: false)
<!-- Bold Reports script manager -->
<bold-script-manager></bold-script-manager>
</body>
`
```

Initialize Report Viewer

1. Open the `Index.cshtml` page.
2. Remove the existing codes and add the following code.

```
`html
<bold-report-viewer id="viewer"></bold-report-viewer>
`
```

Add already created reports

1. Create a folder `Resources` into the `wwwroot` folder in your application to store the RDL reports.
2. Download the `sales-order-detail.rdl` from [here](#). For more sample reports, refer to [samples and demos](#) section.
3. Extract the compressed file and paste the `sales-order-detail.rdl` to `Resources` folder.

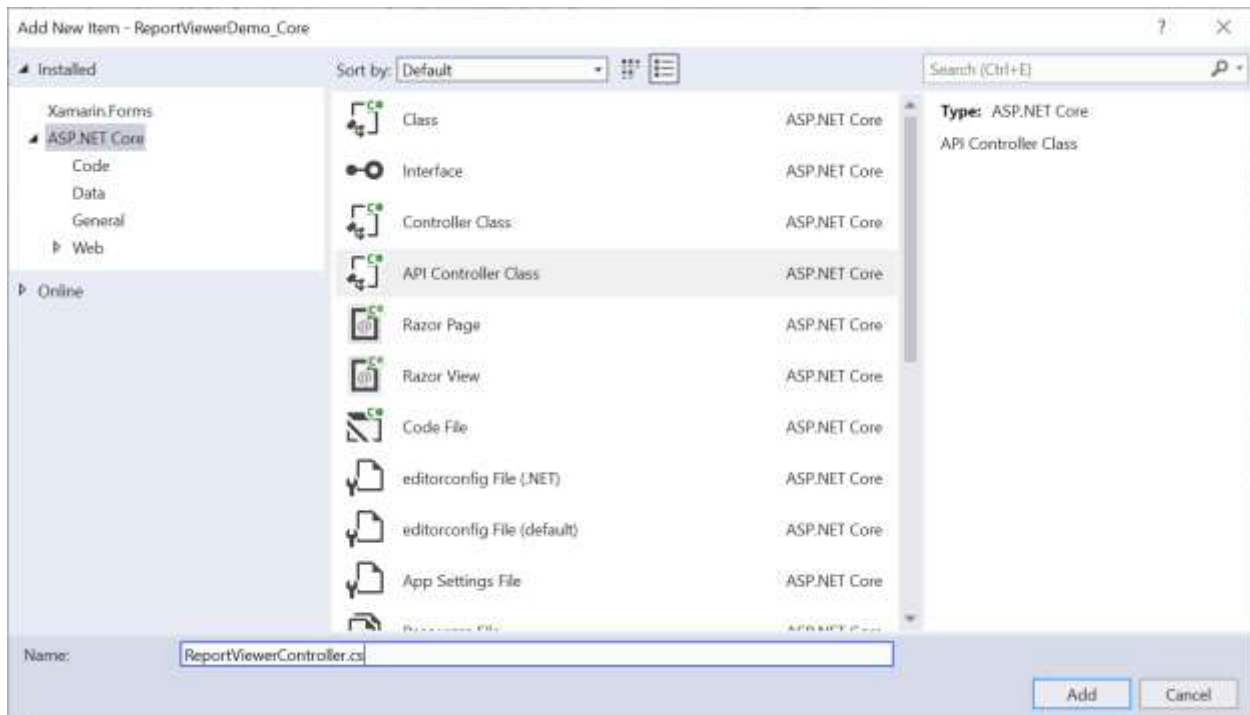
The Report Viewer is only for rendering the reports. Refer to the [create RDL report](#) section to create a report.

Configure Web API

The ASP.NET Core Report Viewer requires a Web API service to process the RDL, RDLC, and SSRS report files.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller** class and name it as `ReportViewerController.cs`



3. Click **Add**.

While adding API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using System.IO;
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and then implement its methods.

It is required for processing the reports and for handling request from the Report Viewer.

6. Create local variables inside the **ReportViewerController** class.

```
`csharp
//Report Viewer requires a memory cache to store the information of consecutive client request and
have the rendered report viewer information in server
private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
// IHostingEnvironment used to get the report stream from application wwwroot\Resources folder..
private Microsoft.AspNetCore.Hosting.IHostingEnvironment _hostingEnvironment;
```


7. Load the report as stream in `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
    reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}
`
```

You cannot load the report stored in application with path information in ASP.NET Core service.

8. Set the `Route` attribute for `ReportViewerController`.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    ...
}
`
```

9. You can replace the template code with the following code.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
```

```
{
// Report viewer requires a memory cache to store the information of consecutive client request and
// have the rendered Report Viewer information in server.
private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
// IHostingEnvironment used with sample to get the application data from wwwroot.
private Microsoft.AspNetCore.Hosting.IHostingEnvironment _hostingEnvironment;
// Post action to process the report from server based json parameters and send the result back to the
client.
public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
Microsoft.AspNetCore.Hosting.IHostingEnvironment hostingEnvironment)
{
_cache = memoryCache;
_hostingEnvironment = hostingEnvironment;
}
// Post action to process the report from server based json parameters and send the result back to the
client.
[HttpPost]
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
{
//Contains helper methods that help to process a Post or Get request from the Report Viewer control
and return the response to the Report Viewer control
return ReportHelper.ProcessReport(jsonArray, this, this._cache);
}
// Method will be called to initialize the report information to load the report with ReportHelper for
processing.
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
string basePath = _hostingEnvironment.WebRootPath;
// Here, we have loaded the sales-order-detail.rdl report from application the folder
wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
FileStream inputStream = new FileStream(basePath + @"\Resources\" +
reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
MemoryStream reportStream = new MemoryStream();
inputStream.CopyTo(reportStream);
}
```

```

reportStream.Position = 0;
inputStream.Close();
reportOption.ReportModel.Stream = reportStream;
}

// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}

//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
return ReportHelper.GetResource(resource, this, _cache);
}

[HttpPost]
public object PostFormReportAction()
{
return ReportHelper.ProcessReport(null, this, _cache);
}
}
`

```

Enable cross-origin requests

Browser security prevents the Report Viewer from making requests to your Web API Service when both server-side and client-side requests run in different domains. To allow access to your Web API service from a different domain, enable the cross-origin requests.

1. Open `Startup.cs` file.
2. Call `AddCors` in `Startup.ConfigureServices` to add CORS services to the app's service container as in below code.

```

`csharp
public void ConfigureServices(IServiceCollection services)

```

Display ssrs rdl report in ASP.NET Core 2.1 application using Report Viewer **Set** report path and service URL

```
{
services.AddMvc();
services.AddCors(o => o.AddPolicy("AllowAllOrigins", builder =>
{
builder.AllowAnyOrigin()
.AllowAnyMethod()
.AllowAnyHeader();
}));
}
```

To specify the CORS policy for `ReportViewerController` add the `[EnableCors]` attribute.

```
`csharp
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
....
....
}
```

If you are looking to load the report directly from SQL Server Reporting Services (SSRS), then you can skip the following steps and move to [SSRS Report](#) section.

Set report path and service URL

To render the reports available in the application, of the Report Viewer. You can replace the following code in your Report Viewer page.

1. Open the `Index.cshtml` page.
2. Set the `report-path` and `report-service-url` properties as in below.

```
`html
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-
url="/api/ReportViewer"></bold-report-viewer>
```


The report path property is set to the RDL report that is added to the project `Resources` folder.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

1 / 1 75%

Sales Order No:



Sales Order

Order ID #SO50750	Billing Date 22-04-2019	Order Date 01-06-2003	Purchase Order PO7192170677	Shipment Method CARGO TRANSPORT 5
-----------------------------	-----------------------------------	---------------------------------	---------------------------------------	---

Billing Address Jean Handley Central Discount Store 259826 Russell Rd. South Kent, Washington 98031 United States	Shipping Address Jean Handley Central Discount Store 259826 Russell Rd. South Kent, Washington 98031 United States	Contact Jean Handley Ph: 562-555-0113
---	--	---

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.04	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	8K-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	8K-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

I/We hereby certify that the information on this invoice is true and correct and that the contents of this shipment are as stated above.

Signature of Authorized Person

See Also

[Create your first app in ASP.Net Core 3.1 version](#)

[Create your first app in NET 5.0 version](#)

[Render report with data visualization report items](#)

[Create RDLC report](#)

[Render RDLC reports](#)

[Preview report in print mode](#)

[Set data source credential for shared data sources](#)

[Change data source connection string](#)

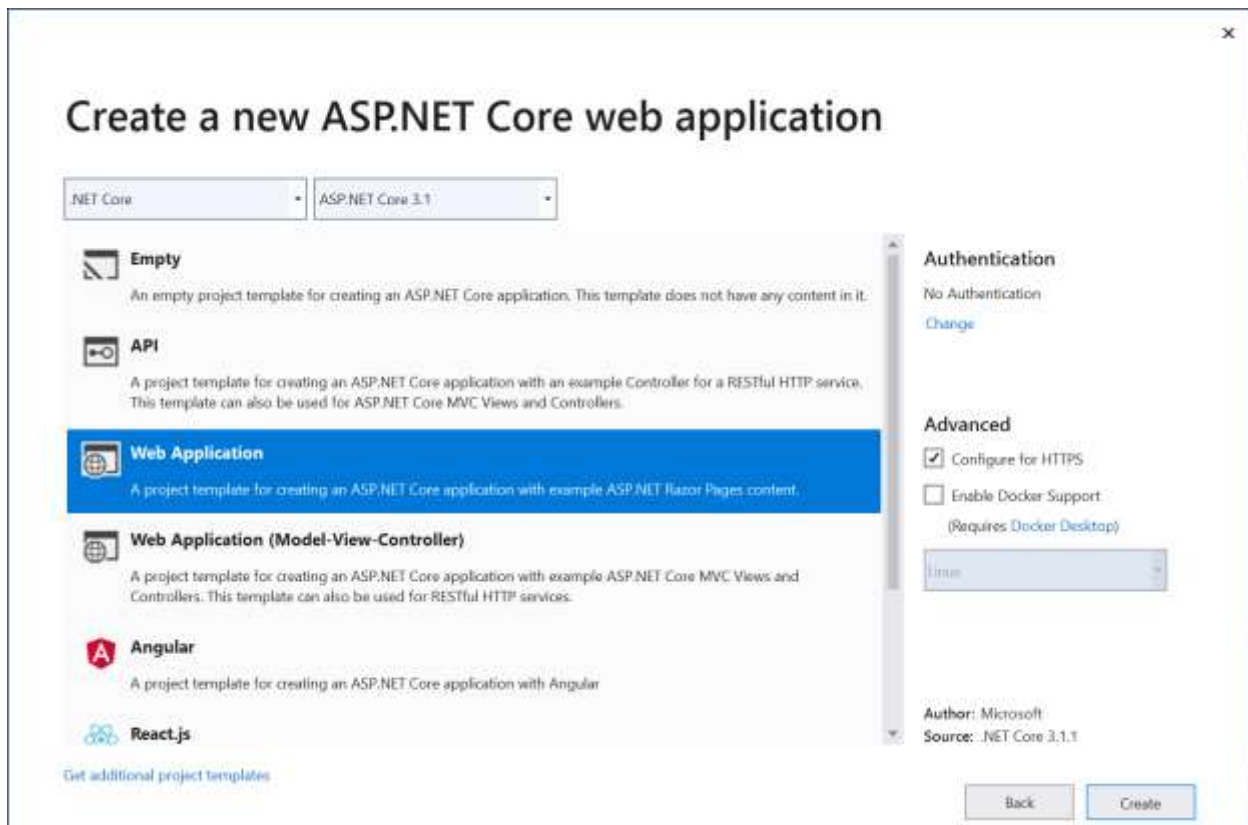
Display ssrs rdl report in ASP.NET Core 3.1 application using Report Viewer

Create your first ASP.NET Core reporting Web application to display already created SSRS RDL report in Bold Reports ASP.NET Core Report Viewer without using a Report Server using this step-by-step instructions.

To create your first application in other .NET Core frameworks, refer to the documentation [ASP .NET Core 2.1](#), [.NET 5.0](#) or [.NET 6.0](#)

Create ASP.NET Core application

1. Start Visual Studio 2019 and click **Create new project**.
2. Choose **ASP.NET Core Web Application**, and then click **Next**.
3. Change the project name, and then click **Create**.
4. In the dropdown with the **ASP.NET Core version**, choose **ASP.NET Core 3.1**.
5. Select the **Web Application (Model-View-Controller)** template, then click **Create**.



If you need to use Bold Reports with ASP.NET Core on Linux or macOS, then refer to this [Can Bold Reports be used with ASP.NET Core on Linux and macOS](#) section.

List of dependency libraries

1. In the Solution Explorer tab right-click the project or solution , and choose **Manage NuGet Packages**. Alternatively, go to **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.AspNet.Core**, **BoldReports.Net.Core** and **System.Data.SqlClient** packages, and install them in your Core application. The following table provides details about the packages and their usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Exports the report to PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core**, and **Syncfusion.XlsIO.Net.Core** packages.

Syncfusion.Pdf.Net.Core | Exports the report to a PDF.

Syncfusion.DocIO.Net.Core | Exports the report to a Word.

Syncfusion.XlsIO.Net.Core | Exports the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serializes and deserialize data for the Report Viewer. It is a mandatory package for Report Viewer, and the package version should be 10.0.1 or higher.

Refer scripts and CSS

Directly refer all the required scripts and style sheets from [CDN](#) links.

- The following scripts and style sheets are mandatorily required to use the Report Viewer.
 - bold.reports.all.min.css**
 - jquery-1.10.2.min.js**
 - ej2-maps.min.js** - Renders the map item. Add this script only if your report contains the map report item.
 - ej2-base.min.js**, **ej2-data.min.js**, **ej2-pdf-export.min.js**, **ej2-svg-base.min.js**, **ej2-lineargauge.min.js** and **ej2-circulargauge.min.js** - Render the gauge item. Add these scripts only if your report contains the gauge report item.
 - bold.reports.common.min.js**
 - bold.reports.widgets.min.js**
 - ej.chart.min.js** - Renders the chart item. Add this script, only if your report contains the chart report item.
 - bold.report-viewer.min.js**
- Open the `\Views\Shared_Layout.cshtml` page.
- Replace the following code in your `\Views\Shared_Layout.cshtml` page tag.

`<html`

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
```

```
<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
```

```
<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
```

```

<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>

<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>

<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>

<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>

```

To learn more about rendering a report with data visualization report items, refer to the [how to render data visualization report items](#) section.

The Report Viewer scripts and styles can be added into your application by installing the **BoldReports.JavaScript** online nuget package.

Tag helper

It is necessary to define the following tag helper within the **_ViewImports.cshtml** page to initialize the Report Viewer component with the tag helper support.

```

`js
@using BoldReports.TagHelpers
@addTagHelper *, BoldReports.AspNet.Core

```

Configure Script Manager

Open the **~/Views/Shared/_Layout.cshtml** page and add the reporting Script Manager at the end of **<body>** element as in the following code sample.

```

`html
<body>

<div style="min-height: 600px;width: 100%;">
@RenderBody()
</div>

@RenderSection("Scripts", required: false)
<!-- Bold Reports script manager -->
<bold-script-manager></bold-script-manager>

```



```
</body>
```

Initialize Report Viewer

1. Open the `Index.cshtml` page.
2. Remove the existing codes and add the following code.

```
`html<br><bold-report-viewer id="viewer"></bold-report-viewer>
```

Add already created reports

1. Create a folder `Resources` into the `wwwroot` folder in your application to store the RDL reports.
2. Download the `sales-order-detail.rdl` from [here](#). For more sample reports, refer to [samples and demos](#) section.
3. Extract the compressed file and paste the `sales-order-detail.rdl` to `Resources` folder.

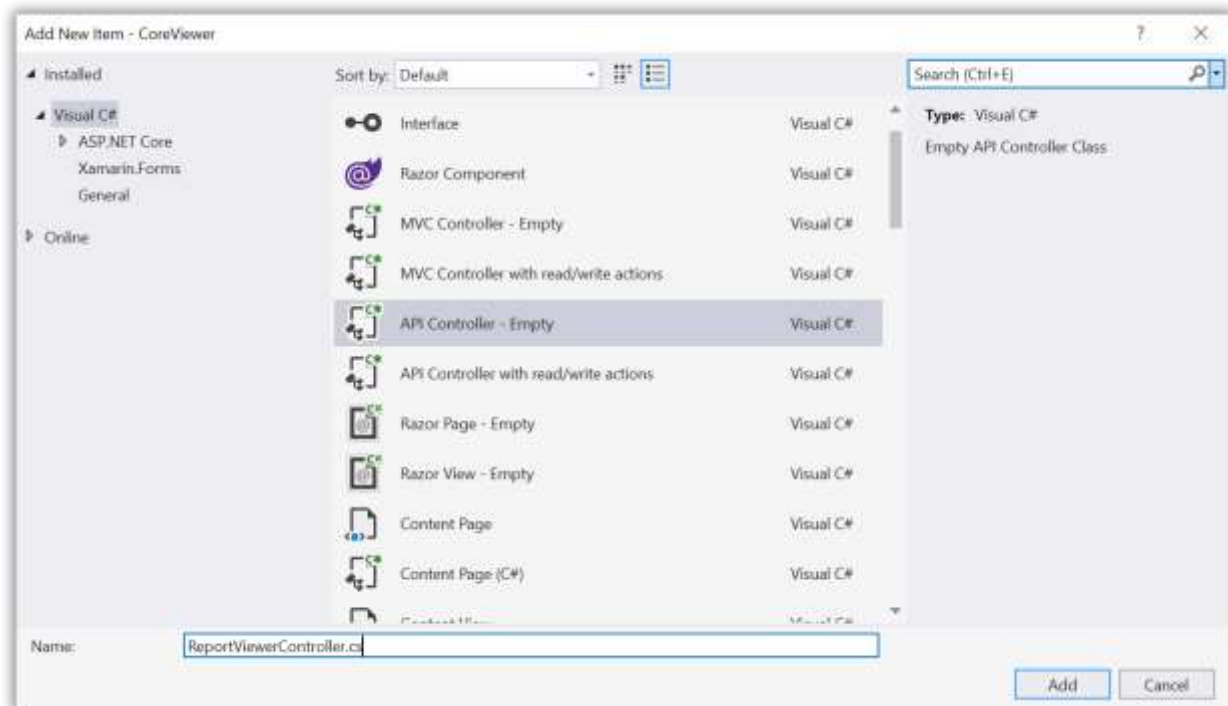
The Report Viewer is only for rendering the reports. Refer to the [create RDL report](#) section to create a report.

Configure Web API

The ASP.NET Core Report Viewer requires a Web API service to process the RDL, RDLC, and SSRS report files.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller Empty** class and name it as `ReportViewerController.cs`



3. Click **Add**.

While adding API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using System.IO;
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and then implement its methods.

It is required for processing the reports and for handling request from the Report Viewer.

6. Create local variables inside the **ReportViewerController** class.

```
`csharp
//Report Viewer requires a memory cache to store the information of consecutive client request and
have the rendered report viewer information in server
private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
// IWebHostEnvironment used to get the report stream from application wwwroot\Resources folder..
```

```
private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
,
```

7. Load the report as stream in `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
    reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}
,
```

You cannot load the report stored in application with path information in ASP.NET Core service.

8. Set the `Route` attribute for `ReportViewerController`.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    ...
}
,
```

9. You can replace the template code with the following code.

```
`csharp
[Route("api/[controller]/[action]")]
```

```
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered Report Viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
    public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
    {
        //Contains helper methods that help to process a Post or Get request from the Report Viewer control
        //and return the response to the Report Viewer control
        return ReportHelper.ProcessReport(jsonArray, this, this._cache);
    }
    // Method will be called to initialize the report information to load the report with ReportHelper for
    // processing.
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sales-order-detail.rdl report from application the folder
        // wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
        FileStream inputStream = new FileStream(basePath + @"\Resources\" +
            reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
        MemoryStream reportStream = new MemoryStream();
    }
}
```

```

inputStream.CopyTo(reportStream);
reportStream.Position = 0;
inputStream.Close();
reportOption.ReportModel.Stream = reportStream;
}

// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}

//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
return ReportHelper.GetResource(resource, this, _cache);
}

[HttpPost]
public object PostFormReportAction()
{
return ReportHelper.ProcessReport(null, this, _cache);
}
}
`

```

Enable cross-origin requests

Browser security prevents the Report Viewer from making requests to your Web API Service when both server-side and client-side requests run in different domains. To allow access to your Web API service from a different domain, enable the cross-origin requests.

1. Open `Startup.cs` file.
2. Call `AddCors` in `Startup.ConfigureServices` to add CORS services to the app's service container as in below code.

```
`csharp
```

Display ssrs rdl report in ASP.NET Core 3.1 application using Report Viewer **Set report path and service URL**

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddCors(o => o.AddPolicy("AllowAllOrigins", builder =>
    {
        builder.AllowAnyOrigin()
        .AllowAnyMethod()
        .AllowAnyHeader();
    }));
}
```

To specify the CORS policy for `ReportViewerController` add the `[EnableCors]` attribute.

```
`csharp
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    ....
    ....
}
```

If you are looking to load the report directly from SQL Server Reporting Services (SSRS), then you can skip the following steps and move to [SSRS Report](#) section.

Set report path and service URL

To render the reports available in the application, of the Report Viewer. You can replace the following code in your Report Viewer page.

1. Open the `Index.cshtml` page.
2. Set the `report-path` and `report-service-url` properties as in below.

```
`html
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-
url="/api/ReportViewer"></bold-report-viewer>
```

The report path property is set to the RDL report that is added to the project `Resources` folder.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

The screenshot shows a report viewer interface with a toolbar at the top containing icons for print, save, zoom, and other functions. Below the toolbar, there is a dropdown menu for 'Sales Order No.' with the value 'SO50750' and a 'View Report' button. The main content area displays a report titled 'Sales Order' for 'Adventure Works Cycles'. The report includes the following information:

- Order ID:** #SO50750
- Billing Date:** 22-04-2019
- Order Date:** 01-06-2003
- Purchase Order:** PO7192170677
- Shipment Method:** CARGO TRANSPORT 5

The report also lists the Billing Address, Shipping Address, and Contact information for Jean Handley at Central Discount Store.

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.04	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	8K-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W291-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	8K-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

At the bottom of the report, there is a signature line for the Authorized Person.

See Also

[Create your first app in ASP.Net Core 2.1 version](#)

[Create your first app in NET 5.0 version](#)

[Render report with data visualization report items](#)

[Create RDLC report](#)

[Render RDLC reports](#)

[Preview report in print mode](#)

[Set data source credential for shared data sources](#)

[Change data source connection string](#)

Display SSRS RDL report in Bold Reports ASP.NET Core Report Viewer

Create your first ASP.NET Core reporting Web application in the .NET 5.0 framework to display an already created SSRS RDL report in the Bold Reports ASP.NET Core Report Viewer without using a Report Server using this step-by-step instructions.

To create your first application on the other .NET Core frameworks, refer to the documentation for [ASP .NET Core 2.1](#), [ASP.NET Core 3.1](#) or [ASP.NET Core 6.0](#)

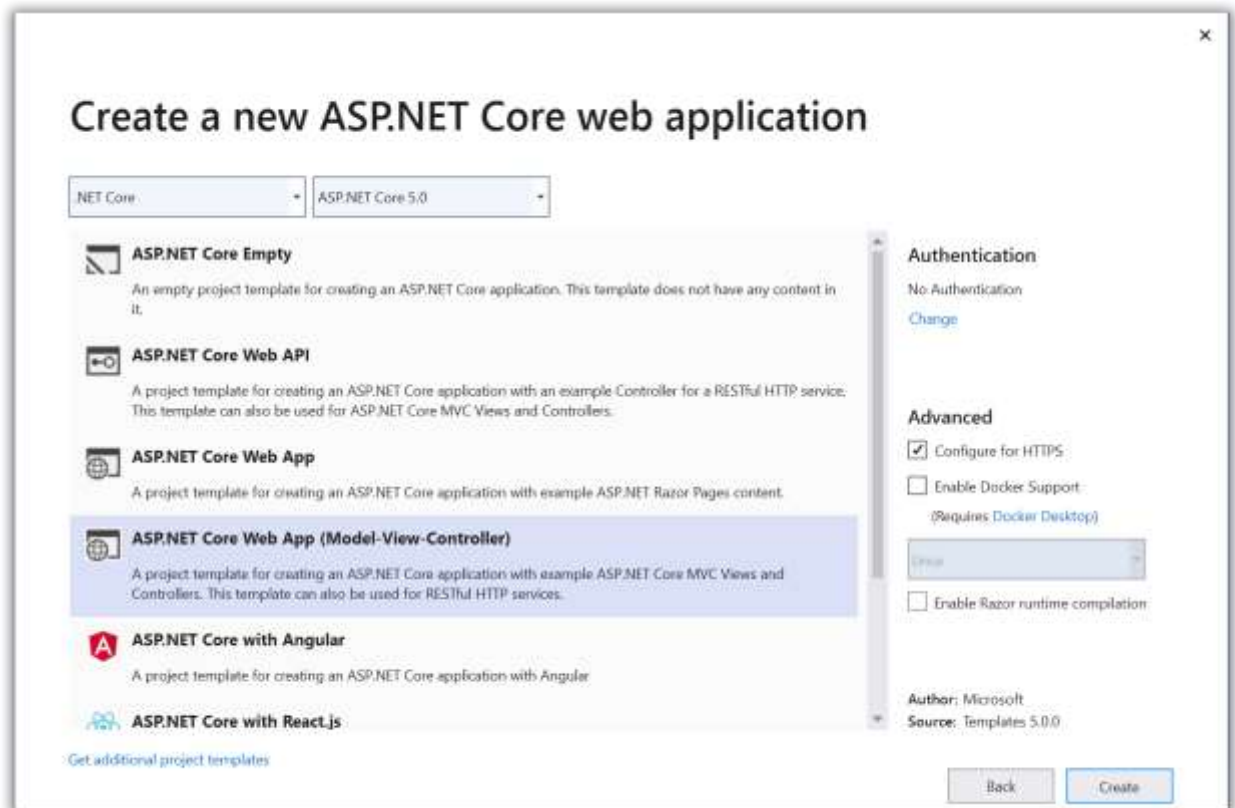
To get start quickly with Report Viewer, you can check on this video:

youtube: <https://youtu.be/-zt8p6RnOPg>

Display SSRS RDL report in Bold Reports ASP.NET Core Report Viewer **Create** an ASP.NET Core application

Create an ASP.NET Core application

1. Start Visual Studio 2019 and click **Create new project**.
2. Choose **ASP.NET Core Web Application**, and then click **Next**.
3. Change the project name, and then click **Create**.
4. In the dropdown for the **ASP.NET Core version**, choose **ASP.NET Core 5.0**.
5. Select the **Web Application (Model-View-Controller)** template, then click **Create**.



If you need to use Bold Reports with ASP.NET Core on Linux or macOS, then refer to the [Can Bold Reports be used with ASP.NET Core on Linux and macOS](#) section.

List of dependency libraries

1. In the Solution Explorer tab, right-click the project or solution, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.AspNet.Core**, **BoldReports.Net.Core**, and **System.Data.SqlClient** packages, and install them in your Core application. The following table provides details about the packages and their usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Exports the report to a PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core**, and **Syncfusion.XlsIO.Net.Core** packages.

Syncfusion.Pdf.Net.Core | Exports the report to a PDF.

Syncfusion.DocIO.Net.Core | Exports the report to a Word.

Syncfusion.XlsIO.Net.Core | Exports the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serializes and deserialize data for the Report Viewer. It is a mandatory package for the Report Viewer, and the package version should be 10.0.1 or higher.

Refer scripts and CSS

Directly refer all the required scripts and style sheets from the [CDN](#) links.

- The following scripts and style sheets are mandatorily required to use the Report Viewer.
 - bold.reports.all.min.css**
 - jquery.min.js**
 - ej2-maps.min.js** - Renders the map item. Add this script only if your report contains the map report item.
 - ej2-base.min.js**, **ej2-data.min.js**, **ej2-pdf-export.min.js**, **ej2-svg-base.min.js**, **ej2-lineargauge.min.js** and **ej2-circulargauge.min.js** - Render the gauge item. Add these scripts only if your report contains the gauge report item.
 - bold.reports.common.min.js**
 - bold.reports.widgets.min.js**
 - ej.chart.min.js** - Renders the chart item. Add this script only if your report contains the chart report item.
 - bold.report-viewer.min.js**
- Open the `\Views\Shared_Layout.cshtml` page.
- Replace the following code in your `\Views\Shared_Layout.cshtml` page tag.

`<html`

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
```

```
<script src="https://cdn.jsdelivr.net/npm/jquery/3.6.0/jquery.min.js"></script>
```

```
<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
```

```

<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>

<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>

<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>

<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>

```

To learn more about rendering a report with data visualization report items, refer to the [how to render data visualization report items](#) section.

The Report Viewer scripts and styles can be added to your application by installing the **BoldReports.JavaScript** online nuget package.

Tag helper

It is necessary to define the following tag helper within the `_ViewImports.cshtml` page to initialize the Report Viewer component with the tag helper support.

```

`js
@using BoldReports.TagHelpers
@addTagHelper *, BoldReports.AspNet.Core

```

Configure Script Manager

Open the `~/Views/Shared/_Layout.cshtml` page and add the reporting Script Manager at the end of the `<body>` element as in the following code sample.

```

`html
<body>

<div style="min-height: 600px;width: 100%;">
@RenderBody()
</div>

@RenderSection("Scripts", required: false)
<!-- Bold Reports script manager -->
<bold-script-manager></bold-script-manager>

```

```
</body>
```

```
,
```

Initialize Report Viewer

1. Open the `Index.cshtml` page.
2. Remove the existing codes and add the following code.

```
`html
```

```
<bold-report-viewer id="viewer"></bold-report-viewer>
```

```
,
```

Add already created reports

1. Create a folder `Resources` in the `wwwroot` folder in your application to store the RDL reports.
2. Download the `sales-order-detail.rdl` from [here](#). For more sample reports, refer to the [samples and demos](#) section.
3. Extract the compressed file and paste the `sales-order-detail.rdl` to the `Resources` folder.

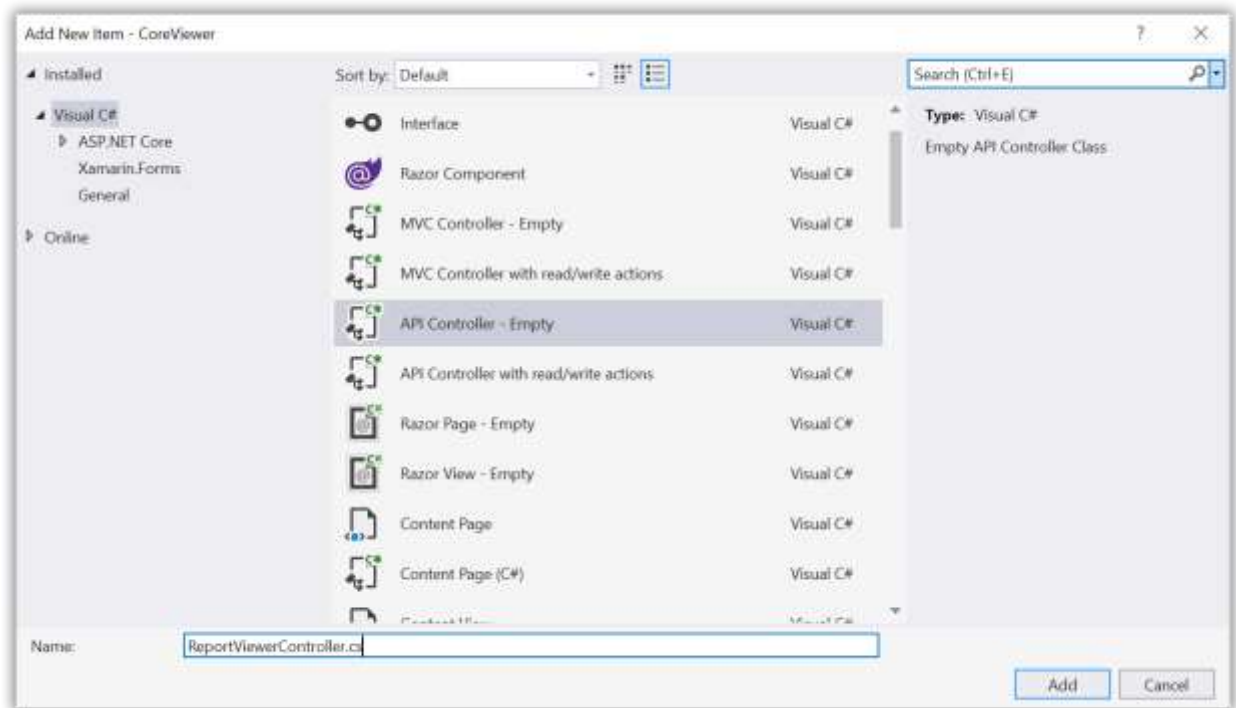
The Report Viewer is only for rendering reports. Refer to the [create RDL report](#) section to create a report.

Configure Web API

The ASP.NET Core Report Viewer requires a Web API service to process the RDL, RDLC, and SSRS report files.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller Empty** class and name it as `ReportViewerController.cs`.



3. Click **Add**.

While adding the API Controller class, naming it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using System.IO;
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and then implement its methods.

It is required for processing the reports and for handling requests from the Report Viewer.

6. Create local variables inside the **ReportViewerController** class.

```
`csharp
//Report Viewer requires a memory cache to store the information of consecutive client request and
have the rendered report viewer information in server
private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
// IWebHostEnvironment used to get the report stream from application wwwroot\Resources folder.
```

```
private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
,
```

7. Load the report as a stream in the `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
    reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}
,
```

You cannot load the report stored in the application with path information from an ASP.NET Core service.

8. Set the `Route` attribute for `ReportViewerController`.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    ...
}
,
```

9. You can replace the template code with the following code.

```
`csharp
```

```
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered Report Viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
    Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }
    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
    public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
    {
        //Contains helper methods that help to process a Post or Get request from the Report Viewer control
        //and return the response to the Report Viewer control
        return ReportHelper.ProcessReport(jsonArray, this, this._cache);
    }
    // Method will be called to initialize the report information to load the report with ReportHelper for
    // processing.
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sales-order-detail.rdl report from application the folder
        // wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
        FileStream inputStream = new FileStream(basePath + @"\Resources\" +
        reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    }
}
```

```

MemoryStream reportStream = new MemoryStream();
inputStream.CopyTo(reportStream);
reportStream.Position = 0;
inputStream.Close();
reportOption.ReportModel.Stream = reportStream;
}

// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}

//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
return ReportHelper.GetResource(resource, this, _cache);
}

[HttpPost]
public object PostFormReportAction()
{
return ReportHelper.ProcessReport(null, this, _cache);
}
}
,

```

Enable cross-origin requests

Browser security prevents the Report Viewer from making requests to your Web API Service when both server-side and client-side requests run in different domains. To allow access to your Web API service from a different domain, enable cross-origin requests.

1. Open the `Startup.cs` file.
2. Call `AddCors` in `Startup.ConfigureServices` to add CORS services to the app's service container as in the below code.

```
`csharp
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddCors(o => o.AddPolicy("AllowAllOrigins", builder =>
    {
        builder.AllowAnyOrigin()
        .AllowAnyMethod()
        .AllowAnyHeader();
    }));
}
```

To specify the CORS policy for `ReportViewerController`, add the `[EnableCors]` attribute.

```
`csharp
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    ....
    ....
}
```

If you are looking to load the report directly from the SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#) section.

Set report path and service URL

To render the reports available in the application, of the Report Viewer. You can replace the following code on your Report Viewer page.

1. Open the `Index.cshtml` page.
2. Set the `report-path` and `report-service-url` properties as shown below.

```
`html
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-
url="/api/ReportViewer"></bold-report-viewer>
```


The report path property is set to the RDL report that is added to the project **Resources** folder.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

The screenshot shows a web application interface for viewing a report. At the top, there is a toolbar with icons for print, save, zoom, and other report functions. Below the toolbar, a dropdown menu shows 'Sales Order No: 5050750'. To the right is a 'View Report' button. The main content area displays a report titled 'Sales Order' with the 'Adventure Works Cycles' logo. The report includes the following information:

Order ID	Billing Date	Order Date	Purchase Order	Shipment Method
#5050750	22-04-2019	01-06-2003	PO7192170677	CARGO TRANSPORT 5

Below this, there are sections for Billing Address, Shipping Address, and Contact, all for 'Jean Handley' at 'Central Discount Store'.

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0102-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.04	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	B6-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	B6-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

At the bottom, there is a signature line for the 'Authorized Person'.

Note: You can refer to our feature tour page for the [ASP.NET Core Report Viewer](#) to see its innovative features. Additionally, you can view our [ASP.NET Core Report Viewer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

See Also

[Create your first app in ASP.Net Core 2.1 version](#)

[Create your first app in ASP.Net Core 3.1 version](#)

[Render report with data visualization report items](#)

[Create RDLC report](#)

[Render RDLC reports](#)

[Preview report in print mode](#)

[Set data source credential for shared data sources](#)

[Change data source connection string](#)

[Create your first app in visual studio 2017](#)

[List of SSRS server versions are supported in Bold Reports](#)

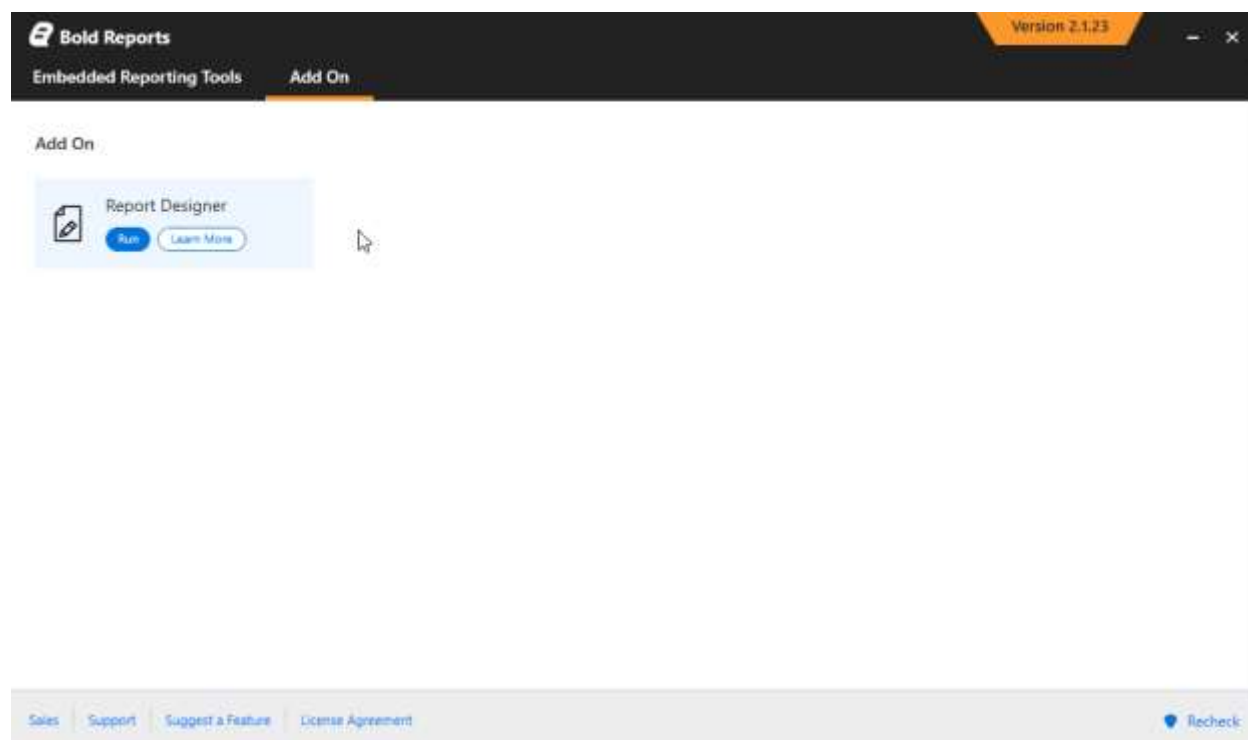
Create a SSRS RDL report

You can create an RDL report using any of the following reporting tools:

- Bold Reports Report Designer.
- Microsoft Report Builder.
- Visual Studio Report Server project template.

Bold Reports Report Designer

Bold Reports Report Designer provides the intuitive user interface to create and edit the RDL reports, which is available in Bold Embedded Reporting Tools Control Panel Add On.



Microsoft SQL Report Builder

You can create an RDL report using the Microsoft stand-alone Report Builder. For more details, refer to this [online documentation](#).

Visual Studio Report Server template

To create an RDL report in Visual Studio, a Report Server project is required where you can save your report definition (.rdl) file. For more details, refer to this [Visual Studio documentation](#).

If you do not have the Business Intelligence or Report Server Project options, you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create a RDLC report using business object data source

This section describes step by step procedure to create an RDLC report using Visual Studio Reporting project type.

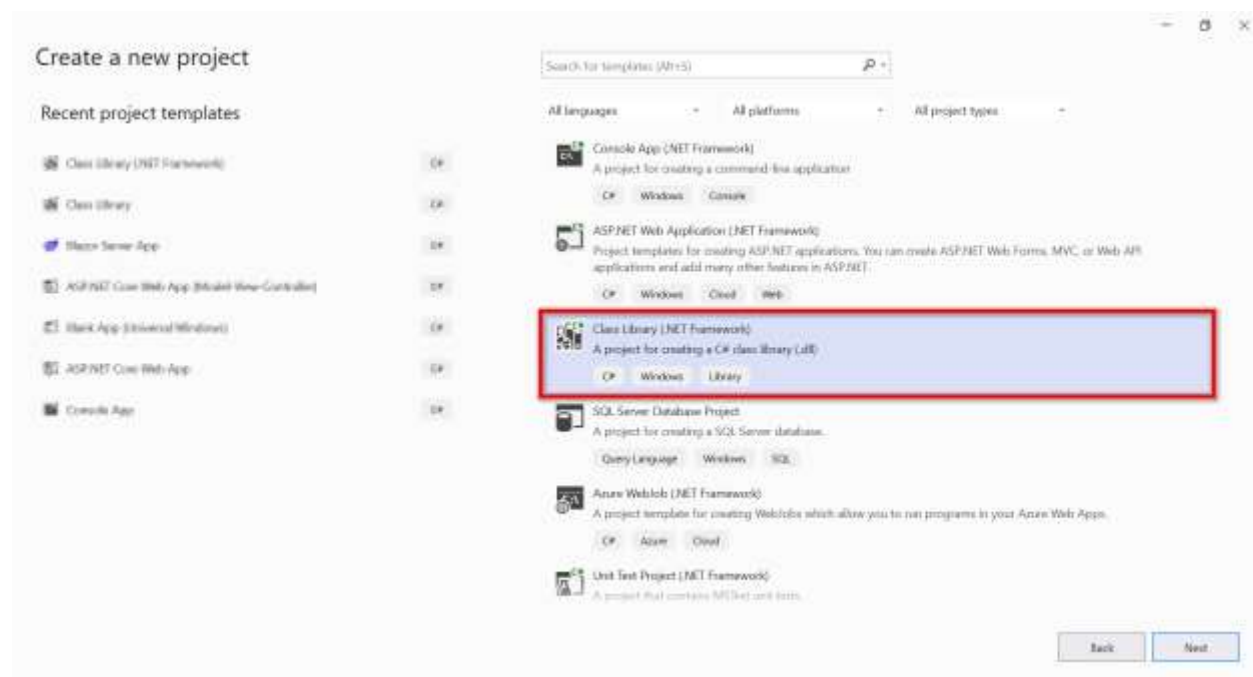
Prerequisites

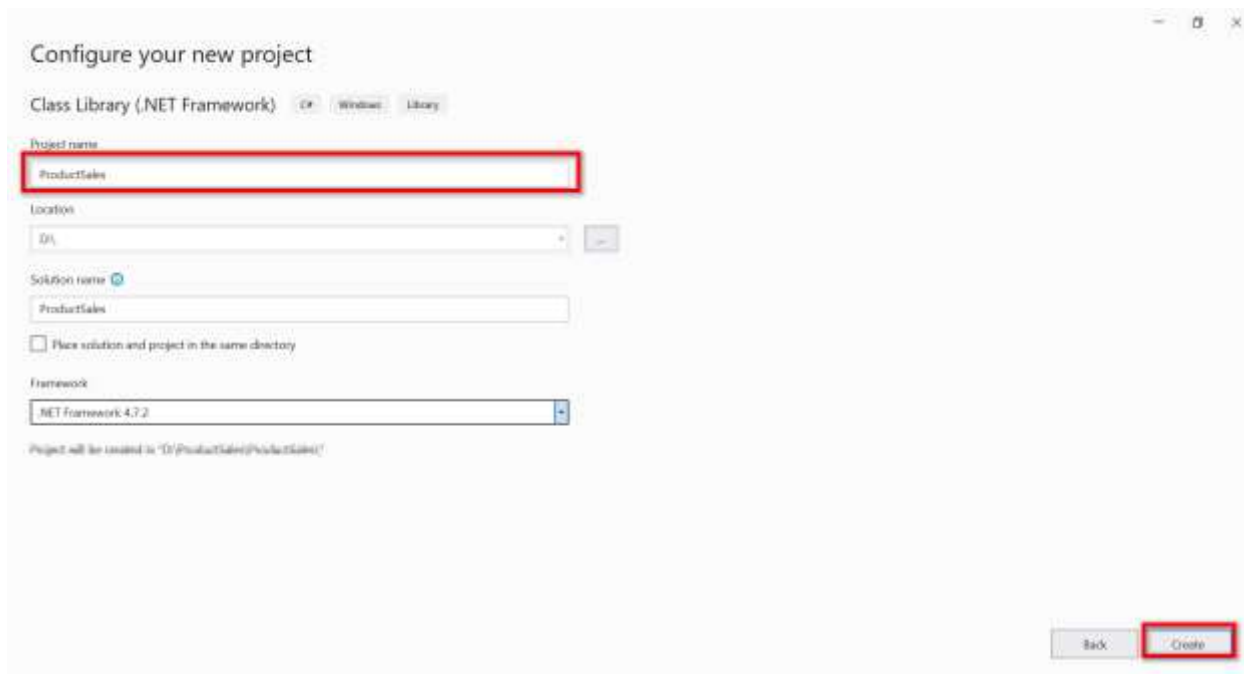
- Microsoft Visual Studio 2022 or higher
- [Microsoft RDLC Report Designer](#)

If you are using Microsoft Visual Studio lower to 2017 version then you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create business object class

1. Open Visual Studio from the File menu and select **Create a new project**.
2. Create project with Class Library(.NET Framework) type from the project type list.





Configure your new project

Class Library (.NET Framework) ☐ WebSite ☐ Library

Project name
ProductSales

Location
D:\

Solution name
ProductSales

☐ Place solution and project in the same directory

Framework
NET Framework 4.7.2

Project will be created in "D:\ProductSales\ProductSales\"

Back Create

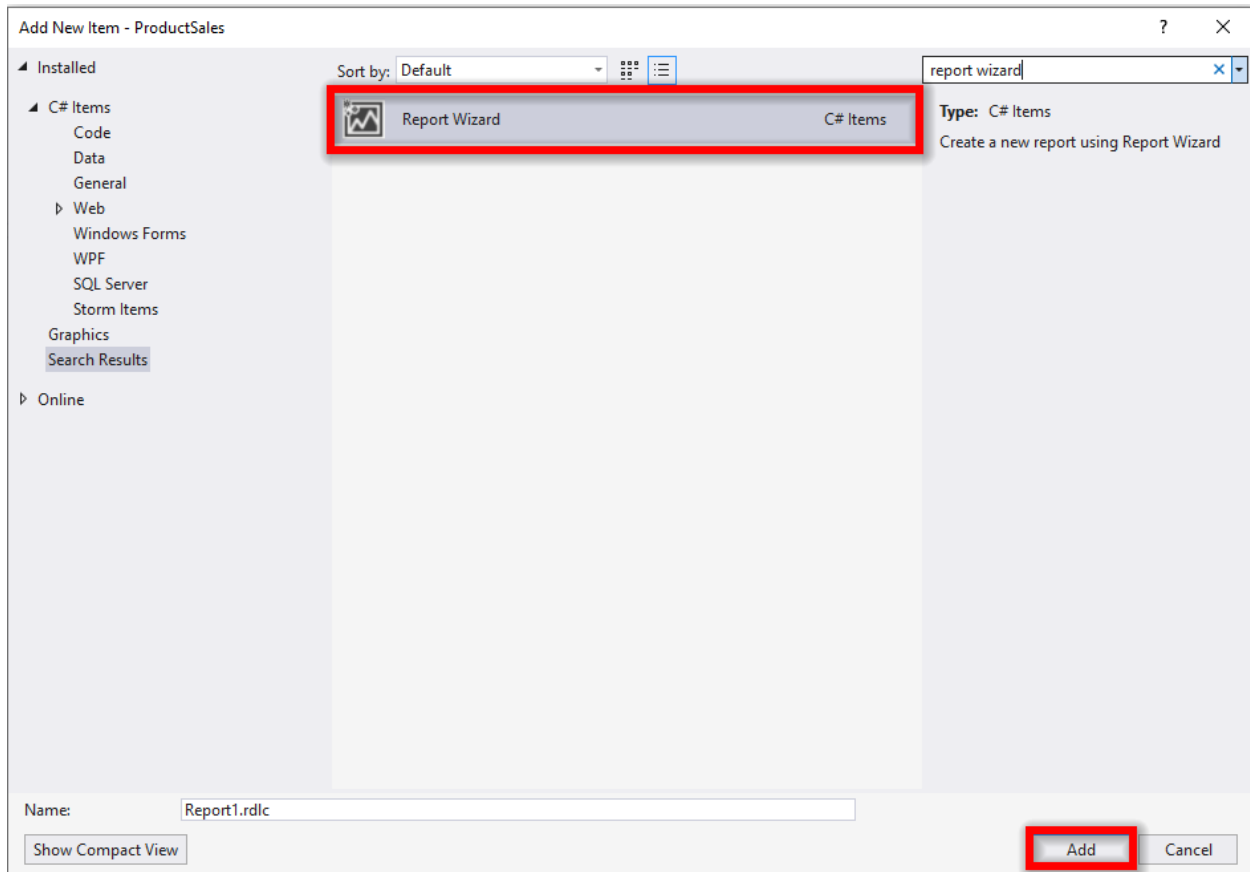
3. Create the class with necessary properties. You can find the reference below,

```
`csharp
public class ProductSales
{
    public string ProdCat { get; set; }
    public string SubCat { get; set; }
    public string OrderYear { get; set; }
    public string OrderQty { get; set; }
    public double Sales { get; set; }
}
```

4. Clean and build the application.

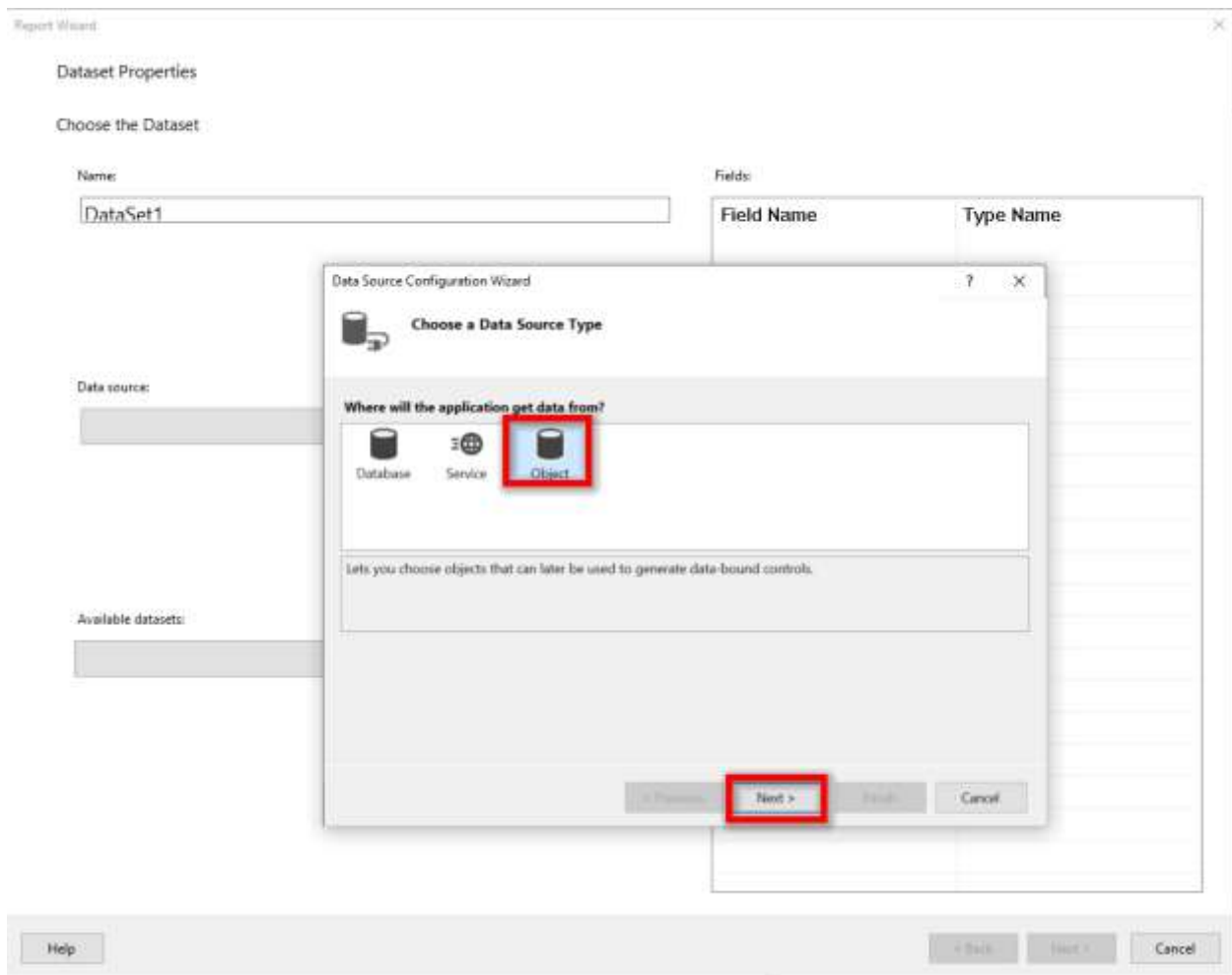
Add an RDLC report

1. Right-click the project and click **Add > New Item**.
2. Search Report with new item and select **Report Wizard** to start the report creation with dataset selection.
3. Click **Add**.

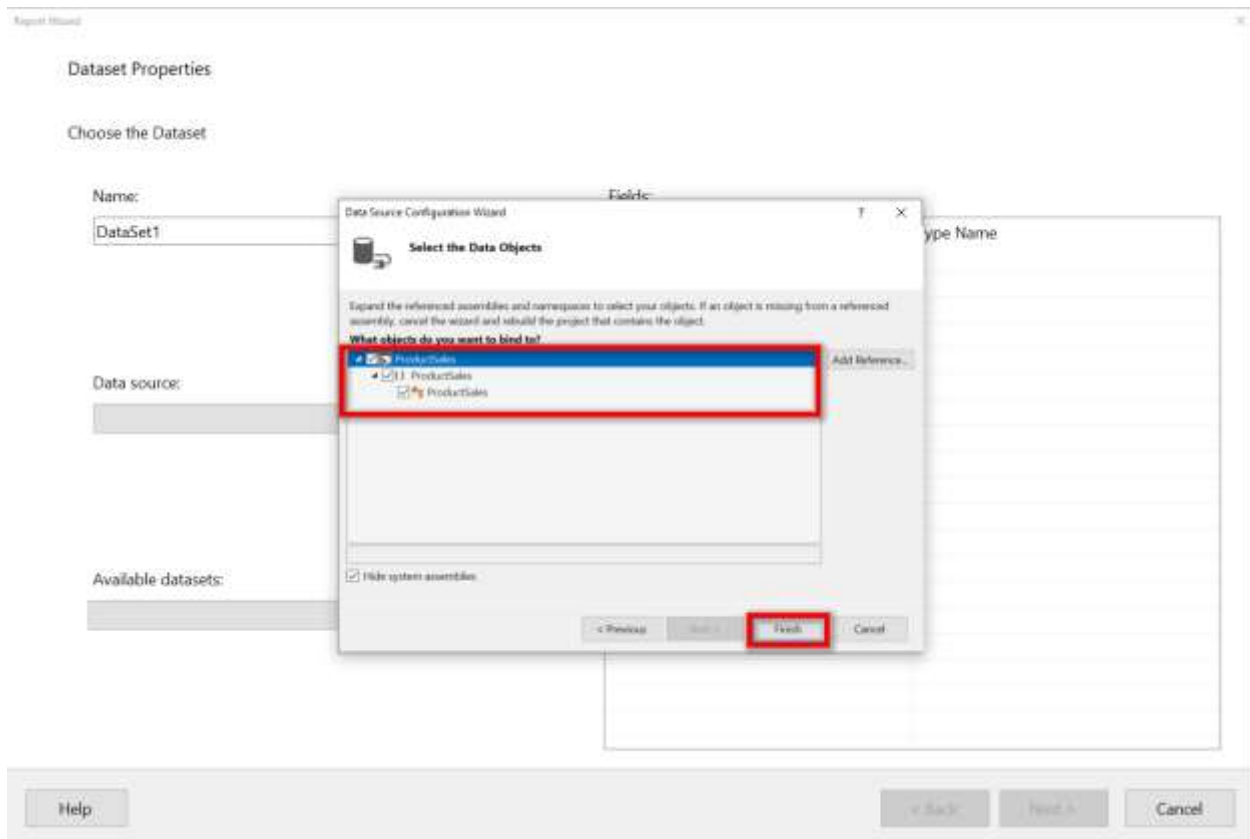


Data source and table configuration wizard

1. Choose **object** type from the Data Source Configuration wizard and click **Next**.



2. Expand the tree view and select **ProductSales**, and then click **Finish**.



3. In the DataSet Properties wizard, specify the dataset name as **SalesData**.

Report Wizard

Dataset Properties

Choose the Dataset

Name:
SalesData

Data source:
ProductSales New...

Available datasets:
ProductSales

Fields:

Field Name	Type Name
OrderQty	System.String
OrderYear	System.String
ProdCat	System.String
Sales	System.Double
SubCat	System.String

Help < Back Next > Cancel

4. Drag the fields into Values, Row, and Column groups, and then click **Next**.

Report Wizard

Arrange fields

Arrange fields to group data in rows, columns, or both, and choose values to display. Data expands across the page in column groups and down the page in row groups. Use functions such as Sum, Avg, and Count on the fields in the Values box.

Available fields:

- OrderQtr
- OrderYear
- ProdCat
- Sales
- SubCat

Column groups

- OrderYear
- OrderQtr

Row groups

- ProdCat
- SubCat

Values

- Sum(Sales)

Help < Back Next > Cancel

5. Choose the table layout and click **Next**.
6. Select table style and click **Finish**.

Report Wizard

Choose the layout

If you choose to show subtotals and grand totals, you can place them above or below the group. Stepped reports show hierarchical structure with indented groups in the same column.

Options:

☒ Show subtotals and grand totals

☒ Blocked, subtotal below

☐ Blocked, subtotal above

☐ Stepped, subtotal above

☒ Expand/collapse groups

Preview

Prod Cat	Sub Cat	[OrderYear]	Total
[ProdCat]	[SubCat]	[Sum(Sales)]	[Sum(Sales)]
Total		[Sum(Sales)]	[Sum(Sales)]
Total		[Sum(Sales)]	[Sum(Sales)]

Help < Back **Next >** Cancel

Report Wizard

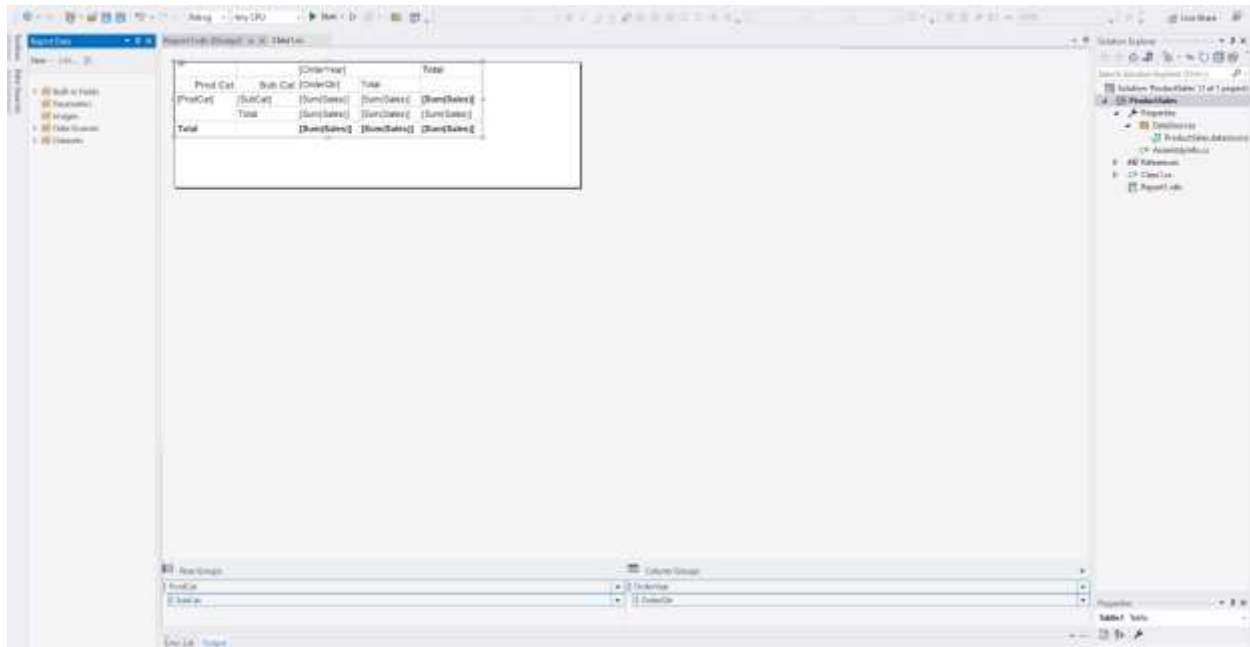
Preview

Preview the report item being created. You can customize the fonts, color schemes and style after you finish the wizard.

Prod Cat	Sub Cat	[OrderYear]	Total
[ProdCat]	[SubCat]	[Sum(Sales)]	[Sum(Sales)]
Total		[Sum(Sales)]	[Sum(Sales)]
Total		[Sum(Sales)]	[Sum(Sales)]

Help < Back **Finish >>** Cancel

Now, the RDLC report is displayed in the Visual Studio as follows.



To render the RDLC using Report Viewer, refer to the [RDLC Report](#) section.

Render data visualization report items

To render the report with data visualization components such as chart, gauge and map items, must add scripts of the visualization element. The following table shows the script reference that need to be added in Report Viewer page for data visualization elements.

Visualization Item | Script File

Gauge | ej2-base.min.js, ej2-data.min.js, ej2-pdf-export.min.js, ej2-svg-base.min.js, ej2-lineargauge.min.js and ej2-circulargauge.min.js

Map | ej2-maps.min.js

Chart | ej.chart.min.js

To render the chart report item, add chart control script `ej.chart.min.js` before the `bold.report-viewer.min.js` reference in `\Views\Shared_Layout.cshtml` page as in following code sample.

```
`html
```

```
<link href="~/Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
```

```
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
```

```
<script src="~/Scripts/bold-reports/common/bold.reports.common.min.js"></script>
```

```
<script src="~/Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
```

```
<!--Used to render the chart item. Add this script, only if your report contains the chart report item.-->
```

```
<script src="~/Scripts/bold-reports/data-visualization/ej.chart.min.js"></script>
```

```
<!-- Report Viewer component script-->
```

```
<script src="~/Scripts/bold-reports/bold.report-viewer.min.js"></script>
```

The following code can be used to render the chart, gauge and map report items in Report Viewer.

```
`html
```

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />

<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>

<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>

<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
```

This section is specifically for Bold Reports version less than 3.x. If you are using a higher version of [Bold Reports](#), recommend you to embed the VB code directly in the report itself, as VBCodeProvider is now available in the .NET Core platform.

How to use the custom code with Report Viewer

Custom code in a report allows to include new custom constants, variables, functions, or subroutines. As the VBCodeProvider is not available in .NET Core platform, report with custom codes requires additional settings to enable this feature. This section describes the steps required to use custom codes with Report Viewer.

Create a custom code class file

1. Create a new C# class file in your application.
2. The namespace should be **BoldReports.Processing.Helper** and class declaration should be **public static class Code**.
3. Write the C# methods equivalent to your report VB codes.
4. Here is the example code to convert the value to USD.

```
`csharp
namespace BoldReports.Processing.Helper
{
    public static class Code
    {
        public static string PrintHelloWorld()
        {
            return "Hello World";
        }
    }
}
```

Create a assembly reference and add it to Report Viewer

If you have created the custom code class, then you need to add the assemblies in the list with what you have used in your application to Report Viewer as shown in following code example.

In our Report Viewer application, assemblies needs to be added before calling the OnReportLoaded method.

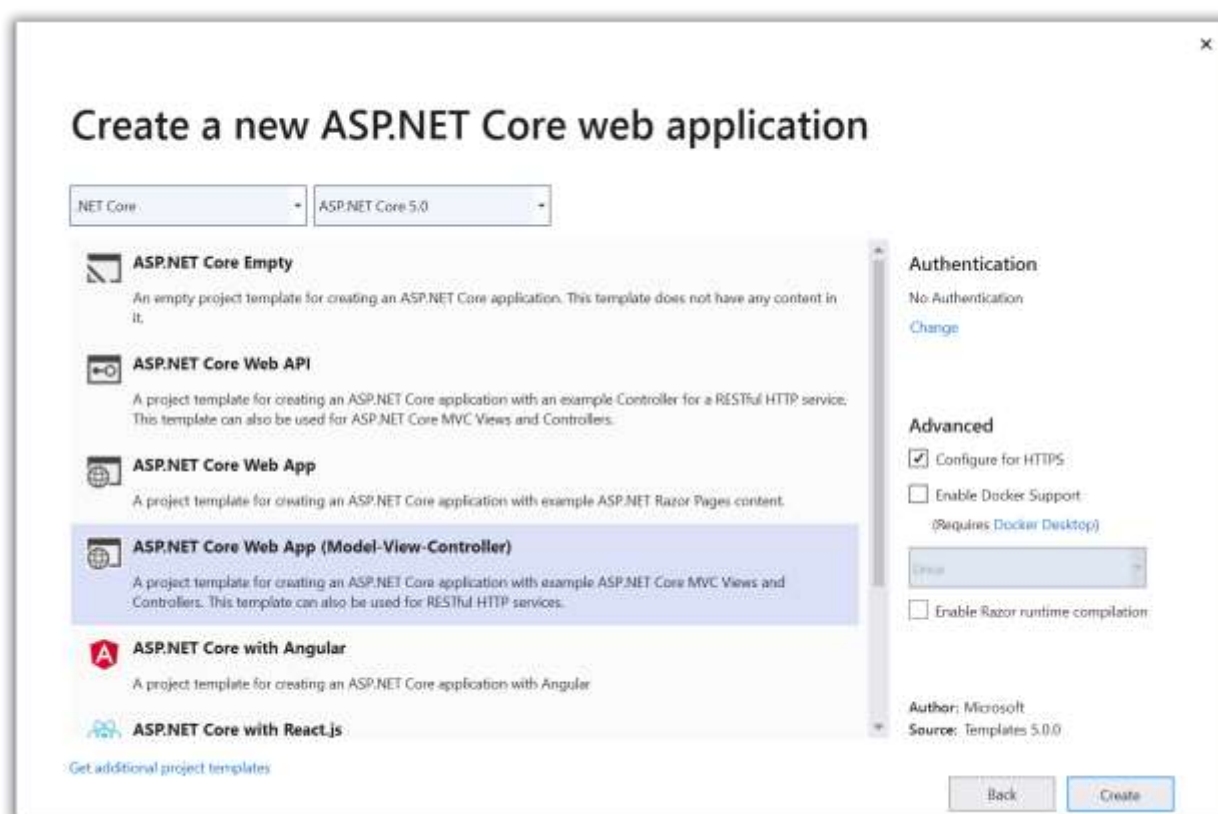
```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    reportOption.ReportModel.ProcessingMode = ProcessingMode.Local;
    FileStream inputStream = new FileStream(basePath + @"\Resources\Product List.rdlc", FileMode.Open,
    FileAccess.Read);
    reportOption.ReportModel.Stream = inputStream;
    reportOption.ReportModel.Assemblies.Add(this.GetType().GetTypeInfo().Assembly);
}
`
```

Display ssrs rdl report in Bold Reports ASP.NET Core Report Viewer

This section explains you the steps required to create your first ASP.NET Core reporting application to display already created SSRS RDL report in Bold Reports ASP.NET Core Report Viewer without using a Report Server.

Create ASP.NET Core application

1. Open Visual Studio 2017, click the **File** menu, go to **New**, and then select **Project**.
2. Go to **Installed > Visual C# > Web**, and then select **ASP.NET Core Web Application**, change the application name, and then click **OK**.
3. Choose the **ASP.NET Core version** and select the Web Application **Model-View-Controller** template and then click **OK**. Do not select Enable Docker Support.



Enable Docker support

We are using `System.Drawing` to measure text size for `CanGrow` feature support with `Textbox ReportItem` it requires native `libgdipplus` library but which doesn't contain in default `microsoft/dotnet` image. So, if we select **Enable Docker Support** then we must add native `libgdipplus` library install command with `DockerFile`.

`command

install `System.Drawing` native dependencies

RUN `apt-get update \`

&& `apt-get install -y --allow-unauthenticated \`

```

libc6-dev \
libgdiplus \
libx11-dev \
&& rm -rf /var/lib/apt/lists/*
`

```

The sample docker file can be downloaded from [here](#)

List of dependency libraries

Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, go to **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command and then search for **BoldReports.AspNet.Core** and **BoldReports.Net.Core** packages, and install them in your Core application. The following table provides details about the packages and their usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Exports the report to PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core**, and **Syncfusion.XlsIO.Net.Core** packages.

Syncfusion.Pdf.Net.Core | Exports the report to a PDF.

Syncfusion.DocIO.Net.Core | Exports the report to a Word.

Syncfusion.XlsIO.Net.Core | Exports the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serializes and deserialize data for the Report Viewer. It is a mandatory package for Report Viewer, and the package version should be higher than 10.0.1 for NET Core 2.0 and others should be higher than 9.0.1.

System.Data.SqlClient | This is an optional package for Report Viewer. It should be referenced in project when the RDL report renders visual data from the SQL Server or SQL Azure data source based on RDL design. The package version should be higher than 4.1.0.

Refer scripts and CSS

Directly refer all the required scripts and style sheets from [CDN](#) links.

- The following scripts and style sheets are mandatorily required to use the Report Viewer.
 - bold.reports.all.min.css**
 - jquery-1.10.2.min.js**
 - bold.reports.common.min.js**
 - bold.reports.widgets.min.js**
 - ej.chart.min.js** - Renders the chart item. Add this script, only if your report contains the chart report item.
 - ej2-base.min.js**, **ej2-data.min.js**, **ej2-pdf-export.min.js**, **ej2-svg-base.min.js**, **ej2-lineargauge.min.js** and **ej2-circulargauge.min.js** - Render the gauge item. Add these scripts only if your report contains the gauge report item.

- `ej2-maps.min.js` - Renders the map item. Add this script only if your report contains the map report item.
 - `bold.report-viewer.min.js`
2. Open the `\Views\Shared_Layout.cshtml` page.
 3. Add the listed references in the same order given in above list. You can replace the following code in your `\Views\Shared_Layout.cshtml` page tag.

```
`html
```

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />

<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>

<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>

<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>

<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>

<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>

<!--Render the chart item. Add this script only if your report contains the chart report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>

<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
`
```

To learn more about rendering a report with data visualization report items, refer to the [how to render data visualization report items](#) section.

Tag helper

It is necessary to define the following tag helper within the `_ViewImports.cshtml` page to initialize the Report Viewer component with the tag helper support.

```
`js
```



```
@using BoldReports.TagHelpers
@addTagHelper *, BoldReports.AspNet.Core
,
```

Configure Script Manager

Open the `~/Views/Shared/_Layout.cshtml` page and add the reporting Script Manager at the end of `<body>` element as in the following code sample.

```
`html
<body>
<div style="min-height: 600px;width: 100%;">
@RenderBody()
</div>
@RenderSection("Scripts", required: false)
<!-- Bold Reports script manager -->
<bold-script-manager></bold-script-manager>
</body>
,
```

Initialize Report Viewer

Initialize the Report Viewer as shown in the following code sample in your Report Viewer CSHTML page. For an example, the `Index.cshtml` page can be replaced with the following code by removing the existing codes.

```
`html
<bold-report-viewer id="viewer"></bold-report-viewer>
,
```

Add already created reports

The Report Viewer is only for rendering the reports. You must use a report generation tool to create a report and to learn more about this, refer to the [create RDL report](#) section.

1. Create a folder `Resources` into the `wwwroot` folder in your application to store the RDL reports.
2. Add already created reports to the newly created folder.

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded from [here](#). You can add the reports from Bold Reports installation location. For more information, refer to [samples and demos](#) section.

Configure Web API

The interface `IReportController` has declaration of action methods that are defined in the Web API Controller for processing the RDL, RDLC, and SSRS reports and for handling request from the Report Viewer control. The `IReportController` has the following action methods declaration:

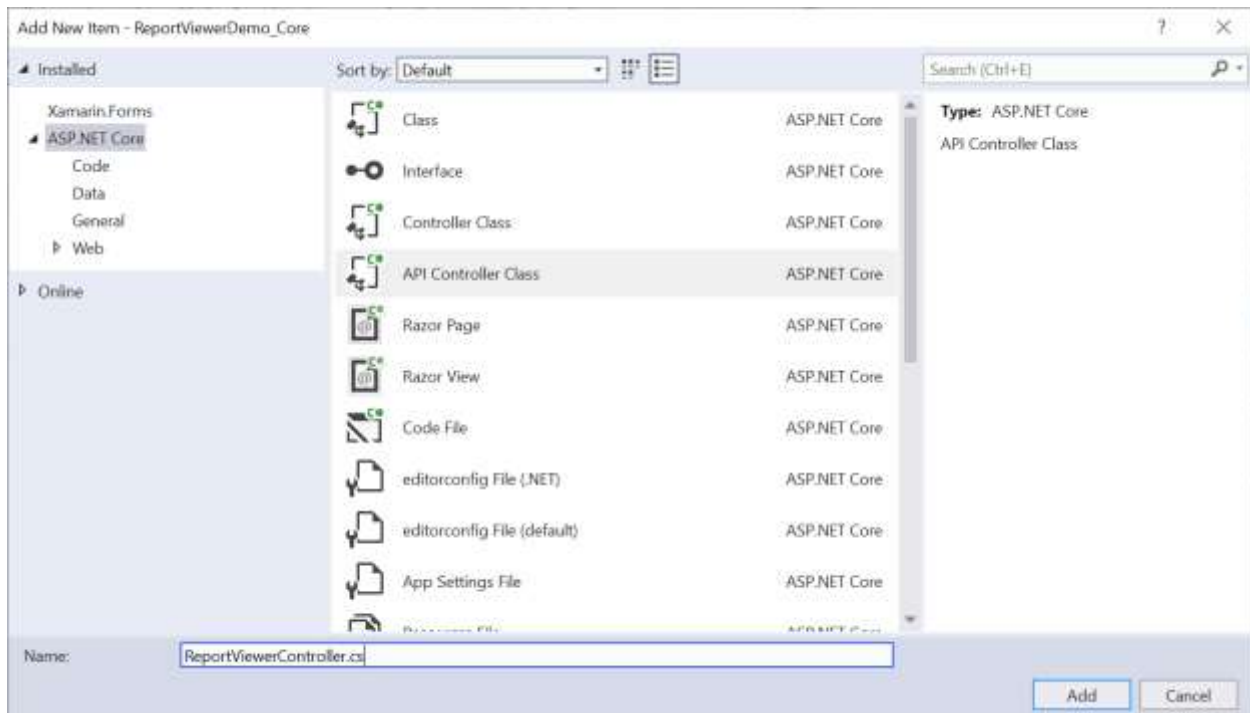
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller** class and name it as **ReportViewerController.cs**



3. Click **Add**.

While adding API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using System.IO;
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and then implement its methods.
6. Create local references for the interfaces given in following table.

Interface | Purpose

IMemoryCache | Report Viewer requires a memory cache to store the information of consecutive client request and have the rendered report viewer information in server.

IHostingEnvironment | IHostingEnvironment used to get the report stream from application `wwwroot\Resources` folder.

7. You cannot load the application report with path information in ASP.NET Core service. So, you should load the report as stream in `OnInitReportOptions`.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
        reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    reportOption.ReportModel.Stream = reportStream;
}
`
```

8. You can replace the template code with the following code.

```
`csharp
[Route("api/[controller]/[action]")]
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered Report Viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IHostingEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IHostingEnvironment _hostingEnvironment;
```

// Post action to process the report from server based json parameters and send the result back to the client.

```
public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
    Microsoft.AspNetCore.Hosting.IHostingEnvironment hostingEnvironment)
```

```
{
```

```
    _cache = memoryCache;
```

```
    _hostingEnvironment = hostingEnvironment;
```

```
}
```

// Post action to process the report from server based json parameters and send the result back to the client.

```
[HttpPost]
```

```
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
```

```
{
```

```
    return ReportHelper.ProcessReport(jsonArray, this, this._cache);
```

```
}
```

// Method will be called to initialize the report information to load the report with ReportHelper for processing.

```
[NonAction]
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
    string basePath = _hostingEnvironment.WebRootPath;
```

// Here, we have loaded the sales-order-detail.rdl report from application the folder wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.

```
    FileStream inputStream = new FileStream(basePath + @"\Resources\" +
        reportOption.ReportModel.ReportPath, FileMode.Open, FileAccess.Read);
```

```
    MemoryStream reportStream = new MemoryStream();
```

```
    inputStream.CopyTo(reportStream);
```

```
    reportStream.Position = 0;
```

```
    inputStream.Close();
```

```
    reportOption.ReportModel.Stream = reportStream;
```

```
}
```

// Method will be called when reported is loaded with internally to start to layout process with ReportHelper.

```
[NonAction]
```

```
public void OnReportLoaded(ReportViewerOptions reportOption)
```

```

{
}

//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
    return ReportHelper.GetResource(resource, this, _cache);
}

[HttpPost]
public object PostFormReportAction()
{
    return ReportHelper.ProcessReport(null, this, _cache);
}
}
`

```

Enable cross-origin requests

Browser security prevents the Report Viewer from making requests to your Web API Service when both server-side and client-side requests run in different domains. To allow access to your Web API service from a different domain, enable the cross-origin requests.

Call `AddCors` in `Startup.ConfigureServices` to add CORS services to the app's service container. Replace the following code to allow any origin requests.

```

`csharp
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddCors(o => o.AddPolicy("AllowAllOrigins", builder =>
    {
        builder.AllowAnyOrigin()
        .AllowAnyMethod()
        .AllowAnyHeader();
    }));
}

```

To specify the CORS policy for home controller, add the `[EnableCors]` attribute to the controller class and specify the policy name.

```
`csharp
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
    public IActionResult Index()
    {
        return View();
    }
    ....
}
```

Set report path and service URL

To render the reports available in the application, set the `report-path` and `report-service-url` properties of the Report Viewer. You can replace the following code in your Report Viewer page.


```
`html
<bold-report-viewer id="viewer" report-path="sales-order-detail.rdl" report-service-
url="/api/ReportViewer"></bold-report-viewer>
```

The report path property is set to the RDL report that is added to the project `Resources` folder.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

Sales Order No: View Report



Sales Order
 Order ID: #SO50750
 Billing Date: 22-04-2019
 Order Date: 01-06-2003
 Purchase Order: PO7192170677
 Shipment Method: CARGO TRANSPORT 5

Billing Address
 Jean Handley
 Central Discount Store
 259626 Russell Rd. South
 Kent, Washington
 98031
 United States

Shipping Address
 Jean Handley
 Central Discount Store
 259626 Russell Rd. South
 Kent, Washington
 98031
 United States

Contact
 Jean Handley
 Ph: 562-555-0113

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.84	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	B6-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	B6-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

I/We hereby certify that the information on this invoice is true and correct and that the contents of this shipment are as stated above.

Signature of Authorized Person

See Also

[Render report with data visualization report items](#)[Create RDLC report](#)[Render RDLC reports](#)[Preview report in print mode](#)[Set data source credential for shared data sources](#)[Change data source connection string](#)

How to change the exporting document file name based on parameter

Find the following steps to change the file based on parameter values in Report.

1. Create the file exporting file name using the parameters in **onRenderingComplete** event and store it in local variable.

`html

```
function onRenderingComplete(event) {
var parameters = event.reportParameters;
if(parameters){
for (var i = 0; i < parameters.length; i++) {
if(parameters[i].Name == "Department"){
this.exportFileName = "Sales for " + parameters[i].Value;
```

```

}
}
`

```

2. Use the file Name property with export, click event to change the file using the value stored in local variable used for having the file using the parameters.

```

`html
function onExportItemClick(event) {
event.fileName = this.exportFileName ;
}
`

```

How to change the data source dynamically

You have to use the `reportOption.ReportModel.DataSourceCredentials` available with the `OnInitReportOptions` method to dynamically change the data source in the web API controller. The following code sample shows how to change the connection string of the `<database>` data source in the report.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
DataSourceCredentials dataSourceCredentials = new DataSourceCredentials();

string connectionString = "Data Source = <instancename>; Initial Catalog = <database>; User ID = 
'<username>'; Password = '<password>'";

//You have to provide the shared data source name used with the report or the data source name
available with the report.

dataSourceCredentials.Name = "<database>";
dataSourceCredentials.ConnectionString = connectionString;
reportOption.ReportModel.DataSourceCredentials = new List<DataSourceCredentials> {
dataSourceCredentials };
}
`

```

How to disable the vertical scrollbar in parameter panel

To disable the vertical scrollbar in parameter panel, set the `enableparameterblockscroller` property to false.

```
<span style="font-weight:bold">Example</span>
```



```
`js
<div id="report viewer"></div>
<script>
$("#report viewer").boldReportViewer(
{
enableParameterBlockScroller: false
});
</script>
`
```

How to generate the RDL and RDLC reports programmatically in the report viewer

The Bold Reports reporting library allows you to generate and preview RDL and RDLC reports programmatically in the report viewer. The `ReportDefinition` class is defined as a report object model. In the report definition instance, you can create, add, and modify the report properties, report sections such as header and footer, and report items.

The following steps illustrates how to create a basic report object model:

Initialize a report definition

The following code snippet guides you in initializing the report definition.

```
`csharp
ReportDefinition CreateReport()
{
ReportDefinition report = new ReportDefinition();
report.ReportSections = new ReportSections();
var reportSection = new ReportSection();
report.ReportSections.Add(reportSection);
reportSection.Width = new BoldReports.RDL.DOM.Size("6in");
report.ReportUnitType = "Inch";
report.RDLType = RDLType.RDL2010;
return report;
}

BoldReports.RDL.DOM.Style AddStyle()
{
BoldReports.RDL.DOM.Style style = new BoldReports.RDL.DOM.Style();
style.Border = new BoldReports.RDL.DOM.Border();
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a Data source for report

```
style.Border.Width = new BoldReports.RDL.DOM.Size("1pt");
style.Border.Style = "Solid";
style.Border.Color = "Black";
return style;
}
`
```

Create a Data source for report

The following code snippet guides you in creating a **Datasource** for the report and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Page = new BoldReports.RDL.DOM.Page();
reportSection.Page.Style = AddStyle();
reportSection.Page.PageHeight = "4in";
reportSection.Page.PageWidth = "6in";
var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
<database>");
report.DataSources = new DataSources();
report.DataSources.Add(newDataSource);
...
...
...
}

//If you are using RDLC report then you can pass any string value as connection string
BoldReports.RDL.DOM.DataSource CreateDataSource(string name, string dataProvider, string
connectionString)
{
var dataSource = new BoldReports.RDL.DOM.DataSource();
dataSource.Name = name;
dataSource.SecurityType = BoldReports.RDL.DOM.SecurityType.Integrated;
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a Dataset for the report

```
dataSource.ConnectionProperties = new BoldReports.RDL.DOM.ConnectionProperties();
dataSource.ConnectionProperties.DataProvider = dataProvider;
dataSource.ConnectionProperties.ConnectionString = connectionString;
dataSource.ConnectionProperties.IntegratedSecurity = true;
return dataSource;
}
`
```

Create a Dataset for the report

The following code snippet guides you in creating a **Dataset** for the report and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Page.PageWidth = "6in";
var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
<database>");
report.DataSources = new DataSources();
report.DataSources.Add(newDataSource);
var newDataSet = CreateDataSet("DataSource1", "DataSet1");
report.DataSets = new DataSets();
report.DataSets.Add(newDataSet);
...
...
...
}

BoldReports.RDL.DOM.DataSet CreateDataSet(string dataSourceName, string dataSetName)
{
var dataSet = new BoldReports.RDL.DOM.DataSet();
dataSet.Name = dataSetName;
dataSet.Query = new Query();
dataSet.Query.DataSourceName = dataSourceName;
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a Dataset for the report

```
dataSet.Query.CommandText = "SELECT  
HumanResources.Department.DepartmentID,HumanResources.Department.Name,HumanResources.De  
partment.GroupName,HumanResources.Department.ModifiedDate FROM  
HumanResources.Department";
```

```
dataSet.Query.QueryDesignerState = new QueryDesignerState();  
var table = CreateTable();  
dataSet.Query.QueryDesignerState.Tables = new List<Table>();  
dataSet.Query.QueryDesignerState.Tables.Add(table);  
dataSet.Fields = new Fields();  
dataSet.Fields.Add(CreateField("DepartmentID", "System.Int16"));  
dataSet.Fields.Add(CreateField("Name", "System.String"));  
dataSet.Fields.Add(CreateField("GroupName", "System.String"));  
dataSet.Fields.Add(CreateField("ModifiedDate", "System.DateTime"));  
return dataSet;  
}
```

```
BoldReports.RDL.DOM.Table CreateTable()
```

```
{  
var table = new Table();  
table.Schema = "HumanResources";  
table.Name = "Department";  
table.Columns = new List<Column>();  
table.Columns.Add(CreateColumn("DepartmentID"));  
table.Columns.Add(CreateColumn("Name"));  
table.Columns.Add(CreateColumn("GroupName"));  
table.Columns.Add(CreateColumn("ModifiedDate"));  
return table;  
}
```

```
BoldReports.RDL.DOM.Column CreateColumn(string columnName)
```

```
{  
var column = new Column();  
column.Name = columnName;  
return column;  
}
```

```
BoldReports.RDL.DOM.Field CreateField(string fieldName, string type)
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a report page header

```
{
var field = new Field();
field.Name = fieldName;
field.TypeName = type;
field.DataField = fieldName;
return field;
}
`
```

Create a report page header

The following code snippet guides you in creating a **PageHeader** report section and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Page = new BoldReports.RDL.DOM.Page();
reportSection.Page.Style = AddStyle();
reportSection.Page.PageHeight = "4in";
reportSection.Page.PageWidth = "6in";
reportSection.Page.PageHeader = CreateHeader();
...
...
...
}
PageHeader CreateHeader()
{
PageHeader pageHeader = new PageHeader();
pageHeader.Height = new BoldReports.RDL.DOM.Size("0.59167in");
pageHeader.Style = AddStyle();
pageHeader.PrintOnFirstPage = true;
pageHeader.PrintOnLastPage = true;
pageHeader.ReportItems = new ReportItems();
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a report page footer

```
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
return pageHeader;
}
\
```

Create a report page footer

The following code snippet guides you in creating a **PageFooter** report section and setting values to their properties.

```
\csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Page.PageHeader = CreateFooter();
...
...
...
}
PageFooter CreateFooter()
{
PageFooter pageFooter = new PageFooter();
pageFooter.Style = AddStyle();
pageFooter.Height = new BoldReports.RDL.DOM.Size("0.59167in");
pageFooter.ReportItems = new ReportItems();
pageFooter.PrintOnFirstPage = true;
pageFooter.PrintOnLastPage = true;
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
return pageFooter;
}
\
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

Initialize a report body section

- Initialize a report body object and set the values to their properties like height, styles, and report items as shown in the following code snippet.

```
`csharp
ReportDefinition CreateReport()
{
    ...
    ...
    ...
    reportSection.Body = CreateBody();
    ...
    ...
    ...
}
Body CreateBody()
{
    var body = new Body();
    body.Height = new BoldReports.RDL.DOM.Size("2.03333in");
    body.Style = AddStyle();
    body.ReportItems = new ReportItems();
    var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
    body.ReportItems.Add(textBox);
    return body;
}
`
```

- Using the following code snippets, you can create a **Textbox** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

```
`csharp
PageHeader CreateHeader()
{
    ...
    ...
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
...
pageHeader.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
...
...
...
}
PageFooter CreateFooter()
{
...
...
...
pageFooter.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
...
...
...
}
Body CreateBody()
{
...
...
...
body.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
body.ReportItems.Add(textBox);
...
...
...
}
```


How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

`BoldReports.RDL.DOM.TextBox CreateTextBox(string name, string text, string left, string top, string width, string height)`

```
{
var textBox = new BoldReports.RDL.DOM.TextBox();
textBox.Name = name;
textBox.Height = new BoldReports.RDL.DOM.Size(height);
textBox.Width = new BoldReports.RDL.DOM.Size(width);
textBox.Left = new BoldReports.RDL.DOM.Size(left);
textBox.Top = new BoldReports.RDL.DOM.Size(top);
textBox.Style = new BoldReports.RDL.DOM.Style();
textBox.Style.TextAlign = "Center";
textBox.Style.VerticalAlign = "Top";
textBox.Paragraphs = new Paragraphs();
BoldReports.RDL.DOM.Paragraph paragraph = new BoldReports.RDL.DOM.Paragraph();
TextRuns runs = new TextRuns();
TextRun run = new TextRun();
run.Style = new BoldReports.RDL.DOM.Style();
run.Style.FontStyle = "Default";
run.Style.TextAlign = "Center";
run.Style.FontFamily = "Arial";
run.Style.FontSize = new BoldReports.RDL.DOM.Size("10pt");
run.Value = text;
runs.Add(run);
paragraph.Style = new BoldReports.RDL.DOM.Style();
paragraph.Style.VerticalAlign = "Top";
paragraph.Style.TextAlign = "Center";
paragraph.TextRuns = runs;
textBox.Paragraphs.Add(paragraph);
return textBox;
}
、
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

Create a Table for the report

- Using the following code snippets, you can create a **Table** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

```
`csharp
```

```
BoldReports.RDL.DOM.Tablix CreateTablix(string name, string text, string left, string top, string width, string height)
```

```
{
```

```
var tableItem = new BoldReports.RDL.DOM.Tablix();
```

```
tableItem.Name = name;
```

```
tableItem.Height = new BoldReports.RDL.DOM.Size(height);
```

```
tableItem.Width = new BoldReports.RDL.DOM.Size(width);
```

```
tableItem.Left = new BoldReports.RDL.DOM.Size(left);
```

```
tableItem.Top = new BoldReports.RDL.DOM.Size(top);
```

```
tableItem.Style = new BoldReports.RDL.DOM.Style();
```

```
tableItem.Style.Border = new BoldReports.RDL.DOM.Border();
```

```
tableItem.Style.Border.Style = "Solid";
```

```
tableItem.DataSetName = "DataSet1";
```

```
tableItem.TablixBody = new TablixBody();
```

```
tableItem.TablixBody.TablixColumns = new TablixColumns();
```

```
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
```

```
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
```

```
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
```

```
tableItem.TablixBody.TablixRows = new TablixRows();
```

```
var rowOne = new List<TablixRowValues>();
```

```
rowOne.Add(new TablixRowValues { TextBoxName = "TextBox1", TextBoxValue = "Department ID" });
```

```
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox3", TextBoxValue = "Name" });
```

```
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox5", TextBoxValue = "Group Name" });
```

```
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowOne));
```

```
var rowTwo = new List<TablixRowValues>();
```

```
rowTwo.Add(new TablixRowValues { TextBoxName = "DepartmentID", TextBoxValue =  
"=Fields!DepartmentID.Value" });
```

```
rowTwo.Add(new TablixRowValues { TextBoxName = "Name", TextBoxValue = "=Fields!Name.Value" });
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
rowTwo.Add(new TablixRowValues { TextBoxName = "GroupName", TextBoxValue =
"=Fields!GroupName.Value" });
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowTwo));
tableItem.TablixColumnHierarchy = new TablixColumnHierarchy();
tableItem.TablixColumnHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixRowHierarchy = new TablixRowHierarchy();
tableItem.TablixRowHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixRowHierarchy.TablixMembers.Add(new TablixMember() { KeepWithGroup =
KeepWithGroup.After });
tableItem.TablixRowHierarchy.TablixMembers.Add(CreateTablixMember("Details"));
return tableItem;
}

BoldReports.RDL.DOM.TablixColumn CreateTablixColumn()
{
var tablixColumn = new TablixColumn();
tablixColumn.Width = "1in";
return tablixColumn;
}

BoldReports.RDL.DOM.TablixRow CreateTablixRow(List<TablixRowValues> values)
{
var tablixRow = new TablixRow();
tablixRow.Height = "0.25in";
tablixRow.TablixCells = new TablixCells();
for(int i = 0; i < values.Count; i++)
{
tablixRow.TablixCells.Add(CreateTablixCell(values[i].TextBoxName, values[i].TextBoxValue));
}
return tablixRow;
}

BoldReports.RDL.DOM.TablixCell CreateTablixCell(string textBoxName, string textBoxValue)
{
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
var tablixCell = new TablixCell();
tablixCell.CellContents = new CellContents();
tablixCell.CellContents.ReportItem = CreateTextBox(textBoxName, textBoxValue);
return tablixCell;
}

BoldReports.RDL.DOM.TablixMember CreateTablixMember(string groupName)
{
    var tablixMember = new TablixMember();
    tablixMember.Group = new Group();
    tablixMember.Group.Name = groupName;
    return tablixMember;
}

internal class TablixRowValues
{
    public string TextBoxName { get; set; }
    public string TextBoxValue { get; set; }
}
`
```

Create a Chart for the report

- Using the following code snippets, you can create a **Chart** report item and assign the values for their properties like name, styles, and dimension values.

```
`csharp
BoldReports.RDL.DOM.Chart CreateChart(string name, string left, string top, string width, string height)
{
    var chartItem = new Chart();
    chartItem.Name = name;
    chartItem.Height = new BoldReports.RDL.DOM.Size("10in");
    chartItem.Width = new BoldReports.RDL.DOM.Size("10in");
    chartItem.Left = new BoldReports.RDL.DOM.Size("5in");
    chartItem.Top = new BoldReports.RDL.DOM.Size("5in");
    chartItem.DataSetName = "DataSet1";
    chartItem.Style = new BoldReports.RDL.DOM.Style();
    chartItem.Bookmark= "=First(Fields!Name.Value, \"DataSet1\")";
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
chartItem.ChartCategoryHierarchy = new ChartCategoryHierarchy();
chartItem.ChartCategoryHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartCategoryHierarchy.ChartMembers.Add(ChartCategoryMember());
chartItem.ChartSeriesHierarchy = new ChartSeriesHierarchy();
chartItem.ChartSeriesHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartSeriesHierarchy.ChartMembers.Add(ChartSeriesMember());
chartItem.ChartData = new ChartData();
chartItem.ChartData.ChartDerivedSeriesCollection = new ChartDerivedSeriesCollection();
chartItem.ChartData.ChartSeriesCollection = CreateChartSeriesCollection();
chartItem.ChartLegends = new ChartLegends();
chartItem.ChartAreas = new ChartAreas();
chartItem.ChartAreas.Add(CreateChartArea());
chartItem.ChartTitles = new ChartTitles();
chartItem.Palette = "Pacific";
chartItem.ChartBorderSkin = new ChartBorderSkin();
chartItem.ChartNoDataMessage = new ChartNoDataMessage();
chartItem.ChartNoDataMessage.Caption = "No Data Available";
chartItem.ChartNoDataMessage.Name = "NoDataMessage";
return chartItem;
}

BoldReports.RDL.DOM.ChartMember ChartCategoryMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = new Group();
    ChartMember.Group = CreateChartGroup();
    ChartMember.Label="=Fields!DepartmentID.Value";
    ChartMember.SortExpressions = SortExpressions();
    return ChartMember;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
BoldReports.RDL.DOM.ChartMember ChartSeriesMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = null;
    ChartMember.Label = "Department ID";
    ChartMember.SortExpressions = new SortExpressions();
    return ChartMember;
}

BoldReports.RDL.DOM.SortExpressions SortExpressions()
{
    var SortExpressions = new SortExpressions();
    var SortExpression = new SortExpression();
    SortExpression.Direction = SortDirection.Ascending;
    SortExpression.Value = "=Fields!DepartmentID.Value";
    SortExpressions.Add(SortExpression);
    return SortExpressions;
}

BoldReports.RDL.DOM.Group CreateChartGroup()
{
    var ChartGroup = new Group();
    ChartGroup.DataElementName = null;
    ChartGroup.DataElementOutput = DataElementOutputs.Auto;
    ChartGroup.DocumentMapLabel = null;
    ChartGroup.DocumentMapLabelLocID = null;
    ChartGroup.DomainScope = null;
    ChartGroup.Filters = new Filters();
    ChartGroup.GroupExpressions = CreateGroupExpressions();
    ChartGroup.Name = "Chart2_CategoryGroup";
    ChartGroup.PageBreak = null;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
ChartGroup.PageName = null;
ChartGroup.Parent = null;
ChartGroup.Variables = new Variables();
return ChartGroup;
}

BoldReports.RDL.DOM.GroupExpressions CreateGroupExpressions()
{
    var GroupExpressions = new GroupExpressions();
    var GroupExpression = new GroupExpression();
    GroupExpression.Value = "=Fields!DepartmentID.Value";
    GroupExpressions.Add(GroupExpression);
    return GroupExpressions;
}

BoldReports.RDL.DOM.ChartSeriesCollection CreateChartSeriesCollection()
{
    var ChartSeriesCollection = new ChartSeriesCollection();
    ChartSeriesCollection.Add(CreateChartSeries());
    return ChartSeriesCollection;
}

BoldReports.RDL.DOM.ChartSeries CreateChartSeries()
{
    var ChartSeries = new ChartSeries();
    ChartSeries.CategoryAxisName = "Primary";
    ChartSeries.ChartAreaName = null;
    ChartSeries.ChartDataLabel = null;
    ChartSeries.ChartDataPoints = new ChartDataPoints();
    ChartSeries.ChartDataPoints.Add(ChartDataPoint());
    ChartSeries.ChartEmptyPoints = new ChartEmptyPoints();
    ChartSeries.ChartEmptyPoints.ChartDataLabel = new ChartDataLabel();
    ChartSeries.ChartEmptyPoints.ChartDataLabel.Style = new Style();
    ChartSeries.ChartEmptyPoints.ChartMarker = new ChartMarker();
    ChartSeries.ChartEmptyPoints.ChartMarker.Style = new Style();
    ChartSeries.ChartSmartLabel = new ChartSmartLabel();
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
ChartSeries.Name = "DepartmentID";
ChartSeries.Type = VisualizationType.Column;
ChartSeries.Subtype = VisualizationSubType.Plain;
ChartSeries.Style = new Style();
return ChartSeries;
}

BoldReports.RDL.DOM.ChartDataPoint ChartDataPoint()
{
var ChartDataPoint = new ChartDataPoint();
ChartDataPoint.ActionInfo = null;
ChartDataPoint.ChartDataLabel = null;
ChartDataPoint.ChartDataLabel = ChartDataLabel();
ChartDataPoint.ChartDataPointValues = new ChartDataPointValues();
ChartDataPoint.ChartDataPointValues.Y = "=Sum(Fields!DepartmentID.Value)";
ChartDataPoint.ChartItemInLegend = null;
ChartDataPoint.ChartMarker = new ChartMarker();
ChartDataPoint.ChartMarker.Size = null;
ChartDataPoint.ChartMarker.Type = null;
ChartDataPoint.ChartMarker.Style= chartStyle("Transparent", "Arial");
ChartDataPoint.CustomProperties = new CustomProperties();
ChartDataPoint.DataElementName = null;
ChartDataPoint.DataElementOutput = DataElementOutputs.Output;
ChartDataPoint.ToolTip = null;
ChartDataPoint.Style= chartStyle("Transparent", "Arial");
return ChartDataPoint;
}

BoldReports.RDL.DOM.ChartDataLabel ChartDataLabel()
{
var ChartDataLabel = new ChartDataLabel();
ChartDataLabel.ActionInfo = null;
ChartDataLabel.Label = null;
ChartDataLabel.Position = null;
ChartDataLabel.Rotation = "0";
```


How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
ChartDataLabel.Style = new Style();
ChartDataLabel.Style = chartStyle("Transparent", "Arial");
ChartDataLabel.ToolTip = null;
return ChartDataLabel;
}

BoldReports.RDL.DOM.Style chartStyle(string BackgroundColor, string FontFamily)
{
    var style = new Style();
    style.BackgroundColor = BackgroundColor;
    style.FontFamily = FontFamily;
    return style;
}

BoldReports.RDL.DOM.ChartArea CreateChartArea()
{
    var chartArea = new ChartArea();
    chartArea.Style = new Style();
    chartArea.Style = chartStyle("Transparent", "Arial");
    chartArea.AlignOrientation = AlignOrientation.None;
    chartArea.AlignWithChartArea = null;
    chartArea.ChartAlignType = null;
    chartArea.ChartElementPosition = null;
    chartArea.ChartInnerPlotPosition = null;
    chartArea.ChartThreeDProperties = null;
    chartArea.Hidden = false;
    chartArea.EquallySizedAxesFont = false;
    chartArea.Name = "Default";
    chartArea.ChartCategoryAxes = new ChartCategoryAxes();
    chartArea.ChartCategoryAxes.Add(createChartAxis("Primary"));
    chartArea.ChartCategoryAxes.Add(createChartAxis("Secondary"));
    chartArea.ChartValueAxes = new ChartValueAxes();
    chartArea.ChartValueAxes.Add(createChartAxis("Primary"));
    chartArea.ChartValueAxes.Add(createChartAxis("Secondary"));
    return chartArea;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Render** the generated report in ReportViewer

```
}  
BoldReports.RDL.DOM.ChartAxis createChartAxis(string Name)  
{  
    var ChartAxis = new ChartAxis();  
    ChartAxis.Style = new Style();  
    ChartAxis.AllowLabelRotation = AllowLabelRotation.None;  
    ChartAxis.Angle = "0";  
    ChartAxis.Arrows = Arrows.None;  
    ChartAxis.ChartAxisScaleBreak = new ChartAxisScaleBreak();  
    ChartAxis.ChartAxisTitle = new ChartAxisTitle();  
    ChartAxis.ChartMajorGridLines = new ChartMajorGridLines();  
    ChartAxis.ChartMajorTickMarks = new ChartMajorTickMarks();  
    ChartAxis.ChartMinorGridLines = new ChartMinorGridLines();  
    ChartAxis.ChartMinorTickMarks = new ChartMinorTickMarks();  
    ChartAxis.ChartStripLines = new ChartStripLines();  
    ChartAxis.CrossAt = "NaN";  
    ChartAxis.CustomProperties = new CustomProperties();  
    ChartAxis.LineStyle = LineStyle.Solid;  
    ChartAxis.Name = Name;  
    return ChartAxis;  
}  
`
```

[Render the generated report in ReportViewer](#)

You can render the report using the ReportViewer, after the report definition is created. The following code snippet illustrates how to render the report using the ReportViewer by report definition.

```
`csharp  
[NonAction]  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
    var report = CreateReport();  
    reportOption.ReportModel.ReportDefinition = reportDefinition;  
}  
`
```

How to pass multiple values using custom data

Pass the multiple data values as JSON in `ajaxBeforeLoad` event using the 'data' property, which needs to be serialized in the server side API using known class type and `JsonConvert.DeserializeObject`.

1. Map the `ajaxBeforeLoad` event with the `onAjaxRequest` function in the client to pass custom data to the server.

```
`html
<bold-report-viewer id="viewer"
report-service-url="/api/ReportViewer"
processing-mode="Remote"
report-path="sales-order-detail.rdl"
ajax-before-load="onAjaxRequest">
</bold-report-viewer>
`js
<script type="text/javascript">
function onAjaxRequest(args) {
//Passing custom data to server
}
</script>
```

2. You need to pass the multiple custom data values as JSON and use the `data` property to send custom data to the server in the Ajax request.

```
`js
function onAjaxRequest(args) {
//Passing custom data to server
var jsonData = {
customerID: "CI0021",
productID: "PO0022"
};
args.data = jsonData;
}
```

3. The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates deserialization of the JSON string and change the data source connection strings based on Customer ID and Product ID in the `OnInitReportOptions` method.

```
`csharp
public SampleData customDatas = new SampleData();
public class SampleData
{
    public string customerID { get; set; }
    public string productID { get; set; }
}
[HttpPost]
public object PostReportAction([FromBody]Dictionary<string, object> jsonResult)
{
    if (jsonResult.ContainsKey("customData"))
    {
        //Get client side JSON custom data, desirialize it and store in local variable.
        customDatas = JsonConvert.DeserializeObject<SampleData>(jsonResult["customData"].ToString());
    }
    return ReportHelper.ProcessReport(jsonResult, this, this._cache);
}
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (customDatas != null)
    {
        if (customDatas.customerID == "CI0021" && customDatas.productID == "PO0022") {
            //If you are changing the connection string based on customer id then could you please change the
            connection string as below.

            //reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

            reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
            Catalog=<database>;"));
        }
    }
}
```

```
}  
`
```

How to resolve InvalidOperationException when creating a new application

Add the **MemoryCache** in **startup** file to resolve the InvalidOperationException when creating the new ASP.NET core application. You can find the following code for your reference.

```
`csharp  
  
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddMemoryCache();  
}  
`
```

How to load the report from the database

You must load the report using the **Stream** option available with **reportOption.ReportModel**. To load the report from the database, please follow these steps:

From byte array

```
`csharp  
  
[NonAction]  
  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
    byte[] bytes = ""; // provide your byte array report data  
    MemoryStream reportStream = new MemoryStream(bytes);  
    reportOption.ReportModel.Stream = reportStream;  
}  
`
```

From base64String

```
`csharp  
  
[NonAction]  
  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
    string base64String = ""; // provide your base64 report data  
    byte[] bytes = System.Convert.FromBase64String(base64String);  
    MemoryStream reportStream = new MemoryStream(bytes);  
}
```

```
reportOption.ReportModel.Stream = reportStream;
}
```

From string

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string reportData = ""; // provide your database report data
    byte[] bytes = System.Text.Encoding.ASCII.GetBytes(reportFile);
    MemoryStream reportStream = new MemoryStream(bytes);
    reportOption.ReportModel.Stream = reportStream;
}
```

How to clear cache when closing the Report Viewer

By using the `clearReportCache` and `destroy` method, you can clear the server side cache. You have to use this method when closing report viewer and switching to another report within your application.

CSHTML

Add following code in `index.cshtml` file.

```
`js
<div id="reportviewer"></div>
<script>
var isSubmit = true;
$(document.body).bind('submit', $.proxy(this.formSubmit, this));
function formSubmit(args)
{
    isSubmit = false;
}
window.onbeforeunload = function () {
    if (isSubmit) {
        var reportviewerObj = $("#reportviewer").data("boldReportViewer");
        reportviewerObj.clearReportCache();
        reportviewerObj.destroy();
    }
}
```

```

}
isSubmit = true;
};
</script>
`

```

Web API

In the `PostReportAction` method, you have to collect the GC with `ClearCache` as shown in the following code sample.

```

`csharp
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
{
    bool isclearcache = false;

    if (jsonArray != null && jsonArray.ContainsKey("reportAction") && jsonArray["reportAction"].ToString()
    == "ClearCache")
    {
        isclearcache = true;
    }

    var reportresult = ReportHelper.ProcessReport(jsonArray, this, this._cache);
    if (isclearcache)
    {
        GC.Collect(); isclearcache = false;
    }

    return reportresult;
}
`

```

Migrate to Report Viewer v2.0 component

This section provides simple step-by-step instructions to update your existing Report Viewer application to our latest modern v2.0 scripts, styles and components.

1. Open the `\Views\Shared_Layout.cshtml` page that contains Report Viewer and its scripts.
2. Remove the following older scripts and CSS references in your `_Layout.cshtml` page.
 - o `bold.reports.all.min.css`
 - o `jquery.min.js`
 - o `ej2-base.min.js`
 - o `ej2-data.min.js`
 - o `ej2-pdf-export.min.js`

- ej2-svg-base.min.js
- ej2-lineargauge.min.js
- ej2-circulargauge.min.js
- ej2-maps.min.js
- ej.chart.min.js
- bold.reports.common.min.js
- bold.reports.widgets.min.js
- bold.report-viewer.min.js

3. Add the following v2.0 scripts and CSS references in your `_Layout.cshtml` page.

```
`html
<!-- Report Viewer component styles -->
<link href="https://cdn.boldreports.com/5.4.20/content/v2.0/tailwind-light/bold.report-viewer.min.css"
rel="stylesheet" />
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<!-- Report Viewer component dependent script -->
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.widgets.min.js"></script>
<!-- Report Viewer component script -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/bold.report-viewer.min.js"></script>
`
```

The final updated `_Layout.cshtml` page for v2.0 Report Viewer. looks as follows.

```
`html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link href="https://cdn.boldreports.com/5.4.20/content/v2.0/tailwind-light/bold.report-viewer.min.css"
rel="stylesheet" />
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/common/bold.reports.widgets.min.js"></script>
`
```



```
<script src="https://cdn.boldreports.com/5.4.20/scripts/v2.0/bold.report-viewer.min.js"></script>
</head>
<body>
<div style="min-height: 600px;width: 100%;">
@RenderBody()
</div>
@RenderSection("Scripts", required: false)
<!-- Bold Reports script manager -->
<bold-script-manager></bold-script-manager>
</body>
</html>
`
```

Frequently asked questions

This section helps to get the answer for the frequently asked questions in Bold Reports ASP.NET Core Report Viewer.

1. [How can improve the performance and handle the large amounts of data with Report Viewer?](#)
2. [Is Report Viewer compatible with latest version of JQuery library?](#)
3. [Is the back action from drillthrough report will load the report again with Report Viewer?](#)
4. [Is Report Viewer supports with .NET Core on Linux Docker?](#)
5. [Is possible to change the culture of the date time parameter?](#)
6. [Is it possible to hide report parameters?](#)
7. [Is it possible to hide the export options in Report Viewer?](#)
8. [Is it possible to load reports providing parameter values at runtime?](#)

Does Report Viewer supports .NET Core on Linux Docker

Yes, Report Viewer supports .NET Core on Linux Docker. You can get more details about this from [Bold Reports Embedded Reporting Tools support .NET Core and Docker on Linux](#).

Is possible to change the culture of the date time parameter

Yes, we can change the culture of the date time parameter for Report Viewer by changing the locale. You can make use the following reference to change the locale of Report Viewer.

[ReportViewer Localization](#)

In Report Viewer, we could not change the culture of specific parameter or UI. So, we have to achieve the requirements only by changing the culture.

Is it possible to hide the parameters in Report Viewer

Yes, it is possible to hide the parameters in Report Viewer. On hiding the parameters, the users will not be able to have the parameter block in the Report Viewer. Refer to this [Hide parameter block and toolbar items](#) section.

Is it possible to hide the export options in Report Viewer

Yes, it is possible to hide the export options in Report Viewer. The Report Viewer provides the `exportOptions` property to show or hide the default export types available in the component. Refer to this [Decide or Hide the export options](#) section.

Is it possible to load reports providing parameter values at runtime

Yes, it is possible to load reports with application inputs as parameters in Report Viewer. You need to provide the input values to the Report Viewer from client side at runtime using the `parameters` property, which accepts JSON array values. Refer to this [Set parameter at client](#) section.

Refer Bold Reports Scripts and Themes from BoldReports.Javascript NuGet in .NET Core application

In this section, we are going to dive into

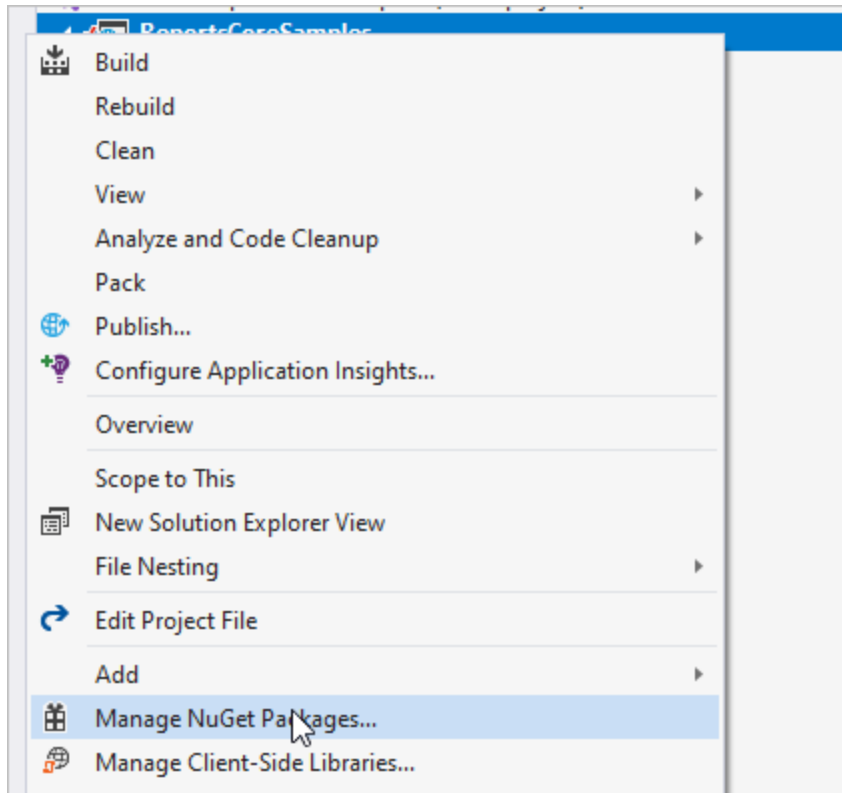
Find and install the package

First, we need to download the `BoldReports.Javascript` NuGet package,

- In Solution Explorer, right-click either References or the project and select `Manage Nuget Packages`.

Refer Bold Reports Scripts and Themes from BoldReports.Javascript NuGet in .NET Core application

Installed Location



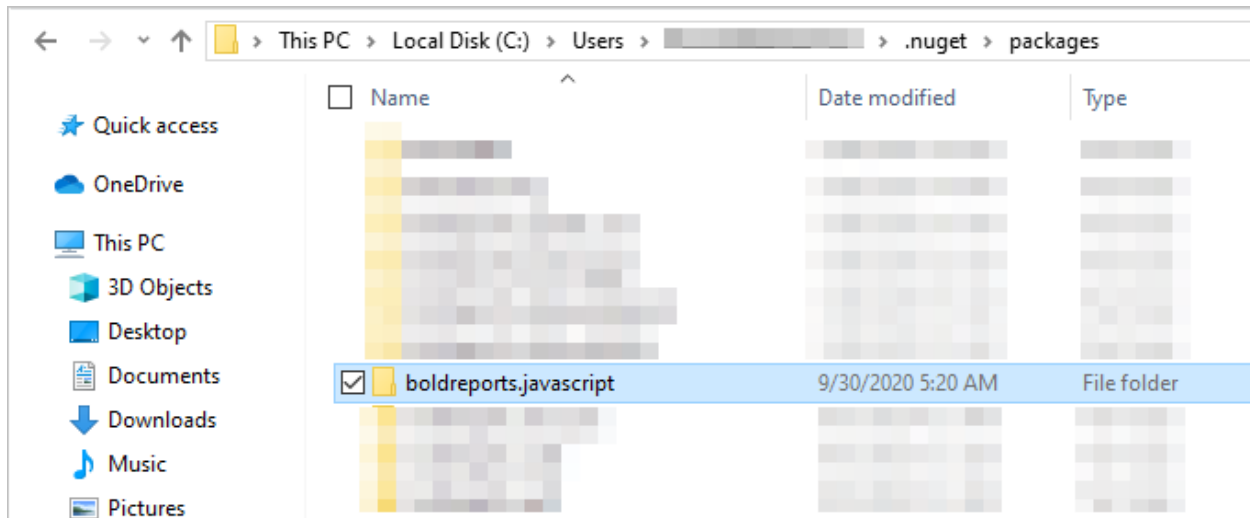
- Under the browser tab search for **BoldReports.Javascript** package and install the required version.



Installed Location

The installed NuGet package will be available under the below path

C:\Users\{CurrentUser}\.nuget\packages\



Copy and refer Scripts and CSS files into project

- Copy the Scripts and CSS files from the installed path to our project location inside 'wwwroot' folder.
- Refer to the below image for reference.



Bold Report Writer

Report Writer is a class library that enables the user to render reports defined in Microsoft's RDL format (2008 or 2008 R2) as PDF, Word, Excel or CSV documents.

The important features of ASP.NET Core Report Writer are listed as follows:

- RDL Specification - Supports RDL specification for the SQL Server 2008 and RDL specification for the SQL Server 2008 R2 only. List of available report definition formats:
[msdn.microsoft.com/library/dd297486\(SQL.100\)](https://msdn.microsoft.com/library/dd297486(SQL.100)).

- Data sources - You can use advanced database servers Data Sources in Report Writer (SQL and Oracle).
- Charts - Show all basic types of charts that are available in Microsoft RDL reports.
- Tablix - Shows the summaries and simple tables.
- Gauge - Shows measurement values by using expression values.
- Textbox - Shows textbox data with expression support.
- Export - Export report as PDF, Word, Excel and CSV.
- Report Parameter - Views the report based on the report parameter value.

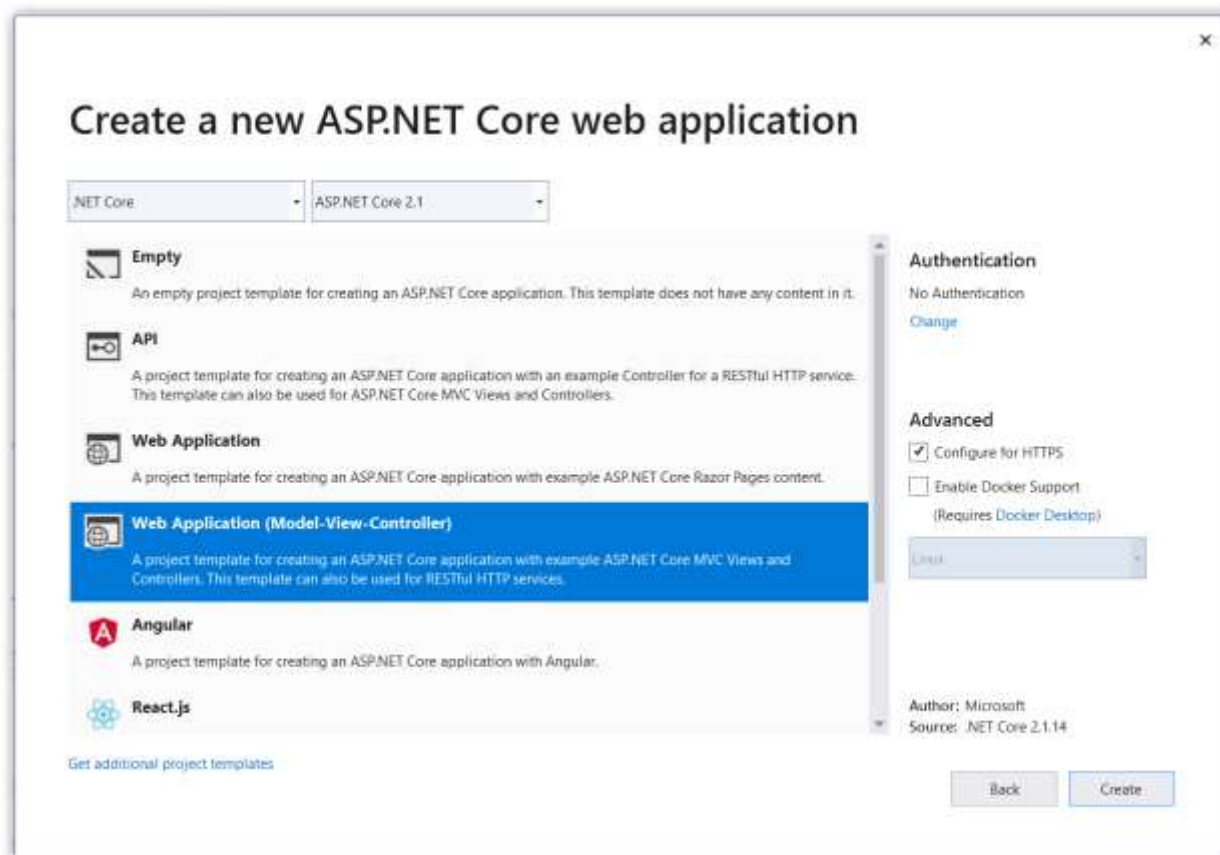
Export SSRS RDL Report in Bold Reports ASP.NET Core Report Writer

The Report Writer is a class library that is used to export the RDL report with popular file formats like PDF, Microsoft Word, Microsoft CSV, and Microsoft Excel without previewing the report in the webpage. This section describes how to export the RDL report in an ASP.NET Core application using the **Report Writer**.

Bold Reports ASP.NET Core Report Writer works on the **ASP .NET Core 2.1**, **ASP.NET Core 2.2**, and **ASP.NET Core 3.x** versions.

Create an ASP.NET Core application

1. Start Visual Studio 2019 and click **Create new project**.
2. Choose **ASP.NET Core Web Application**, and then click **Next**.
3. Change the project name, and then click **Create**.
4. In the dropdown for the **ASP.NET Core version**, choose **ASP.NET Core 2.1**.
5. Select the **Web Application (Model-View-Controller)** template, then click **Create**.



List of dependency libraries

1. In the Solution Explorer tab, right-click the project or solution, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.Net.Core** and **System.Data.SqlClient** packages, and install them in your Core application. The following table provides details about the packages and their usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Exports the report to a PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core**, and **Syncfusion.XlsIO.Net.Core** packages.

Syncfusion.Pdf.Net.Core | Exports the report to a PDF.

Syncfusion.DocIO.Net.Core | Exports the report to a Word.

Syncfusion.XlsIO.Net.Core | Exports the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is the base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serializes and deserializes data for the Report Writer. It is a mandatory package for Report Writer, and the package version should be 10.0.1 or higher.

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded [here](#). You can get the reports from the Bold Reports installation location. For more information, refer to the [samples and demos](#) section.

Server side Report Writer changes

1. Create a folder `Resources` in the `wwwroot` folder in your application. Copy and paste the sample RDL reports into the `Resources` folder.
2. Create local variables inside the `HomeController` class.

```
`csharp
// IWebHostEnvironment used with sample to get the application data from wwwroot.
private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
// IWebHostEnvironment initialized with controller to get the data from application data folder.
public HomeController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
{
    _hostingEnvironment = hostingEnvironment;
}
`
```

3. Open the `HomeController.cs` file in your application and add the `Export()` function to load the report as a stream. Refer to the following code snippet.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    // Here, we have loaded the sales-order-detail sample report from application the folder
    wwwroot\Resources.

    FileStream inputStream = new FileStream(_hostingEnvironment.WebRootPath + @"\Resources\sales-
order-detail.rdl", FileMode.Open, FileAccess.Read);

    MemoryStream reportStream = new MemoryStream();

    inputStream.CopyTo(reportStream);

    reportStream.Position = 0;

    inputStream.Close();

    .....
}
`
```

4. Initialize the Report Writer instance with a report stream and set the specified export format and file name for the export document.

```
`csharp
public IActionResult Export(string writerFormat)
{
    .....
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    string fileName = null;
    WriterFormat format;
    string type = null;
    fileName = "sales-order-detail.pdf";
    type = "pdf";
    format = WriterFormat.PDF;
}
`
```

5. You can use the **Save** method in the Report Writer to generate the export document along with the information of the report stream, it will return the generated file as Stream.

```
`csharp
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
`
```

6. Refer to the following complete code snippet used to get the exported file stream.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
```


// Here, we have loaded the sales-order-detail sample report from application the folder
wwwroot\Resources.

```
FileStream inputStream = new FileStream(_hostingEnvironment.WebRootPath + @"\Resources\sales-  
order-detail.rdl", FileMode.Open, FileAccess.Read);
```

```
MemoryStream reportStream = new MemoryStream();
```

```
inputStream.CopyTo(reportStream);
```

```
reportStream.Position = 0;
```

```
inputStream.Close();
```

```
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
```

```
string fileName = null;
```

```
WriterFormat format;
```

```
string type = null;
```

```
if (writerFormat == "PDF")
```

```
{
```

```
    fileName = "sales-order-detail.pdf";
```

```
    type = "pdf";
```

```
    format = WriterFormat.PDF;
```

```
}
```

```
else if (writerFormat == "Word")
```

```
{
```

```
    fileName = "sales-order-detail.docx";
```

```
    type = "docx";
```

```
    format = WriterFormat.Word;
```

```
}
```

```
else if (writerFormat == "CSV")
```

```
{
```

```
    fileName = "sales-order-detail.csv";
```

```
    type = "csv";
```

```
    format = WriterFormat.CSV;
```

```
}
```

```
else
```

```
{
```

```
    fileName = "sales-order-detail.xlsx";
```

```
    type = "xlsx";
```

```

format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}
`

```

Client side changes

1. Use the following code snippet in `Index.cshtml` home page to invoke the Web API from the client side.

```

`html
@{Html.BeginForm("Export", "Home", FormMethod.Post);
{
<div>
<input type="submit" value="Generate" style="width: 150px;" />
</div>
}
Html.EndForm();
}
`

```

2. You can add the export file types to the home page to choose the format you want to export in the Report Writer. Copy and paste the following code snippet in your application.

```

`html
@{Html.BeginForm("Export", "Home", FormMethod.Post);
{
<div class="Common">
<div class="tablediv">

```

```
<div class="rowdiv">
<label id="design">
Choose the any one of the format to export the document in Report Writer.
<br />
<br />
</label>
</div>
<div class="rowdiv">
<div class="celldiv" style="padding:10px">
<label>
<strong> Save As :</strong>
</label>
<input id="rbtnPDF" type="radio" name="writerFormat" value="PDF" checked="checked" style="margin-left: 15px" />
<label for="rbtnPDF" style="padding:0px 5px 0px 2px">
PDF
</label>
<input id="rbtnWord" type="radio" name="writerFormat" value="Word" style="margin-left: 15px" />
<label for="rbtnWord" style="padding:0px 5px 0px 2px">
Word
</label>
<input id="rbtnxls" type="radio" name="writerFormat" value="xls" style="margin-left: 15px" />
<label for="rbtnxls" style="padding:0px 5px 0px 2px">
Excel
</label>
<input id="rbtnCSV" type="radio" name="writerFormat" value="CSV" style="margin-left: 15px" />
<label for="rbtnCSV" style="padding:0px 25px 0px 2px ">
CSV
</label>
<input class="buttonStyle" type="submit" name="button" value="Generate" style="width:150px;" />
</div>
</div>
</div>
```

```
</div>
```

```
Html.EndForm();
```

```
}}
```

```
,
```

3. Now, Run and export the report with a specified export format in your Report Writer application.

To learn more about export a report with data visualization report items, refer to the [Export data visualization report items](#) section.

- Congratulations! You've completed your first ASP.NET Core Writer application! Click [here](#) to download the already created ASP.NET Core Report Writer application.

Note: You can refer to our feature tour page for the [ASP.NET Core Report Writer](#) to see its innovative features. Additionally, you can view our [ASP.NET Core Report Writer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

Export RDLC Report

You can export the RDLC reports that exist on the local file system with custom business object data collection. The following steps demonstrates how to export a RDLC report using Report Writer.

This section requires an ASP.NET Core Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

Bind data source in Web API controller

The following steps help you to configure the Web API to render the RDLC report with business object data collection.

1. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
```

```
{
List<ProductList> datas = new List<ProductList>();
ProductList data = new ProductList()
{
    ProductName = "Baked Chicken and Cheese",
    OrderId = "323B60",
    Price = 55,
    Category = "Non-Veg",
    Ingredients = "grilled chicken, corn and olives.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Delite",
    OrderId = "323B61",
    Price = 100,
    Category = "Non-Veg",
    Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
```

```

}
}
,

```

2. In this tutorial, the `Product List.rdlc` report is used, and it can be downloaded [here](#). Copy and paste the sample RDLC reports into the `wwwroot/Resources` folder. For more information, see [Samples and demos](#).
3. Create a folder `Resources` into the `wwwroot` folder in your application. Copy and paste the sample RDLC reports into the `Resources` folder.
4. Open the `HomeController.cs` file in your application and add the `Export()` function to load the report as stream. Refer to the following code snippet.

```

`csharp
public class HomeController : Controller
{
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // IWebHostEnvironment initialized with controller to get the data from application data folder.
    public HomeController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _hostingEnvironment = hostingEnvironment;
    }
    [HttpPost]
    public IActionResult Export(string writerFormat)
    {
        // Here, we have loaded the Product List.rdlc sample report from application the Resources folder.
        FileStream inputStream = new FileStream(_hostingEnvironment.WebRootPath + @"\Resources\Product
List.rdlc", FileMode.Open, FileAccess.Read);
        MemoryStream reportStream = new MemoryStream();
        inputStream.CopyTo(reportStream);
        reportStream.Position = 0;
        inputStream.Close();
        .....
    }
}
,

```

5. Initialize the Report Writer instance with created `reportStream` and Set the value of the `ReportProcessingMode` property value `ProcessingMode.Local` to provide the dataset collection for that RDLC report.

```
`csharp
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.ReportProcessingMode = ProcessingMode.Local;
`
```

6. Bind the business object data values collection by adding a new item to the `DataSources` as in the following code snippet.

```
`csharp
//Pass the dataset collection for report
writer.DataSources.Clear();

writer.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list", Value =
ProductList.GetData() });
`
```

Data source `Name` is case sensitive and it should be same as in the data source name in the report definition and the `Value` accepts `IList`, `DataSet`, and `DataTable` inputs.

7. You can use the `Save` method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as `Stream`.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    // Here, we have loaded the Product List.rdlc sample report from application the Resources folder.
    FileStream inputStream = new FileStream(_hostingEnvironment.WebRootPath + @"\Resources\Product
List.rdlc", FileMode.Open, FileAccess.Read);

    MemoryStream reportStream = new MemoryStream();

    inputStream.CopyTo(reportStream);

    reportStream.Position = 0;

    inputStream.Close();

    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();

    writer.ReportProcessingMode = ProcessingMode.Local;

    // Pass the dataset collection for report
```

```
writer.DataSources.Clear();

writer.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list", Value =
ProductList.GetData() });

string fileName = null;
WriterFormat format;
string type = null;
if (writerFormat == "PDF")
{
    fileName = "Product List.pdf";
    type = "pdf";
    format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
    fileName = "Product List.docx";
    type = "docx";
    format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
    fileName = "Product List.csv";
    type = "csv";
    format = WriterFormat.CSV;
}
else
{
    fileName = "Product List.xlsx";
    type = "xlsx";
    format = WriterFormat.Excel;
}

writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
```



```
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileName = fileName;
return fileStreamResult;
}
```

8. Now, the RDLC report exported successfully in your application.

Export SSRS Report Server Report

Report Writer has support to Export RDL reports into popular file formats such as PDF, Microsoft Word, Microsoft Excel, and CSV from SSRS Report Server.

To export the SSRS Reports, set the `ReportProcessingMode`, `ReportServerURL`, and `ReportPath` properties in Web API as shown in the following steps.

1. To create your first ASP.NET Core Report Writer application, refer to the [Getting-started](#) section.
2. Set the `reportProcessignMode` API for Bold Report Writer as shown in the following code snippet.

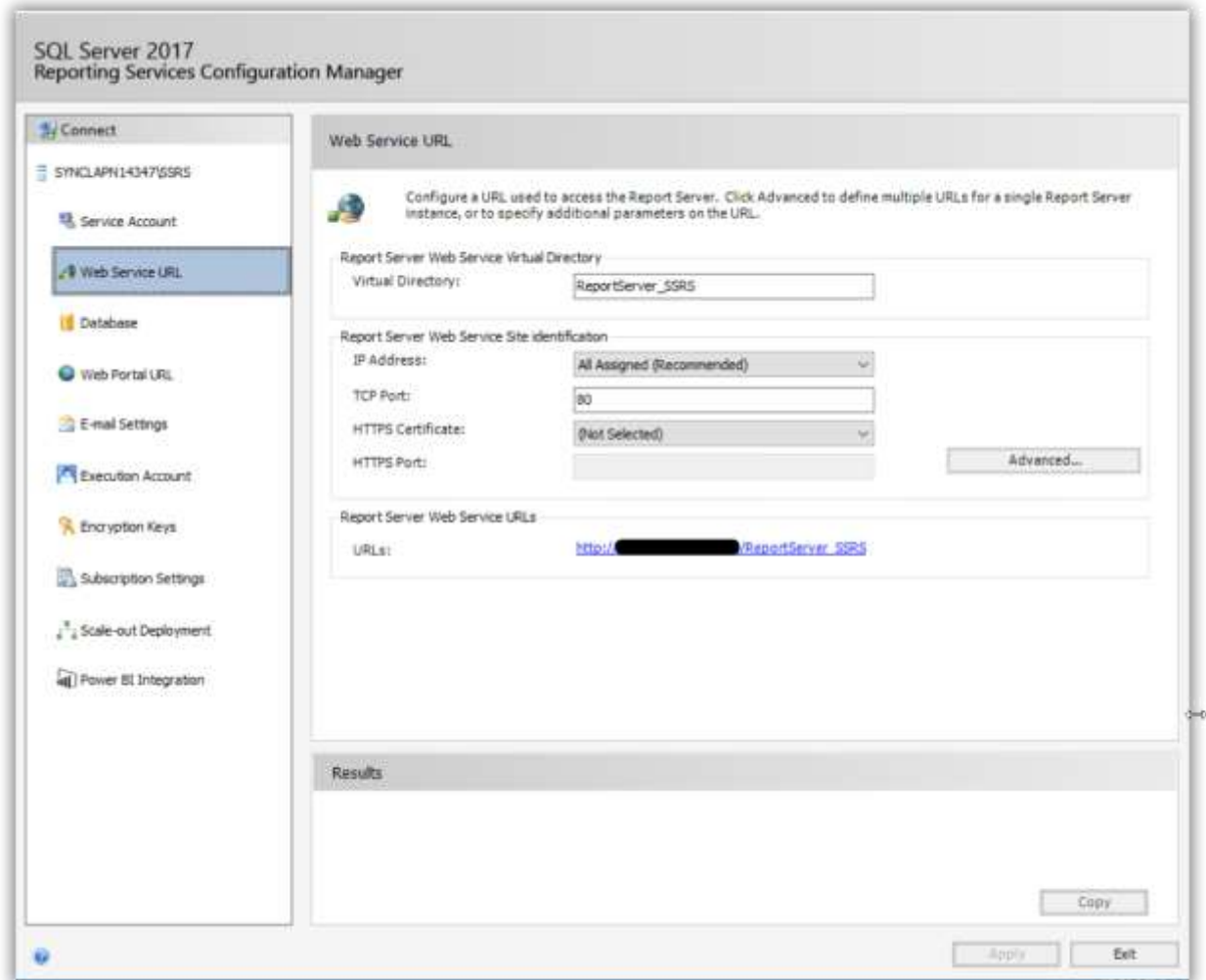
```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
}
```

3. Set the `reportServerUrl` API on Bold Report Writer with `WebServiceURL` in the WebAPI as shown in the following code snippet.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
```

```
writer.ReportServerUrl = "http://<servername>/Reports_SSRS";
}
```

The Web Service URL should be set as `reportServerUrl` in the report writer configuration. The Web Service URL can be found from the Reporting Services Configuration manager under the **Web Service URL** section, as shown in the following image.

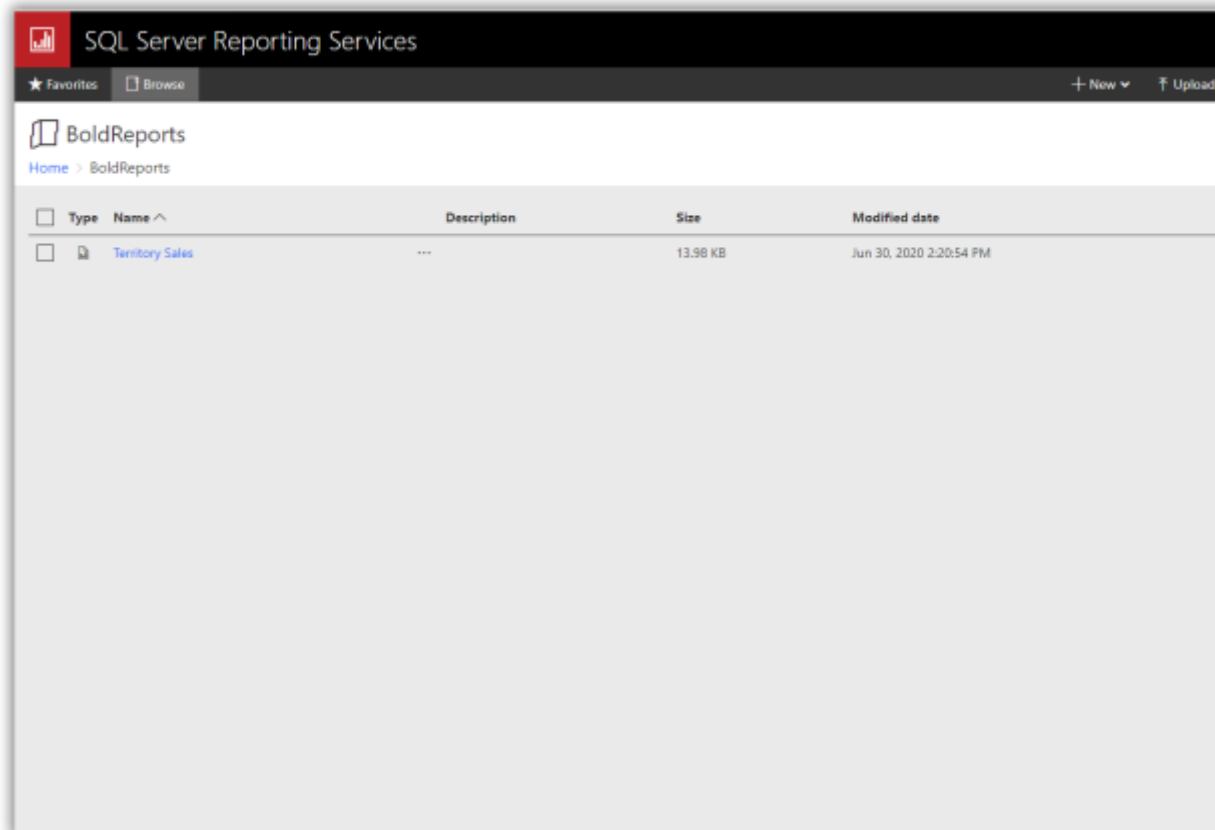


4. Set the report path for loading the reports from the SSRS Report Server. The report path should be in the format of `/folder name/report name`.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
```

```
writer.ReportProcessingMode = ProcessingMode.Remote;
writer.ReportServerUrl = "http://<servername>/Reports_SSRS";
writer.ReportPath = "/SSRSSamples2/Territory Sales";
}
\
```

The report path can be found from the SSRS Report Server by navigating to the path of the report to be loaded, as shown in the following image.



Network credentials for SSRS

The network credentials are required to load specified SSRS report from the specified SSRS Report Server using the Report Writer. Specify the `ReportServerCredential` property in writer instance.

```
`csharp
writer.ReportServerCredential = new System.Net.NetworkCredential("username", "password");
\
```

If you are facing problem to access the SSRS Report server reports, you can refer [How to provide the permission for user to access the SSRS Report Server reports](#).

Set data source credential to shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS Server. If the report has any data source that uses credentials to connect with the

database, then you should specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp
List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new
List<BoldReports.Web.DataSourceCredentials>();

dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",
"username", "password"));

writer.SetDataSourceCredentials(dataSourceCredentialsList);
`
```

Data source credentials should be added to the shared data sources that do not have credentials in the connection strings.

Refer to the following final code snippet example to export the SSRS report server reports.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
    writer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
    writer.ReportPath = "/Report_Path";
    writer.ReportServerCredential = new System.Net.NetworkCredential("username", "password");
    List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new
    List<BoldReports.Web.DataSourceCredentials>();
    dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("Datasource name",
    "username", "password"));
    writer.SetDataSourceCredentials(dataSourceCredentialsList);
    string fileName = null;
    WriterFormat format;
    string type = null;
    if (writerFormat == "PDF")
    {
        fileName = "sales-order-detail.pdf";
        type = "pdf";
        format = WriterFormat.PDF;
    }
}
```

```
else if (writerFormat == "Word")
{
    fileName = "sales-order-detail.docx";
    type = "docx";
    format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
    fileName = "sales-order-detail.csv";
    type = "csv";
    format = WriterFormat.CSV;
}
else
{
    fileName = "sales-order-detail.xlsx";
    type = "xlsx";
    format = WriterFormat.Excel;
}

MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}
,
```

Export SharePoint Server Report

Report Writer has support to Export RDL reports into popular file formats like PDF, Microsoft Word, Microsoft Excel, and CSV from SharePoint server reports.

This section requires an ASP.NET Core Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

To export the SharePoint server reports, Initialize the Report Writer and set the `ReportProcessingMode`, `ReportServerURL`, and `ReportPath` properties in Web API as shown in the following code snippet.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
    writer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
    writer.ReportPath = "http://<servername>/reportserver$instanceName/Report_Path";
}
`
```

In SharePoint server, the `ReportServerUrl` will be same as your site URL. The `ReportPath` is relative to the Report Server URL with the file extension.

Forms credential for SharePoint Server

The forms credentials are required to load the SharePoint integrated SSRS report from the specified SharePoint integrated SSRS Report Server using the Report Viewer. Specify the `ReportServerFormsCredential` property to access the share point reports.

```
`csharp
writer.ReportServerFormsCredential = new
BoldReports.Web.ReportServerFormsCredential("username", "password");
`
```

Set data source credential to shared data sources

The shared data source credentials can be added to the `DataSourceCredentials` property to connect with the database.

```
`csharp
List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new
List<BoldReports.Web.DataSourceCredentials>();

dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",
"username", "password"));

writer.SetDataSourceCredentials(dataSourceCredentialsList);
`
```

Data source credentials should be added to shared data sources that do not have credentials in the connection strings.

Refer to the following final code snippet example to export the SharePoint server reports.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
    writer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
    writer.ReportPath = "http://<servername>/reportserver$instanceName/Report_Path";
    writer.ReportServerFormsCredential = new
    BoldReports.Web.ReportServerFormsCredential("username", "password");
    List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new
    List<BoldReports.Web.DataSourceCredentials>();
    dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",
    "username", "password"));
    writer.SetDataSourceCredentials(dataSourceCredentialsList);
    string fileName = null;
    WriterFormat format;
    string type = null;
    if (writerFormat == "PDF")
    {
        fileName = "sales-order-detail.pdf";
        type = "pdf";
        format = WriterFormat.PDF;
    }
    else if (writerFormat == "Word")
    {
        fileName = "sales-order-detail.docx";
        type = "docx";
        format = WriterFormat.Word;
    }
    else if (writerFormat == "CSV")
    {
        fileName = "sales-order-detail.csv";
        type = "csv";
    }
}
```

```

format = WriterFormat.CSV;
}
else
{
fileName = "sales-order-detail.xlsx";
type = "xlsx";
format = WriterFormat.Excel;
}
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}
`

```

Export Subreport

You can export a Main report with subreport with popular file formats such as PDF, Microsoft Word, and Microsoft Excel without previewing the report in webpage using ASP.NET Report Writer application.

This section requires an ASP.NET Core Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

Export RDL Subreport

In this tutorial, the `SideBySideMainReport.rdl`, `SideBySideSubReport.rdl` reports is used, and it can be downloaded [here](#). You can get the report from Bold Reports installation location. The reports used from installed location requires `NorthwindIO_Reports.sdf` database to run, so add the database to your application. For more information, refer to [Samples and demos](#).

Refer to the following steps to export the RDL sub report with specified format.

1. Create a folder `Resources` into the `wwwroot` folder in your application. Copy and paste the sample RDL reports into the `Resources` folder.
2. Open the `HomeController.cs` file in your application and add the `Export()` function to load the report as stream. Refer to the following code snippet.

```

`csharp
public class HomeController : Controller

```



```

{
// IWebHostEnvironment used with sample to get the application data from wwwroot.
private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
// IWebHostEnvironment initialized with controller to get the data from application data folder.
public HomeController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
{
    _hostingEnvironment = hostingEnvironment;
}
[HttpPost]
public IActionResult Export(string writerFormat)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sample reports from application the wwwroot\Resources folder.
    FileStream inputMainStream = new FileStream(basePath + @"\Resources\SideBySideMainReport.rdl",
        FileMode.Open, FileAccess.Read);
    MemoryStream mainReportStream = new MemoryStream();
    inputMainStream.CopyTo(mainReportStream);
    mainReportStream.Position = 0;
    inputMainStream.Close();
    FileStream inputSubStream = new FileStream(basePath + @"\Resources\SideBySideSubReport.rdl",
        FileMode.Open, FileAccess.Read);
    MemoryStream subReportStream = new MemoryStream();
    inputSubStream.CopyTo(subReportStream);
    subReportStream.Position = 0;
    inputSubStream.Close();
    .....
}
}
,

```

3. Initialize the Report Writer instance and pass the subreport name and stream to the `LoadSubreport()` method in Report Writer instance.

```
`csharp
```

```
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
```

```
writer.LoadSubreport("SideBySideSubReport", subReportStream);
`
```

4. After loading the main report stream using Report Writer instance. Refer to the following code snippet.

```
`csharp
writer.LoadReport(mainReportStream);
`
```

5. You can use the **Save** method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as Stream.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sample reports from application the wwwroot\Resources folder.
    FileStream inputMainStream = new FileStream(basePath + @"\Resources\SideBySideMainReport.rdl",
        FileMode.Open, FileAccess.Read);
    MemoryStream mainReportStream = new MemoryStream();
    inputMainStream.CopyTo(mainReportStream);
    mainReportStream.Position = 0;
    inputMainStream.Close();
    FileStream inputSubStream = new FileStream(basePath + @"\Resources\SideBySideSubReport.rdl",
        FileMode.Open, FileAccess.Read);
    MemoryStream subReportStream = new MemoryStream();
    inputSubStream.CopyTo(subReportStream);
    subReportStream.Position = 0;
    inputSubStream.Close();
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.LoadSubreport("SideBySideSubReport", subReportStream);
    writer.LoadReport(mainReportStream);
    string fileName = null;
    WriterFormat format;
    string type = null;
}
```

```
if (writerFormat == "PDF")
{
    fileName = "SideBySideMainReport.pdf";
    type = "pdf";
    format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
    fileName = "SideBySideMainReport.docx";
    type = "docx";
    format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
    fileName = "SideBySideMainReport.csv";
    type = "csv";
    format = WriterFormat.CSV;
}
else
{
    fileName = "SideBySideMainReport.xlsx";
    type = "xlsx";
    format = WriterFormat.Excel;
}
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}
```

6. Now, the RDL report exported successfully in your application.

Export RDLC subreport

To export the RDLC report along with subreport, we need to load the subreport streams before loading a main report to the Report Writer as in the following code example, then only subreport processing event will work.

```
`csharp
[HttpPost]
public IActionResult Pdf()
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sample report from application the folder wwwroot.
    FileStream inputMainStream = new FileStream(basePath + @"\Reports\MainReport.rdlc",
        FileMode.Open, FileAccess.Read);
    MemoryStream mainReportStream = new MemoryStream();
    inputMainStream.CopyTo(mainReportStream);
    mainReportStream.Position = 0;
    inputMainStream.Close();

    FileStream inputsub1Stream = new FileStream(basePath + @"\Reports\SubReport1.rdlc",
        FileMode.Open, FileAccess.Read);
    MemoryStream subreport1Stream = new MemoryStream();
    inputsub1Stream.CopyTo(subreport1Stream);
    subreport1Stream.Position = 0;
    inputsub1Stream.Close();

    FileStream inputsub2Stream = new FileStream(basePath + @"\Reports\SubReport2.rdlc",
        FileMode.Open, FileAccess.Read);
    MemoryStream subreport2Stream = new MemoryStream();
    inputsub2Stream.CopyTo(subreport2Stream);
    subreport2Stream.Position = 0;
    inputsub2Stream.Close();

    ReportWriter writer = new ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Local;
    writer.SubreportProcessing += ReportWriter_SubreportProcessing;
    //Adding sub report stream going to be used with exporting report.
    writer.LoadSubreport("SubReport1", subreport1Stream);
}
```

```

writer.LoadSubreport("SubReport2", subreport2Stream);
//Loading the report going to export as PDF.
writer.LoadReport(mainReportStream);
writer.DataSources.Clear();
writer.DataSources.Add(new ReportDataSource { Name = "DataSet", Value = MainReport.GetData() });
// Steps to generate PDF report using Report Writer.
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, BoldReports.Writer.WriterFormat.PDF);
// Download the generated from client.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/pdf");
fileStreamResult.FileName = "Invoice.pdf";
return fileStreamResult;
}
private void ReportWriter_SubreportProcessing(object sender, SubreportProcessingEventArgs e)
{
    if (e.ReportPath == "SubReport1")
    {
        //Pass the dataset collection for subreport
        e.DataSources.Clear();
        e.DataSources.Add(new ReportDataSource { Name = "DataSet1", Value = SubReport1.GetData() });
    }
    else if (e.ReportPath == "SubReport2")
    {
        //Pass the dataset collection for subreport
        e.DataSources.Clear();
        e.DataSources.Add(new ReportDataSource { Name = "DataSet2", Value = SubReport2.GetData() });
    }
}

```

Export Parameter Report

You can set report parameter default values or modify the values using the `SetParameters()` method of Report Writer instance. This section describes how to modify the exported document report parameter values.

This section requires an ASP.NET Core Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

In this tutorial, the `sales-order-detail.rdl` report is used and it can be downloaded [here](#). You can get the reports from Bold Reports installation location. For more information, refer to [samples and demos](#) section.

Set report parameters to export file

1. Create a folder `Resources` into the `wwwroot` folder in your application. Copy and paste the sample RDL reports into the `Resources` folder.
2. To load the report as stream from the application `Resources` folder using the `FileStream` class.

```
`csharp
public class HomeController : Controller
{
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // IWebHostEnvironment initialized with controller to get the data from application data folder.
    public HomeController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _hostingEnvironment = hostingEnvironment;
    }
    [HttpPost]
    public IActionResult Export(string writerFormat)
    {
        // Here, we have loaded the sample reports from application the wwwroot\Resources folder.
        FileStream inputStream = new FileStream(_hostingEnvironment.WebRootPath + @"\Resources\sales-order-detail.rdl", FileMode.Open, FileAccess.Read);
        MemoryStream reportStream = new MemoryStream();
        inputStream.CopyTo(reportStream);
        reportStream.Position = 0;
        inputStream.Close();
        BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
```

```
writer.ReportProcessingMode = ProcessingMode.Remote;
```

```
.....
```

```
}
```

```
}
```

```
,
```

3. You can add collection of report parameters programmatically using the **ReportParameter** property that is the list type of Report Parameter class. Refer to the following code snippet to set the report parameter name and value.

```
`csharp
```

```
List<BoldReports.Web.ReportParameter> userParameters = new
```

```
List<BoldReports.Web.ReportParameter>();
```

```
userParameters.Add(new BoldReports.Web.ReportParameter()
```

```
{
```

```
Name = "SalesOrderNumber",
```

```
Values = new List<string>() { "SO50756" }
```

```
});
```

```
.....
```

```
.....
```

```
writer.SetParameters(userParameters);
```

```
,
```

4. You can use the **Save** method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as Stream.

```
`csharp
```

```
[HttpPost]
```

```
public IActionResult Export(string writerFormat)
```

```
{
```

```
// Here, we have loaded the sample reports from application the wwwroot\Resources folder.
```

```
FileStream reportStream = new FileStream(_hostingEnvironment.WebRootPath + @"\Resources\sales-order-detail.rdl", FileMode.Open, FileAccess.Read);
```

```
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
```

```
writer.ReportProcessingMode = ProcessingMode.Remote;
```

```
List<BoldReports.Web.ReportParameter> userParameters = new
```

```
List<BoldReports.Web.ReportParameter>();
```

```
userParameters.Add(new BoldReports.Web.ReportParameter()
{
    Name = "SalesOrderNumber",
    Values = new List<string>() { "SO50756" }
});
string fileName = null;
WriterFormat format;
string type = null;
if (writerFormat == "PDF")
{
    fileName = "sales-order-detail.pdf";
    type = "pdf";
    format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
    fileName = "sales-order-detail.docx";
    type = "docx";
    format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
    fileName = "sales-order-detail.csv";
    type = "csv";
    format = WriterFormat.CSV;
}
else
{
    fileName = "sales-order-detail.xlsx";
    type = "xlsx";
    format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
```



```

writer.SetParameters(userParameters);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileName = fileName;
return fileStreamResult;
}

```

The Report Parameters name should be case sensitive

5. Now, the parameter report exported successfully in your application.

Report Writer Events

You can handle the Report Writer events with reports using the following events.

- SubreportProcessing
- ReportErrorOccurred

Subreport Processing

The SubreportProcessing event occurs when subreport is loaded and it is ready to start the processing. You can handle the event and specify the data source, parameters, and subreport loading. The following sample code loads a subreport by assigning the report data source credentials and parameter values in the SubreportProcessing event.

1. Report Writer instance used to register the SubreportProcessing event in your Report Writer application.

```

`csharp
writer.SubreportProcessing += (sen, arg) =>
{
// Assign the data source and parameter values to the sub report.
};

```

2. Set the subreport data source details to the subreport processing event argument.

```

`csharp

```

```
writer.SubreportProcessing += (sen, arg) =>
{
    arg.DataSources.Add(new BoldReports.Web.ReportDataSource {Name= "Datasource name", Value =
    value });
}
`
```

Data source **Name** is case sensitive and it should be same as in the data source name in the report definition. The **Value** accepts IList, DataSet, and DataTable inputs.

3. Set the subreport parameters value to the subreport processing event argument.

```
`csharp
writer.SubreportProcessing += (sen, arg) =>
{
    arg.Parameters.Add(new BoldReports.Web.ReportParameterInfo { Name = "Parameter name", Values =
    values});
}
`
```

Report Error

The **ReportErrorOccurred** event raises when an error occurs in the report processing. You can handle the event and get the report error details from the event arguments. Refer to the following sample code to handle the report errors in the **ReportErrorOccurred** event.

```
`csharp
writer.ReportErrorOccurred += (sen, arg) =>
{
    // You can handle the report errors using event arguments.
};
`
```

Error logging in ASP.NET Core Report Writer

If an error occurred in report processing, ASP.NET Core Report writer **ReportErrorOccurred** event raised. You can register the event and get the report error details from the event arguments to save all logs, stack trace, and error information into a physical file location.

This section explains how to log the detailed error information to your ASP.NET Core application.

This section requires an ASP.NET Core Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

1. In Solution Explorer, open the Report Writer Controller file.

2. Add the following sample code to register the report errors in the `ReportErrorOccurred` event.

```
`csharp
[HttpPost]
public IActionResult Export(string writerFormat)
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportErrorOccurred += Writer_ReportErrorOccurred;
}

private void Writer_ReportErrorOccurred(object sender, ReportErrorOccurredEventArgs e)
{
    // You can register the report errors using event arguments.
}
`
```

3. Create a method in `HomeController` to write the error text into application folder.

```
`csharp
internal void WriteLogs(string errorMessage)
{
    string filePath = Path.Combine(_hostingEnvironment.WebRootPath, "ErrorDetails.txt");
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
        writer.WriteLine(errorMessage);
    }
}
`
```

4. Invoke the newly created function in `ReportErrorOccurred` as follows.

```
`csharp
private void Writer_ReportErrorOccurred(object sender, ReportErrorOccurredEventArgs e)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2}", e.ClassName,
        e.MethodName, e.Message));
}
```

```
}  
`
```

In cases of any issues faced in the report rendering, share the log file to our technical support team to get assistance on that.

5. The final controller is given as follows, you can replace it in your application.

```
`csharp  
[HttpPost]  
public IActionResult Export(string writerFormat)  
{  
    // Here, we have loaded the sales-order-detail sample report from application the folder  
    wwwroot\Resources.  
  
    FileStream inputStream = new FileStream(_hostingEnvironment.WebRootPath + @"\Resources\sales-  
order-detail.rdl", FileMode.Open, FileAccess.Read);  
  
    MemoryStream reportStream = new MemoryStream();  
  
    inputStream.CopyTo(reportStream);  
  
    reportStream.Position = 0;  
  
    inputStream.Close();  
  
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();  
  
    writer.ReportErrorOccurred += Writer_ReportErrorOccurred;  
  
    string fileName = null;  
  
    WriterFormat format;  
  
    string type = null;  
  
    if (writerFormat == "PDF")  
    {  
        fileName = "sales-order-detail.pdf";  
  
        type = "pdf";  
  
        format = WriterFormat.PDF;  
    }  
  
    else if (writerFormat == "Word")  
    {  
        fileName = "sales-order-detail.docx";  
  
        type = "docx";  
  
        format = WriterFormat.Word;  
    }  
}
```

```
}
else if (writerFormat == "CSV")
{
    fileName = "sales-order-detail.csv";
    type = "csv";
    format = WriterFormat.CSV;
}
else
{
    fileName = "sales-order-detail.xlsx";
    type = "xlsx";
    format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}

private void Writer_ReportErrorOccurred(object sender, ReportErrorOccurredEventArgs e)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2}", e.ClassName,
        e.MethodName, e.Message));
}

internal void WriteLogs(string errorMessage)
{
    string filePath = Path.Combine(_hostingEnvironment.WebRootPath, "ErrorDetails.txt");
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
```

```
writer.WriteLine(errorMessage);  
}  
}  
`
```

Encrypt and Secure Documents

Encrypt and Secure Documents option allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password. The following code snippet explains how to encrypt the exported document with the user defined password.

Password Protected PDF document

You can protect the exported PDF document using the following code snippet.

```
`csharp  
writer.PDFOptions = new BoldReports.Writer.PDFOptions();  
writer.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity  
{  
    UserPassword = "Password"  
};  
`
```

Password Protected Word document

You can protect the exported Word document using the following code snippet.

```
`csharp  
writer.WordOptions = new BoldReports.Writer.WordOptions()  
{  
    EncryptionPassword = "password"  
};  
`
```

Password Protected Excel document

You can protect the exported Excel document using the following code snippet.

```
`csharp  
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()  
{  
    PasswordToModify = "password",  
    PasswordToOpen = "password"  
};  
`
```

Advance Layouts For Merged Cells

Report Writer supports additional export layouts in Word and Excel export formats to eliminate the tiny columns, rows, and merged cells. The benefits of the layout are:

- Overcomes the merging problem with tiny cells, rows, and columns in SSRS Excel and Word exporting.
- Provides clear readability by eliminating the tiny columns, rows, and merge cells.
- Allows you to perform data manipulations such as applying sorting, filters, and grouping in Excel.

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
writer.WordOptions.LayoutOption = WordLayoutOptions.TopLevel;
writer.WordOptions.ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
    Bottom = 0.5f,
    Top = 0.5f
};
`
```

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior. The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, add an empty paragraph between two tables.

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` as `IgnoreCellMerge`.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
};
`
```

Change the Default File Format

Users can change the default file format to any other file format that is supported by the selected file type. It includes APIs to set the formats before exporting the document. Refer to the following section to change the default file formats.

Change the Word document type

You can save the report to the required Word document version by setting the `FormatType` property.

```
`csharp
writer.WordOptions = new BoldReports.Writer.WordOptions()
{
    FormatType = WordFormatType.Docx
};
`
```

Change the Excel document type

You can save the report to the required Excel document version by setting the `ExcelSaveType` property.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013
};
`
```

Change the CSV document type

You can save the report to the required CSV file extension by setting the `FileExtension` property.

```
`csharp
writer.CsvOptions = new BoldReports.Writer.CsvOptions()
{
    FileExtension = ".txt"
};
`
```

PDF Settings

The PDF export options provides properties to manage PDF export behaviors. You can set the customization properties in the `PDFOptions`. Refer to the following code snippet to initialize the `PDFOptions` property.

```
`csharp
writer.PDFOptions = new PDFOptions();
`
```

Export with complex scripts

To export reports with the complex script language texts, set the `ComplexScript` property of `PDFOptions` instance to `true`.


```
`csharp
```

```
writer.PDFOptions.EnableComplexScript = true;
```

```
,
```

PDF conformance

You can export the report as a PDF/A-1b document by specifying the PdfConformanceLevel.Pdf_A1B conformance level in the PdfConformanceLevel property.

```
`csharp
```

```
writer.PDFOptions.PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B;
```

```
,
```

Add custom fonts to PDF document

This section explains the steps to add custom fonts to the PDF exported document. The Report Writer provides Fonts property in PDFOptions to add the new fonts.

Any fonts used in the report that is not installed or not available in the local system, then you must add the font stream to Fonts property.

The following points should be followed while adding new fonts:

1. Key should be same as in FontFamily element of the report item.
2. Key is case sensitive, so casing is must match with FontFamily.
3. To add streams for different font style, weight, styles consider the below,
 - Key should be in the format of {fontname} {weight} {style}. For example Roboto Light Italic.
 - Use only single white space between font name, weight, and style.
 - When FontStyle is set to Normal then use style as Regular. For example Roboto Light Regular.

```
`csharp
```

```
//Load missing font stream
```

```
writer.PDFOptions.Fonts = new Dictionary<string, System.IO.Stream>
```

```
{
```

```
{ "Roboto", new FileStream(basePath + @"\fonts\roboto\Roboto.ttf", FileMode.Open, FileAccess.Read) },
```

```
{ "Roboto Bold", new FileStream(basePath + @"\fonts\roboto\Roboto-Bold.ttf", FileMode.Open, FileAccess.Read) },
```

```
{ "Roboto Bold Regular", new FileStream(basePath + @"\fonts\roboto\Roboto-Bold.ttf", FileMode.Open, FileAccess.Read) },
```

```
{ "Roboto Light Italic", new FileStream(basePath + @"\fonts\roboto\Roboto-Light-Italic.ttf", FileMode.Open, FileAccess.Read) },
```

```
{ "Roboto Thin", new FileStream(basePath + @"\fonts\roboto\Roboto-Thin.ttf", FileMode.Open, FileAccess.Read) }
```

```
};  
`
```

In the above code, loaded the `ttf` streams for `Roboto` font with different style and weight combinations.

Password Protected PDF document

Allows you to protect the exported PDF document from unauthorized users by encrypting the document using encryption password. The following code snippet explains how to encrypt the exported document with user-defined password.

```
`csharp  
writer.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity  
{  
    UserPassword = "Password"  
};  
`
```

Excel Settings

The Excel export options provides properties to manage Excel document export behaviors. You can set the customization properties in the `ExcelOptions`. Refer to the following code snippet to initialize the `ExcelOptions` property.

```
`csharp  
writer.ExcelOptions = new ExcelOptions();  
`
```

Excel document type

You can save the report to the required Excel version by setting the `ExcelSaveType` property.

```
`csharp  
writer.ExcelOptions.ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013;  
`
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` as `IgnoreCellMerge`.

```
`csharp  
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()  
{  
    LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge  
};  
`
```

Protecting Excel document from editing

You can restrict the Excel document from editing by providing the `ExcelSheetProtection` or enabling the `ReadOnlyRecommended` properties.

```
`csharp
writer.ExcelOptions.ReadOnlyRecommended = true;
writer.ExcelOptions.ExcelSheetProtection = Syncfusion.XlsIO.ExcelSheetProtection.DeletingColumns;
`
```

Change Excel export format

Allows you to change the default file format to any other file format using the `ExcelSaveType` properties.

```
`csharp
writer.ExcelOptions.ExcelSaveType = ExcelVersion.Excel2013;
`
```

Password Protected Excel document

Allows you to protect the exported Excel document from unauthorized users by encrypting the document using the encryption password. The following code snippet explains how to encrypt the exported document with user-defined password.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    PasswordToModify = "password",
    PasswordToOpen = "password"
};
`
```

Word Settings

The Word export options provides properties to manage Word document export behaviors. You can set the customization properties in the `WordOptions`. Refer to the following code snippet to initialize the `WordOptions` property.

```
`csharp
writer.WordOptions = new WordOptions();
`
```

Word document type

You can save the report to the required document version by setting the `FormatType` property.

```
`csharp
writer.WordOptions.FormatType = WordFormatType.Docx;
`
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
writer.WordOptions.LayoutOption = WordLayoutOptions.TopLevel;
writer.WordOptions.ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
    Bottom = 0.5f,
    Top = 0.5f
};
```

A paragraph element is inserted between two tables in the exported document to overcome the Word document auto merging behavior. The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, add an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without password. The following are the types of protection:

- `AllowOnlyComments`: Adds or modifies only the comments in the Word document.
- `AllowOnlyFormFields`: Modifies the form field values in the Word document.
- `AllowOnlyRevisions`: Accepts or rejects the revisions in the Word document.
- `AllowOnlyReading`: Views the content only in the Word document.
- `NoProtection`: Accesses or edits the Word document contents as normally.

```
`csharp
writer.WordOptions.ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading;
```

Password Protected Word document

Allows you protect the exported Word document from unauthorized users by encrypting the document using encryption password. The following code snippet explains how to encrypt the exported document with user-defined password.

```
`csharp
writer.WordOptions = new BoldReports.Writer.WordOptions()
{
    EncryptionPassword = "password"
```

```
};  
`
```

CSV Settings

The `CsvOptions` allows you to change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document. You should set the customization properties in the `CsvOptions` property of Report Writer instance.

```
`csharp  
writer.CsvOptions = writer.CsvOptions = new BoldReports.Writer.CsvOptions()  
{  
    Encoding = System.Text.Encoding.Default,  
    FieldDelimiter = ",",  
    UseFormattedValues = false,  
    Qualifier = "#",  
    RecordDelimiter = "@",  
    SuppressLineBreaks = true,  
    FileExtension = ".txt"  
};  
`
```

Export Data Visualization in Core Environment

Report Writer uses [WebBrowser Control](#) to export the data visualization to PDF, PPT, Word, Excel, and HTML file formats. The `WebBrowser` is not supported in Core environment. To overcome this limitation in Core environment, we have provided the below options to export the data visualization report item.

1. [External browser - Puppeteer](#).
2. [PhantomJS - Classic Tool](#)

`External Browser - Puppeteer`

1. It supports `ASP.NET Core 2.1` or above version only.
2. Open the Report Writer application and install the `PuppeteerSharp` package from [NuGet Gallery](#).
3. Create a folder like `wwwroot/ResourcesTemp` in your Report Writer application.
4. Open the Report Writer application Web API file.
5. Set the `BrowserTypes` Enum property to External.
6. Create and initialize `CustomBrowserType` by inheriting the `ExportSettings` class.
7. Set the `ResourcePath`, `Scripts` and `DependentScripts` property in Report Writer instance.
The following code example demonstrates how to configure `Puppeteer` in your ASP.NET Core application.

In this tutorial, the `product-line-sales.rdl` report is used, and it can be downloaded in this [link](#).

```
`csharp
public class HomeController : Controller
{
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // IWebHostEnvironment initialized with controller to get the data from application data folder.
    public HomeController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _hostingEnvironment = hostingEnvironment;
    }
    public IActionResult Index()
    {
        return View();
    }
    [HttpPost]
    public IActionResult Export(string writerFormat)
    {
        string fileName = null;
        WriterFormat format;
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sample report from application the folder wwwroot.
        FileStream inputStream = new FileStream(basePath + @"\Resources\product-line-sales.rdl",
        FileMode.Open, FileAccess.Read);
        MemoryStream reportStream = new MemoryStream();
        inputStream.CopyTo(reportStream);
        reportStream.Position = 0;
        inputStream.Close();
        BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
        // Initialize the class for puppeteer
        writer.ExportSettings = new CustomBrowserType();
        // Puppeteer Option to export the data visualization report items.
        writer.ExportResources.BrowserType = ExportResources.BrowserTypes.External;
```

```
writer.ExportResources.ResourcePath = basePath + @"/ResourcesTemp/";
writer.ExportResources.Scripts = new List<string>
{
    //Gauge component scripts
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-lineargauge.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-circulargauge.min.js",
    //Map component script
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js",
    //Chart component script
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js",
    //Report Viewer Script
    "https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js",
};
writer.ExportResources.DependentScripts = new List<string>
{
    "https://code.jquery.com/jquery-1.10.2.min.js"
};
if (writerFormat == "PDF")
{
    fileName = "ProductLineSales.pdf";
    format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
    fileName = "ProductLineSales.docx";
    format = WriterFormat.Word;
}
```

```
else if (writerFormat == "CSV")
{
    fileName = "ProductLineSales.CSV";
    format = WriterFormat.CSV;
}
else
{
    fileName = "ProductLineSales.xlsx";
    format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated document from client.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/pdf");
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}
// Add the class to override the method for puppeteer
public class CustomBrowserType : ExportSettings
{
    // Can be used to convert the image from external browser
    public override string GetImageFromHTML(string url)
    {
        return LaunchPuppeteer(url).Result;
    }
    // Can be used to convert the image from Puppeteer
    public async Task<string> LaunchPuppeteer(string url)
    {
        var browserFetcher = new BrowserFetcher();
        await browserFetcher.DownloadAsync();
        await using var browser = await Puppeteer.LaunchAsync(new LaunchOptions { Headless = true });
    }
}
```



```

await using var page = await browser.NewPageAsync();
await page.GoToAsync(url);

return await
page.WaitForSelectorAsync("#imagejsonData").Result.GetPropertyAsync("innerText").Result.JsonValueA
sync<string>();
}
}
}
`

```

The **Scripts** and **Dependent** scripts must be added to export the data visualization report items.

`PhantomJS - Classic Tool`

It supports **ASP.NET Core 2.1** or above version only.

To download **PhantomJS** application and deploy it on your machine, you should accept its license terms on [LICENSE](#) and [Third-Party](#) document.

1. Download PhantomJS for windows [here](#) and for linux [here](#) and extract the download file.
 2. Copy the PhantomJS file from the extracted bin folder and paste into **wwwroot/PhantomJS** folder in your application.
 3. Open the Report Writer application Web API file.
 4. Set the **UsePhantomJS** property to **true**.
 5. Set the **PhantomJSPath**, **Scripts** and **DependentScripts** property in **Report Writer** instance.
- The following code example demonstrates how to configure **PhantomJS** in your ASP.NET Core application.

In this tutorial, the **product-line-sales.rdl** report is used, and it can be downloaded in this [link](#).

```

`csharp
public class HomeController : Controller
{
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // IWebHostEnvironment initialized with controller to get the data from application data folder.
    public HomeController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _hostingEnvironment = hostingEnvironment;
    }

    public IActionResult Index()
    {

```

```
return View();
}
[HttpPost]
public IActionResult Export(string writerFormat)
{
    string fileName = null;
    WriterFormat format;
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sample report from application the folder wwwroot.
    FileStream inputStream = new FileStream(basePath + @"\Resources\product-line-sales.rdl",
    FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    inputStream.CopyTo(reportStream);
    reportStream.Position = 0;
    inputStream.Close();
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    // PhantomJS Option to export the data visualization report items.
    writer.ExportResources.UsePhantomJS = true;
    // Set the IncludeText property to true, when text not displayed in the data visualization images.
    writer.ExportResources.IncludeText = true;
    writer.ExportResources.PhantomJSPath = basePath + @"\PhantomJS\";
    writer.ExportResources.Scripts = new List<string>
    {
        //Gauge component scripts
        "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js",
        "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js",
        "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js",
        "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js",
        "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-lineargauge.min.js",
        "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-circulargauge.min.js",
        //Map component script
        "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js",
        "https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js",
```

```
"https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js",
//Chart component script
"https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js",
//Report Viewer Script
"https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js",
};
writer.ExportResources.DependentScripts = new List<string>
{
"https://code.jquery.com/jquery-1.10.2.min.js"
};
if (writerFormat == "PDF")
{
fileName = "ProductLineSales.pdf";
format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
fileName = "ProductLineSales.docx";
format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
fileName = "ProductLineSales.CSV";
format = WriterFormat.CSV;
}
else
{
fileName = "ProductLineSales.xlsx";
format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
```

```
// Download the generated document from client.  
memoryStream.Position = 0;  
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/pdf");  
fileStreamResult.FileName = fileName;  
return fileStreamResult;  
}  
}  
`
```

The **Scripts** and **Dependent** scripts must be added to export the data visualization report items.

Limitations

Linux

Report Writer uses **PhantomJS.exe** for data visualization report items rendering, which is not supported with Linux environment. So, visualization report items will not be available in Report Writer generated document.

Custom code

Custom code in a report allows to include new custom constants, variables, functions, or subroutines. As the VBCodeProvider is not available in .NET Core platform, report with custom codes requires additional settings to enable this feature. You can make use of the following solution using custom codes with Report Writer.

- [How to use custom code with Report Writer](#)

Data source

For RDL, built-in data source processing support is available only for SQL data source. For other datasources, you have to create a custom data extension for data processing.

PDF fonts

.NET Core platform does not have a support to get font details from system environment to embed report fonts with PDF document. So, Report Writer will generate PDF documents only with TimesNewRoman font. You have to provide report font details as separate with PDF Options to get the PDF documents with expected font used in Report. Refer [Add custom fonts to PDF document](../pdf-settings/#Add-custom-fonts-to-pdf-document

) to resolve this problem.

Excel export

Microsoft Excel has a limitation that you could not create a Excel document with more then 1,048,576 rows and 16,384 columns. So, as per limitation Report Writer does not support to create the Excel document from report above of 1,048,576 rows and 16,384 columns.

Image format

PNG and **JPEG** format images only supports in the ASP.NET Core Report Writer for PDF export. This limitation is from our **Syncfusion.Pdf.Net.Core** supports.

How to section for Report Writer component

1. [how to use custom code with Report Writer](#)
2. [Generate and export RDL and RDLC reports programmatically](#)
3. [How to combine the exported pdf with another pdf file](#)
 - o [How to load the report from database?](#)

This section is specifically for Bold Reports version less than 3.x. If you are using a higher version of [Bold Reports](#), recommend you to embed the VB code directly in the report itself, as VBCodeProvider is now available in the .NET Core platform.

How to use custom code with Report Writer

Custom code in a report allows to include new custom constants, variables, functions, or subroutines. As the VBCodeProvider is not available in .NET Core platform, report with custom codes requires additional settings to enable this feature. This section describes the steps required to use custom codes with Report Writer.

Create a custom code class file

1. Create a new C# class file in your application.
2. The namespace should be **BoldReports.Processing.Helper** and class declaration should be **public static class Code**.
3. Write the C# methods equivalent to your report VB codes.
4. Here is the example code to convert the value to USD.

```
`csharp
namespace BoldReports.Processing.Helper
{
    public static class Code
    {
        public static string PrintHelloWorld()
        {
            return "Hello World";
        }
    }
}
```

Create a assembly reference and add it to Report Writer

If you have created the custom code class, then you need to add the assemblies in the list with what you have used in your application to Report Writer as shown in following code example.

```
`csharp
```

```
using (var reportWriter = new ReportWriter())
{
    reportWriter.Assemblies.Add(this.GetType().GetTypeInfo().Assembly);
    reportWriter.LoadReport(inputStream);
}
,
```

If you want to use the custom code feature, then you should not use the **ReportWriter(Stream ReportStream)** constructor to load the report and assemblies added with **Assemblies** property before using **LoadReport** method.

How to generate and export RDL and RDLC reports programmatically

The Bold Reports reporting library allows you to generate and export RDL and RDLC reports programmatically. The **ReportDefinition** class is defined as an object model of a report. You can create, add, and modify the report properties, report sections like header and footer and report items in the report definition instance.

The following steps illustrates how to create a basic report object model:

Initialize a report definition

The following code snippet guides you in initializing the report definition.

```
`csharp
ReportDefinition CreateReport()
{
    ReportDefinition report = new ReportDefinition();
    report.ReportSections = new ReportSections();
    var reportSection = new ReportSection();
    report.ReportSections.Add(reportSection);
    reportSection.Width = new BoldReports.RDL.DOM.Size("6in");
    report.ReportUnitType = "Inch";
    report.RDLType = RDLType.RDL2010;
    return report;
}

BoldReports.RDL.DOM.Style AddStyle()
{
    BoldReports.RDL.DOM.Style style = new BoldReports.RDL.DOM.Style();
    style.Border = new BoldReports.RDL.DOM.Border();
    style.Border.Width = new BoldReports.RDL.DOM.Size("1pt");
```

```
style.Border.Style = "Solid";  
style.Border.Color = "Black";  
return style;  
}  
`
```

Create a Data source for report

The following code snippet guides you in creating a **Datasource** for the report and setting values to their properties.

```
`csharp  
ReportDefinition CreateReport()  
{  
...  
...  
...  
reportSection.Page = new BoldReports.RDL.DOM.Page();  
reportSection.Page.Style = AddStyle();  
reportSection.Page.PageHeight = "4in";  
reportSection.Page.PageWidth = "6in";  
var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =  
<database>");  
report.DataSources = new DataSources();  
report.DataSources.Add(newDataSource);  
...  
...  
...  
}  
//If you are using RDLC report then you can pass any string value as connection string  
BoldReports.RDL.DOM.DataSource CreateDataSource(string name, string dataProvider, string  
connectionString)  
{  
var dataSource = new BoldReports.RDL.DOM.DataSource();  
dataSource.Name = name;  
dataSource.SecurityType = BoldReports.RDL.DOM.SecurityType.Integrated;  
dataSource.ConnectionProperties = new BoldReports.RDL.DOM.ConnectionProperties();
```

```
dataSource.ConnectionProperties.DataProvider = dataProvider;
dataSource.ConnectionProperties.ConnectionString = connectionString;
dataSource.ConnectionProperties.IntegratedSecurity = true;
return dataSource;
}
`
```

Create a Dataset for the report

The following code snippet guides you in creating a **Dataset** for the report and setting values to their properties.

```
`csharp
```

```
ReportDefinition CreateReport()
```

```
{
```

```
...
```

```
...
```

```
...
```

```
reportSection.Page.PageWidth = "6in";
```

```
var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog = <database>");
```

```
report.DataSources = new DataSources();
```

```
report.DataSources.Add(newDataSource);
```

```
var newDataSet = CreateDataSet("DataSource1", "DataSet1");
```

```
report.DataSets = new DataSets();
```

```
report.DataSets.Add(newDataSet);
```

```
...
```

```
...
```

```
...
```

```
}
```

```
BoldReports.RDL.DOM.DataSet CreateDataSet(string dataSourceName, string dataSetName)
```

```
{
```

```
var dataSet = new BoldReports.RDL.DOM.DataSet();
```

```
dataSet.Name = dataSetName;
```

```
dataSet.Query = new Query();
```

```
dataSet.Query.DataSourceName = dataSourceName;
```



```
dataSet.Query.CommandText = "SELECT
HumanResources.Department.DepartmentID,HumanResources.Department.Name,HumanResources.De
partment.GroupName,HumanResources.Department.ModifiedDate FROM
HumanResources.Department";

dataSet.Query.QueryDesignerState = new QueryDesignerState();
var table = CreateTable();
dataSet.Query.QueryDesignerState.Tables = new List<Table>();
dataSet.Query.QueryDesignerState.Tables.Add(table);
dataSet.Fields = new Fields();
dataSet.Fields.Add(CreateField("DepartmentID", "System.Int16"));
dataSet.Fields.Add(CreateField("Name", "System.String"));
dataSet.Fields.Add(CreateField("GroupName", "System.String"));
dataSet.Fields.Add(CreateField("ModifiedDate", "System.DateTime"));
return dataSet;
}

BoldReports.RDL.DOM.Table CreateTable()
{
var table = new Table();
table.Schema = "HumanResources";
table.Name = "Department";
table.Columns = new List<Column>();
table.Columns.Add(CreateColumn("DepartmentID"));
table.Columns.Add(CreateColumn("Name"));
table.Columns.Add(CreateColumn("GroupName"));
table.Columns.Add(CreateColumn("ModifiedDate"));
return table;
}

BoldReports.RDL.DOM.Column CreateColumn(string columnName)
{
var column = new Column();
column.Name = columnName;
return column;
}

BoldReports.RDL.DOM.Field CreateField(string fieldName, string type)
```

```
{  
var field = new Field();  
field.Name = fieldName;  
field.TypeName = type;  
field.DataField = fieldName;  
return field;  
}  
`
```

Create a report page header

The following code snippet guides you in creating a **PageHeader** report section and setting values to their properties.

```
`csharp  
ReportDefinition CreateReport()  
{  
...  
...  
...  
reportSection.Page = new BoldReports.RDL.DOM.Page();  
reportSection.Page.Style = AddStyle();  
reportSection.Page.PageHeight = "4in";  
reportSection.Page.PageWidth = "6in";  
reportSection.Page.PageHeader = CreateHeader();  
...  
...  
...  
}  
PageHeader CreateHeader()  
{  
PageHeader pageHeader = new PageHeader();  
pageHeader.Height = new BoldReports.RDL.DOM.Size("0.59167in");  
pageHeader.Style = AddStyle();  
pageHeader.PrintOnFirstPage = true;  
pageHeader.PrintOnLastPage = true;  
pageHeader.ReportItems = new ReportItems();  
}
```

```
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
return pageHeader;
}
\
```

Create a report page footer

The following code snippet guides you in creating a **PageFooter** report section and setting values to their properties.

```
\csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Page.PageHeader = CreateFooter();
...
...
...
}
PageFooter CreateFooter()
{
PageFooter pageFooter = new PageFooter();
pageFooter.Style = AddStyle();
pageFooter.Height = new BoldReports.RDL.DOM.Size("0.59167in");
pageFooter.ReportItems = new ReportItems();
pageFooter.PrintOnFirstPage = true;
pageFooter.PrintOnLastPage = true;
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
return pageFooter;
}
\
```

Initialize a report body section

- Initialize a report body object and set the values to their properties like height, styles, and report items as shown in the following code snippet.

```
`csharp
ReportDefinition CreateReport()
{
    ...
    ...
    ...
    reportSection.Body = CreateBody();
    ...
    ...
    ...
}
Body CreateBody()
{
    var body = new Body();
    body.Height = new BoldReports.RDL.DOM.Size("2.03333in");
    body.Style = AddStyle();
    body.ReportItems = new ReportItems();
    var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
    body.ReportItems.Add(textBox);
    return body;
}
`
```

- Using the following code snippets, you can create a **Textbox** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

```
`csharp
PageHeader CreateHeader()
{
    ...
    ...
}
```

```
...
pageHeader.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
...
...
...
}
PageFooter CreateFooter()
{
...
...
...
pageFooter.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
...
...
...
}
Body CreateBody()
{
...
...
...
body.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
body.ReportItems.Add(textBox);
...
...
...
}
```

BoldReports.RDL.DOM.TextBox CreateTextBox(string name, string text, string left, string top, string width, string height)

```
{
var textBox = new BoldReports.RDL.DOM.TextBox();
textBox.Name = name;
textBox.Height = new BoldReports.RDL.DOM.Size(height);
textBox.Width = new BoldReports.RDL.DOM.Size(width);
textBox.Left = new BoldReports.RDL.DOM.Size(left);
textBox.Top = new BoldReports.RDL.DOM.Size(top);
textBox.Style = new BoldReports.RDL.DOM.Style();
textBox.Style.TextAlign = "Center";
textBox.Style.VerticalAlign = "Top";
textBox.Paragraphs = new Paragraphs();
BoldReports.RDL.DOM.Paragraph paragraph = new BoldReports.RDL.DOM.Paragraph();
TextRuns runs = new TextRuns();
TextRun run = new TextRun();
run.Style = new BoldReports.RDL.DOM.Style();
run.Style.FontStyle = "Default";
run.Style.TextAlign = "Center";
run.Style.FontFamily = "Arial";
run.Style.FontSize = new BoldReports.RDL.DOM.Size("10pt");
run.Value = text;
runs.Add(run);
paragraph.Style = new BoldReports.RDL.DOM.Style();
paragraph.Style.VerticalAlign = "Top";
paragraph.Style.TextAlign = "Center";
paragraph.TextRuns = runs;
textBox.Paragraphs.Add(paragraph);
return textBox;
}
```

Create a Table for the report

- Using the following code snippets, you can create a **Table** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

```
`csharp
```

```
BoldReports.RDL.DOM.Tablix CreateTablix(string name, string text, string left, string top, string width, string height)
```

```
{  
var tableItem = new BoldReports.RDL.DOM.Tablix();  
tableItem.Name = name;  
tableItem.Height = new BoldReports.RDL.DOM.Size(height);  
tableItem.Width = new BoldReports.RDL.DOM.Size(width);  
tableItem.Left = new BoldReports.RDL.DOM.Size(left);  
tableItem.Top = new BoldReports.RDL.DOM.Size(top);  
tableItem.Style = new BoldReports.RDL.DOM.Style();  
tableItem.Style.Border = new BoldReports.RDL.DOM.Border();  
tableItem.Style.Border.Style = "Solid";  
tableItem.DataSetName = "DataSet1";  
tableItem.TablixBody = new TablixBody();  
tableItem.TablixBody.TablixColumns = new TablixColumns();  
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());  
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());  
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());  
tableItem.TablixBody.TablixRows = new TablixRows();  
var rowOne = new List<TablixRowValues>();  
rowOne.Add(new TablixRowValues { TextBoxName = "TextBox1", TextBoxValue = "Department ID" });  
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox3", TextBoxValue = "Name" });  
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox5", TextBoxValue = "Group Name" });  
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowOne));  
var rowTwo = new List<TablixRowValues>();  
rowTwo.Add(new TablixRowValues { TextBoxName = "DepartmentID", TextBoxValue =  
"=Fields!DepartmentID.Value" });  
rowTwo.Add(new TablixRowValues { TextBoxName = "Name", TextBoxValue = "=Fields!Name.Value" });
```

```

rowTwo.Add(new TablixRowValues { TextBoxName = "GroupName", TextBoxValue =
"=Fields!GroupName.Value" });
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowTwo));
tableItem.TablixColumnHierarchy = new TablixColumnHierarchy();
tableItem.TablixColumnHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixRowHierarchy = new TablixRowHierarchy();
tableItem.TablixRowHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixRowHierarchy.TablixMembers.Add(new TablixMember() { KeepWithGroup =
KeepWithGroup.After });
tableItem.TablixRowHierarchy.TablixMembers.Add(CreateTablixMember("Details"));
return tableItem;
}

BoldReports.RDL.DOM.TablixColumn CreateTablixColumn()
{
var tablixColumn = new TablixColumn();
tablixColumn.Width = "1in";
return tablixColumn;
}

BoldReports.RDL.DOM.TablixRow CreateTablixRow(List<TablixRowValues> values)
{
var tablixRow = new TablixRow();
tablixRow.Height = "0.25in";
tablixRow.TablixCells = new TablixCells();
for(int i = 0; i < values.Count; i++)
{
tablixRow.TablixCells.Add(CreateTablixCell(values[i].TextBoxName, values[i].TextBoxValue));
}
return tablixRow;
}

BoldReports.RDL.DOM.TablixCell CreateTablixCell(string textBoxName, string textBoxValue)
{

```



```

var tablixCell = new TablixCell();
tablixCell.CellContents = new CellContents();
tablixCell.CellContents.ReportItem = CreateTextBox(textBoxName, textBoxValue);
return tablixCell;
}

BoldReports.RDL.DOM.TablixMember CreateTablixMember(string groupName)
{
var tablixMember = new TablixMember();
tablixMember.Group = new Group();
tablixMember.Group.Name = groupName;
return tablixMember;
}

internal class TablixRowValues
{
public string TextBoxName { get; set; }
public string TextBoxValue { get; set; }
}
`

```

Create a Chart for the report

- Using the following code snippets, you can create a **Chart** report item and assign the values for their properties like name, styles, and dimension values.

```

`csharp
BoldReports.RDL.DOM.Chart CreateChart(string name, string left, string top, string width, string height)
{
var chartItem = new Chart();
chartItem.Name = name;
chartItem.Height = new BoldReports.RDL.DOM.Size("10in");
chartItem.Width = new BoldReports.RDL.DOM.Size("10in");
chartItem.Left = new BoldReports.RDL.DOM.Size("5in");
chartItem.Top = new BoldReports.RDL.DOM.Size("5in");
chartItem.DataSetName = "DataSet1";
chartItem.Style = new BoldReports.RDL.DOM.Style();
chartItem.Bookmark= "=First(Fields!Name.Value, \"DataSet1\")";
}

```

```
chartItem.ChartCategoryHierarchy = new ChartCategoryHierarchy();
chartItem.ChartCategoryHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartCategoryHierarchy.ChartMembers.Add(ChartCategoryMember());
chartItem.ChartSeriesHierarchy = new ChartSeriesHierarchy();
chartItem.ChartSeriesHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartSeriesHierarchy.ChartMembers.Add(ChartSeriesMember());
chartItem.ChartData = new ChartData();
chartItem.ChartData.ChartDerivedSeriesCollection = new ChartDerivedSeriesCollection();
chartItem.ChartData.ChartSeriesCollection = CreateChartSeriesCollection();
chartItem.ChartLegends = new ChartLegends();
chartItem.ChartAreas = new ChartAreas();
chartItem.ChartAreas.Add(CreateChartArea());
chartItem.ChartTitles = new ChartTitles();
chartItem.Palette = "Pacific";
chartItem.ChartBorderSkin = new ChartBorderSkin();
chartItem.ChartNoDataMessage = new ChartNoDataMessage();
chartItem.ChartNoDataMessage.Caption = "No Data Available";
chartItem.ChartNoDataMessage.Name = "NoDataMessage";
return chartItem;
}

BoldReports.RDL.DOM.ChartMember ChartCategoryMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = new Group();
    ChartMember.Group = CreateChartGroup();
    ChartMember.Label="=Fields!DepartmentID.Value";
    ChartMember.SortExpressions = SortExpressions();
    return ChartMember;
}
```

```
BoldReports.RDL.DOM.ChartMember ChartSeriesMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = null;
    ChartMember.Label = "Department ID";
    ChartMember.SortExpressions = new SortExpressions();
    return ChartMember;
}

BoldReports.RDL.DOM.SortExpressions SortExpressions()
{
    var SortExpressions = new SortExpressions();
    var SortExpression = new SortExpression();
    SortExpression.Direction = SortDirection.Ascending;
    SortExpression.Value = "=Fields!DepartmentID.Value";
    SortExpressions.Add(SortExpression);
    return SortExpressions;
}

BoldReports.RDL.DOM.Group CreateChartGroup()
{
    var ChartGroup = new Group();
    ChartGroup.DataElementName = null;
    ChartGroup.DataElementOutput = DataElementOutputs.Auto;
    ChartGroup.DocumentMapLabel = null;
    ChartGroup.DocumentMapLabelLocID = null;
    ChartGroup.DomainScope = null;
    ChartGroup.Filters = new Filters();
    ChartGroup.GroupExpressions = CreateGroupExpressions();
    ChartGroup.Name = "Chart2_CategoryGroup";
    ChartGroup.PageBreak = null;
}
```

```
ChartGroup.PageName = null;
ChartGroup.Parent = null;
ChartGroup.Variables = new Variables();
return ChartGroup;
}

BoldReports.RDL.DOM.GroupExpressions CreateGroupExpressions()
{
    var GroupExpressions = new GroupExpressions();
    var GroupExpression = new GroupExpression();
    GroupExpression.Value = "=Fields!DepartmentID.Value";
    GroupExpressions.Add(GroupExpression);
    return GroupExpressions;
}

BoldReports.RDL.DOM.ChartSeriesCollection CreateChartSeriesCollection()
{
    var ChartSeriesCollection = new ChartSeriesCollection();
    ChartSeriesCollection.Add(CreateChartSeries());
    return ChartSeriesCollection;
}

BoldReports.RDL.DOM.ChartSeries CreateChartSeries()
{
    var ChartSeries = new ChartSeries();
    ChartSeries.CategoryAxisName = "Primary";
    ChartSeries.ChartAreaName = null;
    ChartSeries.ChartDataLabel = null;
    ChartSeries.ChartDataPoints = new ChartDataPoints();
    ChartSeries.ChartDataPoints.Add(ChartDataPoint());
    ChartSeries.ChartEmptyPoints = new ChartEmptyPoints();
    ChartSeries.ChartEmptyPoints.ChartDataLabel = new ChartDataLabel();
    ChartSeries.ChartEmptyPoints.ChartDataLabel.Style = new Style();
    ChartSeries.ChartEmptyPoints.ChartMarker = new ChartMarker();
    ChartSeries.ChartEmptyPoints.ChartMarker.Style = new Style();
    ChartSeries.ChartSmartLabel = new ChartSmartLabel();
}
```

```
ChartSeries.Name = "DepartmentID";
ChartSeries.Type = VisualizationType.Column;
ChartSeries.Subtype = VisualizationSubType.Plain;
ChartSeries.Style = new Style();
return ChartSeries;
}

BoldReports.RDL.DOM.ChartDataPoint ChartDataPoint()
{
    var ChartDataPoint = new ChartDataPoint();
    ChartDataPoint.ActionInfo = null;
    ChartDataPoint.ChartDataLabel = null;
    ChartDataPoint.ChartDataLabel = ChartDataLabel();
    ChartDataPoint.ChartDataPointValues = new ChartDataPointValues();
    ChartDataPoint.ChartDataPointValues.Y = "=Sum(Fields!DepartmentID.Value)";
    ChartDataPoint.ChartItemInLegend = null;
    ChartDataPoint.ChartMarker = new ChartMarker();
    ChartDataPoint.ChartMarker.Size = null;
    ChartDataPoint.ChartMarker.Type = null;
    ChartDataPoint.ChartMarker.Style= chartStyle("Transparent", "Arial");
    ChartDataPoint.CustomProperties = new CustomProperties();
    ChartDataPoint.DataElementName = null;
    ChartDataPoint.DataElementOutput = DataElementOutputs.Output;
    ChartDataPoint.ToolTip = null;
    ChartDataPoint.Style= chartStyle("Transparent", "Arial");
    return ChartDataPoint;
}

BoldReports.RDL.DOM.ChartDataLabel ChartDataLabel()
{
    var ChartDataLabel = new ChartDataLabel();
    ChartDataLabel.ActionInfo = null;
    ChartDataLabel.Label = null;
    ChartDataLabel.Position = null;
    ChartDataLabel.Rotation = "0";
```

```
ChartDataLabel.Style = new Style();
ChartDataLabel.Style = chartStyle("Transparent", "Arial");
ChartDataLabel.ToolTip = null;
return ChartDataLabel;
}

BoldReports.RDL.DOM.Style chartStyle(string BackgroundColor, string FontFamily)
{
    var style = new Style();
    style.BackgroundColor = BackgroundColor;
    style.FontFamily = FontFamily;
    return style;
}

BoldReports.RDL.DOM.ChartArea CreateChartArea()
{
    var chartArea = new ChartArea();
    chartArea.Style = new Style();
    chartArea.Style = chartStyle("Transparent", "Arial");
    chartArea.AlignOrientation = AlignOrientation.None;
    chartArea.AlignWithChartArea = null;
    chartArea.ChartAlignType = null;
    chartArea.ChartElementPosition = null;
    chartArea.ChartInnerPlotPosition = null;
    chartArea.ChartThreeDProperties = null;
    chartArea.Hidden = false;
    chartArea.EquallySizedAxesFont = false;
    chartArea.Name = "Default";
    chartArea.ChartCategoryAxes = new ChartCategoryAxes();
    chartArea.ChartCategoryAxes.Add(createChartAxis("Primary"));
    chartArea.ChartCategoryAxes.Add(createChartAxis("Secondary"));
    chartArea.ChartValueAxes = new ChartValueAxes();
    chartArea.ChartValueAxes.Add(createChartAxis("Primary"));
    chartArea.ChartValueAxes.Add(createChartAxis("Secondary"));
    return chartArea;
}
```

```

}
BoldReports.RDL.DOM.ChartAxis createChartAxis(string Name)
{
var ChartAxis = new ChartAxis();
ChartAxis.Style = new Style();
ChartAxis.AllowLabelRotation = AllowLabelRotation.None;
ChartAxis.Angle = "0";
ChartAxis.Arrows = Arrows.None;
ChartAxis.ChartAxisScaleBreak = new ChartAxisScaleBreak();
ChartAxis.ChartAxisTitle = new ChartAxisTitle();
ChartAxis.ChartMajorGridLines = new ChartMajorGridLines();
ChartAxis.ChartMajorTickMarks = new ChartMajorTickMarks();
ChartAxis.ChartMinorGridLines = new ChartMinorGridLines();
ChartAxis.ChartMinorTickMarks = new ChartMinorTickMarks();
ChartAxis.ChartStripLines = new ChartStripLines();
ChartAxis.CrossAt = "NaN";
ChartAxis.CustomProperties = new CustomProperties();
ChartAxis.LineStyle = LineStyle.Solid;
ChartAxis.Name = Name;
return ChartAxis;
}
`

```

Export the generated report in ReportWriter

You can export the report using the ReportWriter, after the report definition or stream is created. The following code snippet illustrates how to export the report into PDF format using the Report Writer by report definition.

```

`csharp
ReportWriter reportWriter = new ReportWriter();
reportWriter.LoadReport(reportDefinition);
reportWriter.Save(fileName, WriterFormat.PDF);
`

```

How to combine the exported PDF with another PDF file

You have to use the [PDF combine](#) for merging the exported PDF file with another PDF file as license agreement template. Refer to the following code sample controller,

```
`csharp
public IActionResult Export()
{
    // Export the RDL To PDF.
    FileStream mainReportStream = new FileStream(_hostingEnvironment.WebRootPath +
        @"\"Reports\Sales Order Detail.rdl", FileMode.Open, FileAccess.Read);
    MemoryStream reportStream = new MemoryStream();
    mainReportStream.CopyTo(reportStream);
    reportStream.Position = 0;
    mainReportStream.Close();

    //Template File
    FileStream templateStream = new FileStream(_hostingEnvironment.WebRootPath +
        @"\"PDF\Template.pdf", FileMode.Open, FileAccess.Read);

    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    MemoryStream memoryStream = new MemoryStream();
    writer.LoadReport(reportStream);
    writer.Save(memoryStream, BoldReports.Writer.WriterFormat.PDF);
    // Download the generated export document to the client side.
    memoryStream.Position = 0;
    FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/pdf");
    var exportedPdfSream = fileStreamResult.FileStream;
    // Combine the Exported PDF and template
    PdfDocument finalDoc = new PdfDocument();
    // Creates a PDF stream for merging
    Stream[] streams = { exportedPdfSream, templateStream };
    // Merges the PDF document.
    PdfDocumentBase.Merge(finalDoc, streams);
    //Save the document into a stream
    MemoryStream combinestream = new MemoryStream();
    finalDoc.Save(combinestream);
    combinestream.Position = 0;
    FileStreamResult fileStreamResult1 = new FileStreamResult(combinestream, "application/pdf");
    var combinePDF = fileStreamResult1.FileStream;
    // Add Page number for the combined PDF
```



```

PdfLoadedDocument loadedDoc = new PdfLoadedDocument(combinePDF);
//Set the font.
PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 12f);
//Create a page number field.
PdfPageNumberField pageNumber = new PdfPageNumberField(font, PdfBrushes.Black);
//Create a page count field.
PdfPageCountField count = new PdfPageCountField(font, PdfBrushes.Black);
//Add the fields in composite fields.
PdfCompositeField compositeField = new PdfCompositeField(font, PdfBrushes.Black, "Page {0} of {1}",
pageNumber, count);
for (int i = 0; i < loadedDoc.Pages.Count; i++)
{
//Draw the composite field.
compositeField.Draw(loadedDoc.Pages[i].Graphics, new PointF(loadedDoc.Pages[i].Size.Width / 2 - 20,
loadedDoc.Pages[i].Size.Height - 20));
}
MemoryStream combineWithPagenumber = new MemoryStream();
loadedDoc.Save(combineWithPagenumber);
//Save the document.
combineWithPagenumber.Position = 0;
//Close the document.
loadedDoc.Close(true);
finalDoc.Close(true);
string contentType = "application/pdf";
//Define the file name
string fileName = "Combinewithpagenumber.pdf";
//Creates a FileContentResult object by using the file contents, content type, and file name
return File(combineWithPagenumber, contentType, fileName);
}

```

How to load the report from the database

You must load the report using the **Stream** option available with **writer.LoadReport()** method. To load the report from the database, please follow these steps:

From byte array

```
`csharp
byte[] bytes = ""; // provide your byte array report data
MemoryStream reportStream = new MemoryStream(bytes);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadReport(reportStream);
`
```

From base64String

```
`csharp
string base64String = ""; // provide your base64 report data
byte[] bytes = System.Convert.FromBase64String(base64String);
MemoryStream reportStream = new MemoryStream(bytes);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadReport(reportStream);
`
```

From string

```
`csharp
string reportData = ""; // provide your Report data
byte[] bytes = System.Text.Encoding.ASCII.GetBytes(reportData);
MemoryStream reportStream = new MemoryStream(bytes);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadReport(reportStream);
`
```

Reporting tools for ASP.NET MVC

Enterprise-class reporting tools for ASP.NET MVC development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your MVC applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

Reporting tools for ASP.NET MVC

Enterprise-class reporting tools for ASP.NET MVC development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your MVC applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

System Requirements

This topic describes the software and hardware requirements for setting up development environment of Bold Reports ASP.NET MVC.

Supported Operating Systems

- Windows 7+, 8+
- Windows Server 2008 R2+

Hardware Environments

The following hardware environments are necessary for setting up the Bold Reports ASP.NET MVC.

- 1 GHZ or faster, 32 bit or 64 bit processor.
- 1 GB RAM for 32 bit or 2 GB RAM for 64 bit.

Development Environments

By using the following IDEs, you can develop the Bold Reports ASP.NET MVC.

- [Microsoft Visual Studio 2010](#) or [later](#)
- Internet Information Services (IIS) 7.0+

Framework

The below tool is required for Bold Reports ASP.NET MVC.

.NET Framework 4.5 or higher

Browser Compatibility

- IE 9+
- Microsoft Edge
- Mozilla Firefox 22+
- Chrome 17+
- Opera 12+
- Safari 5+

See Also

- [Licensing procedure for deployment](#)

Overview

The Report Viewer is a visualization control used to display SSRS, RDL, RDLC, and Bold Report Server reports within web applications. It allows you to view RDL/RDLC reports with or without using SSRS or Bold Report Server. You can bind data sources, parameters, and render reports with all major capabilities of RDL reporting and export the report to PDF, Microsoft Excel, CSV, Microsoft Word, and HTML formats. Some of the key features are,

- Renders interactive reports with drill down, drill through, hyperlinks, and interactive sorting.
- Easily customize each element of Report Viewer and provide events for report processing customization.
- Supports jQuery, Angular, React, Blazor, ASP.NET Core, ASP.NET MVC, ASP.NET WebForms, WPF, and UWP.

Introducing the newly redesigned Report Viewer! We have given it a refreshing new look that embraces sleek and modern design, aligning perfectly with the latest web design trends. It allows seamless integration into your application. For detailed guidance on the migration process, we recommend you to refer our [Report Viewer v2.0 migration document](#).

Display SSRS RDL report in Bold Reports ASP.NET MVC Report Viewer

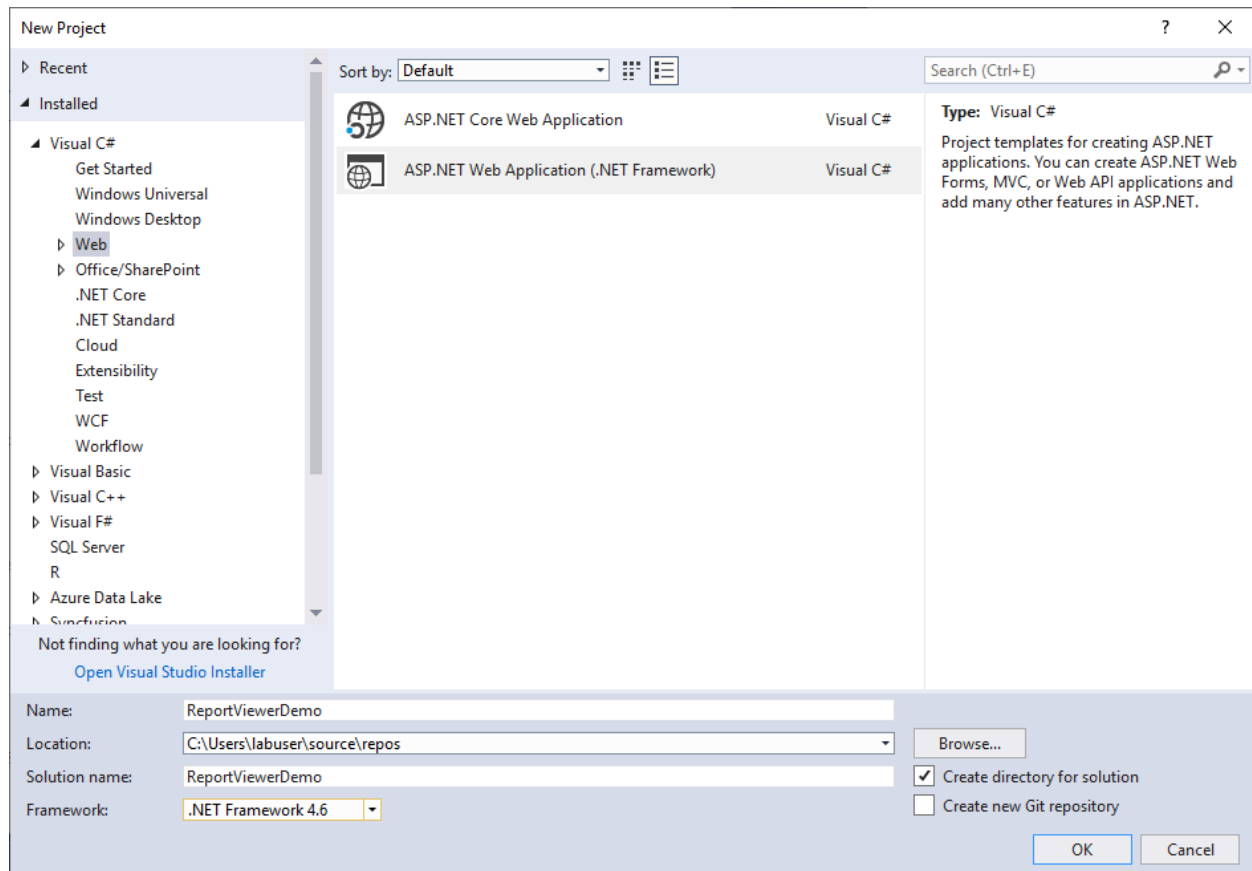
This section explains you the steps required to create your first ASP.NET MVC reporting application to display an already created SSRS RDL report in the Bold Reports ASP.NET MVC Report Viewer without using a Report Server.

To get start quickly with Report Viewer, you can check on this video:

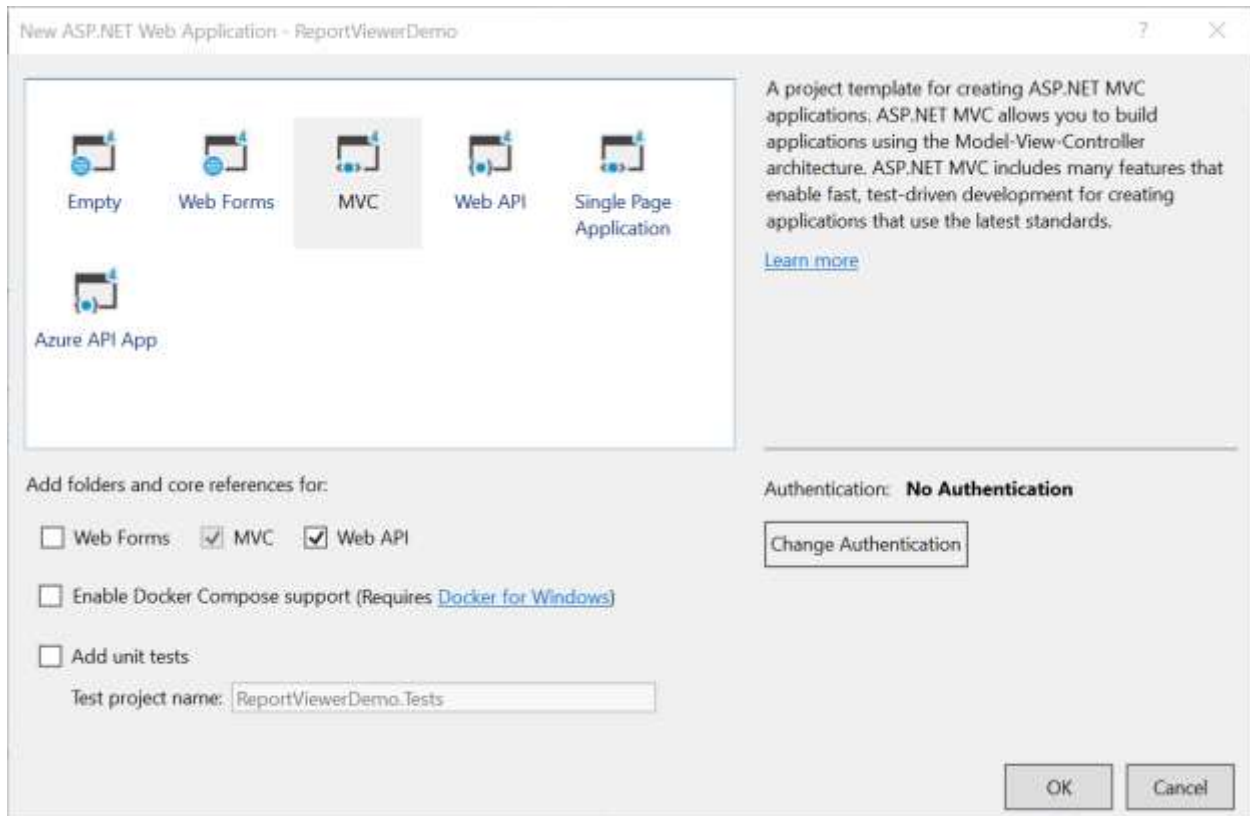
youtube:<https://youtu.be/2sKWxZYNLeI>

Create an ASP.NET MVC 5 application

1. Open Visual Studio 2017, click the **File** menu, go to **New**, and then select **Project**.
2. Go to **Installed > Visual C# > Web**, and then select the required **.NET Framework** in the drop-down.
3. Select **ASP.NET Web Application (.NET Framework)**, change the application name, and then click **OK**.



4. Choose **MVC** and **Web API**, and then click **OK**.



Configure Report Viewer in an application

1. Right-click the project or solution on the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select **Tools > NuGet Package Manager > Manage NuGet Packages for Solution**.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring Report Viewer NuGet packages.

2. Search for **BoldReports.Web**, **BoldReports.JavaScript**, and **BoldReports.Mvc5** NuGet packages, and install them in your MVC application.

Package | Purpose

PostReportAction | Action (HttpPost) method for posting the request in the report process.

OnInitReportOptions | Report initialization method occurs when the report is about to be processed.

OnReportLoaded | Report loaded method occurs when the report and sub report start loading.

GetResource | Action (HttpGet) method is used to get resources for the report.

ReportHelper

The class **ReportHelper** contains helper methods that help to process a Post or Get request from the Report Viewer control and return the response to the Report Viewer control. It has the following methods:

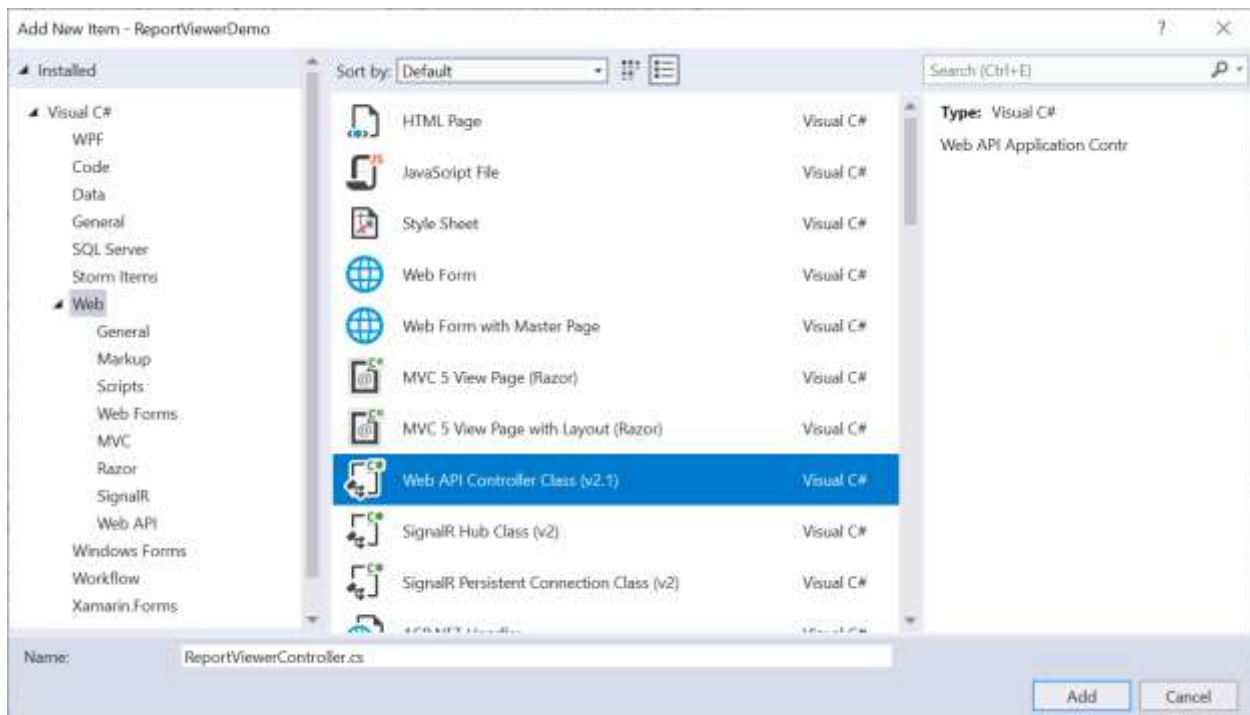
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click the **Controller** folder in your project and select **Add > New Item** from the context menu.
2. Select the **Web API Controller Class** from the listed templates and name it as **ReportViewerController.cs**.



3. Click **Add**.

While adding the Web API Controller class, naming it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and implement its methods (replace the following code in the newly created Web API controller).

```
`csharp
```

```
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }

    // Get action for getting resources from the report
    [System.Web.Http.ActionName("GetResource")]
    [AcceptVerbs("GET")]
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }

    // Method that will be called when initialize the report options before start processing the report
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }

    // Method that will be called when reported is loaded
    [NonAction]
    public void OnReportLoaded(ReportViewerOptions reportOption)
    {
        //You can update report options here
    }
}
```

Add routing information

To configure routing to include an action name in the URI, open the `WebApiConfig.cs` file and change the `routeTemplate` in the **Register** method as follows.

```
`csharp
public static class WebApiConfig
{

```



```

public static void Register(HttpConfiguration config)
{
    // Web API configuration and services
    // Web API routes
    config.MapHttpAttributeRoutes();
    config.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{action}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
}

```

If you are looking to load the report directly from the SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#) section.

Set report path and service URL

To render the reports available in the application, set the **ReportPath** and **ReportServiceUrl** properties of the Report Viewer. You can replace the following code on your Report Viewer page.

```

`js
@(Html.Bold().ReportViewer("viewer")
    .ReportServiceUrl("/api/ReportViewer")
    .ReportPath("~/Resources/sales-order-detail.rdl")
)

```


The report path property is set for the RDL report that is added to the project **Resources** folder.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

1 / 1 75%

Sales Order No:



Sales Order

Order ID #5050750	Billing Date 22-04-2019	Order Date 01-06-2003	Purchase Order PO7192170677	Shipment Method CARGO TRANSPORT 5
-----------------------------	-----------------------------------	---------------------------------	---------------------------------------	---

Billing Address Jean Handley Central Discount Store 259826 Russell Rd. South Kent, Washington 98031 United States	Shipping Address Jean Handley Central Discount Store 259826 Russell Rd. South Kent, Washington 98031 United States	Contact Jean Handley Ph: 562-555-0113
---	--	---

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.84	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	B6-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	B6-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

I/We hereby certify that the information on this invoice is true and correct and that the contents of this shipment are as stated above.

Signature of Authorized Person

Note: You can refer to our feature tour page for the [ASP.NET MVC Report Viewer](#) to see its innovative features. Additionally, you can view our [ASP.NET MVC Report Viewer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

See Also

[Render report with data visualization report items](#)

[Create RDLC report](#)

[List of SSRS server versions are supported in Bold Reports](#)

Load SSRS Report Server reports

Report Viewer has support to load RDL reports from SSRS Report Server. To render SSRS Reports, set the `reportServerUrl`, `reportPath`, and `reportServiceUrl` properties as shown in the following steps.

1. To create your first ASP.NET MVC reporting application, refer to the [Getting-started](#) section.

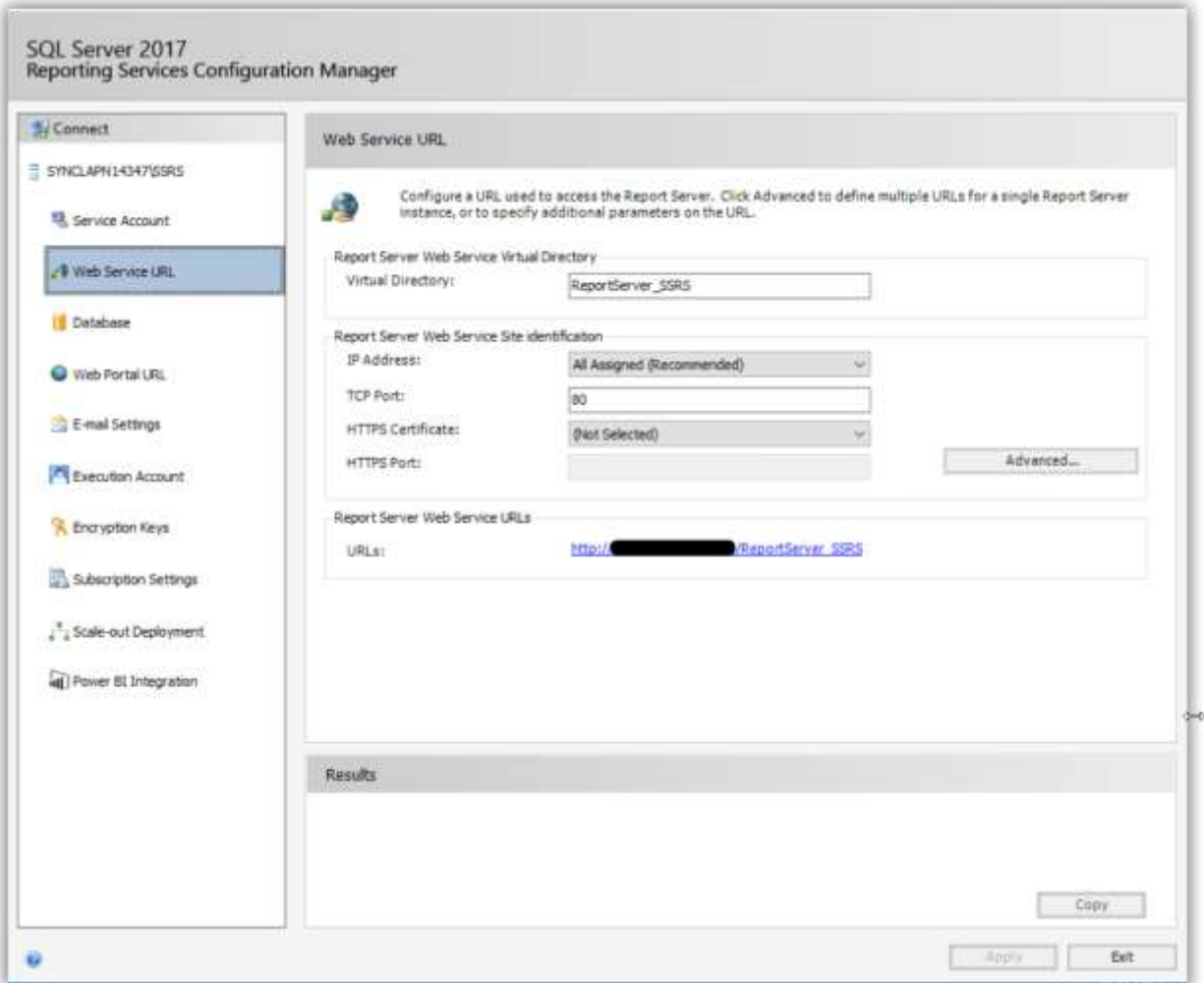
If you need to know about the difference between `reportServiceUrl` and `reportServerUrl`, then refer to [Difference between Report Service URL and Report Server URL](#).

2. Set the `reportServerUrl` API on Bold Report Viewer with `WebServiceURL`. Open the `App.js` or `app.component.ts` and replace the following code example.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/SSRSReports")
```

```
.ReportServerUrl("http://<servername>/Reports_SSRS")
)
`
```

The Web Service URL should be set as `reportServerUrl` when using ASP.NET MVC service. The Web Service URL can be found from the Reporting Services Configuration manager under the **Web Service URL** section, as shown in the following image.

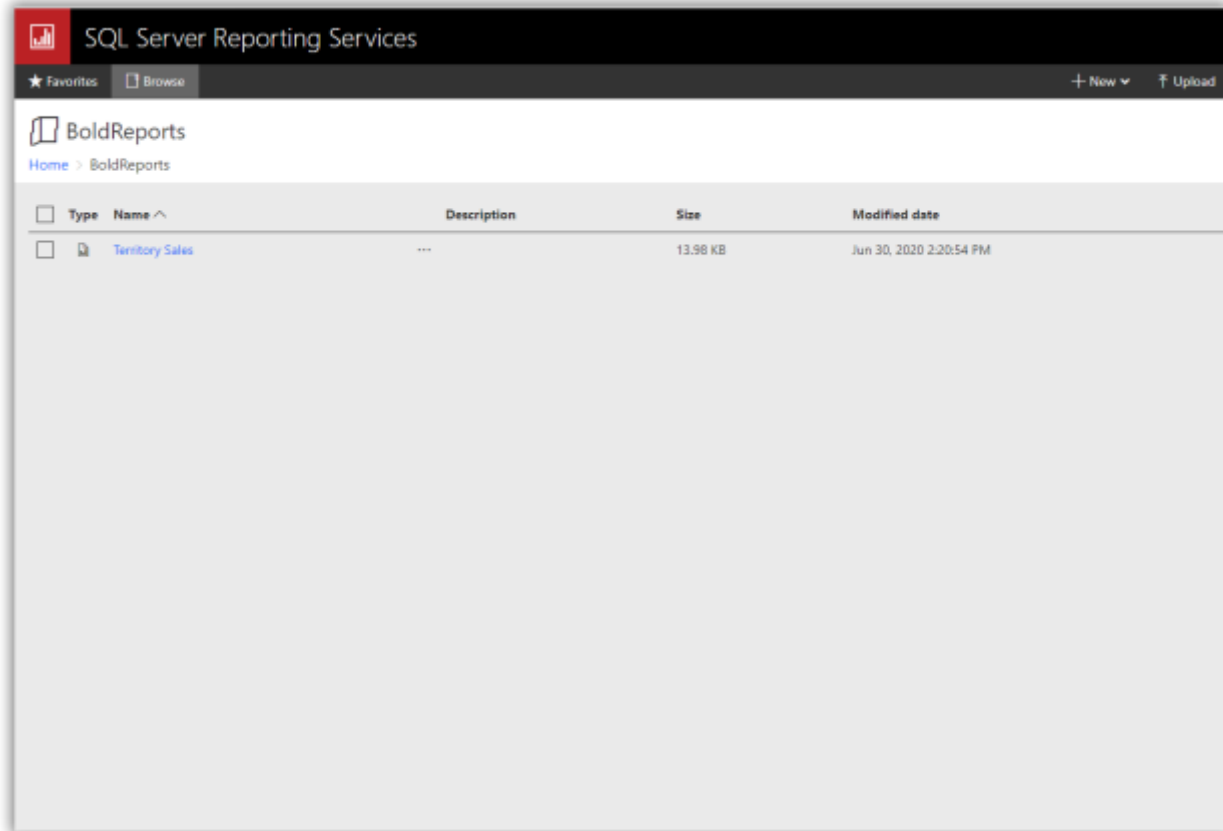


3. Set the report path for loading the reports from the SSRS Report Server. The report path should be in the format of `/folder name/report name`. Open the `Index.cshtml` file and replace the following code example.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/SSRSReports")
.ReportServerUrl("http://<servername>/reportserver$instanceName")
```

```
.ReportPath("/SSRSSamples2/Territory Sales")
)
,
```

The report path can be found from the SSRS Report Server by navigating to the path of the report to be loaded, as shown in the following image.



Network credentials for SSRS

The network credentials are required to load specified SSRS report from the specified SSRS Report Server using the Report Viewer. Specify the `ReportServerCredential` property in the Web API Controller `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server credential
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
    "RDLReport1");
}
,
```

If you are facing problem to access the SSRS Report server reports, you can refer [How to provide the permission for user to access the SSRS Report Server reports](#).

Set data source credential to shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS Server. If the report has any data source that uses credentials to connect with the database, then you should specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server and data source credentials
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
    "RDLReport1");
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
}
`
```

Data source credentials should be added to the shared data sources that do not have credentials in the connection strings.

Change data source connection string

You can change the connection string of a report data source before it is loaded in the Report Viewer. The `DataSourceCredentials` class provides the option to set and update the modified connection string as in the following code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "<username>", "<password>", "Data
    Source=<instancename>;Initial Catalog=<database>;"));
}
`
```

The previous code shows an option to change the connection string only, but the class provides multiple options to change data source information. To learn more about this, refer to this `DataSourceCredentials` class.

[See also](#)

[Does Bold Report Viewer use SSRS Report processing?](#)

Load SharePoint Server reports

To render SharePoint server reports, set the `ReportServerUrl`, `ReportPath`, and `ReportServiceUrl` properties as shown in the following code snippet.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/SharePointReports")
.ReportServerUrl("http://<servername>/reportserver$instanceName")
.ReportPath("http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl")
)
`
```

In SharePoint integrated mode, the `ReportServerUrl` will be same as your site URL. The `ReportPath` is relative to the Report Server URL with the file extension.

Forms credential for SharePoint Server

The forms credentials are required to load the SharePoint integrated SSRS report from the specified SharePoint integrated SSRS Report Server using the Report Viewer. Specify the `ReportServerFormsCredential` property in the Web API Controller `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add ReportServerFormsCredential for server
    reportOption.ReportModel.ReportServerFormsCredential = new ReportServerFormsCredential("ssrs",
    "RDLReport1");
}
`
```

Set data source credential to shared data sources

The shared data source credentials can be added to the `DataSourceCredentials` property to connect with the database.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add ReportServerFormsCredential and data source credentials
}
```

```
reportOption.ReportModel.ReportServerFormsCredential = new ReportServerFormsCredential("ssrs",
"RDLReport1");

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
}
```

Data source credentials should be added to shared data sources that do not have credentials in the connection strings.

Render RDLC report

The data binding support, allows you to view RDLC reports that exist on the local file system with JSON array and custom business object data collection. The following steps demonstrates how to render a RDLC report with JSON array and custom business object data collection.

Add the RDLC report `product-list.rdlc` from Bold Reports installation location to your application `Resources` folder. For more information, see [Samples and demos](#).

Bind data source at client side

1. Set the RDLC report path to the `ReportPath` property.
2. Assign the `ProcessingMode` property to `ProcessingMode.Local`.
3. Bind the data from `viewdata` collection to the `DataSources` property as shown in following code.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ProcessingMode(BoldReports.ReportViewerEnums.ProcessingMode.Local)
.ReportPath("~/Resources/product-list.rdlc")
.ReportServiceUrl("/api/ReportViewer")
.DataSources(ds => ds.Name("list").Value(ViewData["DataSource"]).Add())
)
```

4. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
public partial class HomeController : Controller
{
    public ActionResult Index()
    {
```

```

ProductList productList = new ProductList();
ViewData["DataSource"] = productList.GetData();
return View();
}
}

public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public IList GetData()
    {
        List<ProductList> dataList = new List<ProductList>();
        ProductList data = null;

        data = new ProductList() {ProductName = "Baked Chicken and Cheese",OrderId = "323B60",Price = 55,Category = "Non-Veg",Ingredients = "grilled chicken, corn and olives.",ProductImage = "" };
        dataList.Add(data);

        data = new ProductList() {ProductName = "Chicken Delite",OrderId = "323B61",Price = 100,Category = "Non-Veg",Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",ProductImage = ""};
        dataList.Add(data);

        data = new ProductList() {ProductName = "Chicken Tikka",OrderId = "323B62",Price = 64,Category = "Non-Veg",Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",ProductImage = ""};
        dataList.Add(data);

        return dataList;
    }
}

```

[Bind data source in Web API controller](#)

The following steps help you to configure the Web API to render the RDLC report with business object data collection.

- Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> dataList = new List<ProductList>();
        ProductList data = null;
        data = new ProductList() {ProductName = "Baked Chicken and Cheese",OrderId = "323B60",Price = 55,Category = "Non-Veg",Ingredients = "grilled chicken, corn and olives.",ProductImage = "" };
        dataList.Add(data);
        data = new ProductList() {ProductName = "Chicken Delite",OrderId = "323B61",Price = 100,Category = "Non-Veg",Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",ProductImage = ""};
        dataList.Add(data);
        data = new ProductList() {ProductName = "Chicken Tikka",OrderId = "323B62",Price = 64,Category = "Non-Veg",Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",ProductImage = ""};
        dataList.Add(data);
        return dataList;
    }
}
```

- Set the **ProcessingMode** to **ProcessingMode.Local** and **ReportPath** to the RDLC report location.
- Bind the business object data values collection by adding new item to the **DataSources** as in the following code snippet.

```
`csharp
[NonAction]
```

```

public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ProcessingMode = ProcessingMode.Local;
    reportOption.ReportModel.ReportPath =
        System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/product-list.rdlc");
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
        Value = ProductList.GetData() });
}

```

Here the **Name** is case sensitive and it should be same as in the data source name in the report definition.

The **Value** accepts IList, DataSet, and DataTable inputs.

Load report as stream

To load report as a stream, create a report stream using the **FileStream** class and assign the report stream to the **Stream** property.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    // Opens the report from application Resources folder using FileStream
    FileStream reportStream = new
        FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/product-list.rdlc"),
        FileMode.Open, FileAccess.Read);
    reportOption.ReportModel.Stream = reportStream;
    reportOption.ReportModel.ProcessingMode = ProcessingMode.Local;
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
        Value = ProductList.GetData() });
}

```

In the above code, **product-list.rdlc** report is loaded from the **Resources** folder location.

Render subreport

You can display another report inside the body of a main report using the Report Viewer. The following steps help you to customize the subreport properties such as data source, report path, and parameters.

1. Add the sub report and main reports to the application **Resources** folder. In this tutorial, the already created reports are used. Refer to the [Create RDL Report section](#) or [Create RDLC Report section](#).

Download the **side-by-side-main-report.rdl**, **side-by-side-sub-report.rdl** reports from [here](#). You can add a report from the Syncfusion installation location. For more information, refer to [Samples and demos](#).

- Set the **ReportPath** and **ReportServiceUrl** properties of the Report Viewer as in the following code snippet.

The following code example demonstrates how to load a subreport in the Report Viewer at client side.

```
`js
@($.Html.Bold().ReportViewer("viewer"))
.ReportPath("~/Resources/side-by-side-main-report.rdl")
.ReportServiceUrl("/api/ReportViewer")
)
```

Change subreport path

To change the subreport file path, set the **Stream** property of **SubReportModel** in the **OnInitReportOptions** method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string reportPath = System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/" +
        reportOption.SubReportModel.ReportPath);
    if (reportOption.SubReportModel != null)
    {
        FileStream SubStream = new FileStream(reportPath, FileMode.Open, FileAccess.Read);
        reportOption.SubReportModel.Stream = SubStream;
    }
}
```

Set subreport parameter

You can change the parameter default values of a subreport in the **OnReportLoaded** method of the Web API Controller as given in the following code snippet.

```

`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.Parameters = new BoldReports.Web.ReportParameterInfoCollection();
        reportOption.SubReportModel.Parameters.Add(new BoldReports.Web.ReportParameterInfo()
        {
            Name = "SalesPersonID",
            Values = new List<string>() { "2" }
        });
    }
}

```

Modify subreport data source connection string

You can change the credential and connection information of the data sources used in the subreport using the SubReportModel in the `OnInitReportOptions` method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSourceCredentials = new
        List<BoldReports.Web.DataSourceCredentials>();
        reportOption.SubReportModel.DataSourceCredentials.Add(new
        BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
        Catalog=<database>;user id=<username>;password=<password>"));
    }
}

```

Set subreport data source

You can bind local business object data source collection only for RDLC reports. To specify data source of a RDLC subreport, set the `ReportDataSource` property in the `OnReportLoaded` method.

The RDL report has the connection information in report definition itself, so no need to bind data source.

1. Add the RDLC sub report and main reports to your application **Resources** folder. For more information, see [Samples and demos](#).
2. Set the **ReportPath**, **ProcessingMode**, and **ReportServiceUrl** properties of the Report Viewer as shown in following code snippet.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ProcessingMode(BoldReports.ReportViewerEnums.ProcessingMode.Local)
.ReportServiceUrl("/api/SubreportDataSources")
.ReportPath("~/Resources/product-list.rdlc")
)
```

3. Create a class and methods that returns business object data collection. Use the following code in the application Web API Service.

```
`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = null;
        data = new ProductList()
        {
            ProductName = "Baked Chicken and Cheese",
            OrderId = "323B60",
            Price = 55,
```

```

Category = "Non-Veg",
Ingredients = "grilled chicken, corn and olives.",
ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Delite",
    OrderId = "323B61",
    Price = 100,
    Category = "Non-Veg",
    Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
`

```

4. Bind the business object data values collection to the subreport by adding a new item to the DataSources as shown in the following code snippet.

```

`csharp
[NonAction]

```

```

public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'Product List.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
        "list", Value = ProductList.GetData() });
    }
}

```

The data source name is case sensitive, it should be same as in the report definition.

Load subreport stream

To load subreport as stream, set the **Stream** property in the **OnInitReportOptions** method.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        FileStream reportStream = new
        FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/" +
        reportOption.SubReportModel.ReportPath), FileMode.Open, FileAccess.Read);
        reportOption.SubReportModel.Stream = reportStream;
    }
}
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
        "list", Value = ProductList.GetData() });
    }
}

```

```

}
}
,

```

Report parameters

Provides property options to pass or set report parameters default values at run time using the `parameters` property. You can set the report parameters while creating the Report Viewer control in a script or in the Web API Controller.

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded from [here](#).

Set parameter at client

To set parameter default value in the client side, The following code example illustrates how to set report parameter,

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ReportServiceUrl("/api/ReportViewer")
.Parameters(param => { param.Name("SalesOrderNumber").Labels(new List<string>() { "SO50755"
}).Values(new List<string>() { "SO50755" }).Add(); })
)
,

```

Set parameters in Web API Controller

To set parameter default value in Web API Controller, use the following code in the `OnReportLoaded` method.

```

`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    List<BoldReports.Web.ReportParameter> userParameters = new
    List<BoldReports.Web.ReportParameter>();
    userParameters.Add(new BoldReports.Web.ReportParameter()
    {
        Name = "SalesOrderNumber",
        Values = new List<string>() { "SO50755" }
    });
    reportOption.ReportModel.Parameters = userParameters;
}

```


,

The Report Parameters name should be case sensitive

Get report parameter

The **ReportHelper** class provides methods to get the report parameters used in the report. The following helper methods used to get parameter with or without values.

Methods | Description

MaxDateTime | Specify minimum range value of a date parameter

MinDateTime | Specify maximum range value of a date parameter

Refer to the following code sample to set data range using **parameterSettings** in Report Viewer controller.

```
`csharp
public ActionResult Index()
{
    ViewBag.parameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
    ViewBag.parameterSettings.MaxDateTime = new DateTime(2003, 8, 22);
    ViewBag.parameterSettings.MinDateTime = new DateTime(2003, 4, 22);
    return View();
}
```

,

Refer to the following code sample to set data range to the report viewer initialization.

```
`js
@(Html.Bold().ReportViewer("viewer")
    .ReportServiceUrl("/api/ReportViewer")
    .ReportPath("~/Resources/sales-order-detail.rdl")
    .ParameterSettings(parameterSettings =>
        parameterSettings.MaxDateTime(@ViewBag.parameterSettings.MaxDateTime))
    .ParameterSettings(parameterSettings =>
        parameterSettings.MinDateTime(@ViewBag.parameterSettings.MinDateTime))
)
```

,

The above code sets date range for all the date parameters used in the report.

To set different date range for each date parameter used in the report, register the event **beforeParameterAdd** and specify range based on parameter name as in below code sample.

```
`js
@(Html.Bold().ReportViewer("viewer")
```

```

.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/product-line-sales.rdl")
.BeforeParameterAdd("beforeParameterAdd")
)
<script type="text/javascript">
function beforeParameterAdd(args) {
if (args.parameterModel.Name === "StartDate") {
args.parameterSettings.minDateTime = new Date("4/5/2003 5:00:00 AM");
args.parameterSettings.maxDateTime = new Date("4/15/2003 5:00:00 AM");
}
if (args.parameterModel.Name === "EndDate") {
args.parameterSettings.minDateTime = new Date("5/10/2003 5:00:00 AM");
args.parameterSettings.maxDateTime = new Date("5/20/2003 5:00:00 AM");
}
}
}
</script>
`

```

Set date time display format for date parameter

The properties `dateTimeFormat` and `timeDisplayFormat` in the `parameterSettings` are used to set date and time format to be displayed in the `DateTimePicker` control in a report.

Format | Display in `DateTimePicker`

Short Date and Time - `d/M/yy h:mm tt` | 9/12/2014 2:04 PM

Medium Date - `d MMM yy h:mm tt` | 12 Sep 14 2:04: PM

Full Date and short time - `dddd, MMMM dd, yyyy HH:mm tt` | Friday, September 12,2014 2:04 PM

Full Date and Long Time - `dddd, MMMM dd, yyyy HH:mm:ss tt` | Friday, September 12,2014 2:04:00 PM

UTC - `yyyy-MM-dThh:mm:ssz` | 2014-09-12T2:04:00+5

Refer to the following code sample to set date and time format to be displayed, using **parameterSettings** in Report Viewer controller.

```

`csharp
public ActionResult Index()
{
    ViewBag.parameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
}

```

```

ViewBag.parameterSettings.DateTimeFormat = "d/M/yyyy h:mm tt";
ViewBag.parameterSettings.TimeDisplayFormat = "HH:mm";
ViewBag.parameterSettings.TimeInterval = 60;
return View();
}

```

Refer to the following code sample to set date and time format to be displayed in the report viewer initialization.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ParameterSettings(parameterSettings =>
parameterSettings.DateTimeFormat(@ViewBag.parameterSettings.DateTimeFormat))
.ParameterSettings(parameterSettings =>
parameterSettings.TimeDisplayFormat(@ViewBag.parameterSettings.TimeDisplayFormat))
.ParameterSettings(parameterSettings =>
parameterSettings.TimeInterval(@ViewBag.parameterSettings.TimeInterval))
)

```

The above code sets date and time value to be display for all the date parameters used in the report.

To set different date and time value to be display for each date parameter used in the report, register the event `beforeParameterAdd` and specify date and time value based on parameter name as in below code sample.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/product-line-sales.rdl")
.BeforeParameterAdd("beforeParameterAdd")
)
<script type="text/javascript">
function beforeParameterAdd(args) {
if (args.parameterModel.Name === "StartDate") {
args.parameterSettings.dateTimeFormat = "d/M/yyyy h:mm tt";
args.parameterSettings.timeDisplayFormat = "HH:mm",

```

```

args.parameterSettings.timeInterval = 60;
}
if (args.parameterModel.Name === "EndDate") {
args.parameterSettings.dateTimeFormat = "d/M/yyyy h:mm tt";
args.parameterSettings.timeDisplayFormat = "HH:mm",
args.parameterSettings.timeInterval = 60;
}
}
</script>
`

```

Change the Parameter drop-down height and width

The `parameterSettings` helps you to change the height and width of the parameter available in parameter panel.

```

`html
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ParameterSettings(parameter => parameter.PopupHeight("100px").PopupWidth("150px"))
)
`

```

Hide a Parameter scroller

The `enableparameterblockscroller` helps you to hide the scrollbar in parameter panel.

```

`html
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.EnableParameterBlockScroller(false)
)
`

```

Hide a Parameter Pane on load

The `parameterSettings` helps you to hide and show the parameter block.

```

`html
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ParameterSettings(parameter => parameter.HideParameterBlock(true))
)
`

```

,

Change the Parameter Item Width and Label Width

The `parameterSettings` helps you to change the parameter Item width and label width.

```
`html
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ParameterSettings(parameter => parameter.ItemWidth("250px").LabelWidth("auto")))
)
```

,

Access the hidden or internal parameter information

The `accessInternalValue` property in the `parameterSettings` helps you to expose the `hidden` or `internal` report parameter information used in report to the user.

```
`html
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ParameterSettings(parameter => parameter.AccessInternalValue(true)))
)
```

,

Set the report parameter visibility in Web API Controller

The `Hidden` property of `ReportParameter` allows you to show or hide the parameter at the top of the report viewer panel. The following code example shows hiding a report parameter in the Web API controller's `OnReportLoaded` method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    var reportParameters = ReportHelper.GetParameters(jsonArray, this);
    List<BoldReports.Web.ReportParameter> modifiedParameters = new
    List<BoldReports.Web.ReportParameter>();
    if (reportParameters != null)
    {
        foreach (var rptParameter in reportParameters)
        {
            modifiedParameters.Add(new BoldReports.Web.ReportParameter()
            {
```

```

Name = rptParameter.Name,
Hidden = true
});
}
reportOption.ReportModel.Parameters = modifiedParameters;
}
}
`

```

IReportController

The **IReportController** interface has the declaration of action methods that is defined in Web API Controller to process the RDL, RDLC, SSRS report and handling request from Report Viewer control. The IReportController has the following action methods declaration.

Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

`csharp

```

public class ReportsWebApiController: ApiController,IReportController
{
    /// <summary>
    /// Action (HttpGet) method for getting resource for report.
    /// </summary>
    /// <param name="key">The unique key to get the required resource.</param>
    /// <param name="resourceType">The type of the requested resource.</param>
    /// <param name="isPrinting">If set to <see langword="true"/>, then the resource is generated for
    printing.</param>
    /// <returns>The object data.</returns>
    public object GetResource(string key, string resourceType, bool isPrinting)
    {
        //Returns the report resource for the requested key.
        return ReportHelper.GetResource(key, resourceType, isPrinting);
    }
    /// <summary>
    /// Report Initialization method that is triggered when report begin processed.
    /// </summary>

```

```

/// <param name="reportOptions">The Report Viewer options.</param>
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOptions)
{
    //You can update report options here
}
/// <summary>
/// Report loaded method that is triggered when report and sub report begins to be loaded.
/// </summary>
/// <param name="reportOptions">The Report Viewer options.</param>
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOptions)
{
    //You can update report options here
}
/// <summary>
/// Action (HttpPost) method for posting the request for report process.
/// </summary>
/// <param name="jsonData">The JSON data posted for processing report.</param>
/// <returns>The object data.</returns>
public object PostReportAction(Dictionary < string, object > jsonData)
{
    //Processes the report request and returns the result.
    return ReportHelper.ProcessReport(jsonData, this);
}
}
,

```

Extensions

See Also

- [Custom Data Extension](#)

Custom Data Extension

This section explains the steps required to create and load data extensions in Web ASP.NET MVC Report Viewer Application

See Also

- [Configure Data Extension](#)

Configure a new data source extension in ASP.NET MVC Report Viewer

The ASP.NET MVC Report Viewer provides SQL, ODBC and OLEDB data sources as built in support data sources. Other data source like Web API, JSON, XML, and OData are provided as extension data sources.

This documentation provides step by step procedure to register and connect with new extension data sources in Report Viewer Application.

Create ASP.NET MVC Report Viewer Application

Refer [Getting Started](#) and create a ASP.NET MVC Report Viewer Application.

Install data source extension from NuGet

Based on the required data connector install the respective NuGet package to the application. The NuGet packages name for each data connectors are provided in below table,

Data source	Package Name	Assembly Name
-----	-----	-----
Web data sources(WebAPI, JSON, XML, and OData)	BoldReports.Data.WebData	BoldReports.Data.WebData.dll
PostgreSQL data sources	BoldReports.Data.PostgreSQL	BoldReports.Data.PostgreSQL.dll
CSV data sources	BoldReports.Data.Csv	BoldReports.Data.Csv.dll
Excel data sources	BoldReports.Data.Excel	BoldReports.Data.Excel.dll
MySQL data sources	BoldReports.Data.MySQL	BoldReports.Data.MySQL.dll
Oracle data sources	BoldReports.Data.Oracle	BoldReports.Data.Oracle.dll

For example, to register and load web data sources in the application install *BoldReports.Data.WebData* package.

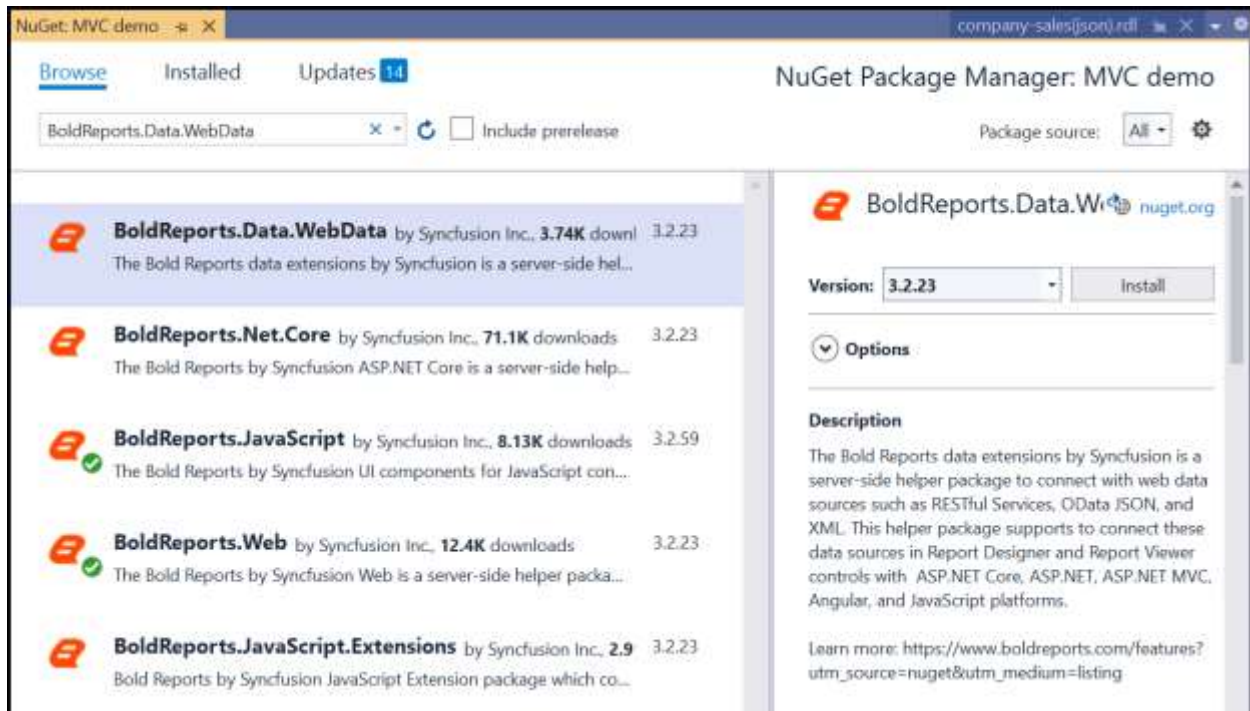
Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring NuGet packages.

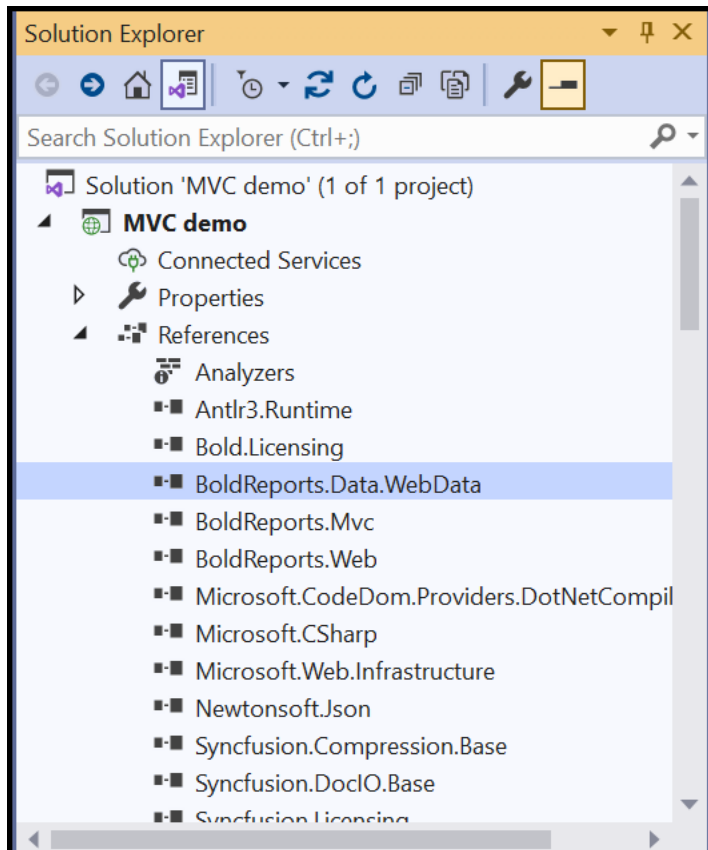
Search for **BoldReports.Data.WebData** NuGet package, and install it in your application.

Configure a new data source extension in ASP.NET MVC Report Viewer from NuGet

Install data source extension



BoldReports.Data.WebData will install into your application. Click OK. Now, the assembly will be added in the respective project references.



Register data source extension

1. Open the code-behind file `WebApiConfig.cs` and add the following using statement.

```
`csharp
using BoldReports.Web;
`
```

2. Then add the following code to register extension assembly in `Register` method.

```
`csharp
public static void Register(HttpConfiguration config)
{
    //Use the below code to register extensions assembly into report viewer
    ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {
        "BoldReports.Data.WebData" });

    //To register multiple data extensions, provide the assembly name's as list of strings. For example:
    "ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {
        "BoldReports.Data.WebData", "BoldReports.Data.Excel"};"

    //Incase the data source extensions fails to register or any error occurs replace the code as below,
    "ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string>
    {System.IO.Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory) +
    "BoldReports.Data.WebData.dll" });"

    ...

    ...

}
`
```

Run the application

1. Run the ASP.NET MVC Report Viewer Application
2. Now we can able to see the company sales report rendered with `JSON` data


```

function onReportLoaded(args) {
var dataSource = [
{
ProductName: "Baked Chicken and Cheese", OrderId: "323B60", Price: 55, Category: "Non-Veg",
Ingredients: "Grilled chicken, Corn and Olives.", ProductImage: ""
},
{
ProductName: "Chicken Delite", OrderId: "323B61", Price: 100, Category: "Non-Veg", Ingredients:
"Cheese, Chicken chunks, Onions & Pineapple chunks.", ProductImage: ""
},
{
ProductName: "Chicken Tikka", OrderId: "323B62", Price: 64, Category: "Non-Veg", Ingredients: "Onions,
Grilled chicken, Chicken salami & Tomatoes.", ProductImage: ""
}
];
var reportObj = $('#viewer').data("boldReportViewer");
reportObj.model.dataSources = [{
value: ej.DataManager(dataSource).executeLocal(ej.Query()),
name: "list"
}
];
}
</script>

```

Report error

When an error occurs in the report processing, it raises the **ReportError** event. You can handle the event and show the details in your custom dialog instead of component default error detail interface.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ProcessingMode(BoldReports.ReportViewerEnums.ProcessingMode.Local)
.ReportPath("~/Resources/product-list.rdlc")
.ReportServiceUrl("/api/ReportViewer")
.ReportError("onReportError")
)
<script type="text/javascript">
function onReportError(args) {
alert(args.errmsg);

```

```
args.cancel = true;
}
</script>
`
```

To suppress the default error dialog, set the cancel argument to true.

Show error

The `showError` event is invoked whenever users click a report item that contains an error in processing. It provides detailed information about the cause of error. You can hide the default dialog and show your customized dialog.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ProcessingMode(BoldReports.ReportViewerEnums.ProcessingMode.Local)
.ReportPath("~/Resources/product-list.rdlc")
.ReportServiceUrl("/api/ReportViewer")
.showError("onShowError")
)
<script type="text/javascript">
function onShowError(args) {
alert("Error code : " + args.errorCode + "\n" +
"Error Detail : " + args.errorDetail + "\n" +
"Error Message : " + args.errorMessage);
args.cancel = true;
}
</script>
`
```

Drill through

When a drill through item is selected in a report, it invokes the `DrillThrough` event. You can change the drill through arguments such as report parameter and data sources. The following sample code can be used to change the drill through report name and set the parameter value before the drill through report is rendered.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-person-details.rdl")
.DrillThrough("onDrillThrough")
```

```

)
<script type="text/javascript">
function onDrillThrough(args) {
args.actionInfo.ReportName = "personal-details";
args.actionInfo.Parameters = [{ name: 'EmployeeID', value: ['3'] }];
}
</script>
`

```

Hyperlink

The **Hyperlink** event occurs when users click a hyperlink in a report, before the hyperlink is followed. The following sample code redirects to a new custom URL and cancels the component default action.

```

`js
@(Html.Bold()).ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-person-details.rdl")
.Hyperlink("onHyperlink")
)
<script type="text/javascript">
function onHyperlink(args) {
args.cancel = true;
//You can modify the URL here
window.open(args.actionInfo.Hyperlink);
}
</script>
`

```

Handle post actions

Report processing actions are sent in an Ajax request to exchange data with the Web API service. You can handle post actions event to customize the Ajax requests.

- **AjaxBeforeLoad**
- **AjaxSuccess**
- **AjaxError**

AjaxBeforeLoad

This event can be triggered before an Ajax request is sent to the Report Viewer Web API service. It allows you to set additional headers and custom data in the Ajax request. The following code sample demonstrates adding custom authorization header and passing default parameter values to service.

Add custom header to Ajax request

Initialize the `AjaxBeforeLoad` event in the script and add the authorization token to the `headers` property.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.AjaxBeforeLoad("onAjaxRequest")
)
<script type="text/javascript">
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
}
</script>
`
```

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded from [here](#).

Get the custom header value from the `HttpContext` header collection using the key name specified at client side.

```
`csharp
string authenticationHeader;

public object PostReportAction([FromBody] Dictionary<string, object> jsonResult)
{
if (jsonResult != null)
{
//Get client side custom ajax header and store in local variable
authenticationHeader = HttpContext.Current.Request.Headers["Authorization"];
//Perform your custom validation here
if (authenticationHeader == "")
{
return new Exception("Authentication failed!!!");
}
}
}
```

```

else
{
return ReportHelper.ProcessReport(jsonResult, this);
}
}
return null;
}
`

```

Perform your own action to validate the header values.

Pass custom data in Ajax request

Use the `data` property to set custom data to the server in the Ajax request. In the following code sample, parameter values are passed to the server side.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.AjaxBeforeLoad("onAjaxRequest")
)
<script type="text/javascript">
function onAjaxRequest(args) {
//Passing custom data to server
var customerID = "CI0021";
args.data = customerID;
}
</script>
`

```

The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates to change the datasource connection strings based on Customer ID in the `OnInitReportOptions` method.

```

`csharp
string customerID = null;
public object PostReportAction(Dictionary<string, object> jsonResult)
{
if (jsonResult != null)

```



```
{
if (jsonResult.ContainsKey("customData"))
{
//Get client side custom data and store in local variable.
customerID = jsonResult["customData"].ToString();
}
}
return ReportHelper.ProcessReport(jsonResult, this);
}
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
if (customerID != null)
{
if(customerID.Contains("CI0021"))
{
//If you are changing the connection string based on customer id then could you please change the
connection string as below.
//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));
reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>;"));
}
else if(customerID.Contains("CI0022"))
{
//If you are changing the connection string based on customer id then could you please change the
connection string as below.
//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));
reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>;"));
}
}
}
```

`

AjaxSuccess

To perform custom action or show user defined message, use the **AjaxSuccess** event on the successful Ajax request.

`js

```
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.AjaxBeforeLoad("onAjaxRequest")
.AjaxSuccess("onAjaxSuccess")
)
<script type="text/javascript">
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
//Passing custom parameter data to server
args.data = [{ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'] }];
}
function onAjaxSuccess(args) {
//Perform your custom success message here
alert("Ajax request success!!!");
}
</script>
```

`

AjaxError

The **AjaxError** event is called, if an error occurred with the request, you can display the customized error detail in the event method.

`js

```
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.AjaxBeforeLoad("onAjaxRequest")
.AjaxError("onAjaxFailure")
)
<script type="text/javascript">
```

```
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
//Passing custom parameter data to server
args.data = [{ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'] }];
}
function onAjaxFailure(args) {
alert("Status: " + args.status + "\n" +
"Error: " + args.responseText);
}
</script>
`
```

You can never have both an error and a success callback with a request.

[Change Report Viewer default WEB API action with custom endpoint](#)

The `actionName` event argument allows to change the methods of `IReportController` to custom WEB API action endpoints.

[Change report processing endpoint](#)

Create a new Web Api action method in Report Viewer API controller as in the following code snippet,

```
`csharp
public object PostReportCustomAction(Dictionary<string, object> jsonResult)
{
return ReportHelper.ProcessReport(jsonResult, this);
}
`
```

The custom method must have the `Dictionary<TKey, TValue>` argument and add code `ReportHelper.ProcessReport` for processing the report.

Register the event `ajaxBeforeLoad` in your html page and set the newly created name to `actionName` property as in the below code snippet.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.AjaxBeforeLoad("onAjaxRequest")
)
<script type="text/javascript">
```

Error logging in ASP.NET MVC Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

```
function onAjaxRequest(args) {  
    args.actionName = "PostReportCustomAction";  
}  
</script>  
`
```

Change export action endpoint

Create a new Web Api action method in Report Viewer API controller as in the following code snippet,

```
`csharp  
public object ExportReportCustomAction()  
{  
    return ReportHelper.ProcessReport(null, this);  
}  
`
```

The custom method must have the code `ReportHelper.ProcessReport` for exporting the report.

Register the event `onExportProgressChanged` in your html page and set the newly created name to `actionName` property as in the below code snippet.

```
`js  
@(Html.Bold().ReportViewer("viewer")  
    .ReportServiceUrl("/api/ReportViewer")  
    .ReportPath("~/Resources/sales-order-detail.rdl")  
    .ExportProgressChanged("onExportProgressChanged")  
)  
<script type="text/javascript">  
    function onExportProgressChanged(args) {  
        if(args.stage === "exportStarted"){  
            args.actionName = "ExportReportCustomAction";  
        }  
    }  
</script>  
`
```

Error logging in ASP.NET MVC Report Viewer

If an error occurred in report processing, ASP.NET MVC Report Viewer displays short messages about the error in view. You need to configure the `ApiController` to save all logs, stack trace, and error information into a physical file location.

Error logging in ASP.NET MVC Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

This section explains how to log the detailed error information to your ASP.NET MVC application.

This section requires a ASP.NET MVC Report Viewer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

1. In Solution Explorer, open the Report Viewer Controller file.
2. Inherit the `IReportLogger` interface and implement the interface methods.

```
`csharp
public class ReportViewerController : ApiController, IReportController, IReportLogger
{
    public void LogError(string message, Exception exception, MethodBase methodType, ErrorType
    errorType)
    {
        throw new NotImplementedException();
    }

    public void LogError(string errorCode, string message, Exception exception, string errorDetail, string
    methodName, string className)
    {
        throw new NotImplementedException();
    }
}
```

3. Create a method in `ReportViewerController` to write the error text into application folder.

```
`csharp
internal void WriteLogs(string errorMessage)
{
    string filePath = HttpContext.Current.Server.MapPath("/App_Data/Errordetails.txt");
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
        writer.WriteLine(errorMessage);
    }
}
```

Error logging in ASP.NET MVC Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

4. Invoke the newly created function in `LogError` as follows.

```
`csharp
public void LogError(string message, Exception exception, MethodBase methodType, ErrorType
errorType)
{
WriteLogs(string.Format("Error Message: {0} \n Stack Trace: {1}", message, exception.StackTrace));
}

public void LogError(string errorCode, string message, Exception exception, string errorDetail, string
methodName, string className)
{
WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2} \n Stack Trace:
{3}", className, methodName, errorDetail, exception.StackTrace));
}
`
```

In cases of any issues faced in the report rendering, share the log file to our technical support team to get assistance on that.

5. The final controller is given as follows, you can replace it in your application.

```
`csharp
public class ReportViewerController : ApiController, IReportController, IReportLogger
{
// Post action for processing the RDL/RDLC report
public object PostReportAction(Dictionary<string, object> jsonResult)
{
return ReportHelper.ProcessReport(jsonResult, this);
}

// Get action for getting resources from the report
[System.Web.Http.ActionName("GetResource")]
[AcceptVerbs("GET")]
public object GetResource(string key, string resourcetype, bool isPrint)
{
return ReportHelper.GetResource(key, resourcetype, isPrint);
}

// Method that will be called when initialize the report options before start processing the report
```

Error logging in ASP.NET MVC Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

[NonAction]

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
// You can update report options here
```

```
}
```

```
// Method that will be called when reported is loaded
```

[NonAction]

```
public void OnReportLoaded(ReportViewerOptions reportOption)
```

```
{
```

```
// You can update report options here
```

```
}
```

```
public void LogError(string message, Exception exception, MethodBase methodType, ErrorType  
errorType)
```

```
{
```

```
WriteLogs(string.Format("Error Message: {0} \n Stack Trace: {1}", message, exception.StackTrace));
```

```
}
```

```
public void LogError(string errorCode, string message, Exception exception, string errorDetail, string  
methodName, string className)
```

```
{
```

```
WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2} \n Stack Trace:  
{3}", className, methodName, errorDetail, exception.StackTrace));
```

```
}
```

```
internal void WriteLogs(string errorMessage)
```

```
{
```

```
// Error details text file path location
```

```
string filePath = HttpContext.Current.Server.MapPath("/App_Data/Errordetails.txt");
```

```
using (StreamWriter writer = new StreamWriter(filePath, true))
```

```
{
```

```
writer.AutoFlush = true;
```

```
writer.WriteLine(errorMessage);
```

```
}
```

```
}
```

```
}
```

```
,
```

Print report

The Report Viewer provides print report option in the toolbar to print a copy of the report. The page setup dialog allows you to set the paper size or other page setup properties. To see print margins, click **Print Layout** on the toolbar.

You can set values in the Page Setup dialog box for current session only. When you close the report and reopen it, it will have the default values again. The default values of the Page Setup dialog is based on the report properties set in the design view.

View report in print mode

Print margins are displayed in the print layout only. To view report in print mode by default, set the `PrintMode` property to true.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.PrintMode(true)
)
`
```

By default, the Report Viewer renders report in normal layout in which the print margins are not displayed.

Print in new page

To open the print in a new tab of the current browser, set the `printOption` property to `NewTab`. By default, it shows the print dialog in the same page.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.PrintMode(true)
.PrintOption(BoldReports.ReportViewerEnums.PrintOptions.NewTab)
)
`
```

The pop-up blocker should be enabled for the page to open the print view in new tab.

Set page orientation and paper size

You can specify the print page paper size and orientation at client-side to change the page setup properties by setting the `PageSettings` property.

The following code example illustrates how to set page orientation and paper size in the Report Viewer for client side.


```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.PrintMode(true)
.PageSettings(page =>
page.Orientation(BoldReports.ReportViewerEnums.Orientation.Landscape).PaperSize(BoldReports.Repo
rtViewerEnums.PaperSize.Letter))
)
`
```

Set report margin

To set margin values to the report page setup, use the **Margins** property and specify the value to top, right, bottom, and left.

The following code example illustrates how to set report margin in the Report Viewer for client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.PrintMode(true)
.PageSettings(page => page.Margins(margin => margin.Top(0.5).Left(0.5).Bottom(0.5).Right(0.5)))
)
`
```

The values set in the margin property is considered as inches input.

Set page height and width

To set height and width values to the report page setup, use the **Height** and **Width** properties.

The following code example illustrates how to set page height and width in the Report Viewer for client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.PrintMode(true)
.PageSettings(page => page.Height(11.69).Width(8.27))
)
`
```

The values set in the height and width properties are considered as inches input.

Print report with images

When the report has more images, the browser will send the report stream to the print dialog before the images are completely loaded. To load the print report stream with complete images, you should set the `EmbedImageData` property to true in `OnInitReportOptions` as shown in the following code.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.EmbedImageData = true;
}
`
```

Replace the following code sample in the client-side.

```
`js
@(Html.Bold().ReportViewer("viewer")
    .ReportServiceUrl("/api/PrintWithImages")
    .ReportPath("~/Resources/product-details.rdl")
)
`
```

In this tutorial, the `product-details.rdl` report is used, and it can be downloaded from [here](#).

External styles in report printing

While printing report, the external styles are used in the application overrides printable page style and prints output with incorrect alignments. To avoid the external script overriding, set the `isStyleLoad` property to false, which will print the page using only the Report Viewer styles.

```
`js
@(Html.Bold().ReportViewer("viewer")
    .ReportServiceUrl("/api/PrintWithImages")
    .ReportPath("~/Resources/product-details.rdl")
    .ReportPrint("onReportPrint")
)
<script type="text/javascript">
function onReportPrint(args) {
    args.isStyleLoad = false;
}
</script>
```

Show print progress

Report Viewer provides events to show the progress information when the printing takes long time to complete.

To show print progress, follow these steps:

1. Set the `PrintProgressChanged` event in Report Viewer initialization.
2. Implement the function and add code samples to show a custom message based on the print progress status as shown in the following code snippet.

```
`js
@({Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/PrintWithImages")
.ReportPath("~/Resources/product-details.rdl")
.PrintProgressChanged("onPrintProgressChanged")
})
<script type="text/javascript">
function onPrintProgressChanged(args) {
if (args.stage == "beginPrint") {
$('#viewer').ejWaitingPopup({ showOnInit: true, cssClass: "customStyle", text: "Preparing print data..
Please wait..." });
}
if (args.stage == "printStarted") {
var popupObj = $('#viewer').data('ejWaitingPopup');
popupObj.hide();
}
else if (args.stage == "preparation") {
console.log(args.stage);
if (args.preparationStage == "dataPreparation") {
console.log(args.preparationStage);
console.log(args.totalPages);
console.log(args.currentPage);
if (args.totalPages > 1 && args.currentPage > 1) {
var progressPercentage = Math.floor((args.currentPage / args.totalPages) * 100);
if (progressPercentage > 0) {
var popupObj = $('#viewer').data('ejWaitingPopup');
```

```

popupObj.setModel({ text: "Preparing print data.." + progressPercentage + " % completed.. Please
wait..." });
}
}
}
}
args.handled = true;
}
</script>
`

```

Remove empty spaces in printing

The extra blank page is created when the body of your report is too wide for your page. To make the report appear on a single page, all the content within the report body must fit on the physical page, and the body width should be as the following formula.

Body Width <= Page Width - (Left Margin + Right Margin)

For more details about removing the empty pages in the report while designing, refer to the knowledge base article of [report page sizing](#).

Export report

The Report Viewer provides events and properties to control and customize the report exporting functionality.

Export event handling

You can show the progress information, when the exporting process takes long time to complete using the `ExportProgressChanged` event.

1. Set the `ExportProgressChanged` event in Report Viewer initialization.
2. Implement the function and replace the following code samples to show a custom message based on the progress stage.

The following code example demonstrates how to export event handling in the Report Viewer at client side.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ExportProgressChanged("onExportProgressChanged")
)
<script type="text/javascript">

```

```
function onExportProgressChanged(args) {
  if (args.stage === "beginExport") {
    console.log(args.stage);
    args.format =
    $('#viewer').ejWaitingPopup({ showOnInit: true, cssClass: "customStyle", text: "Preparing exporting
    document.. Please wait..." });
  }
  else if (args.stage === "exportStarted") {
    console.log(args.stage);
    var popupObj1 = $('#viewer').data('ejWaitingPopup');
    popupObj1.hide();
  }
  else if (args.stage === "preparation") {
    console.log(args.stage);
    console.log(args.format);
    console.log(args.preparationStage);
    if (args.format === "PDF" && args.preparationStage === "documentPreparation") {
      console.log(args.totalPages);
      console.log(args.currentPage);
      if (args.totalPages > 1 && args.currentPage > 1) {
        var progressPercentage = Math.floor((args.currentPage / args.totalPages) * 100);
        if (progressPercentage > 0) {
          var popupObj2 = $('#viewer').data('ejWaitingPopup');
          popupObj2.setModel({ text: "Preparing exporting document.." + progressPercentage + " % completed..
          Please wait..." });
        }
      }
    }
    args.handled = true;
  }
}</script>
```

Change Excel and Word export format

Allows you change the default file format to any other file format using the **ExcelFormats** and **WordFormats** properties.

The following code example demonstrates how to change Excel and Word export format in the Report Viewer at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ExportSettings(export =>
export.ExcelVersion(BoldReports.ReportViewerEnums.ExcelFormats.Excel2013).WordFormatType(BoldR
eports.ReportViewerEnums.WordFormats.Word2013))
)
`
```

Hide specific export type for report

Show or hide the default export types available in the component using the **ExportOptions** property.

The following code example demonstrates how to hide specific export type to the report in the Report Viewer at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ExportSettings(export => export.ExportOptions(BoldReports.ReportViewerEnums.ExportOptions.All &
~BoldReports.ReportViewerEnums.ExportOptions.PDF))
)
`
```

PDF export options

The **PDFOptions** provides properties to manage PDF export behaviors. You should set the properties in the **OnInitReportOptions** method of the Web API service.

Export with complex scripts

To export reports with the complex scripts, set the **ComplexScript** property of **PDFOptions** instance to true.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
```

```
{
EnableComplexScript = true
};
}
```

PDF conformance

You can export the report as a PDF/A-1b document by specifying the PdfConformanceLevel.Pdf_A1B conformance level in the PdfConformanceLevel property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
{
PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B
};
}
```

Add custom PDF fonts

You can add custom fonts to the PDF exported document by adding the font streams to Fonts collection in PDFOptions instance.

To add custom fonts to the PDF exported document, follow these steps:

1. Add the font .ttf files into your application Resourecs folder.
2. In the Solution Explorer, open the properties of the font file and set the Copy property to Output Directory as Copy always.
3. Initialize the Font collection and add the font stream to it.

The key value provided in the font collection should be same as in the report item font property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
string basePath = _hostingEnvironment.WebRootPath;
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
{
//Load Missing font stream
Fonts = new Dictionary<string, System.IO.Stream>
```

```
{
{ "Segoe UI", new FileStream(basePath + @"\Resourecs\font_symbols.ttf", FileMode.Open,
FileAccess.Read) }
}
};
}
,
```

If any fonts used in the report definition is not installed or available in the local system, then you should load the font stream. In the above code, `font_symbols` font stream is loaded to export the `sales-order-detail.rdl` report as symbols.

Word export options

The `WordOptions` provides properties to manage Word document export behaviors.

Word document type

You can save the report to the required document version by setting the `FormatType` property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
FormatType = BoldReports.Writer.WordFormatType.Docx
};
}
,
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
LayoutOption = BoldReports.Writer.WordLayoutOptions.TopLevel,
ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{

```


Bottom = 0.5f,

Top = 0.5f

}

};

}

,

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior.

The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, add an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without password. The following are the types of protection:

- **AllowOnlyComments**: Adds or modifies only the comments in the Word document.
- **AllowOnlyFormFields**: Modifies the form field values in the Word document.
- **AllowOnlyRevisions**: Accepts or rejects the revisions in the Word document.
- **AllowOnlyReading**: Views the content only in the Word document.
- **NoProtection**: Accesses or edits the Word document contents as normally.

```
`csharp
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
    {
        ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading
    };
}
```

Excel export options

The **ExcelOptions** provides properties to manage Excel document export behaviors.

Excel document type

You can save the report to the required excel version by setting the **ExcelSaveType** property.

```
`csharp
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
```

```
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013
};
}
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` to `IgnoreCellMerge`.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
    {
        LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
    };
}
```

Protecting Excel document from editing

You can restrict the Excel document from editing either by providing the `ExcelSheetProtection` or enabling the `ReadOnlyRecommended` properties.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
    {
        ReadOnlyRecommended = true,
        ExcelSheetProtection = Syncfusion.XlsIO.ExcelSheetProtection.DeletingColumns
    };
}
```

CSV export options

The `CsvOptions` allows you to change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document.

```
`csharp
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.CsvOptions = new BoldReports.Writer.CsvOptions()
    {
        Encoding = System.Text.Encoding.Default,
        FieldDelimiter = ",",
        UseFormattedValues = false,
        Qualifier = "#",
        RecordDelimiter = "@",
        SuppressLineBreaks = true,
        FileExtension = ".txt"
    };
}
```

Password protect exported document

Allows you protect the exported document such as PDF, Word, Excel, and PowerPoint from unauthorized users by encrypting the document using encryption password. The following code snippet illustrates how to encrypt the exported document with user-defined password.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //PDF encryption
    reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions();
    reportOption.ReportModel.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity()
    {
        UserPassword = "password"
    };
    //Word encryption
    reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
    {
        EncryptionPassword = "password"
    };
    //Excel encryption
    reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
```

```
{
    PasswordToModify = "password",
    PasswordToOpen = "password"
};
//PPT encryption
reportOption.ReportModel.PPTOptions = new BoldReports.Writer.PPTOptions()
{
    EncryptionPassword = "password"
};
}
```

Password protection is not supported for HTML export format.

Change file name in export

You can change the file name of report in export using the **FileName** property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ExportSettings = new BoldReports.Writer.ExportSettings()
    {
        FileName = "Invoice"
    };
}
```

Change image quality in export

You can change image quality of data visualization items in report export using the **ImageQuality** property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ExportSettings = new BoldReports.Writer.ExportSettings()
    {
        ImageQuality = 4
    };
}
```

Custom Actions

Add user defined buttons to the toolbar and invoke custom actions using the **Report Viewer** property. You can create a custom email button to send an email with the rendered report to users.

Add email button

1. Create an email button in the toolbar using the **CustomItems** property with the values such as **GroupIndex**, **Index**, **Type**, **CssClass**, and **Tooltip**. The **ToolBarItemClick** event triggers when you click the email button.
2. Access the Report Viewer model and create a JSON array for sending requests to the Web API Server. You can use the following codes to create an event with custom action.
3. You can use the following codes to add email button from controller and passing the data to view using **ViewBag**.

```
`csharp
public ActionResult Index()
{
    ToolbarSettings toolbarSettings = new ToolbarSettings();
    toolbarSettings.CustomItems = new List<CustomItem>();
    var customItem = new CustomItem()
    {
        GroupIndex = 1,
        Index = 1,
        CssClass = "e-icon e-mail e-reportviewer-icon",
        Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
        Id = "E-Mail",
        Tooltip = new ToolTip() { Header = "E-Mail", Content = "Send rendered report as mail attachment" }
    };
    toolbarSettings.CustomItems.Add(customItem);
    ViewBag.toolbarSettings = toolbarSettings;
    return View();
}
```

4. You can use the following codes to set an **ToolbarSettings** property at client side.

```
`js
```

```

@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbarSettings =>
toolbarSettings.CustomItems(@ViewBag.toolbarSettings.CustomItems))
)

```

5. You can use the following codes to create an **ToolBarItemClick** event at client side.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolBarItemClick("onToolBarItemClick")
)
<script type="text/javascript">
function onToolBarItemClick(args) {
alert('Action Triggered');
}
</script>

```

6. You can use the following codes to get **ToolBarSettings** properties on a dynamic object using **ViewBag.toolbarSettings.CustomItems** and invoke custom actions using custom email button at client side.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbarSettings =>
toolbarSettings.CustomItems(@ViewBag.toolbarSettings.CustomItems))
.ToolBarItemClick("onToolBarItemClick")
)
<script type="text/javascript">
function onToolBarItemClick(args) {

```

```

if (args.value == "E-Mail") {
var proxy = $('#viewer').data('boldReportViewer');
var Report = proxy.model.reportPath;
var lastsIndex = Report.lastIndexOf("/");
var reportName = Report.substring(lastsIndex + 1);
var requurl = proxy.model.reportServiceUrl + '/SendEmail';
var _json = {
exportType: "PDF", reportViewerToken: proxy._reportViewerToken, ReportName: reportName
};
$.ajax({
type: "POST",
contentType: "application/json; charset=utf-8",
url: requurl,
data: JSON.stringify(_json),
dataType: "json",
crossDomain: true
})
}
}
</script>
`

```

Create custom email action

1. Create a new action method **SendEmail** in the Web API service.
2. Export the report to the required type using the **ReportHelper.GetReport** method to send a report stream as an attachment.
3. The following code sample exports the report to stream and send it as an attachment to a specified mail address. In the code, the **SmtptClient** method is used to send the report as an email attachment.

```

`csharp
public object SendEmail(Dictionary<string, object> jsonResult)
{
string _token = jsonResult["reportViewerToken"].ToString();
var stream = ReportHelper.GetReport(_token, jsonResult["exportType"].ToString());
stream.Position = 0;

```

```
if (!ComposeEmail(stream, jsonResult["reportName"].ToString()))
{
    return "Mail not sent !!!";
}
return "Mail Sent !!!";
}

public bool ComposeEmail(Stream stream, string reportName)
{
    try
    {
        MailMessage mail = new MailMessage();
        SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
        mail.IsBodyHtml = true;
        mail.From = new MailAddress("xx@gmail.com");
        mail.To.Add("xx@gmail.com");
        mail.Subject = "Report Name : " + reportName;
        stream.Position = 0;
        if (stream != null)
        {
            ContentType ct = new ContentType();
            ct.Name = reportName + DateTime.Now.ToString() + ".pdf";
            System.Net.Mail.Attachment attachment = new System.Net.Mail.Attachment(stream, ct);
            mail.Attachments.Add(attachment);
        }
        SmtpServer.Port = 587;
        SmtpServer.Credentials = new System.Net.NetworkCredential("xx@gmail.com", "xx");
        SmtpServer.EnableSsl = true;
        SmtpServer.Send(mail);
        return true;
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```



```
}  
return false;  
}  
`
```

In the above code sample, the report is exported to PDF format and send to users using the `SmptClient` method.

Custom properties

The custom properties helps you to include additional features that are not natively supported in the RDL reporting. This topic explains the list of custom properties supported in ASP.NET MVC Report Viewer.

See Also

- [Textbox Custom Properties](#)
- [Table Custom Properties](#)
- [Image Custom Properties](#)
- [Chart Custom Properties](#)
- [Report Custom Properties](#)
- [Parameter Custom Properties](#)
- [Export Custom Properties](#)

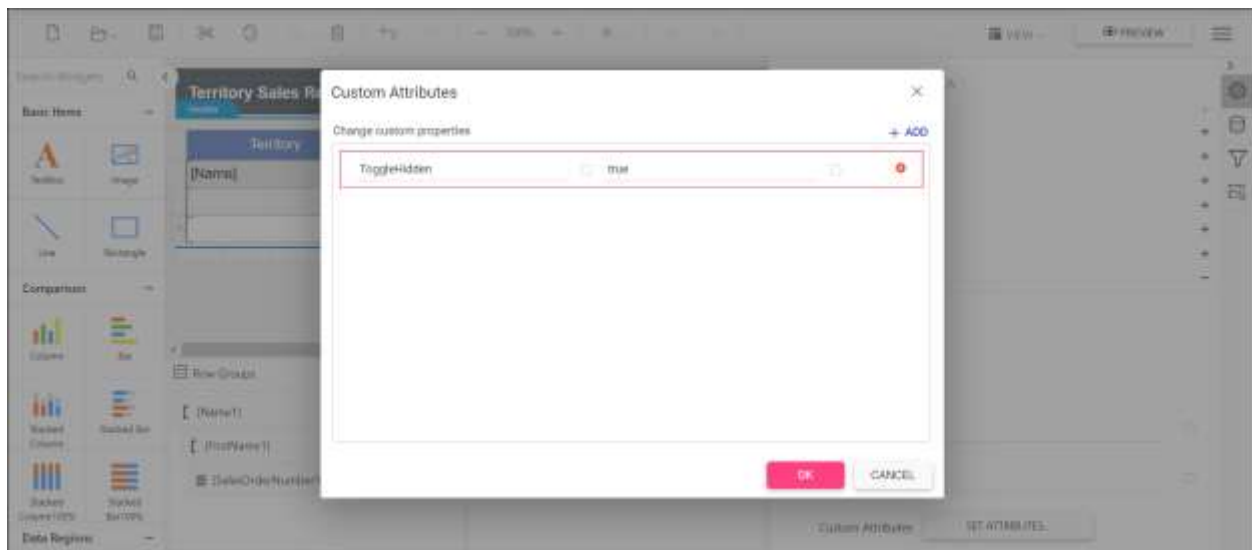
Textbox custom properties

This topic explains about the list of textbox report item custom properties that are supported to render in ASP.NET MVC Report Viewer.

Show or hide toggle icon in text box report item

The `ToggleHidden` custom property is used to show or hide the toggle icon in the textbox.

You can set the `ToggleHidden` property value, as shown in the below.



Before setting the toggle hidden property, the default value will be displayed as below.



The screenshot shows a report titled "Territory Sales Report" in a preview window. The table has four columns: Territory, Sales Person, Order Number, and Total Sales. The Territory column contains expandable items with a small triangle icon. The data is as follows:

Territory	Sales Person	Order Number	Total Sales
▶ Australia			\$1,943,016
	▶ Lynn Tsotlias		\$1,943,016
▶ Canada			\$12,808,458
	▶ Garrett Vargas		\$4,840,689
	▶ José Saravia		\$7,967,769
▶ Central			\$13,434,510
	▶ Jillian Carson		\$13,434,510
▶ France			\$6,083,691
	▶ Ranjit Varkey Chudukatti		\$6,083,691
▶ Germany			\$2,476,530
	▶ Rachel Valdez		\$2,476,530
▶ Northeast			\$12,433,503
	▶ Michael Blythe		\$12,433,503
▶ Northwest			\$6,305,407

Preview the report and see the toggle icon is hidden in the report.



The screenshot shows the same "Territory Sales Report" but with the toggle icons removed from the Territory column. The data is as follows:

Territory	Sales Person	Order Number	Total Sales
Australia			\$1,943,016
	Lynn Tsotlias		\$1,943,016
Canada			\$12,808,458
	Garrett Vargas		\$4,840,689
	José Saravia		\$7,967,769
Central			\$13,434,510
	Jillian Carson		\$13,434,510
France			\$6,083,691
	Ranjit Varkey Chudukatti		\$6,083,691
Germany			\$2,476,530
	Rachel Valdez		\$2,476,530
Northeast			\$12,433,503
	Michael Blythe		\$12,433,503
Northwest			\$6,305,407

Table custom properties

This topic explains about the list of table report item custom properties that are supported to render in ASP.NET MVC Report Viewer.

Limit number of table records on each page

The `RowsPerPage` custom property is used to specify the number of table records to display on each page. It supports integer data value greater than zero.

This property is ignored when table rows heights higher than current page size. Increase the report page height or reduce `RowsPerPage` count that fits within the page.

You can set the `RowsPerPage` property value, as shown in the below.

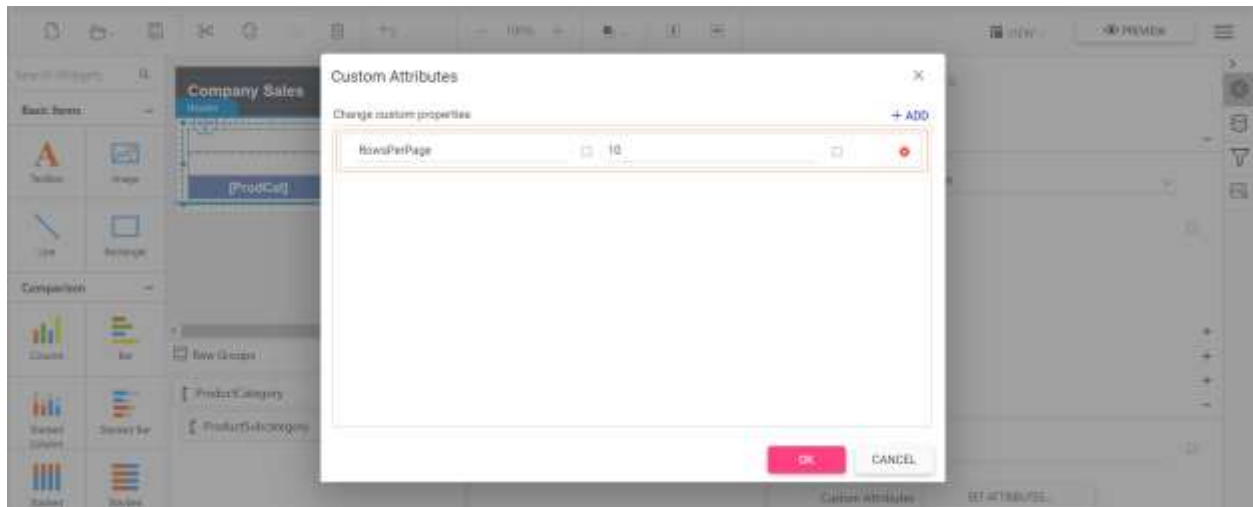


Image custom properties

This topic explains about the list of image custom properties that are supported to render in the ASP.NET MVC Report Viewer.

Angle

Set **Angle** custom property to image report item to rotate the image on a specified angle. It supports the angle values 0, 90, 180, and 270. You can set the property value, as shown in the below.

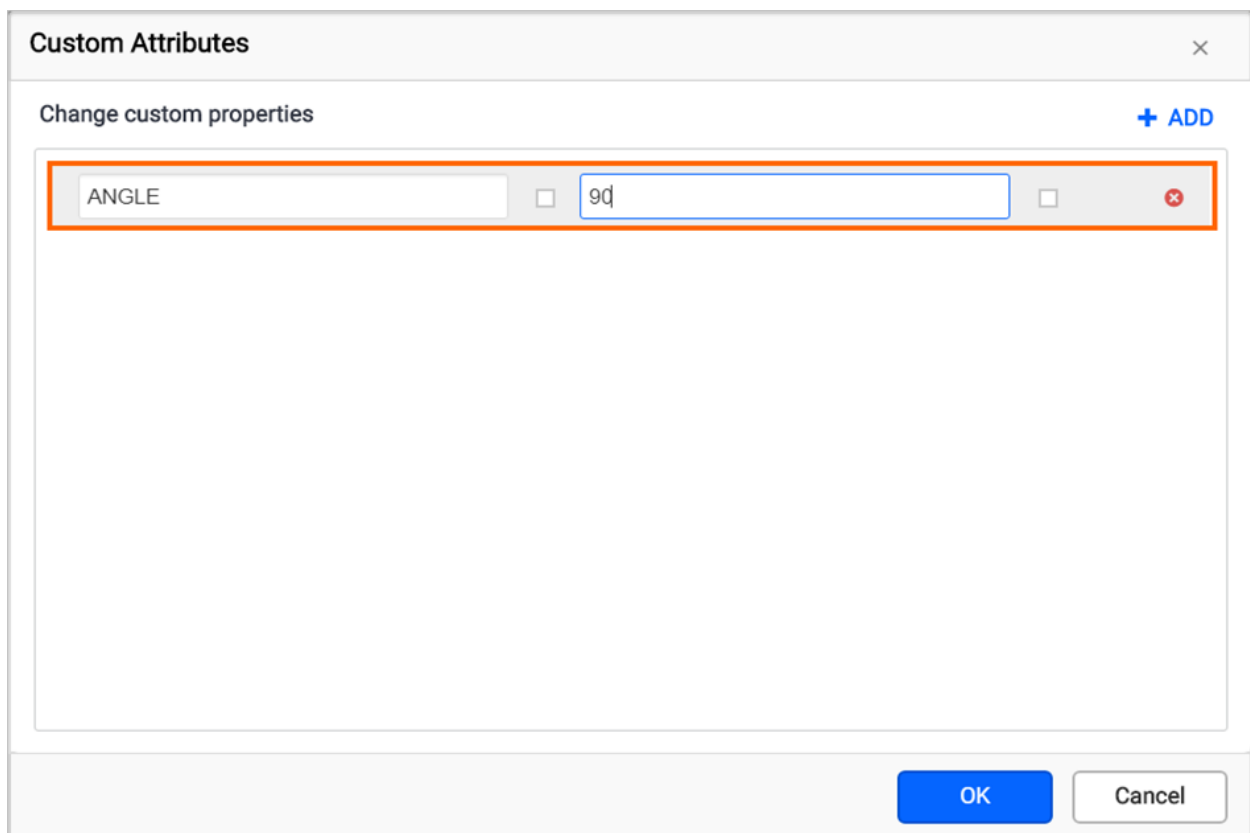
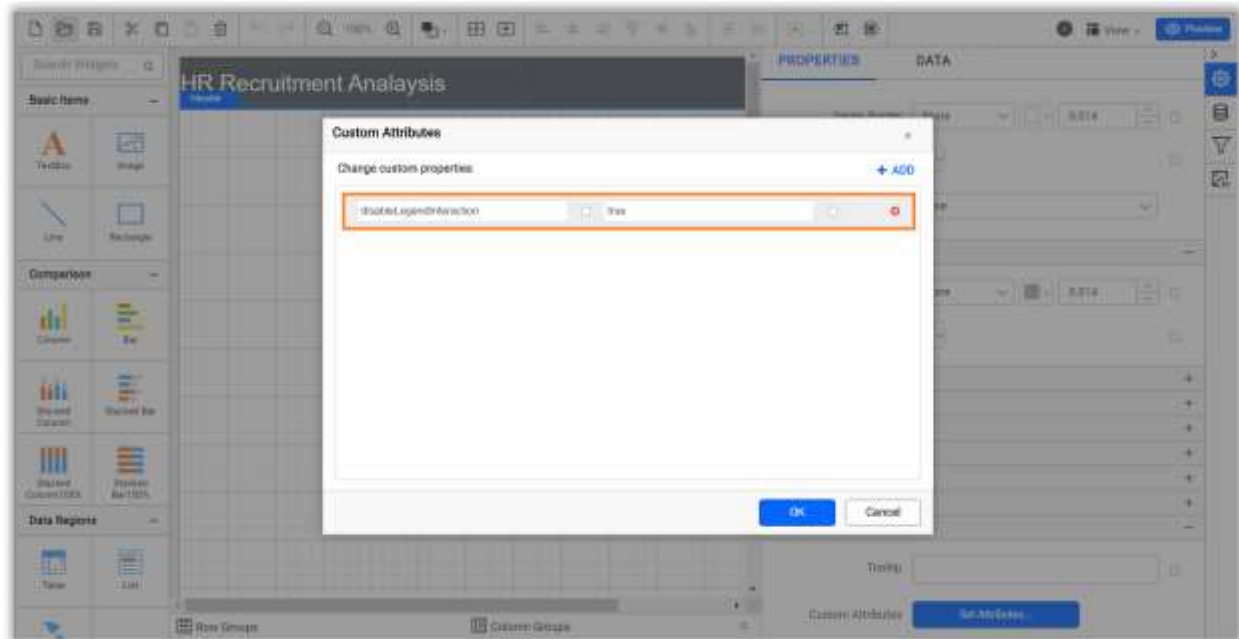


Chart custom properties

This topic explains about the list of chart custom properties that are supported to render in the ASP.NET MVC Report Viewer.

Disable legend item interaction

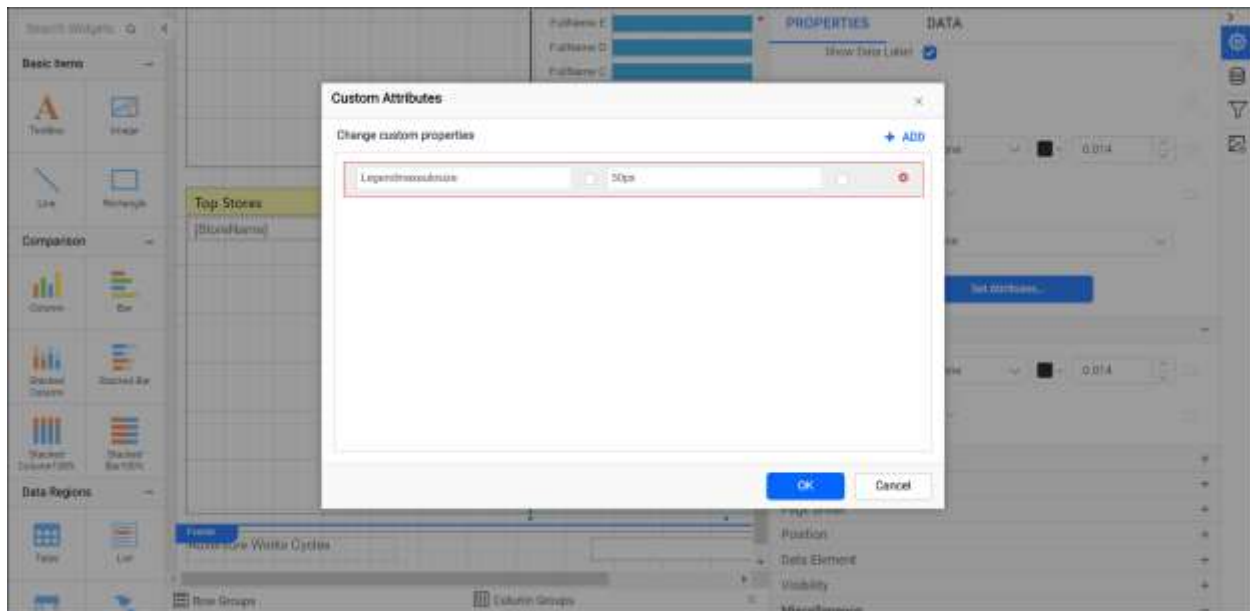
Set `DisableLegendInteraction` custom property value as `true` to stop the legend item interaction. The property value should be boolean. You can set the property value, as shown in the below.



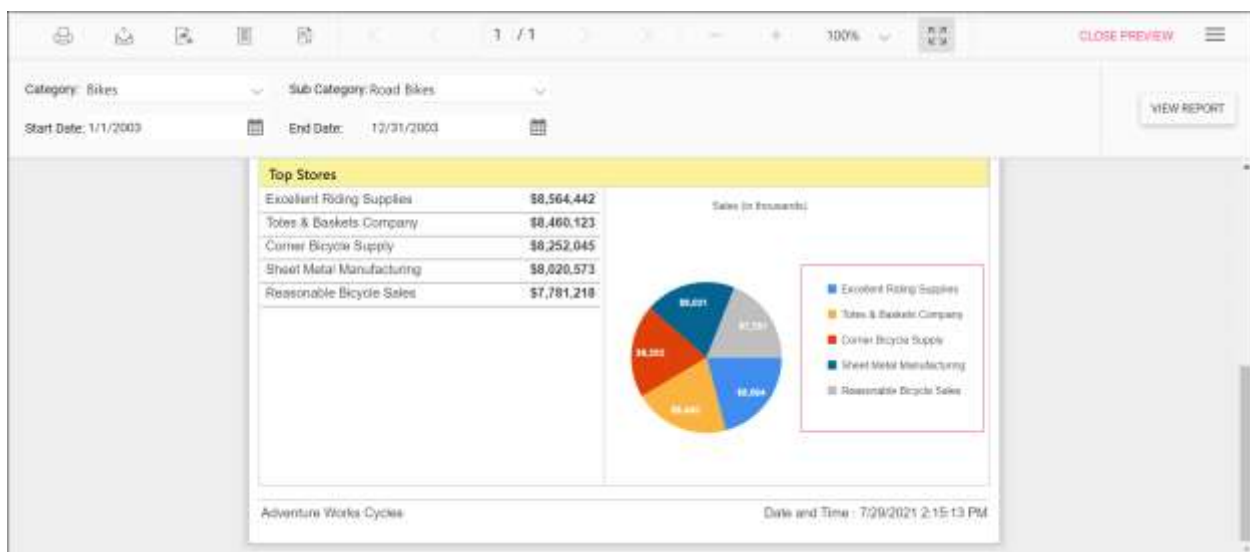
Set maximum size for chart legend

The `LegendMaxAutoSize` custom property specifies the maximum size of the legend container in the report.

You can set the `LegendMaxAutoSize` property value, as shown in the below.



Preview the report and see legend container size in chart report.

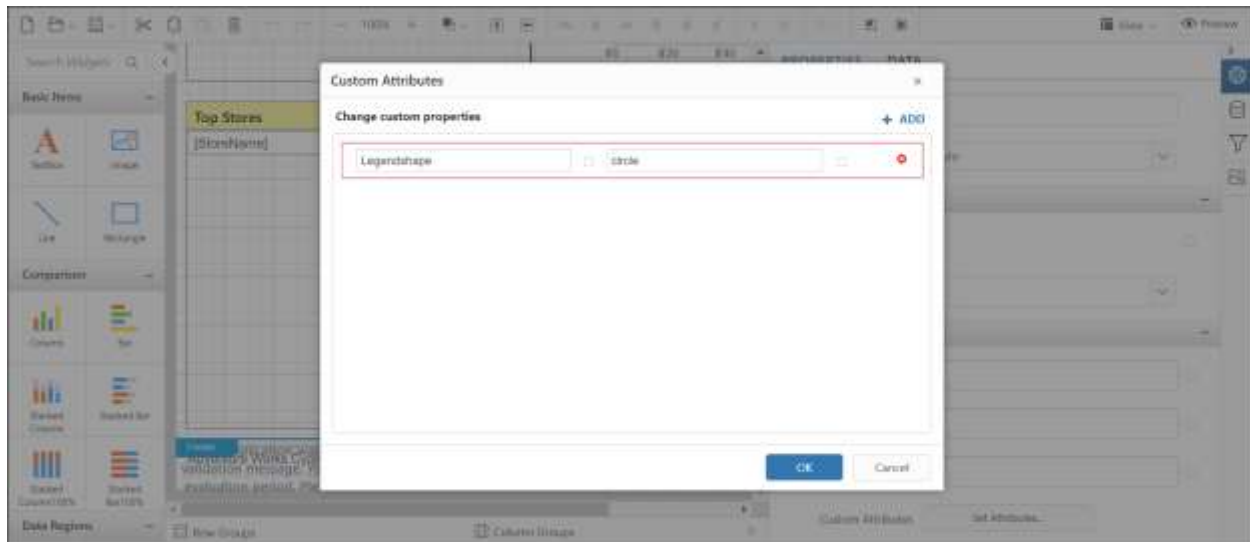


Change chart legend shape

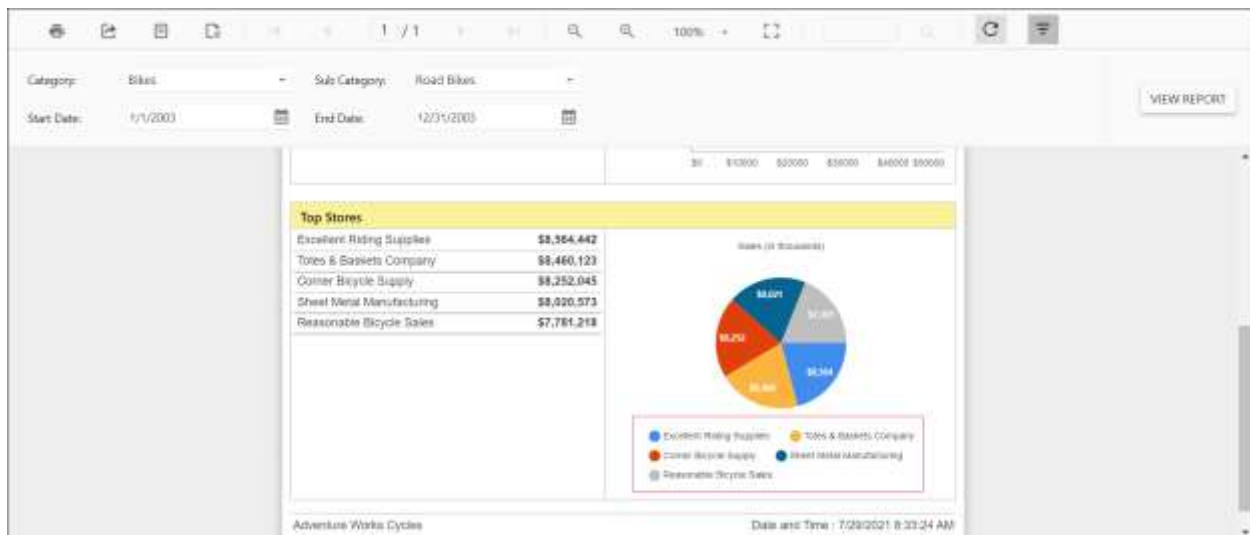
The `LegendShape` custom property allows changing the shape of the legend in the chart report item. By default, the `LegendShape` value is `rectangle`.

- Rectangle - legend shapes displayed in `rectangle`.
- Circle - legends are displayed in `circle`.
- Thumbnail - legends are displayed in `seriestype`.

You can set the `LegendShape` property value, as shown in the below.



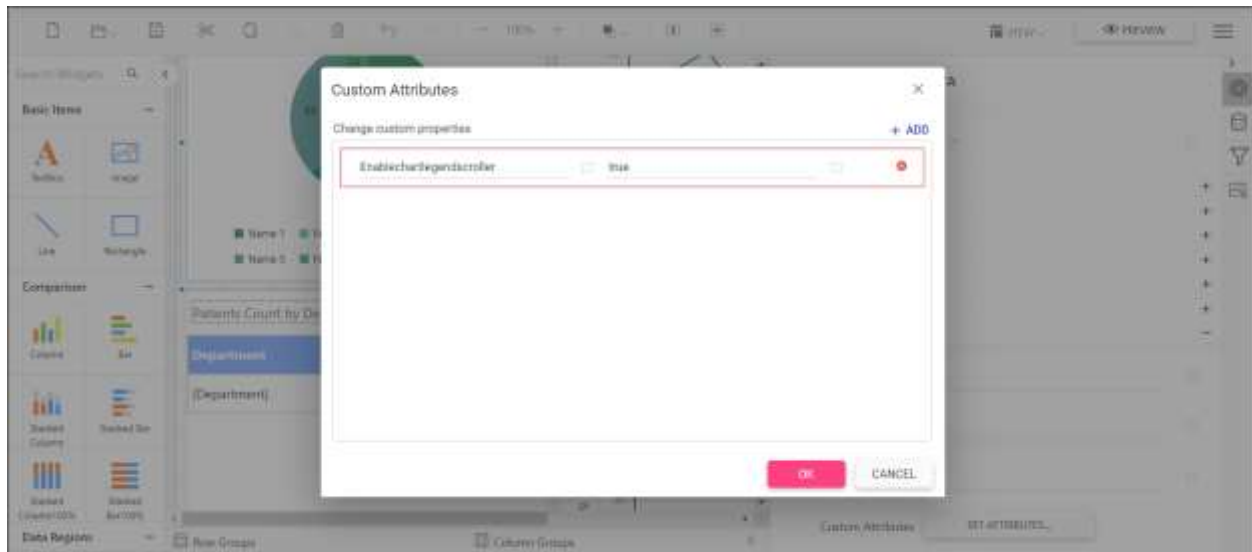
Preview the report and the see legend shape in chart report.



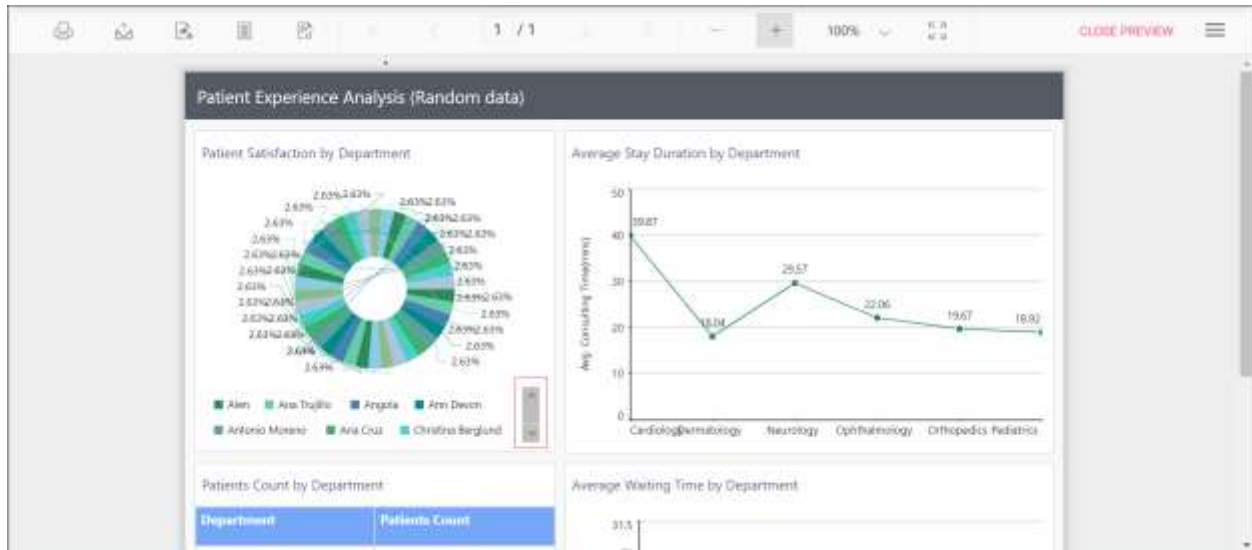
Show or hide chart legend scroller

The `EnableChartLegendScroller` custom property controls whether legend has to use scrollbar or not. The scrollbar appears depending upon size and position properties of legend. By default, the `EnableChartLegendScroller` value is `false`.

You can set the `EnableChartLegendScroller` property value, as shown in the below.



Preview the report and the see the legend scrollbar in chart report.



Set range padding for X-axis and Y-axis

Padding can be applied to the minimum and maximum extremes of the axis range by using the `XAxisRangePadding` and `YAxisRangePadding` property. The default value is `none`.

Numeric axis supports the following types of padding.

Name | Description

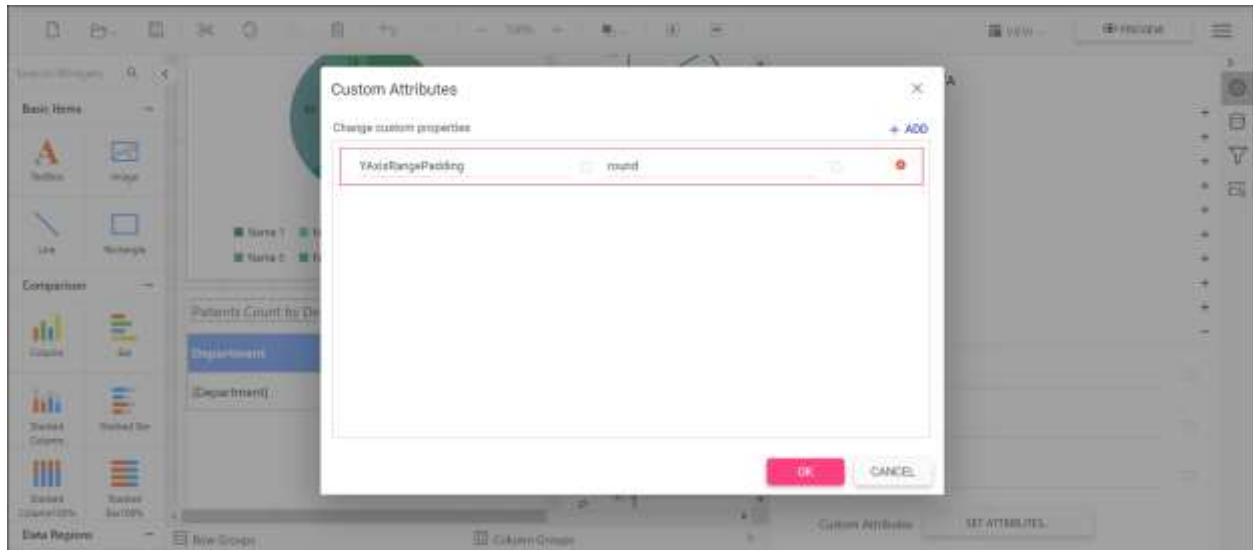
Additional | Interval of the axis is added as padding to the minimum and maximum values of the range

Normal | Padding is applied to the axis based on the range calculation

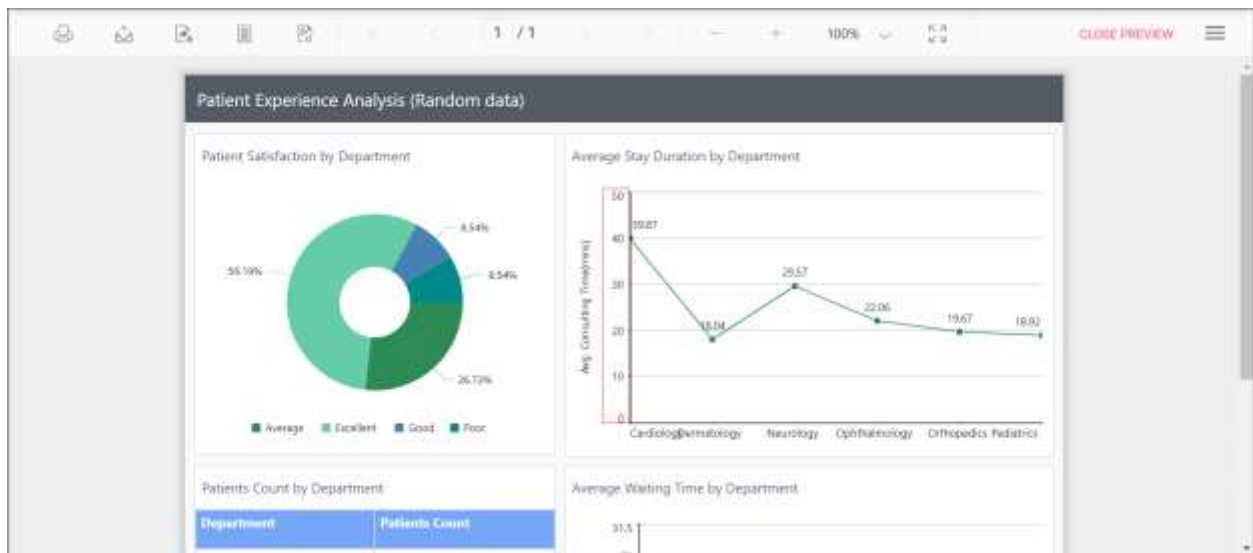
None | Padding cannot be applied to the axis

Round | Axis range is rounded to the nearest possible value divided by the interval

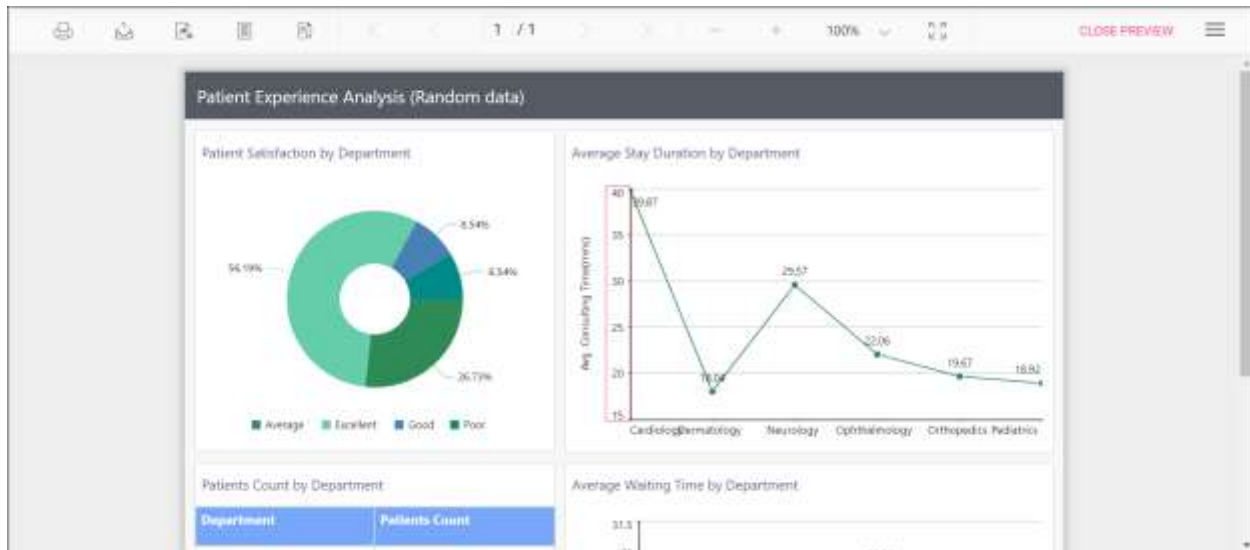
You can set the `XAxisRangePadding` and `YAxisRangePadding` property value, as shown in the below.



Before setting the range padding, the default value will be displayed as below.



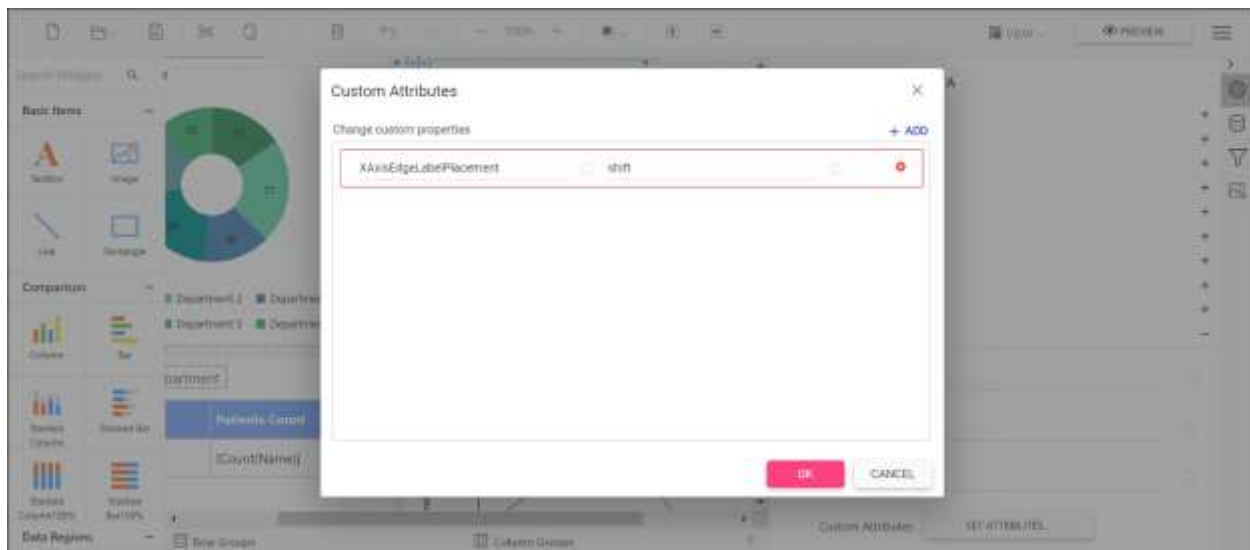
Set the padding range and see the changes in chart report as below.



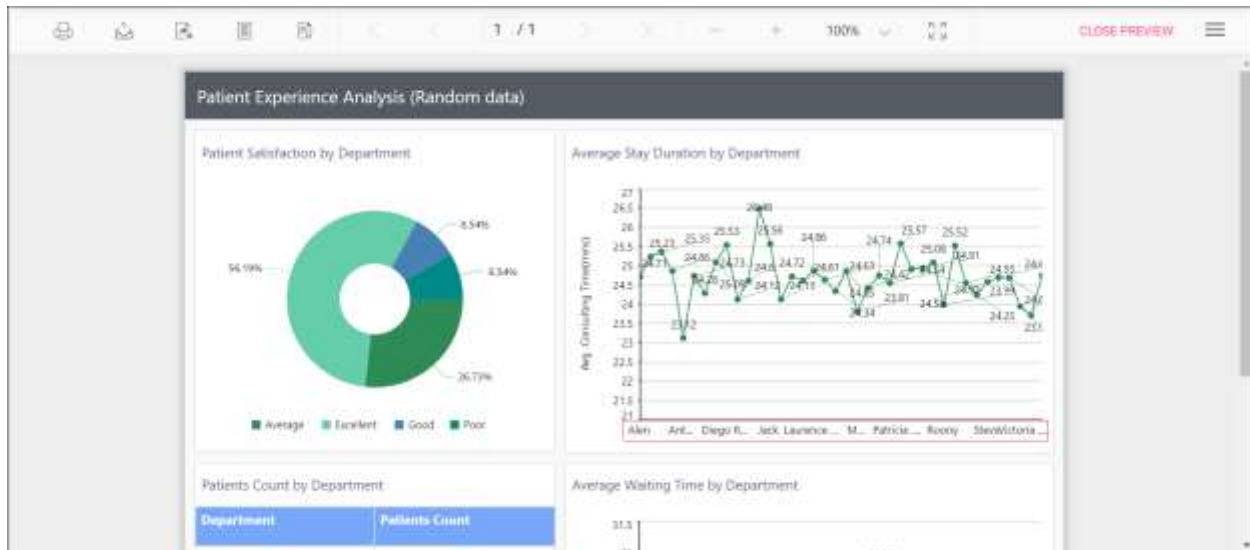
Control edge label placement in chart axis

Labels with long text at the edges of an axis may appear partially outside the chart. The `XAxisEdgeLabelPlacement` and `YAxisEdgeLabelPlacement` custom property can be used to avoid the partial appearance of the labels at the corners. The default value is `none`.

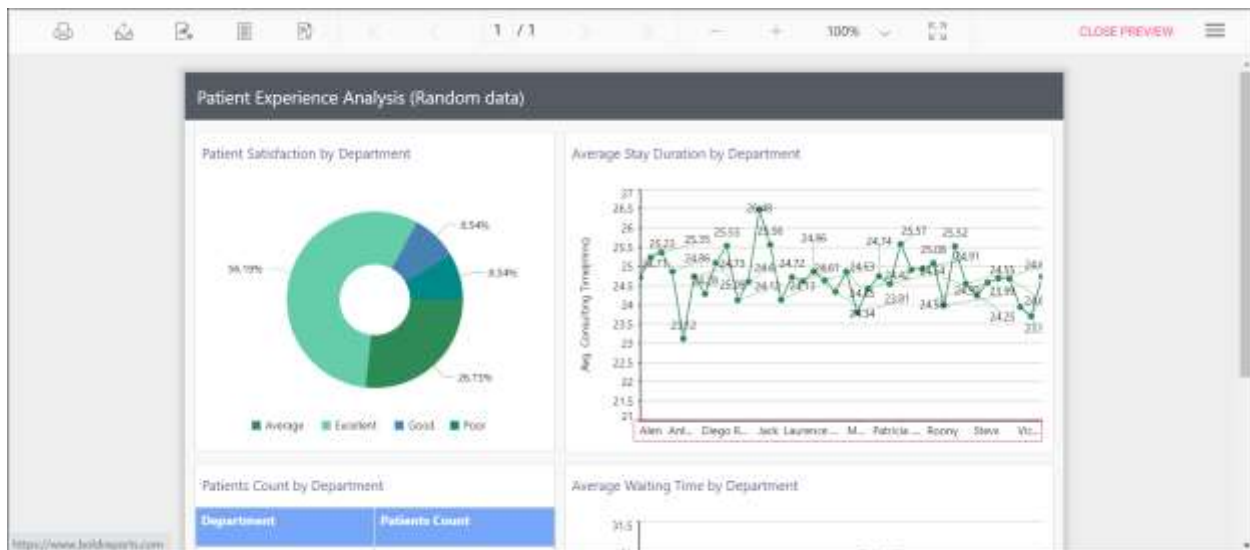
You can set the `XAxisEdgeLabelPlacement` property value, as shown in the below.



Before setting the edge label placement, the default value will be displayed as below.



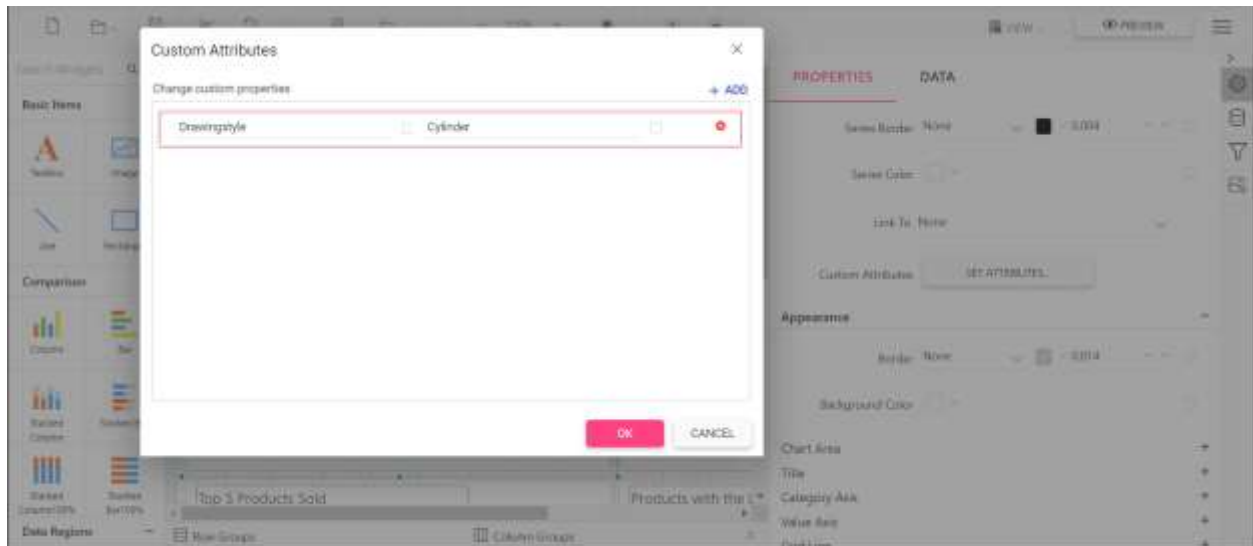
Set the edge label placement and see the changes in chart report as below.



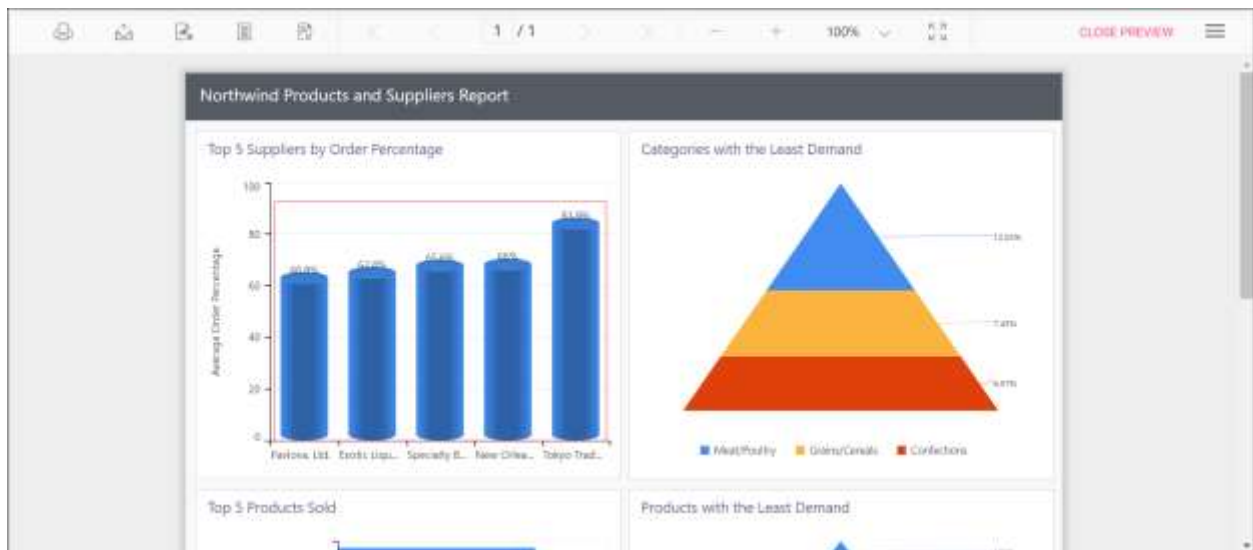
Change the drawing style of a chart column or bar series

The shape of the chart column or bar can be changed using this **Drawingstyle** custom property. By default, the **Drawingstyle** property value is **rectangle**.

You can set the **Drawingstyle** property value, as shown in the below.



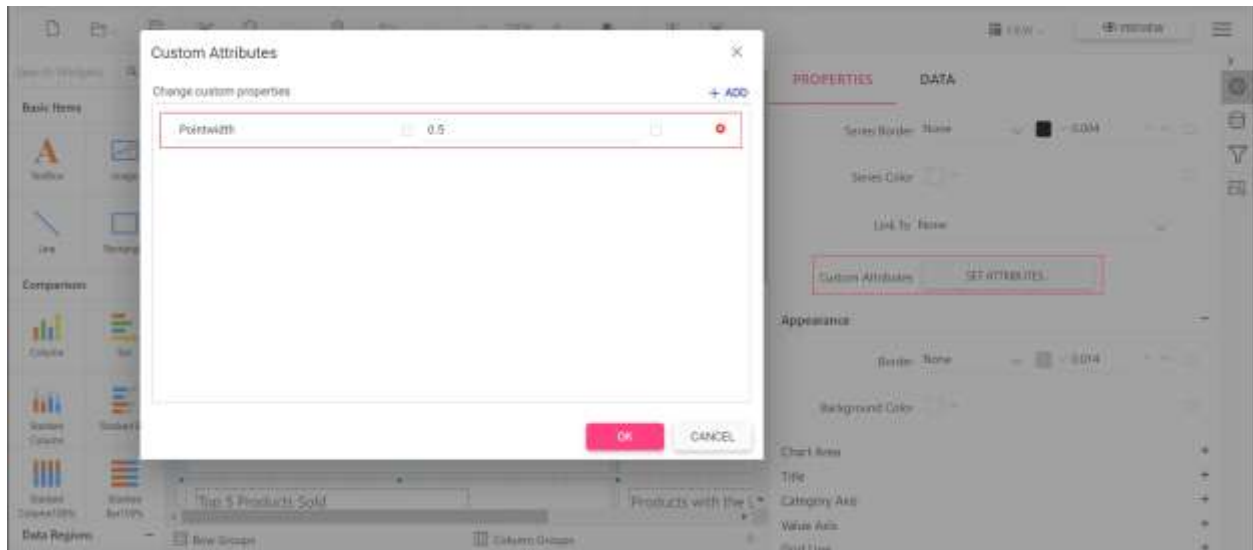
Preview the report and the see the column or bar shape in chart report.



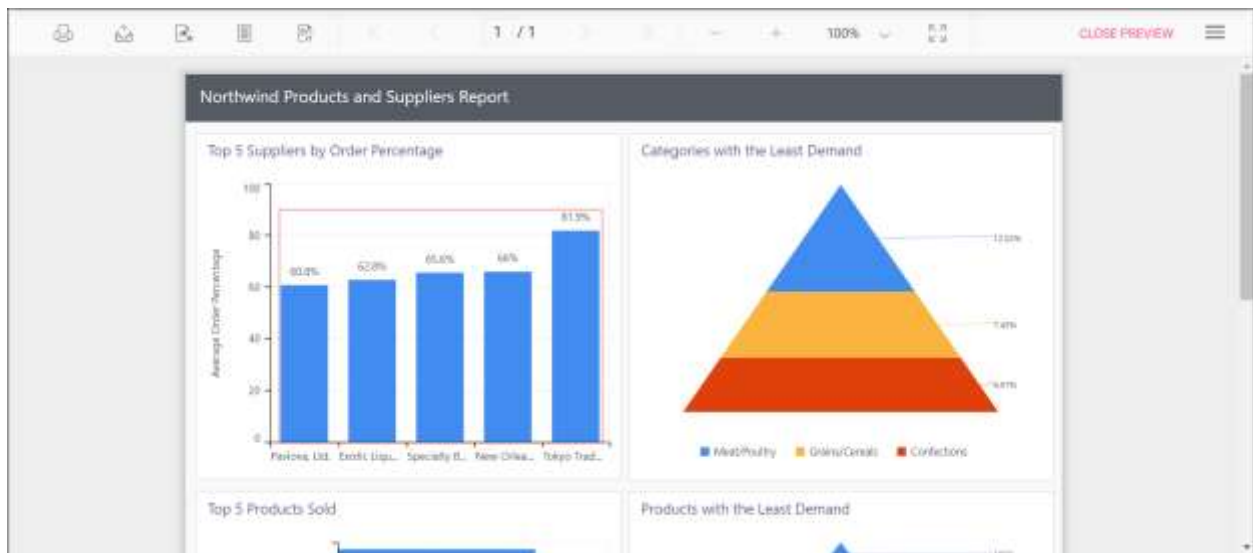
Change width of the column type series in chart

Width of the column type series can be customized by using the **Pointwidth** property. Default value of **Pointwidth** is 0.7. Value ranges from 0 to 1. Here 1 corresponds to 100% of available width and 0 corresponds to 0% of available width.

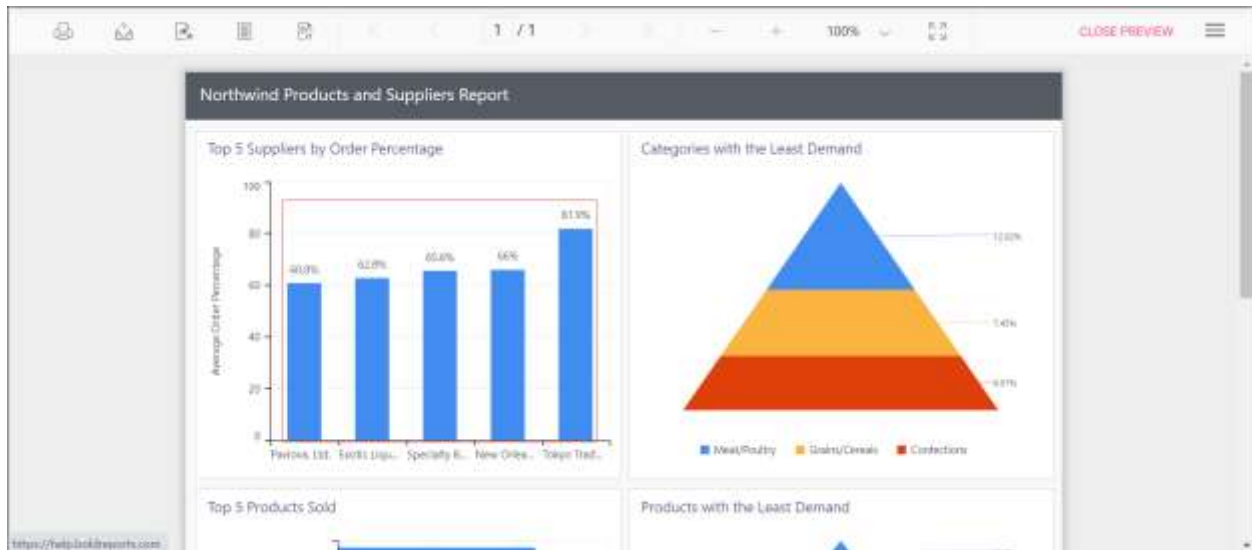
You can set the **Pointwidth** property value, as shown in the below.



Before setting the point width, the default value will be displayed as below.



Preview the report and the see the column width in chart report.



Report custom properties

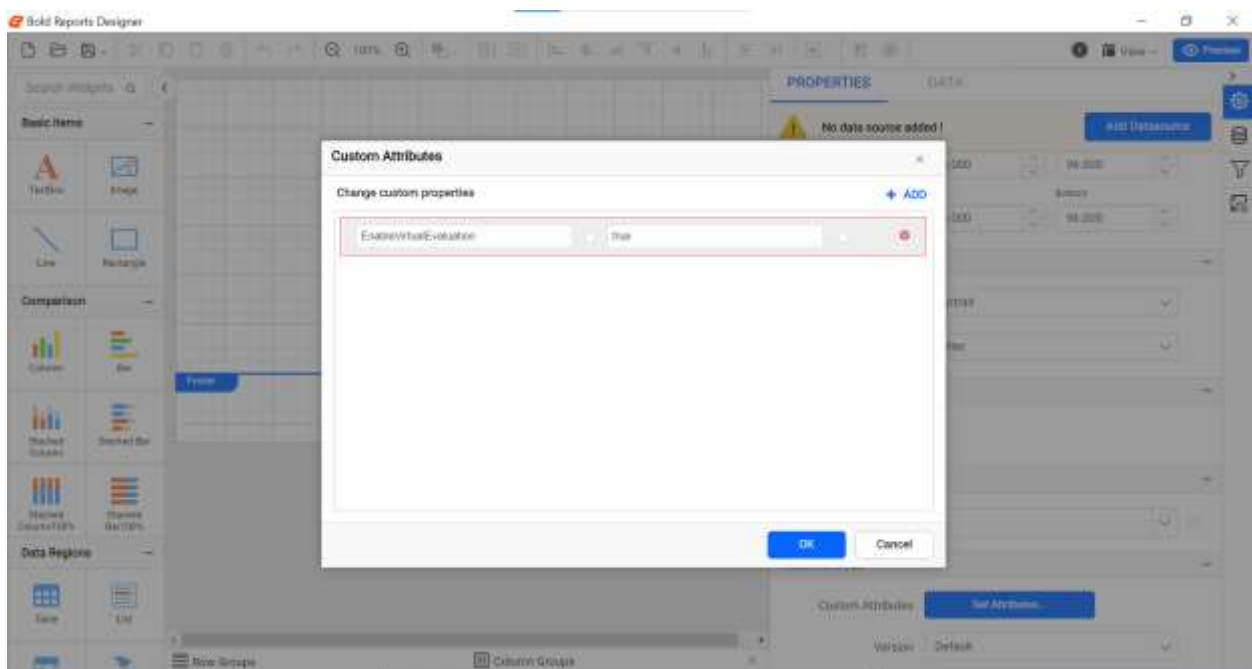
This topic explains about the list of report custom properties that are supported to render in ASP.NET MVC Report Viewer.

Improve performance and handle large amount of data

Set the `EnableVirtualEvaluation` and `DisablePageSplitting` custom properties in a report to improve performance and handle the larger amount of data with less memory footprint.

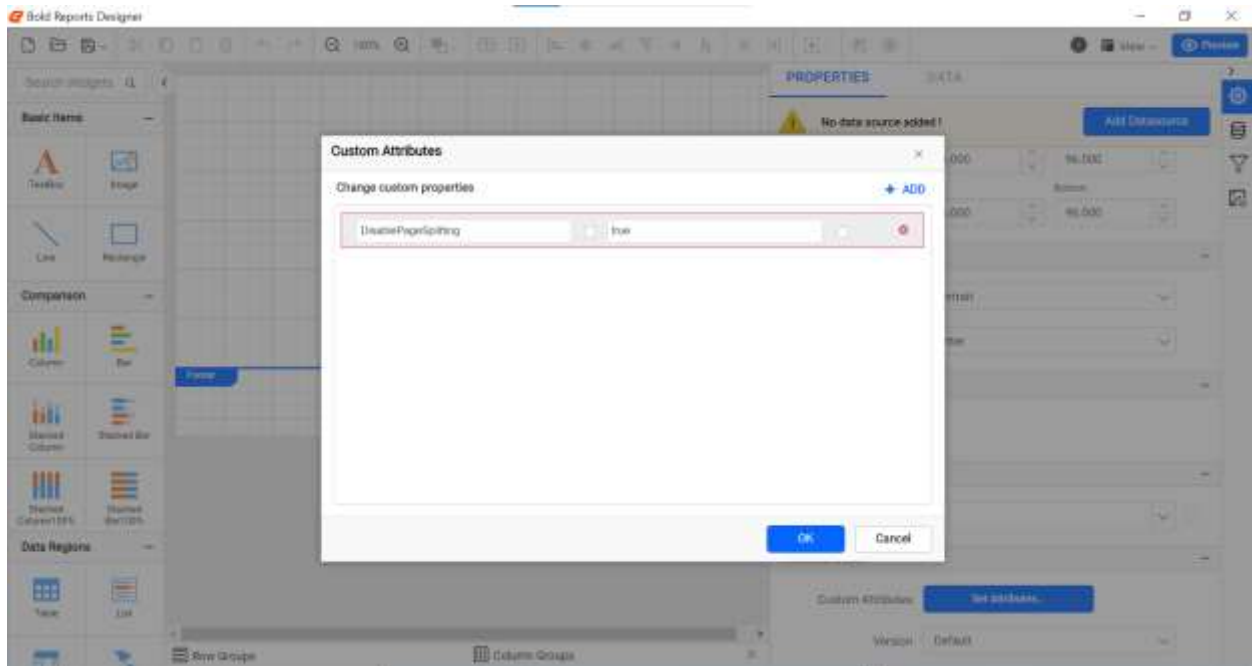
Render large data report faster

The `EnableVirtualEvaluation` custom property is used to render the large data report faster. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Reduce memory footprint for large data report

The `DisablePageSplitting` custom property is used to reduce the memory footprint for large data report. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.

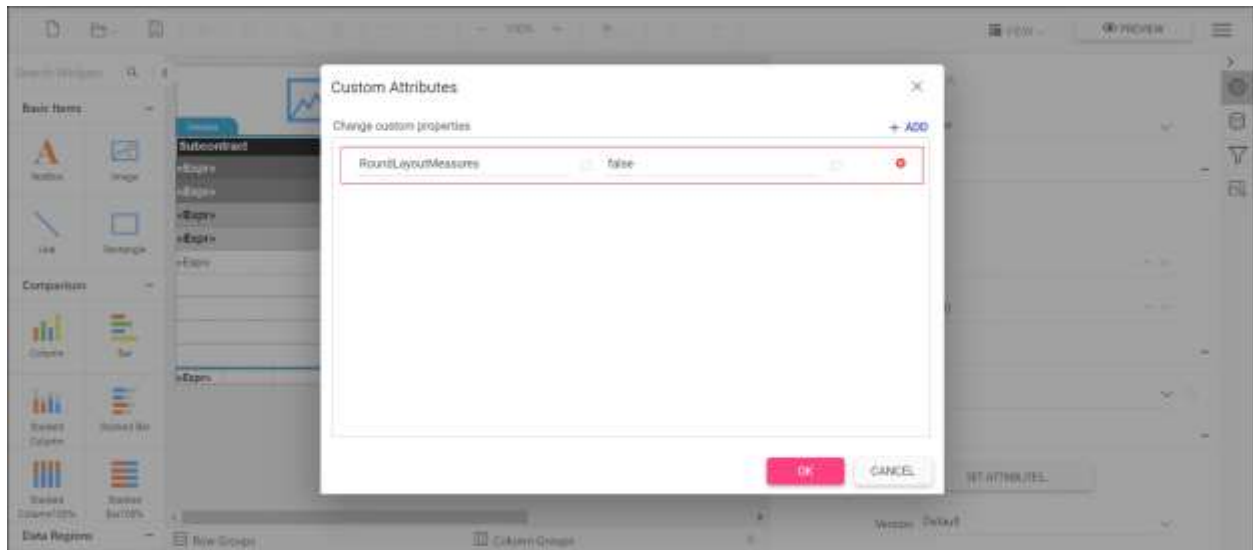


Improve report items layout and avoid extra blank pages

When the `RoundLayoutMeasures` property is false, all non-integral values that are calculated during the report processing are rounded to whole pixel values. It provides following improvements,

- Report text rendered without cuts.
- Eliminates extra blank pages.
- Removes item and text overlaps.
- Eliminates the blur semi-transparent edges that are produced by anti-aliasing.
- Produces identical look in report view and export output.

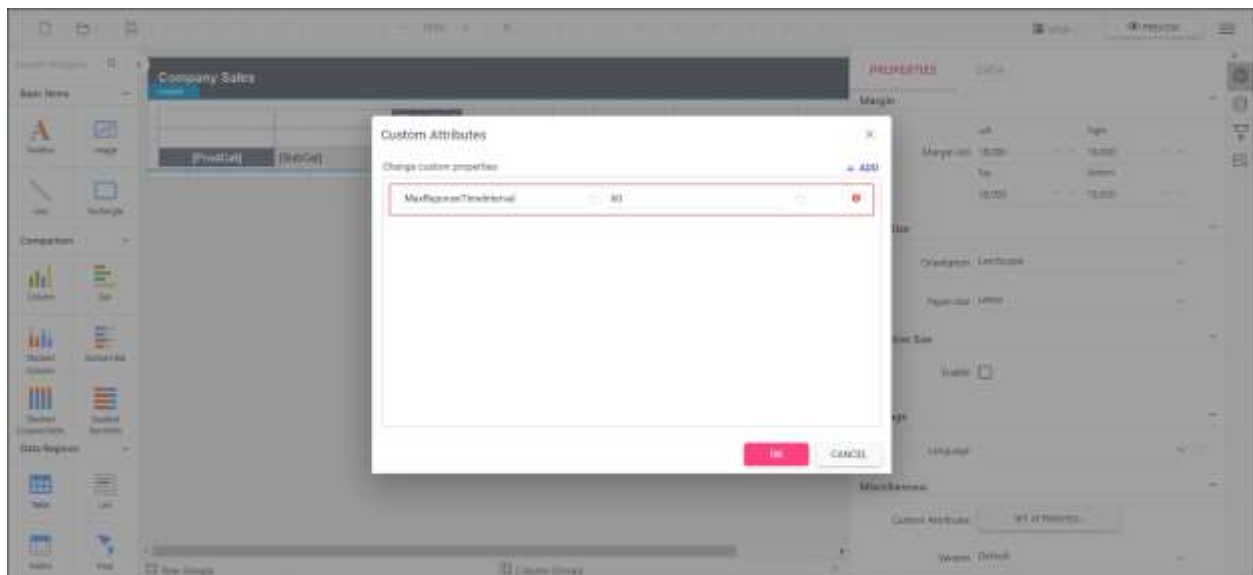
You can set the property value, as shown in the below.



Handling failure in long-running HTTP requests, report processes, and timeouts

When a report process for a long time due to huge records or a HTTP request takes too long to respond then it results in Gateway Timeout or report rendering errors. The `MaxResponseTimeInterval` allows to specify the seconds to process an HTTP request and respond back. It helps to keep the client and server connection live by avoiding the timeout.

You can set the property value as shown in the below.



Parameter custom properties

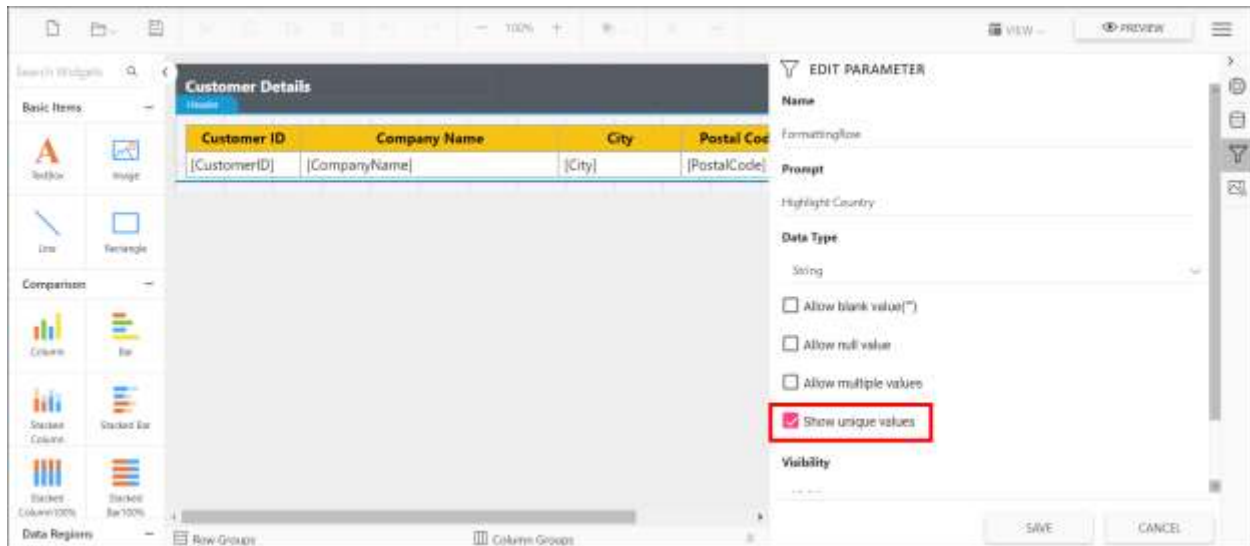
This topic explains about the list of parameter custom properties that are supported to customize the reports preview in Report Viewer.

Show only distinct values in parameter

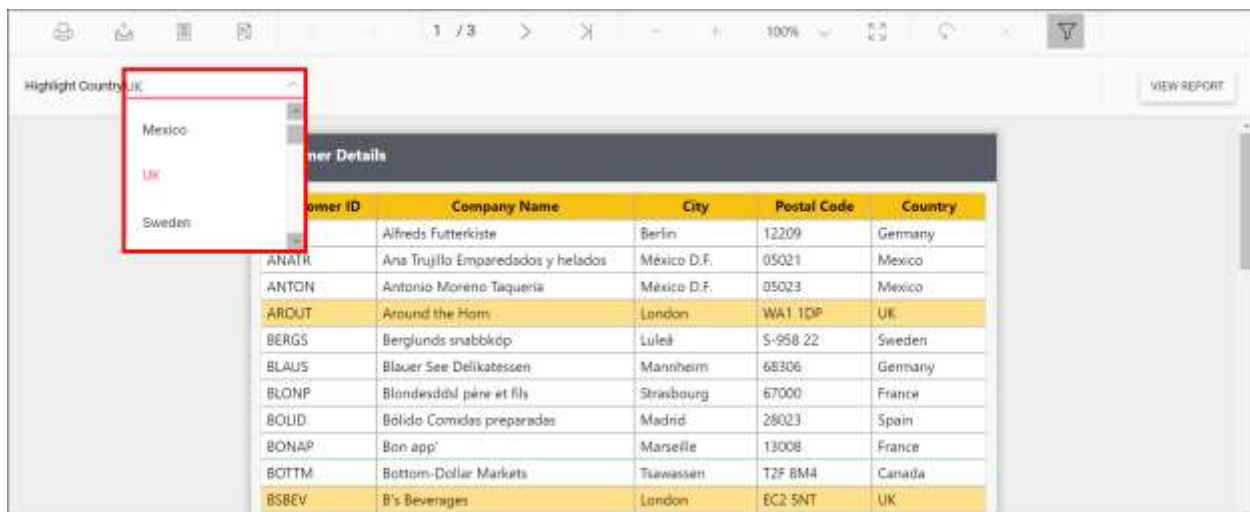
A parameter created with data set query values may contain duplicate values. The Report Viewer supports `UniqueValueParameters` custom property that helps show only unique values in the

parameter drop-down while viewing the report, without creating new a data set. Refer to the below image for property setting option.

You can also provide the parameter names with comma (,) separator to set for multiple parameters.



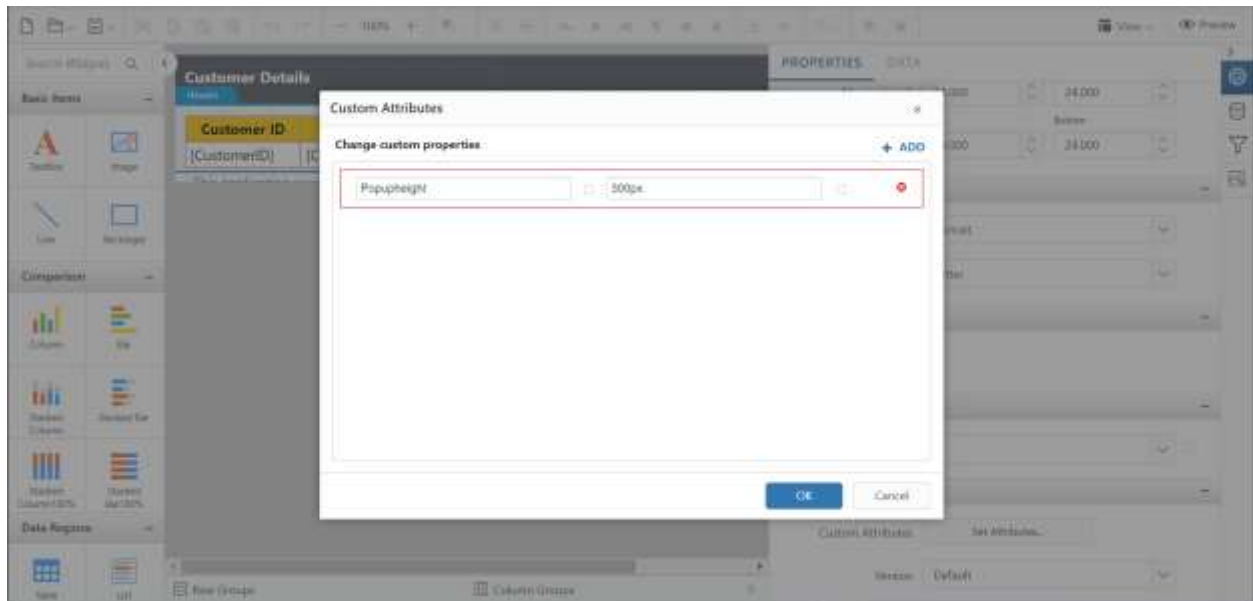
Preview the report and the unique values showed in the parameter drop down.



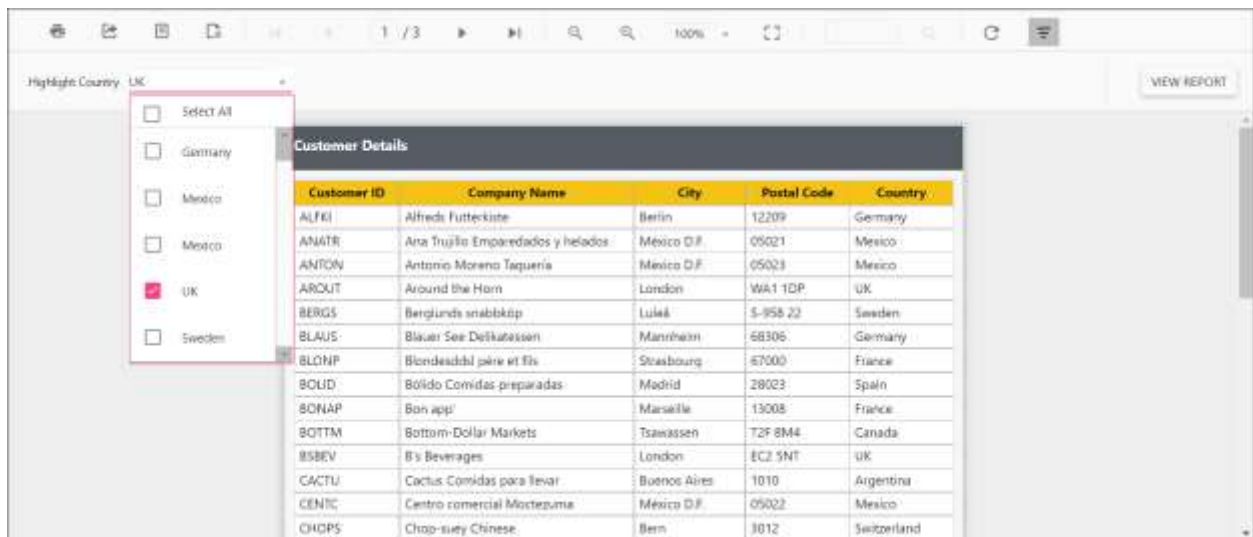
Change the dropdown parameter pop up height value

The **PopupHeight** custom property specifies the height of the parameter combobox popup list in the report. By default, the **PopupHeight** value is **152px**.

You can set the **PopupHeight** property value, as shown in the below.



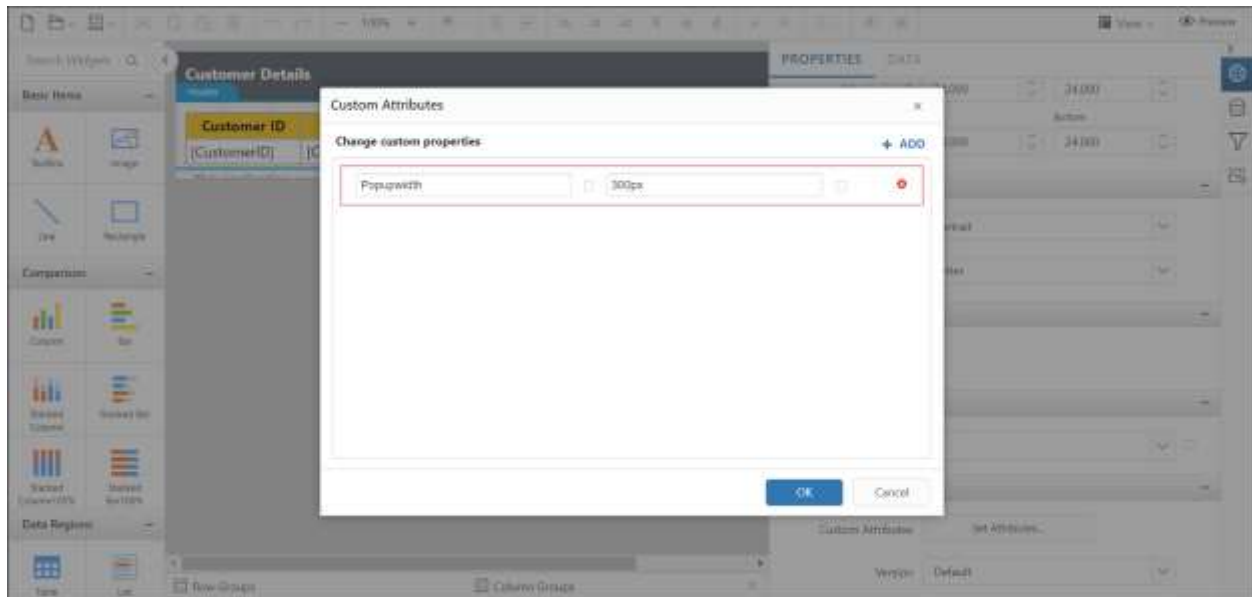
Preview the report and the pop up height showed in the parameter drop down.



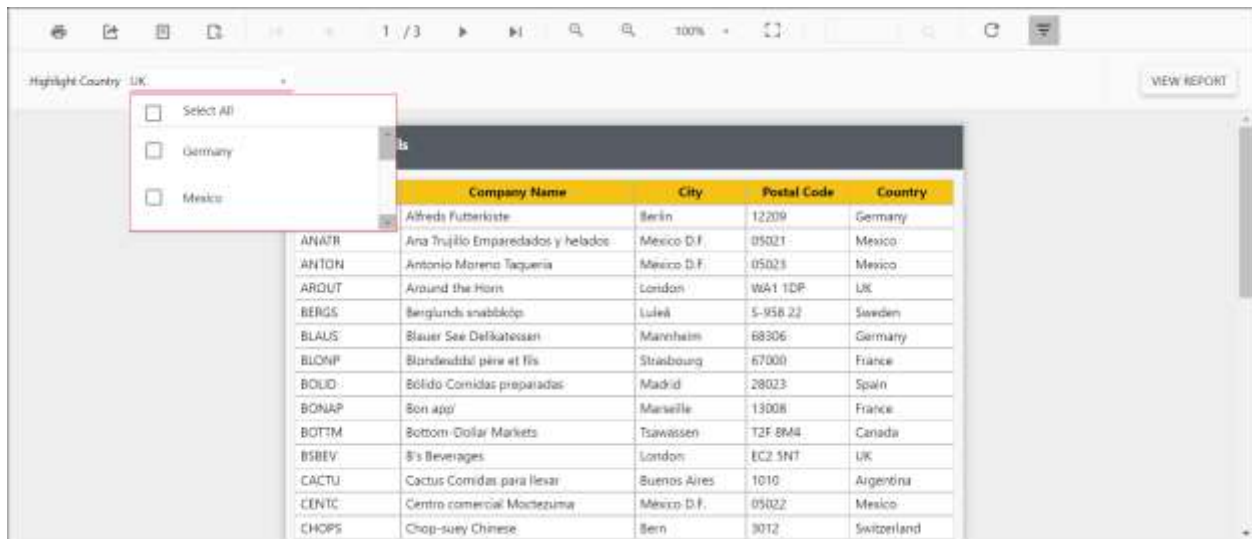
Change the dropdown parameter pop up width value

The **PopupWidth** custom property specifies the width of the parameter combobox popup list in the report. By default, the popup width sets based on the width of the component.

You can set the **PopupWidth** property value, as shown in the below.



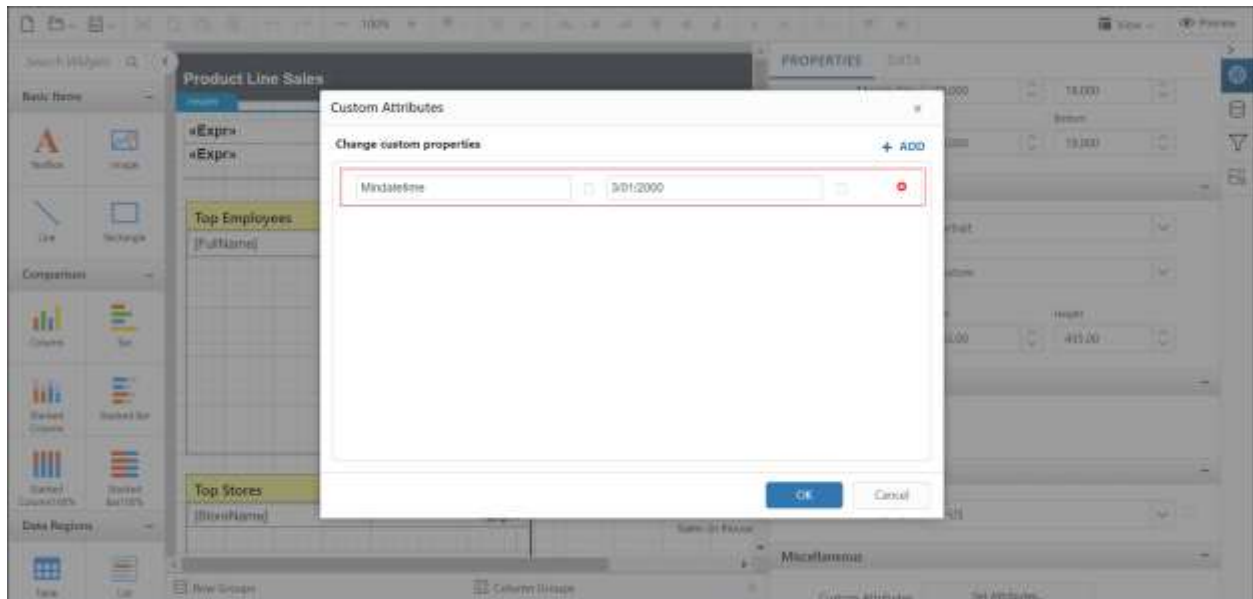
Preview the report and the pop up width showed in the parameter drop down.



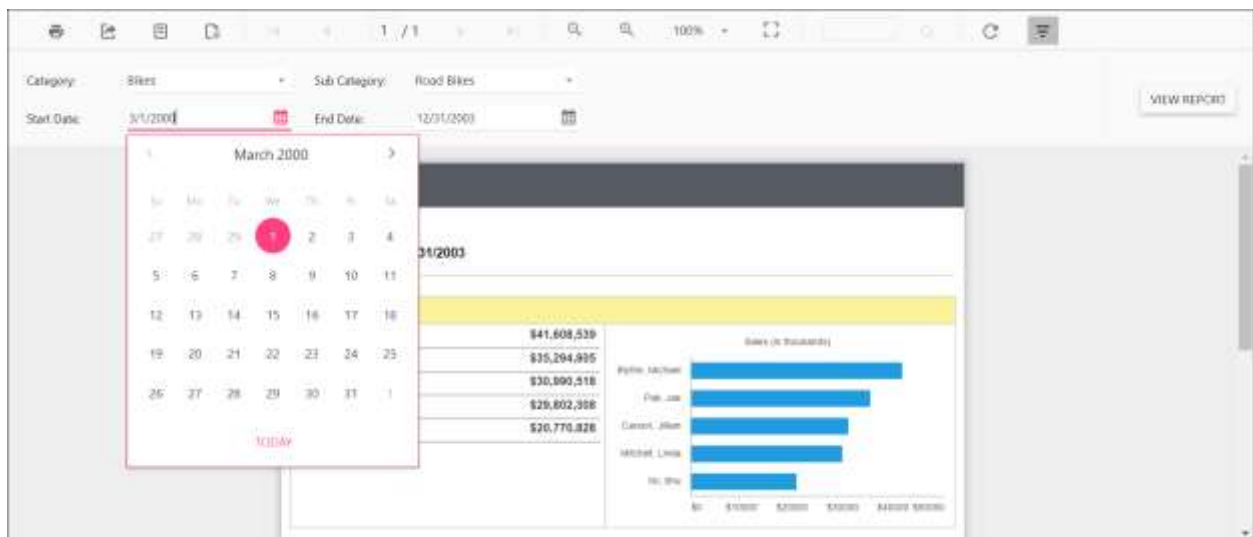
Set the minimum date range for the date report parameter

The **MinDateTime** custom property specifies the minimum date in the datetime parameter item. By default, the **MinDateTime** value is set as **null**.

You can set the **MinDateTime** property value, as shown in the below.



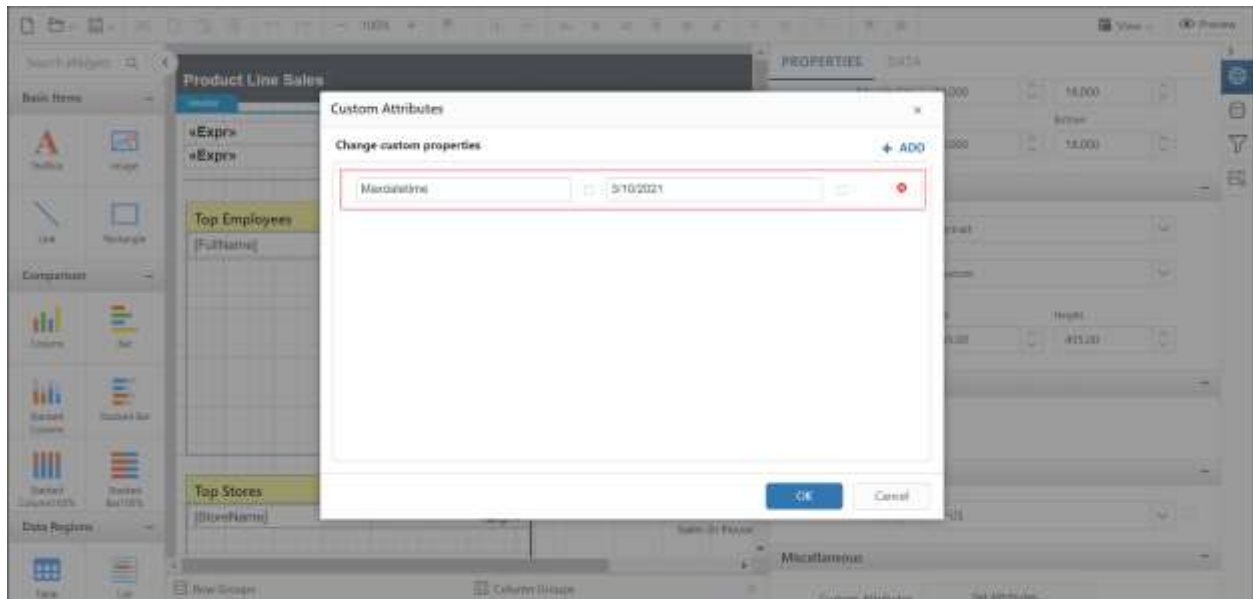
Preview the report and the minimum date showed in the datetime parameter drop down.



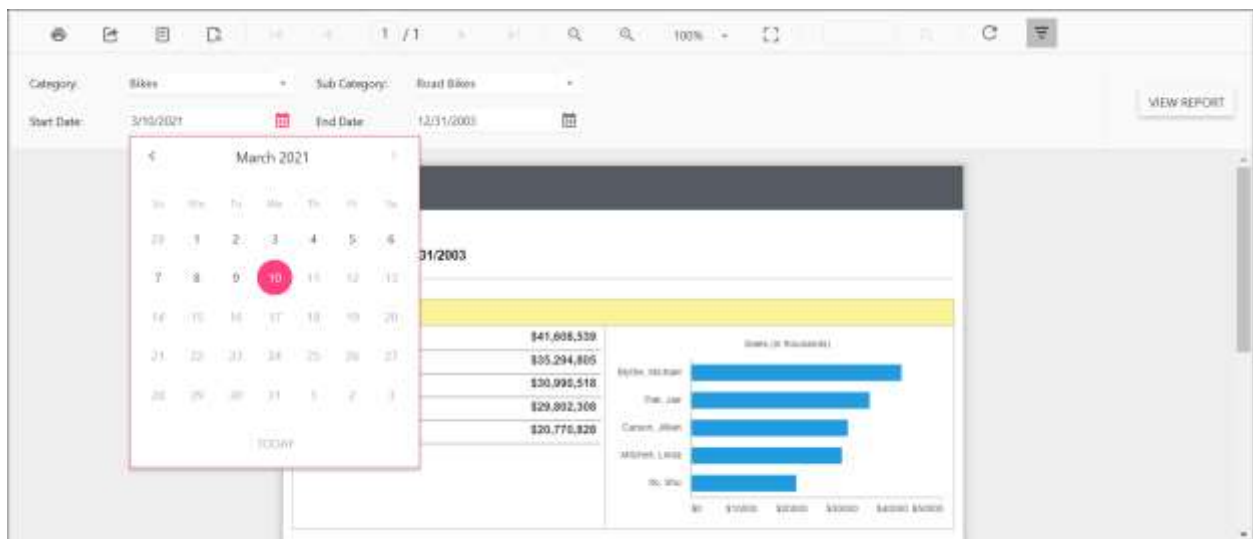
Set the maximum date range for date report parameter

The `MaxDateTime` custom property specifies the maximum date in the datetime parameter item. By default, the `MaxDateTime` value is set as `null`.

You can set the `MaxDateTime` property value, as shown in the below.



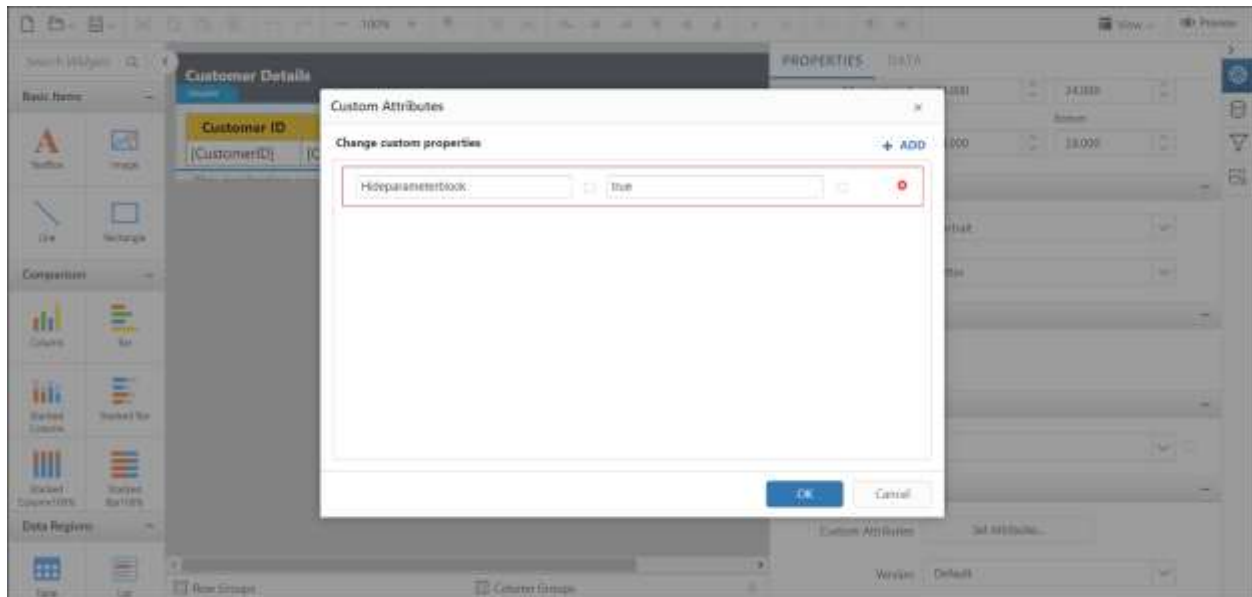
Preview the report and the maximum date showed in the datetime parameter drop down.



Hide the report parameter block

The `HideParameterBlock` custom property is used to hide the parameter block on report initial rendering. The property value should be boolean. By default, the `HideParameterBlock` value is `false`.

You can set the `HideParameterBlock` property value, as shown in the below.



Preview the report and see the parameter block is hidden.

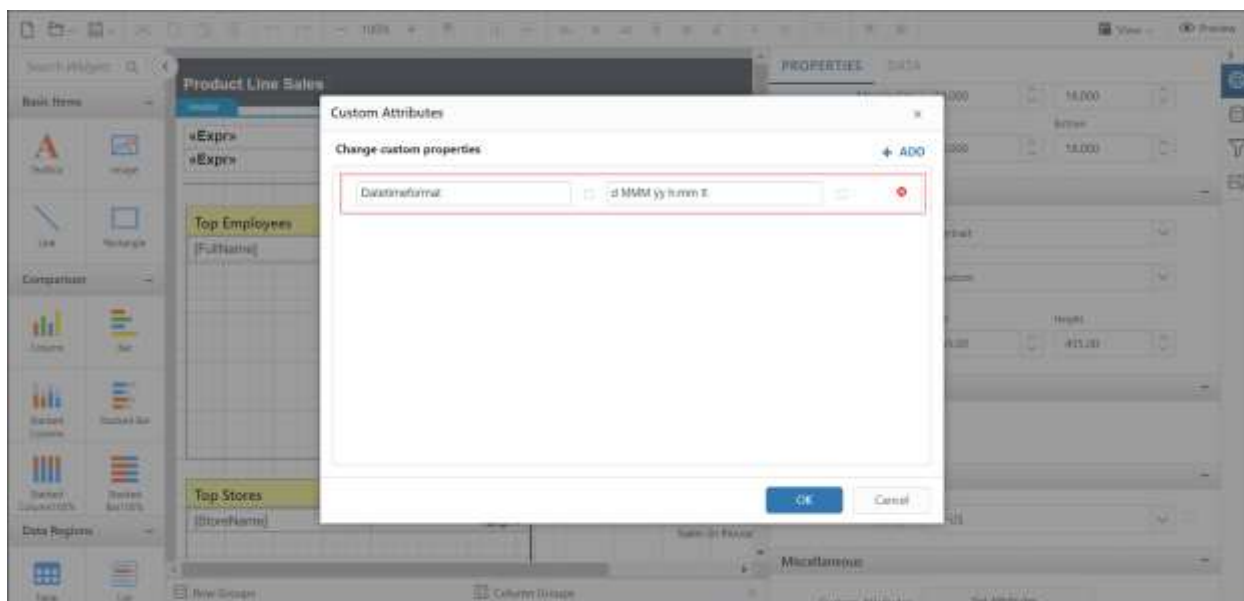
The screenshot shows a report preview titled 'Customer Details'. It displays a table with the following data:

Customer ID	Company Name	City	Postal Code	Country
ALFKI	Alfreds Futterkiste	Berlin	12209	Germany
ANATR	Ana Trujillo Emparedados y helados	México D.F.	05021	Mexico
ANTON	Antonio Moreno Taquería	México D.F.	05023	Mexico
AROUT	Around the Horn	London	WA1 1DP	UK
BERGS	Berglunds snabbköp	Luleå	S-958 22	Sweden
BLAUS	Blaauw Sea Delicatessen	Mannheim	68306	Germany
BONAP	Bonaparte's pizzeria	Strasbourg	67000	France
BOLID	Bolido Comidas preparadas	Madrid	28023	Spain
BONAP	Bonaparte's pizzeria	Marseille	13008	France
BOITM	Bottom-Dollar Markets	Toronto	T2F 8M4	Canada
BSBEV	B's Beverages	London	EC2 5NT	UK
CACTU	Cactus Comidas para llevar	Buenos Aires	1010	Argentina
CENTC	Centro comercial Mochizuki	México D.F.	05022	Mexico
CHOPS	Chop-uey Chinese	Bern	3012	Switzerland
COMM1	Comércio Mineiro	São Paulo	05432-043	Brazil
CONSH	Consolidated Holdings	London	WX1 6LT	UK
DRACD	Drachendorff & Göttsche	Berlin	10709	Germany

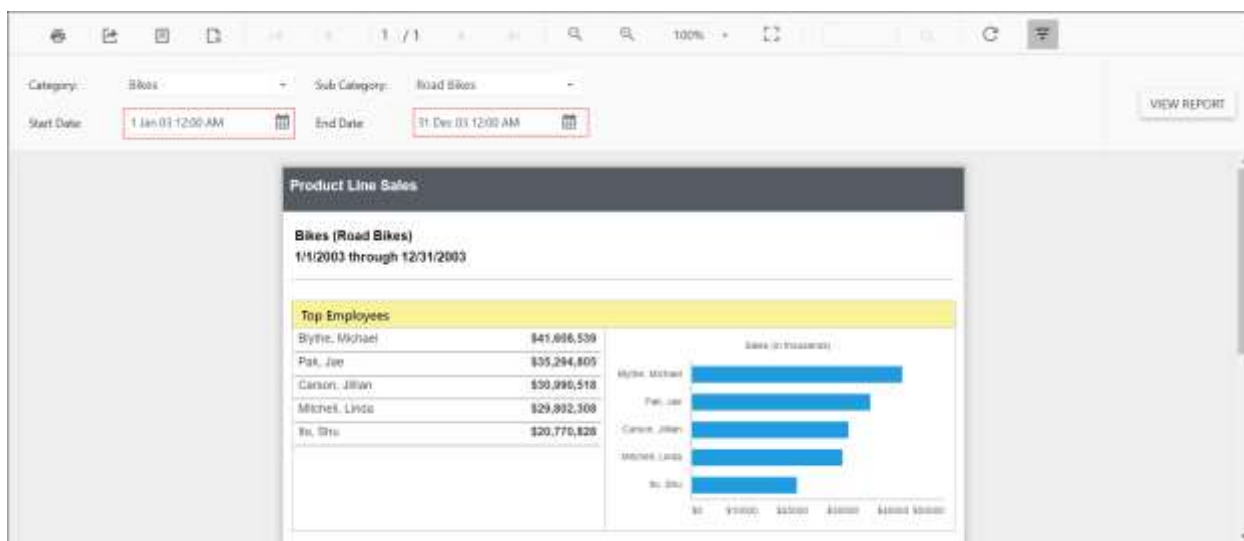
Set date and time format for parameter

The `DateTimeFormat` custom property defines the date time format to be displayed in the `DateTimePicker` popup. By default, the `DateTimeFormat` value is set as `empty`.

You can set the `DateTimeFormat` property value, as shown in the below.



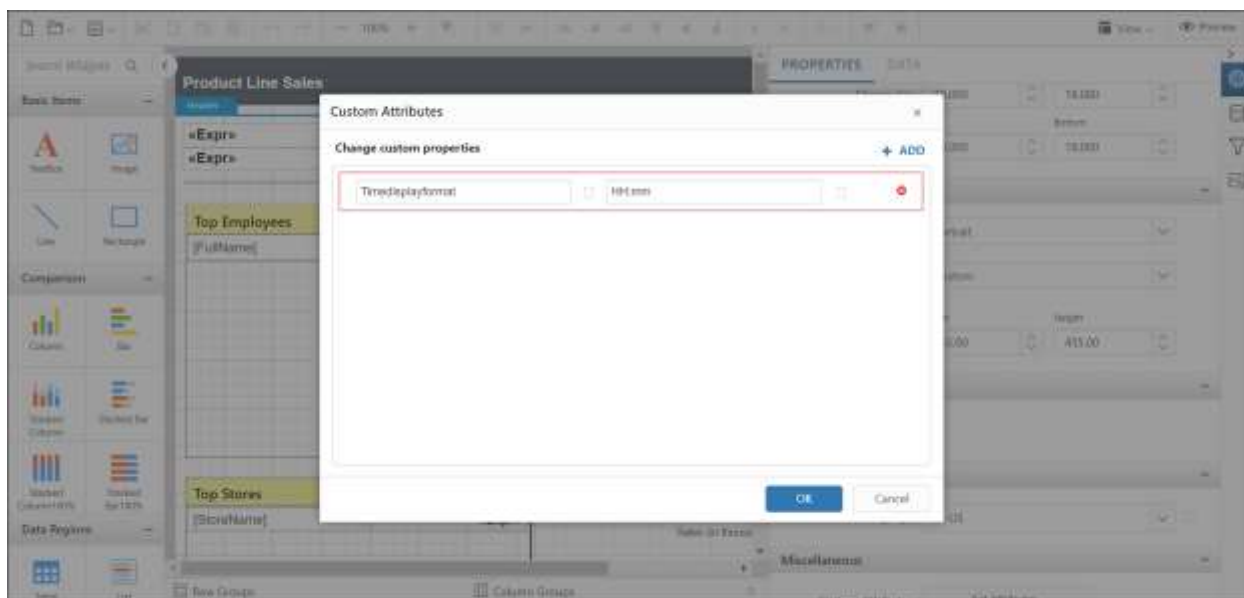
Preview the report and the see date time format in datetime parameter.



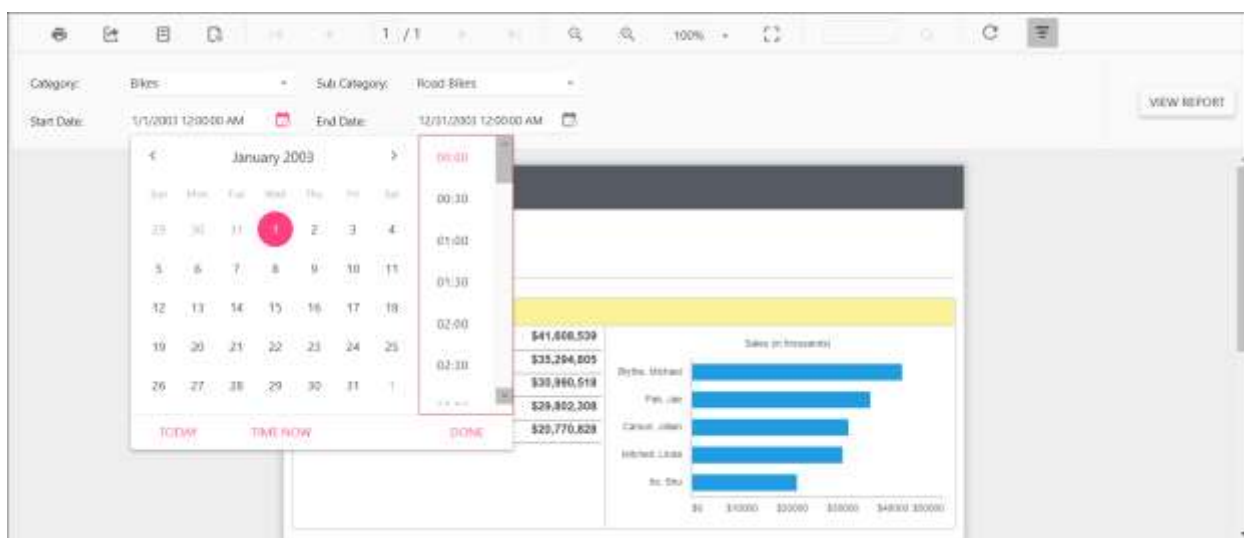
Change time display format for parameter

The `TimeDisplayFormat` custom property defines the time format to be displayed in the time dropdown inside the `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeDisplayFormat`. By default, the `TimeDisplayFormat` value is set as empty.

You can set the `TimeDisplayFormat` property value, as shown in the below.



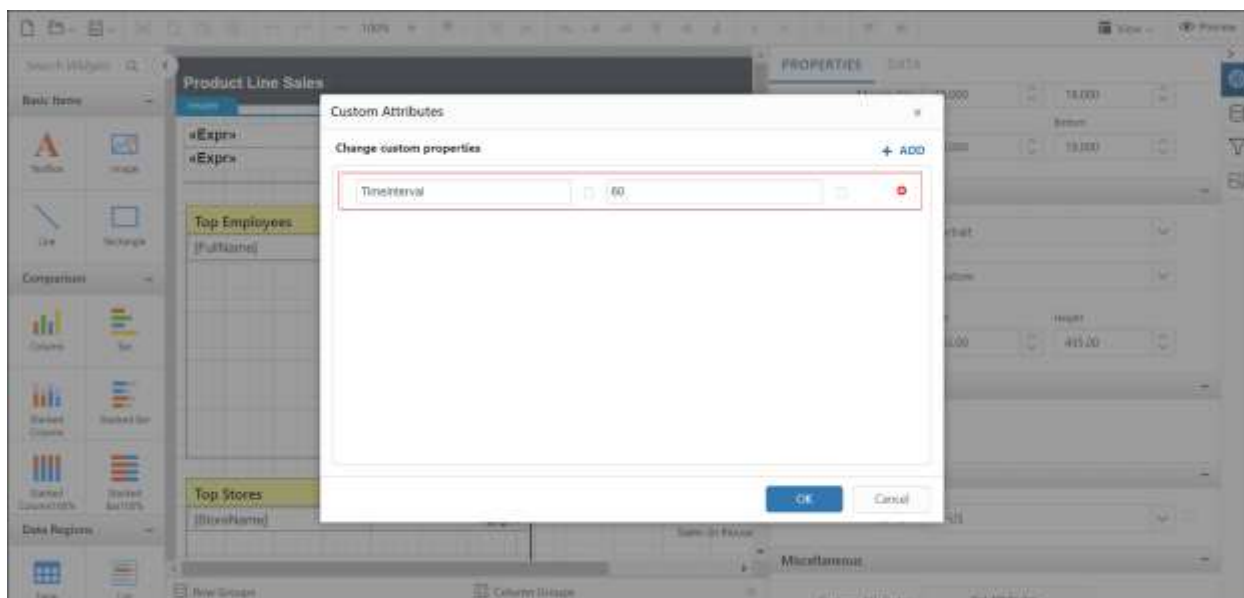
Preview the report and the see time format in datetime parameter.



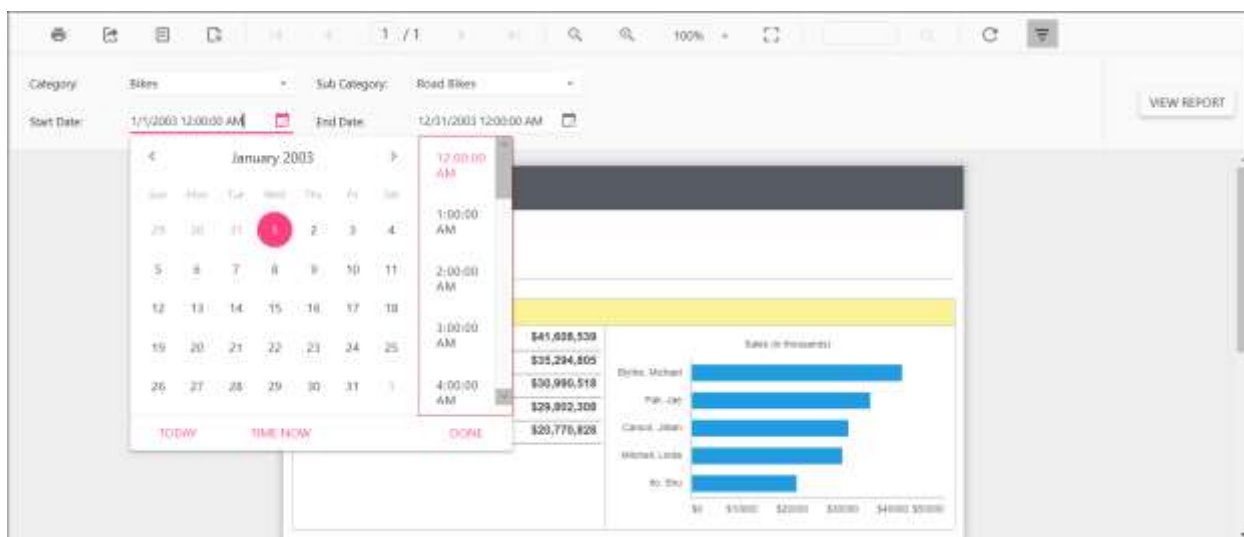
Set time interval in datetime parameter

The `TimeInterval` custom property is used to set the interval between the two adjacent time values in `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeInterval`. By default, the `TimeInterval` value is set as 30.

You can set the `TimeInterval` property value, as shown in the below.



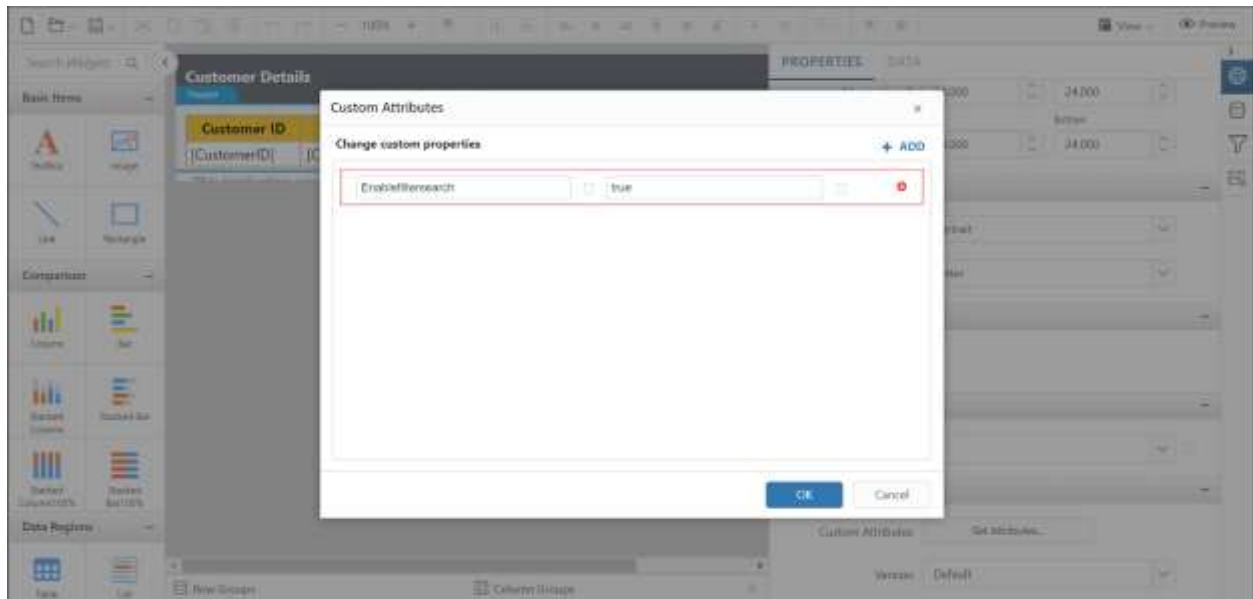
Preview the report and the see time interval in datetime parameter.



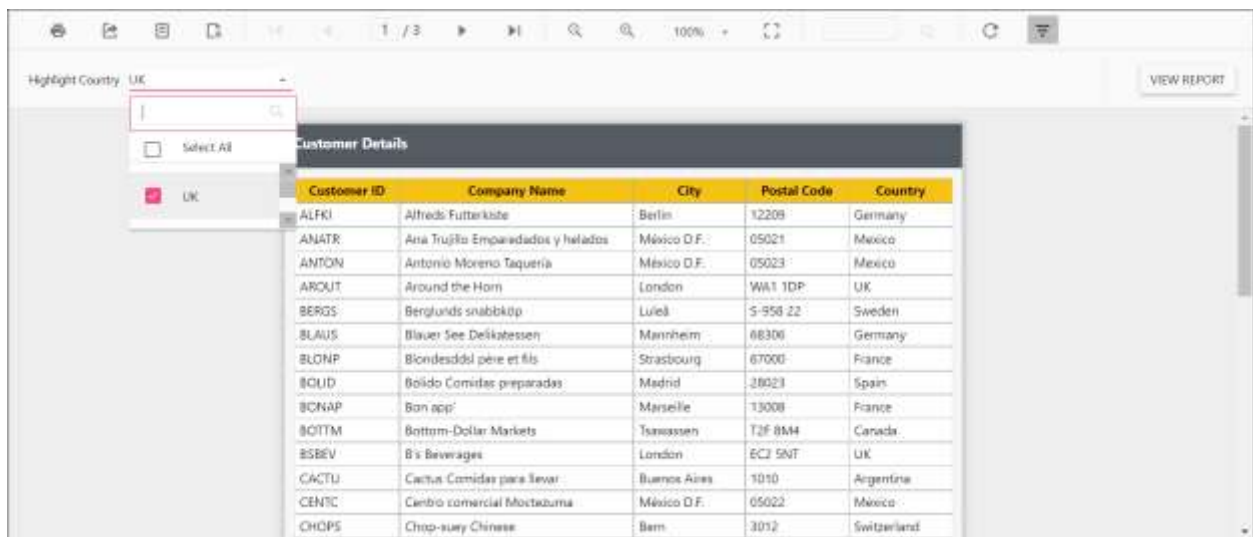
Enable filtering and searching in drop-down report parameter

Setting `EnableFilterSearch` custom property enables search and filtering option in dropdown parameter to easily find the values. The property value should be boolean. By default, the `EnableFilterSearch` value is `false`.

You can set the `EnableFilterSearch` property value, as shown in the below.



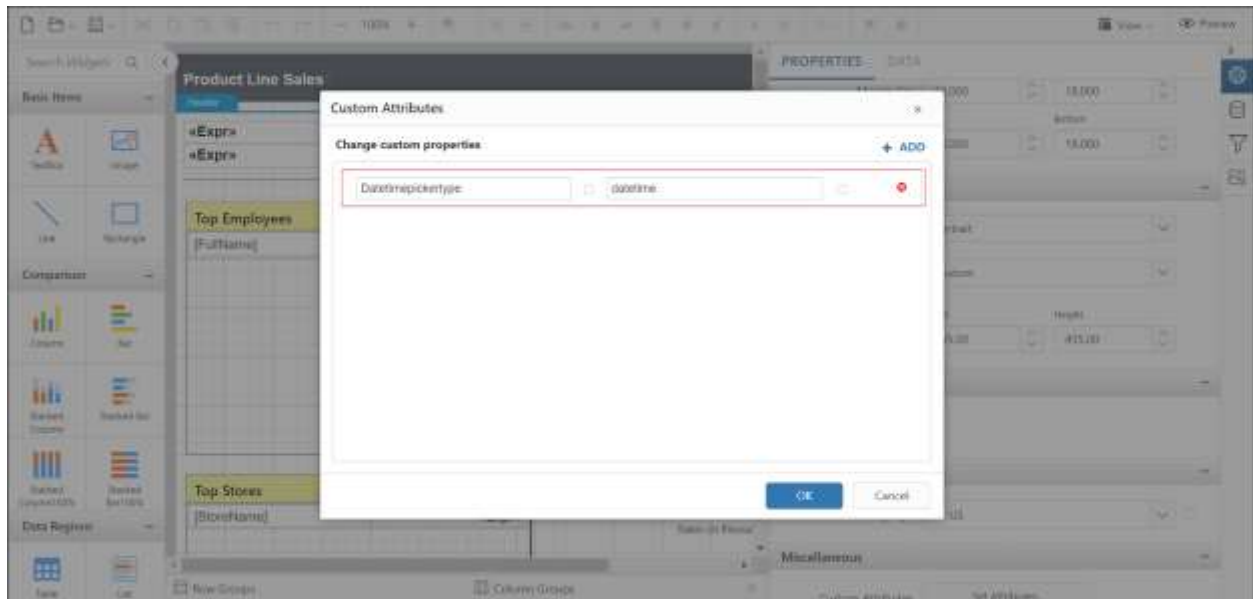
Preview the report and the see search filter in parameter.



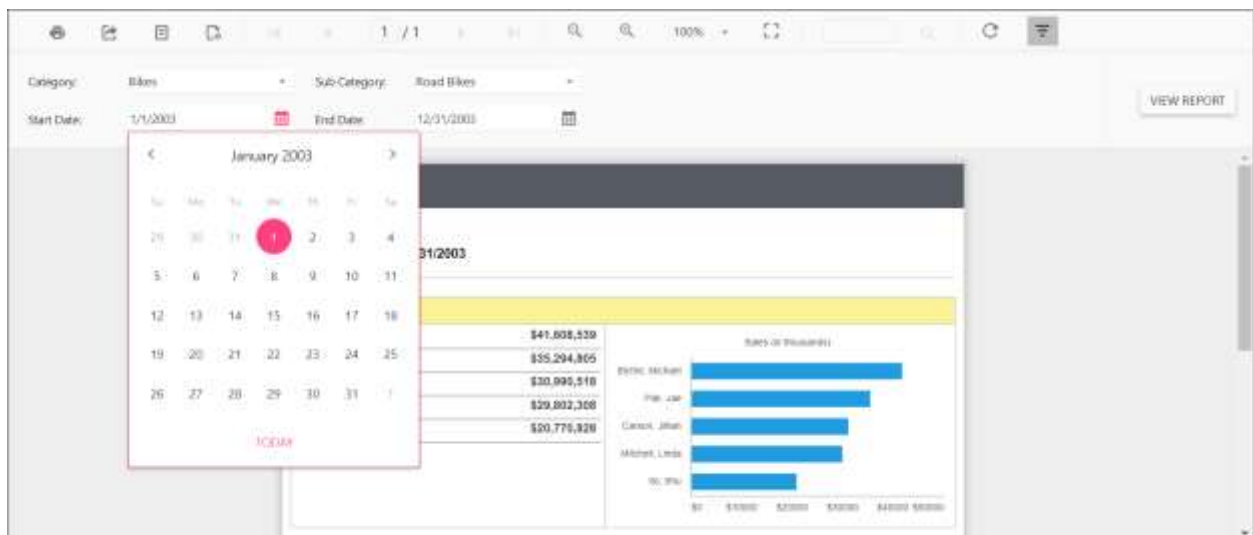
Change date report parameter to display date time picker

You can set `DateTimePickerType` custom property value as `DateTime` to change date parameter item to display `DateTimePicker` for value selection as shown the below.

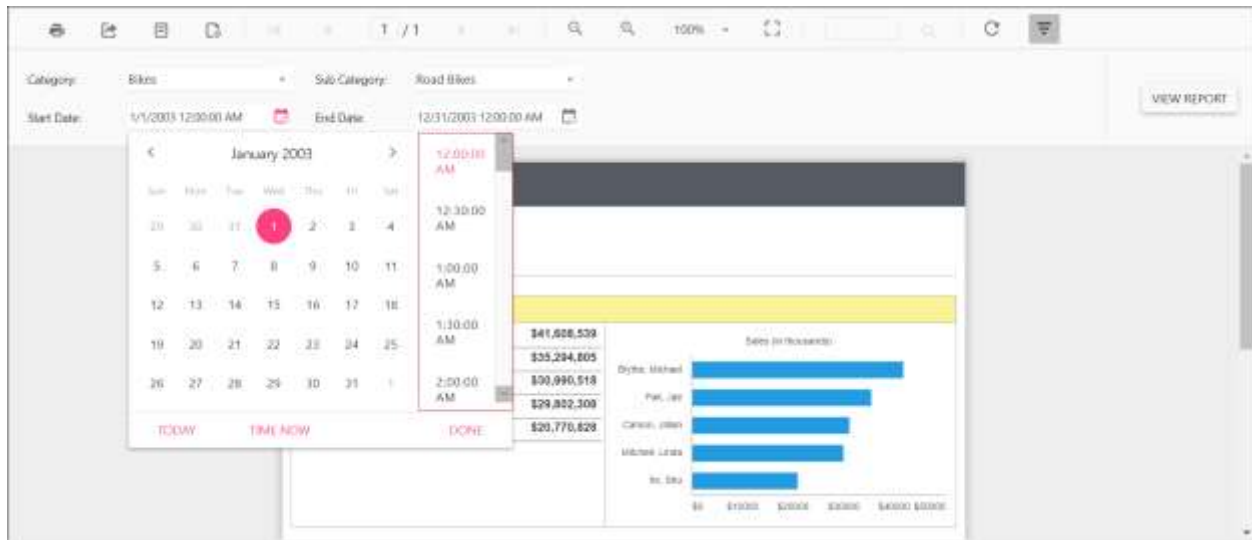
You can set the `DateTimePickerType` property value, as shown in the below.



Before enabling date time picker type, the default value will be displayed as below..



Enable date time picker type and see the time in `DateTimePicker` as in below output.



Export custom properties

This topic explains the list of custom properties that are supported at the report level to control the export behaviour in ASP.NET MVC Report Viewer.

Improve excel export performance

The custom property `DisableExcelCellFormat` helps to improve the excel export performance by ignoring the rendering of report item styles. It allows property value from anyone of the following values.

- **Style** - Disables rendering of the cell styles like padding, background, color, and text style
- **Border** - Disables rendering of the cell border
- **All** - Disables rendering of the cell border, padding, background, color, and text style

Set the property value as **All** to improve maximum excel export performance.

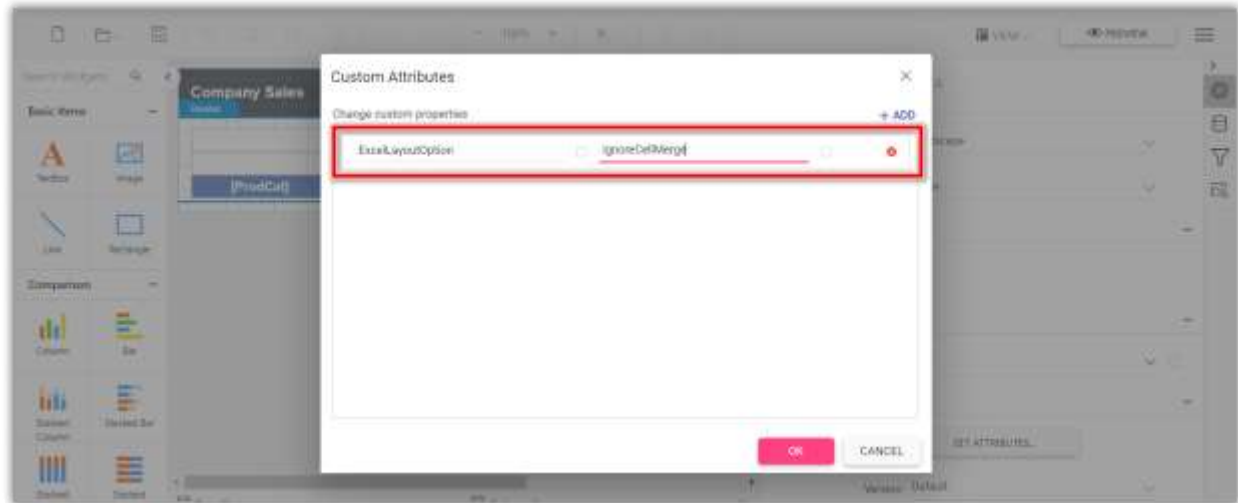
You can set the `DisableExcelCellFormat` custom property as shown below,



The `DisableExcelCellFormat` property must be added to report properties.

Improve excel export readability

Set `ExcelLayoutOption` custom property value as `IgnoreCellMerge` to improve the readability of the excel document by eliminating the tiny columns, rows, and merged cells. You can set the property value, as shown below,



The `ExcelLayoutOption` property must be added to report properties.

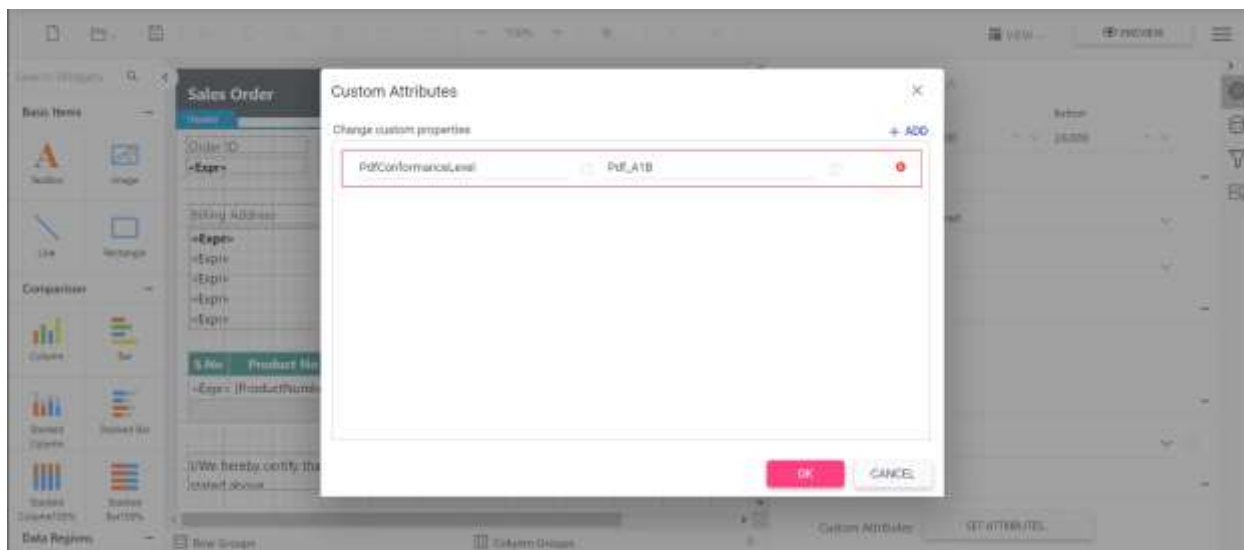
Set pdf conformance level

The `PdfConformanceLevel` allows you set PDF document versions.

You can set anyone of the following PDF conformance option.

- **PdfA1B** - You can create a PdfA1B document by specifying the conformance level as Pdf_A1B through PdfConformanceLevel Enum when creating the new PDF document.
- **PdfA2B** - You can create a PdfA2B document by specifying the conformance level as Pdf_A2B through PdfConformanceLevel Enum when creating the new PDF document
- **PdfA3B** - The PDF/A3B conformance supports the external files as attachment to the PDF document, so you can attach any document format such as Excel, Word, HTML, CAD, or XML files.
- **PdfA1A** - This conformance includes all PdfA1B requirements in addition to the features intended to improve a document's accessibility. PDF/A-1a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2A** - This conformance includes all PDF/A2B requirements in addition to the features intended to improve a document's accessibility. PDF/A-2a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2U** - This conformance includes all PdfA2U requirements, and additionally Unicode mapping for all text in the document.
- **PdfX1A2001** - You can create a PDF/X-1a document by specifying the conformance level as PdfX1A2001 through PdfConformanceLevel Enum when creating the new PDF document.

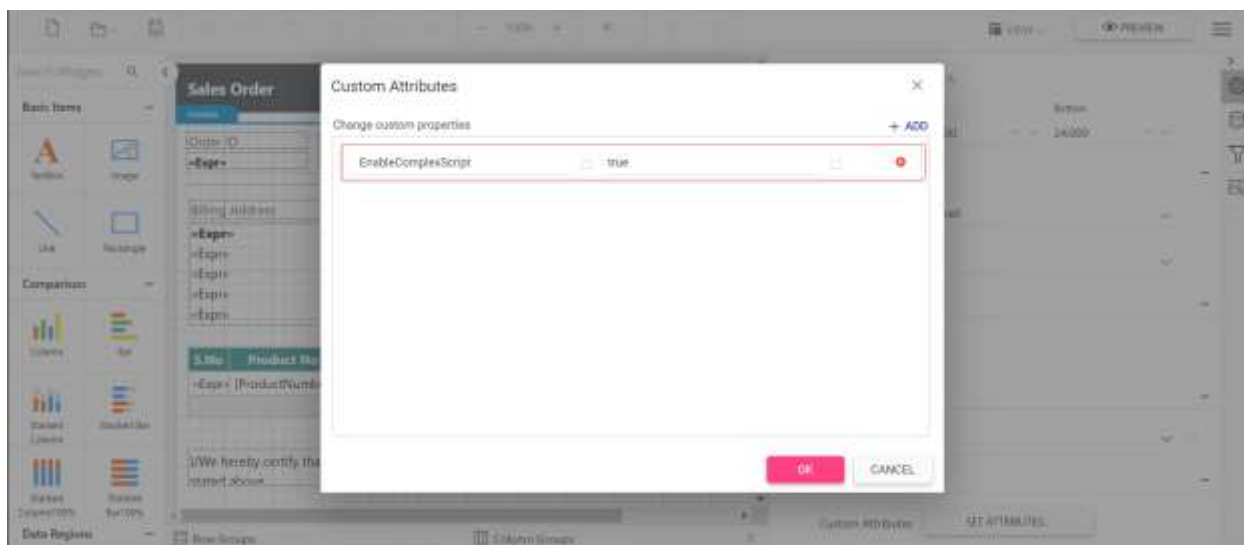
You can set the `PdfConformanceLevel` custom property as shown below,



Export pdf document with complex script

The `EnableComplexScript` custom property allows you to export pdf documents with complex script language texts.

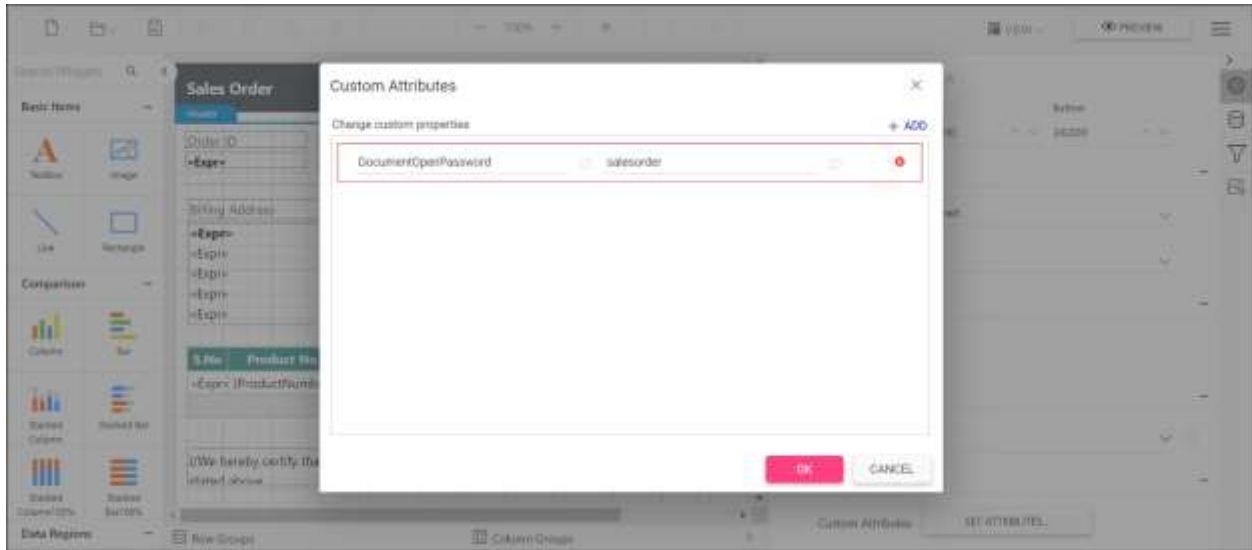
You can set the property value, as shown below,



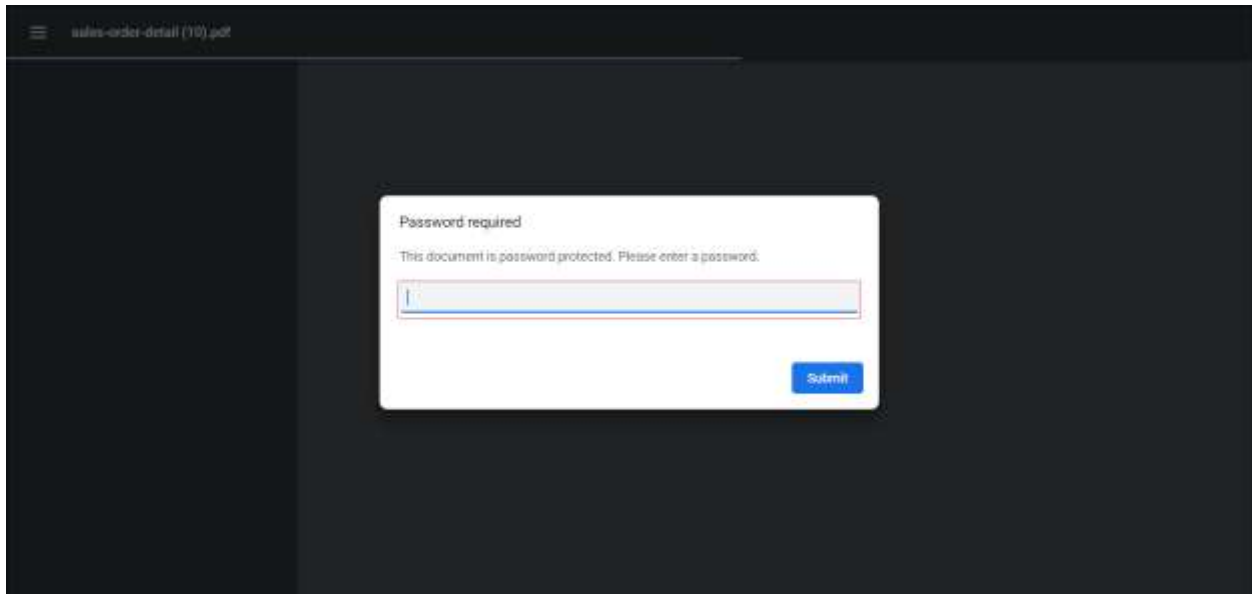
Encrypt and secure export documents

The custom property `DocumentOpenPassword` allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password.

You can set the property value, as shown below,



open the pdf document and see the below output.

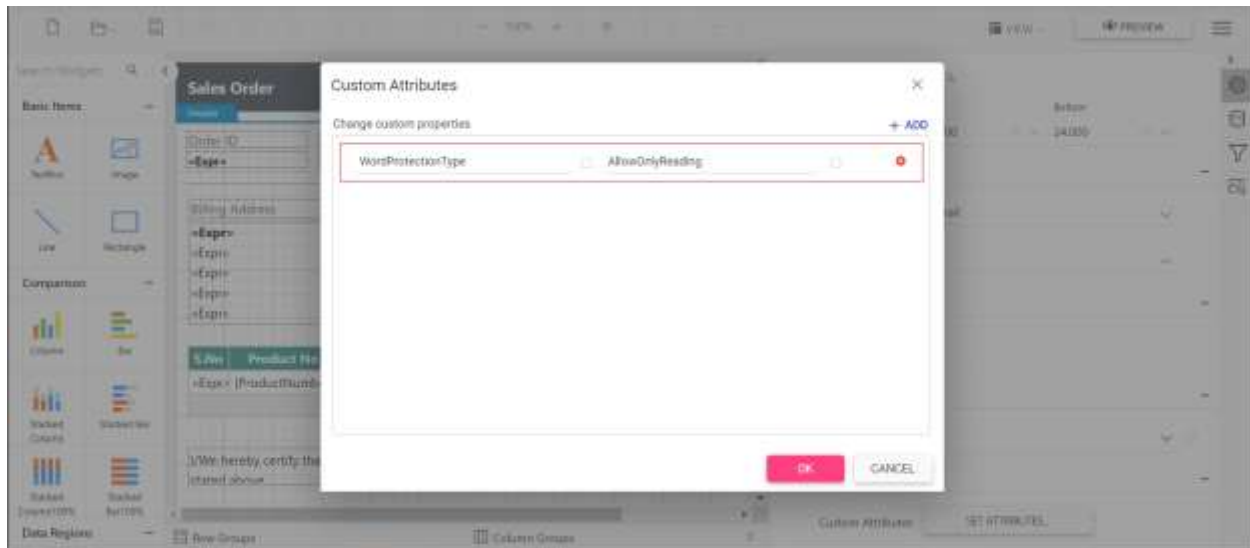


Protecting Word document from editing

The custom property `WordProtectionType` allows you to restrict a Word document from editing either by providing a password or without a password by using following types of protection.

- `AllowOnlyComments`- You can add or modify only the comments in the Word document.
- `AllowOnlyFormFields`- You can modify the form field values in the Word document.
- `AllowOnlyRevisions`- You can accept or reject the revisions in the Word document.
- `AllowOnlyReading`- You can only view the content in the Word document.
- `NoProtection`- You can access or edit the Word document contents as normally.

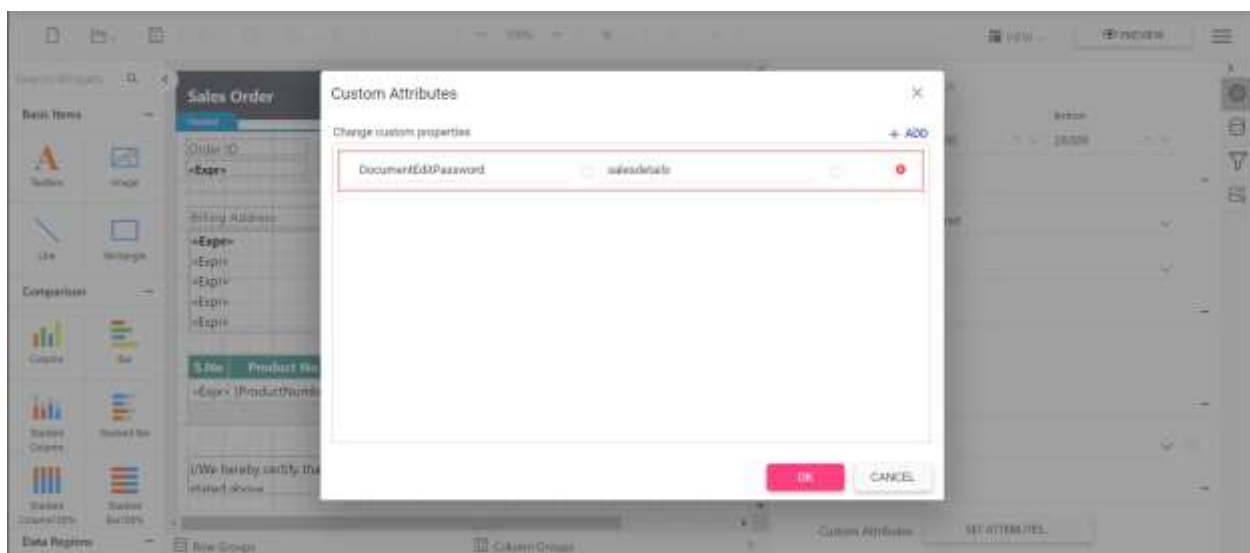
You can set the `WordProtectionType` custom property as shown below,



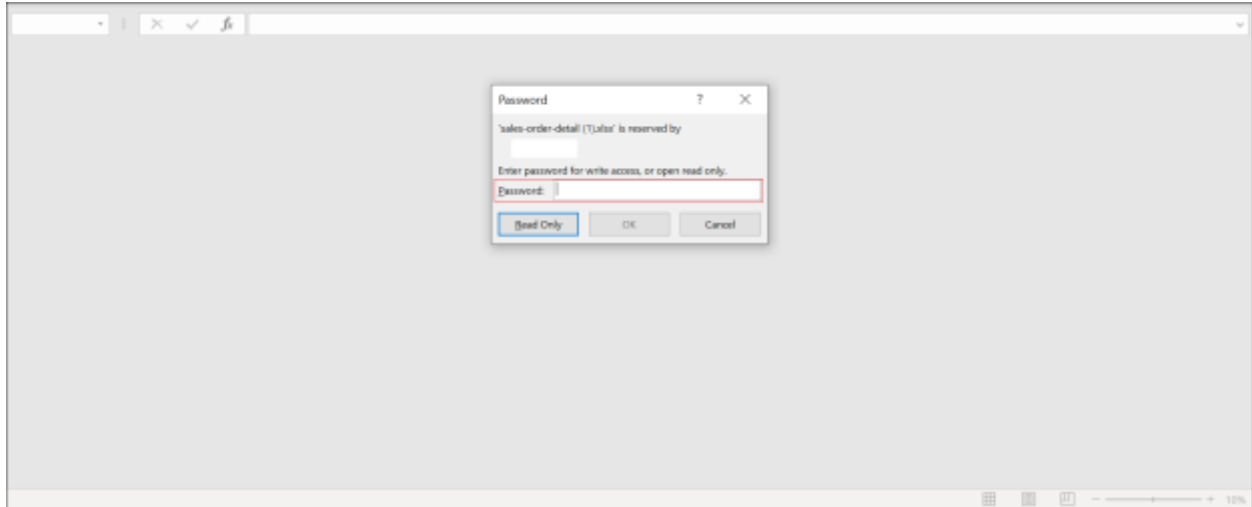
Set excel document edit password

The custom property `DocumentEditPassword` helps to allow specific users permission to modify the workbook data and save changes to the file in the excel document.

You can set the property value, as shown below,



open the excel document and see the below output.

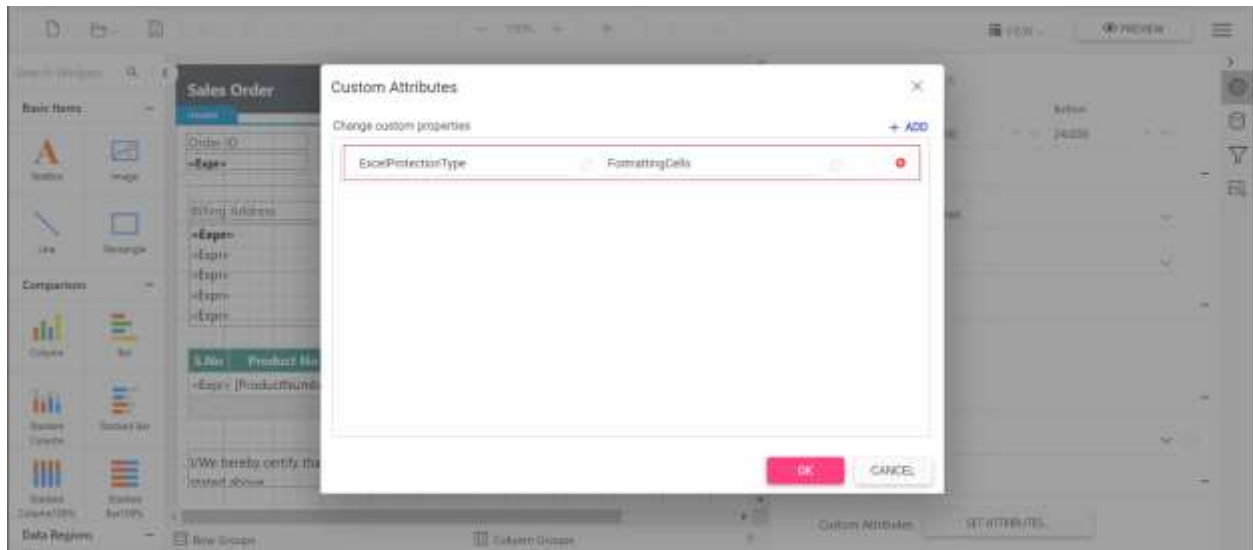


Set excel document protection

The custom property `ExcelProtectionType` allows you to protect the worksheet of excel document either by providing a password or without a password by using following types of protection.

- `None` - Represents no protection in excel sheet
- `Objects` - allows to protect shapes in excel sheet
- `Scenarios` - allows to protect scenarios.
- `FormattingCells` - allows the user to format any cell on a protected worksheet.
- `FormattingColumns` - allows the user to format any column on a protected worksheet.
- `FormattingRows` - allows the user to format any rows on a protected worksheet.
- `InsertingColumns` - allows the user to insert columns on the protected worksheet.
- `InsertingRows` - allows the user to insert rows on the protected worksheet.
- `InsertingHyperlinks` - allows the user to insert hyperlinks on the worksheet.
- `DeletingColumns` - allows the user to delete columns on the protected worksheet, where every cell in the column to be deleted is unlocked.
- `DeletingRows` - allows the user to delete rows on the protected worksheet, where every cell in the row to be deleted is unlocked.
- `LockedCells` - allows to protect locked cells.
- `Sorting` - allows the user to sort on the protected worksheet
- `Filtering` - allows the user to set filters on the protected worksheet. Users can change filter criteria but can not enable or disable an auto filter.
- `UsingPivotTables` - allows the user to use pivot table reports on the protected worksheet.
- `UnLockedCells` - allows to protect the user interface, but not macros.
- `Content` - allows to protect the contents in the excel sheet.
- `All` - allows to protect all type of protection.

You can set the `ExcelProtectionType` custom property as shown below,



Toolbar customization

You can hide the component toolbar to show customized user interface or to customize the toolbar icons and element's appearances using the templates and Report Viewer toolbar customization properties.

In this tutorial, the `sales-order-detail.rdl` report is used and it can be downloaded from [here](#). You can add the reports from the Syncfusion installation location. For more information, refer to [Samples and demos](#).

Hide toolbar items

To hide toolbar items, set the `ToolbarSettings` property. The following code can be used to remove the parameter option from the toolbar and hide the parameter block.

Similarly, you can show or hide all other toolbar options with the help of [toolbarSettings.items](#) enum.

The following code example demonstrates how to hide the parameter block in the Report Viewer at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
    .ReportServiceUrl("/api/ReportViewer")
    .ReportPath("~/Resources/sales-order-detail.rdl")
    .ToolbarSettings(toolbar => toolbar.Items(BoldReports.ReportViewerEnums.ToolbarItems.All &
        ~BoldReports.ReportViewerEnums.ToolbarItems.Parameters))
)
```

Enable stop option in toolbar

To enable stop option in toolbar, set the `ToolbarSettings.Items` property to `BoldReports.ReportViewerEnums.ToolbarItems.All`. The following code can be used to enable stop option in toolbar.

The following code example demonstrates how to enable stop option in the Report Viewer toolbar at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbar => toolbar.Items(BoldReports.ReportViewerEnums.ToolbarItems.All))
)
`
```

Enable export setup option in toolbar

To enable export setup option in toolbar, set the **ToolbarSettings.Items** property to **BoldReports.ReportViewerEnums.ToolbarItems.All**. The following code can be used to enable export setup option in toolbar.

The following code example demonstrates how to enable export setup option in the Report Viewer toolbar at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbar => toolbar.Items(BoldReports.ReportViewerEnums.ToolbarItems.All))
)
`
```

Enable search text option in toolbar

To enable search text option in toolbar, set the **ToolbarSettings.Items** property to **BoldReports.ReportViewerEnums.ToolbarItems.All**. The following code can be used to enable search text option in toolbar.

The following code example demonstrates how to enable search text option in the Report Viewer toolbar at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbar => toolbar.Items(BoldReports.ReportViewerEnums.ToolbarItems.All))
)
`
```

Hide toolbar

To hide the Report Viewer toolbar, set the `ShowToolbar` property to false.

The following code example demonstrates how to hide the toolbar in the Report Viewer at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbar => toolbar.ShowToolbar(false))
)
`
```

Decide or hide the export option

The Report Viewer provides the `ExportOptions` property to show or hide the default export types available in the component. The following code hides the HTML export type from the default export options.

The following code example demonstrates how to decide or hide the export option in the Report Viewer at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ExportSettings(export => export.ExportOptions(BoldReports.ReportViewerEnums.ExportOptions.All &
~BoldReports.ReportViewerEnums.ExportOptions.Html))
)
`
```

Add custom items to the export drop-down

To add custom items to the export drop-down available in the Report Viewer toolbar, use the property `CustomItems` and provide the JSON array of collection input with the `Index`, `CssClass` name, and `Value` properties. Register the `ExportItemClick` event to handle the custom item actions as given in following code snippet.

You can use the following codes to add custom items to the export drop-down from controller and passing the data to view using `ViewBag`.

```
`csharp
public ActionResult Index()
{
    ExportSettings exportSettings = new ExportSettings();
    exportSettings.CustomItems = new List<CustomExportItem>();
}
```

```
var exportItem1 = new CustomExportItem() { Index = 2, CssClass = "", Value = "Text File" };
var exportItem2 = new CustomExportItem() { Index = 4, CssClass = "", Value = "DOT" };
exportSettings.CustomItems.Add(exportItem1);
exportSettings.CustomItems.Add(exportItem2);
ViewBag.exportSettings = exportSettings;
return View();
}
```

You can use the following codes to set an **ExportSettings** property at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ExportSettings(exportSettings => exportSettings.CustomItems(@ViewBag.exportSettings.CustomItems))
)
```

You can use the following codes to create an **ExportItemClick** event at client side.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ExportItemClick("onExportItemClick")
)
<script type="text/javascript">
function onExportItemClick(args) {
alert('Action Triggered');
}
</script>
```

You can use the following codes to get **ExportSettings** properties on a dynamic object using **ViewBag.exportSettings.CustomItems** and invoke custom actions.

```
`js
@(Html.Bold().ReportViewer("viewer")
```

```

.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ExportSettings(exportSettings =>
exportSettings.CustomItems(@ViewBag.exportSettings.CustomItems)).ExportItemClick("onExportItemClick")
)
<script type="text/javascript">
//Export click event handler
function onExportItemClick(args) {
if (args.value === "Text File") {
//Implement the code to export report as Text
alert("Text File export option clicked");
} else if (args.value === "DOT") {
//Implement the code to export report as DOT
alert("DOT export option clicked");
}
}
</script>

```

Add custom toolbar item

You can add custom items to Report Viewer toolbar using the `ToolbarSettings` property. You must register the `ToolBarItemClick` event to handle the newly added custom items action.

You can use the following codes to add custom toolbar item from controller and passing the data to view using `ViewBag`.

```

`csharp
public ActionResult Index()
{
    ToolbarSettings toolbarSettings = new ToolbarSettings();
    toolbarSettings.CustomItems = new List<CustomItem>();
    var customItem = new CustomItem()
    {
        GroupIndex = 1,
        Index = 1,
        CssClass = "e-icon e-mail e-reportviewer-icon",
    }
}

```

```

Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
Id = "E-Mail",
Tooltip = new ToolTip() { Header = "E-Mail", Content = "Send rendered report as mail attachment" }
};
toolbarSettings.CustomItems.Add(customItem);
ViewBag.toolbarSettings = toolbarSettings;
return View();
}
`

```

You can use the following codes to set an **ToolbarSettings** property at client side.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbarSettings =>
toolbarSettings.CustomItems(@ViewBag.toolbarSettings.CustomItems))
)
`

```

You can use the following codes to create an **ToolBarItemClick** event at client side.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolBarItemClick("onToolBarItemClick")
)
<script type="text/javascript">
function onToolBarItemClick(args) {
alert('Action Triggered');
}
</script>
`

```

You can use the following codes to get **ToolbarSettings** properties on a dynamic object using **ViewBag.toolbarSettings.CustomItems** and invoke custom actions.

```

`js

```

```
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbarSettings =>
toolbarSettings.CustomItems(@ViewBag.toolbarSettings.CustomItems))
.ToolBarItemClick("onToolBarItemClick")
)
<script type="text/javascript">
function onToolBarItemClick(args) {
alert('Action Triggered');
}
</script>
`
```

Add custom item to exiting toolbar group

To add a custom item to existing toolbar group use the property **CustomGroups** in **ToolbarSettings** and provide the JSON array of collection input with the **GroupIndex**, **Items**, **Type**, **CssClass** **Id**, and **Tooltip** properties as given in following code snippet.

You can use the following codes to add custom item to exiting toolbar group from controller and passing the data to view using **ViewBag**.

```
`csharp
public ActionResult Index()
{
ToolbarSettings toolbarSettings = new ToolbarSettings();
var groupItem = new List<CustomItem>();
groupItem.Add(new CustomItem()
{
CssClass = "e-icon e-mail e-reportviewer-icon CustomGroup",
Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
Id = "CustomGroup",
Tooltip = new ToolTip() { Header = "CustomGroup", Content = "toolbargroups" }
});
groupItem.Add(new CustomItem()
{
CssClass = "e-icon e-mail e-reportviewer-icon subCustomGroup",
```

```

Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
Id = "subCustomGroup",
Tooltip = new ToolTip() { Header = "subCustomGroup", Content = "subtoolbar groups" }
});
toolbarSettings.CustomGroups.Add(new CustomGroup() { Items = groupItem, GroupIndex = 4 });
ViewBag.toolbarSettings = toolbarSettings;
return View();
}
`

```

You can use the following codes to set an **ToolbarSettings** property at client side.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbarSettings =>
toolbarSettings.CustomGroups(@ViewBag.toolbarSettings.CustomGroups))
)
`

```

You can use the following codes to create an **ToolBarItemClick** event at client side.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolBarItemClick("onToolBarItemClick")
)
<script type="text/javascript">
function onToolBarItemClick(args) {
alert('Action Triggered');
}
</script>
`

```

You can use the following codes to get **ToolbarSettings** properties on a dynamic object using **ViewBag.toolbarSettings.CustomGroups** and invoke custom actions.

```

`js

```



```

@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbarSettings =>
toolbarSettings.CustomGroups(@ViewBag.toolbarSettings.CustomGroups))
.ToolBarItemClick("onToolBarItemClick")
)
<script type="text/javascript">
function onToolBarItemClick(args) {
if (args.value === "CustomGroup") {
//Implement the code to CustomGroup toolbar option
alert("CustomGroup toolbar option clicked");
}
if (args.value === "subCustomGroup") {
//Implement the code to subCustomGroup toolbar option
alert("SubCustomGroup toolbar option clicked");
}
}
}
</script>
`

```

Add new toolbar group

To add new toolbar group and custom items to it, use the property `CustomItems` in `ToolbarSettings` and provide the JSON array of collection input with the `GroupIndex`, `Index` properties. The `CustomItem` must have the properties `Type`, `CssClass` and `Tooltip` as given in following code snippet.

You can use the following codes to add new toolbar group from controller and passing the data to view using `ViewBag`.

```

`csharp
public ActionResult Index()
{
ToolbarSettings toolbarSettings = new ToolbarSettings();
toolbarSettings.CustomItems = new List<CustomItem>();
var customItem = new CustomItem()
{
GroupIndex = 1,

```

```

Index = 1,
CssClass = "e-icon e-mail e-reportviewer-icon",
Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
Id = "E-Mail",
Tooltip = new ToolTip() { Header = "E-Mail", Content = "Send rendered report as mail attachment" }
};
toolbarSettings.CustomItems.Add(customItem);
ViewBag.toolbarSettings = toolbarSettings;
return View();
}
`

```

You can use the following codes to set an **ToolbarSettings** property at client side.

```

`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.ToolbarSettings(toolbarSettings =>
toolbarSettings.CustomItems(@ViewBag.toolbarSettings.CustomItems))
)
`

```

Localization of Bold Reports ASP.NET MVC Report Viewer

Localization of ASP.NET MVC Report Viewer allows you to localize the static text such as tooltip, parameter block, and dialog text based on a specific culture. To render the static text with specific culture, refer to the following corresponding culture script files and set culture name to the **locale** property of the Report Viewer.

- **ej.localtexts.fr-FR.min.js**
- **ej.culture.fr-FR.min.js**

Refer this [CDN links for Localization and Culture](#) to get the Localization and Culture scripts for available Culture Code.

1. Install the **BoldReports.Global** Nuget package in your MVC application.
2. Refer to this **ej.localtexts.fr-FR.min.js** script file from the **Scripts** folder of your application.

```

`html
<script src="~/Scripts/bold-reports/l10n/reportdesigner/ej.localtexts.fr-FR.min.js"></script>

```

,

3. Refer to this `ej.culture.fr-FR.min.js` script file from the `Scripts` folder of your application.

```
`html
```

```
<script src="~/Scripts/bold-reports/i18n/ej.culture.fr-FR.min.js"></script>
```

,

4. To render the localization Report Viewer, use the following code snippet.
 - The following code example illustrates how to localize Report Viewer in the `Index.cshtml` page.

```
`js
```

```
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.Locale("fr-FR")
)
```

,

- The following code example illustrates how to localize Report Viewer in the `~/Views/Shared/_Layout.cshtml` page.

```
`js
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>@ViewBag.Title - Render Report Viewer in French localization</title>
@Styles.Render("~/Content/css")
@Styles.Render("~/Content/material/bold.reports.all.min.css")
@Scripts.Render("~/bundles/modernizr")
</head>
<body>
<div style="min-height: 600px; width: 100%;">
@RenderBody()
```

```

</div>
@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
<!--Render the gauge item. Add these scripts only if your report contains the gauge report item.-->
@Scripts.Render("~/Scripts/bold-reports/common/ej2-base.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/ej2-data.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/ej2-pdf-export.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/ej2-svg-base.min.js")
@Scripts.Render("~/Scripts/data-visualization/ej2-circulargauge.min.js")
@Scripts.Render("~/Scripts/data-visualization/ej2-lineargauge.min.js")
<!--Renders the map item. Add this script only if your report contains the map report item.-->
@Scripts.Render("~/Scripts/data-visualization/ej2-maps.min.js")
@Scripts.Render("~/Scripts/common/bold.reports.common.min.js")
@Scripts.Render("~/Scripts/common/bold.reports.widgets.min.js")
<!--Render the chart item. Add these scripts only if your report contains the chart report item.-->
@Scripts.Render("~/Scripts/data-visualization/ej.chart.min.js")
<!-- Report Viewer component script-->
@Scripts.Render("~/Scripts/bold.report-viewer.min.js")
<!-- Report Viewer french localization scripts-->
@Scripts.Render("~/Scripts/l10n/ej.localetexts.fr-FR.min.js")
@Scripts.Render("~/Scripts/i18n/ej.culture.fr-FR.min.js")
@RenderSection("scripts", required: false)
@Html.Bold().ScriptManager()
</body>
</html>

```

Responsive layout rendering of ASP.NET MVC Report Viewer

Report Viewer will adaptively render itself with optimal user interfaces for phone, tablet, or desktop form factors. This helps your application to scale elegantly on all form factors with ease. You can enable responsive layout rendering in Report Viewer by setting `isResponsive` property to true.

```

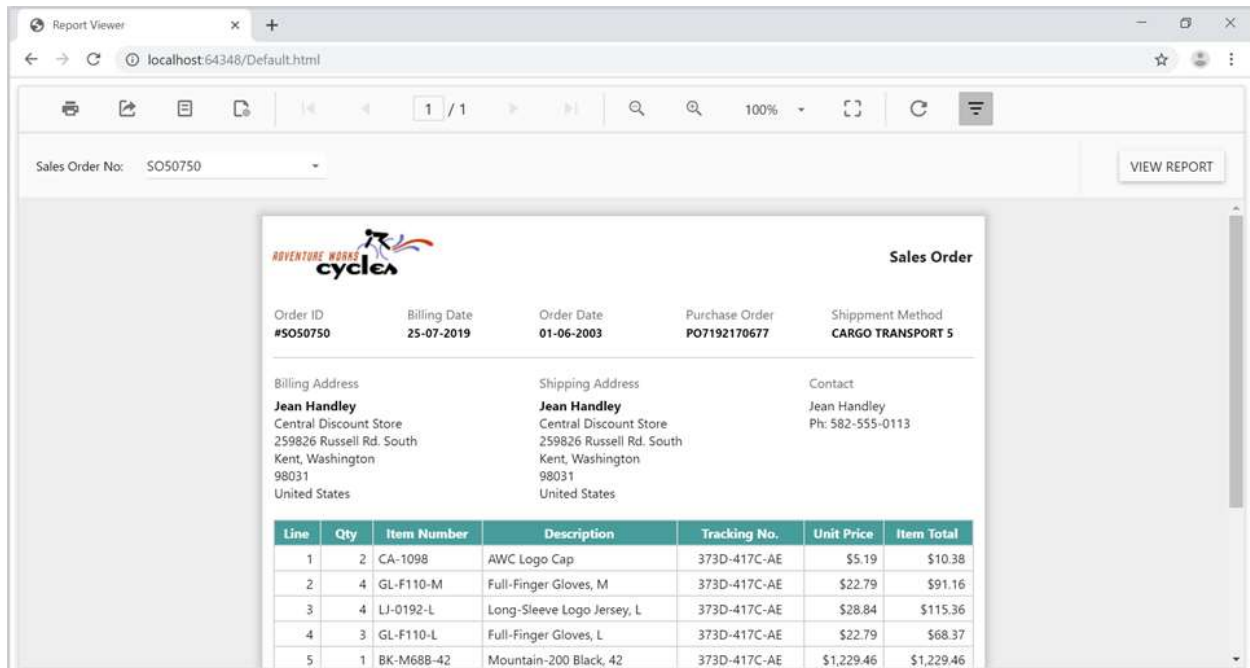
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")

```

```
.ReportPath("~/Resources/sales-order-detail.rdl")
.IsResponsive(true)
)
```

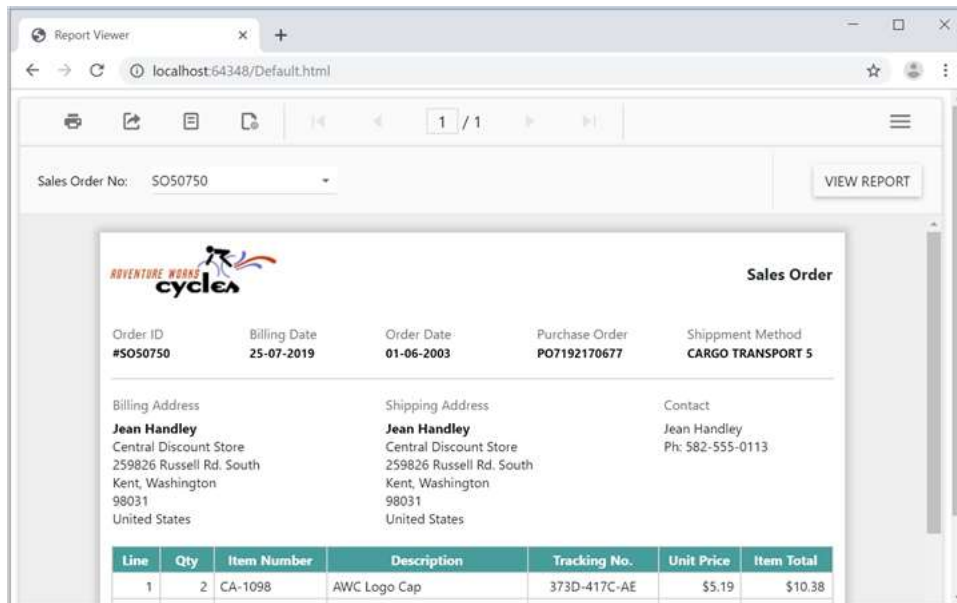
Normal layout

The following output shows the normal layout rendering of Report Viewer tool bar items.



Responsive layout

The following output shows the responsive layout rendering of Report Viewer tool bar items.



Limitations

RDL specification

The Report Viewer control does not support RDL specification for SQL Server 2000 and SQL Server 2005.

Report layout

- In the Tablix cell split layout process, the entire cell moves to the next page to display the complete cell items, when the table cell width value exceeds the page width.

Expressions

The object function and VB function do not have complete support.

SSRS

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the server. If the report has any data source that uses credentials to connect with the database, then you should specify the data source credentials for each report data source to establish database connection.

HTML Formatted Data with Report

Report viewer supports showing the HTML formatted data with Textbox report items along with the inline CSS. This section provides the information about supported tags and limitations.

Supported HTML Tags

- Hyperlinks:
- Fonts:
- Header, style and block elements: `

`,`

,

,

- ,`

- Text format: ,,,

Limitations of Cascading Style Sheet Attributes

The following is a list of attributes that are supported:

- text-align, text-indent
- font-family
- font-size
- Only valid RDL size values are supported in absolute CSS length units. Supported units are: in, cm, mm, pt, pc, px, ex, and em.
- Relative CSS length units are ignored and are not supported. Unsupported units include percentage (%), and rem.

- color
- padding, padding-bottom, padding-top, padding-right, and padding-left
- font-weight

Samples and Demos

Browse and explore the ready-to-use RDL, RDLC reports, samples, online, and offline demos.

Locally installed reports

You can obtain the sample RDL and RDLC files from the Bold Reports installed location

```
%localappdata%\Bold Reports\Embedded Reporting\Samples\Common\Data\ReportTemplate.
```

Offline demos

The offline samples are provided in the Bold Reporting Tools setup. For more details, refer to the [Bold Reporting Tools sample deployment](#).

Online demos

You can view the ASP.NET MVC Report Viewer online demo samples from [here](#).

GitHub demo samples

Click [here](#) to view the GitHub Report Viewer demo samples.

Migrate Report Viewer application

In our Bold Reports new assemblies are introduced for both client and server-side to resolve the compatibility problem between Essential Studio Report Viewer versions. It has changes in both Web API service and client-side scripts.

This section provides step-by-step instructions for migrating Report Viewer from Syncfusion Essential Studio release version to Bold Reports version of ASP.NET MVC Report Viewer application:

Client-side migration

1. In the Solution Explorer, right-click the **References** and remove the following assembly references:
 - Syncfusion.EJ
 - Syncfusion.EJ.MVC
2. Add the assembly from the Syncfusion NuGet package **BoldReports.Mvc5**. To add from NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search for **BoldReports.Mvc5** NuGet package, and install in your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Scripts and CSS references

1. Remove the following scripts and CSS references from the Report Viewer `\Views\Shared_Layout.cshtml` page:
 - ej.web.all.min.css
 - ej.web.all.min.js

- `ej.unobtrusive.min.js`
- 2. The Report Viewer scripts and style sheets are added to the `Scripts` and `Content` folders in your application when installing the `BoldReports.Mvc5` NuGet package. Add scripts as in the following code sample.

```
`js
@Styles.Render("~/Content/bold-reports/material/bold.reports.all.min.css")
<!--Render the gauge item. Add these scripts only if your report contains the gauge report item.-->
@Scripts.Render("~/Scripts/bold-reports/common/ej2-base.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/ej2-data.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/ej2-pdf-export.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/ej2-svg-base.min.js")
@Scripts.Render("~/Scripts/bold-reports/data-visualization/ej2-circulargauge.min.js")
@Scripts.Render("~/Scripts/bold-reports/data-visualization/ej2-lineargauge.min.js")
<!--Renders the map item. Add this script only if your report contains the map report item.-->
@Scripts.Render("~/Scripts/bold-reports/data-visualization/ej2-maps.min.js")
<!-- Report Viewer component script-->
@Scripts.Render("~/Scripts/bold-reports/common/bold.reports.common.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/bold.reports.widgets.min.js")
<!--Renders the chart item. Add this script only if your report contains the chart report item.-->
@Scripts.Render("~/Scripts/bold-reports/data-visualization/ej.chart.min.js")
@Scripts.Render("~/Scripts/bold-reports/bold.report-viewer.min.js")
`
```

- 3. Open the the `~/Views/Shared/_Layout.cshtml` page and replace the script manager tag at the end of element from `@(Html.EJ().ScriptManager())` to `@(Html.Bold().ScriptManager())` as in the following code sample.

```
`html
<body>
....
....
<!-- Bold Report Viewer ScriptManager -->
@Html.Bold().ScriptManager()
</body>
`
```


Adding data visualization scripts

To render the report with data visualization components such as chart, gauge, and map items, add scripts of the visualization element. The following table shows the script reference that needs to be added in Report Viewer page for data visualization elements.

Visualization item | Script file

Gauge | ej2-base.min.js, ej2-data.min.js, ej2-pdf-export.min.js, ej2-svg-base.min.js, ej2-lineargauge.min.js and ej2-circulargauge.min.js

Map | ej2-maps.min.js

Chart | ej.chart.min.js

To render the chart report item, add chart control script `ej.chart.min.js` before the `bold.report-viewer.min.js` reference in the `\Views\Shared_Layout.cshtml` page as demonstrated in the following code sample.

```
`js
@Styles.Render("~/Content/bold-reports/material/bold.reports.all.min.css")
@Scripts.Render("~/Scripts/bold-reports/common/bold.reports.common.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/bold.reports.widgets.min.js")
<!--Renders the chart item. Add this script, only if your report contains the chart report item.-->
@Scripts.Render("~/Scripts/bold-reports/data-visualization/ej.chart.min.js")
<!-- Report Viewer component script-->
@Scripts.Render("~/Scripts/bold-reports/bold.report-viewer.min.js")
`
```

The following code can be used to render the chart, gauge, and map report items in Report Viewer.

```
`js
@Styles.Render("~/Content/bold-reports/material/bold.reports.all.min.css")
<!--Render the gauge item. Add these scripts only if your report contains the gauge report item.-->
@Scripts.Render("~/Scripts/bold-reports/common/ej2-base.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/ej2-data.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/ej2-pdf-export.min.js")
@Scripts.Render("~/Scripts/bold-reports/common/ej2-svg-base.min.js")
@Scripts.Render("~/Scripts/bold-reports/data-visualization/ej2-circulargauge.min.js")
@Scripts.Render("~/Scripts/bold-reports/data-visualization/ej2-lineargauge.min.js")
<!--Renders the map item. Add this script only if your report contains the map report item.-->
@Scripts.Render("~/Scripts/bold-reports/data-visualization/ej2-maps.min.js")
<!-- Report Viewer component script-->
@Scripts.Render("~/Scripts/bold-reports/common/bold.reports.common.min.js")
`
```

```
@Scripts.Render("~/Scripts/bold-reports/common/bold.reports.widgets.min.js")
<!-- Chart component script-->
@Scripts.Render("~/Scripts/bold-reports/data-visualization/ej.chart.min.js")
@Scripts.Render("~/Scripts/bold-reports/bold.report-viewer.min.js")
`
```

Control initialization

1. The component prefix has been changed from `EJ()` to `SF()`.
2. Open the `~/Views/Web.config` file and add the `BoldReports.Mvc` assembly reference to the element.

```
`html
<configuration>
....
....
<system.web.webPages.razor>
....
....
<pages pageBaseType="System.Web.Mvc.WebViewPage">
<namespaces>
....
....
<add namespace="BoldReports.Mvc"/>
</namespaces>
</pages>
</system.web.webPages.razor>
....
....
</configuration>
`
```

3. Set the value of `UnobtrusiveJavaScriptEnabled` to `false` in Root directory `web.config` file as demonstrated in the following code example.

```
`js
<configuration>
```

```

<appSettings>
.....
.....
<add key="UnobtrusiveJavaScriptEnabled" value="false" />
</appSettings>
.....
.....
</configuration>
`

```

4. Open the Report Viewer page in `Index.cshtml` page.
5. Replace the component tag `Html.EJ().ReportViewer("viewer")` with `Html.Bold().ReportViewer("viewer")`.

```

`js
<div style="min-height:600px">
@ (Html.Bold().ReportViewer("viewer"))
</div>
`

```

Server-side migration

1. In the Solution Explorer, right-click the **References** and remove the `Syncfusion.EJ.ReportViewer` assembly reference.
2. Add the assembly from the Bold Reports NuGet package `BoldReports.Web`. To add from NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search for `BoldReports.Web` NuGet package, and then install in your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Web API Controller

1. The `IRreportController` interface is moved to `BoldReports.Web.ReportViewer`. Open the Report Viewer Web API Controller file and remove the following using statement.

```

`csharp
using Syncfusion.EJ.ReportViewer;
`

```

2. Add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

Your application is successfully upgraded to the latest version of Report Viewer, and you can run the application with new assemblies.

Report export configuration

Now, the **BoldReports.Web** can export the reports with data visualization components only using web components. It is mandatory to configure the web scripts in Report Viewer Web API controller for exporting data visualization components such as chart, gauge, and map that are used in report definition. To configure the scripts in Web API, refer to the following steps:

1. Open the Report Viewer Web API controller.
2. Configure the following scripts and styles in **OnInitReportOptions** on Web API controller:
 - o **jquery-1.10.2.min.js**
 - o **bold.reports.common.min.js**
 - o **bold.reports.widgets.min.js**
 - o **ej.chart.min.js** - Exports the chart item. Add this script only if your report contains the chart report item.
 - o **ej2-base.min.js**, **ej2-data.min.js**, **ej2-pdf-export.min.js**, **ej2-svg-base.min.js**, **ej2-lineargauge.min.js** and **ej2-circulargauge.min.js** - Exports the gauge item. Add this script only if your report contains the gauge report item.
 - o **ej2-maps.min.js** - Exports the map item. Add this script only if your report contains the map report item.
 - o **bold.report-viewer.min.js**
3. Replace the following codes in Report Viewer Web API controller.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    var resourcesPath = System.Web.Hosting.HostingEnvironment.MapPath("~/Scripts");
    reportOption.ReportModel.ExportResources.Scripts = new List<string>
    {
        //Gauge component scripts
        resourcesPath + @"\"bold-reports\\common\\ej2-base.min.js",
        resourcesPath + @"\"bold-reports\\common\\ej2-data.min.js",
        resourcesPath + @"\"bold-reports\\common\\ej2-pdf-export.min.js",
        resourcesPath + @"\"bold-reports\\common\\ej2-svg-base.min.js",
    }
}
```

```

resourcesPath + @"\"bold-reports\data-visualization\ej2-lineargauge.min.js",
resourcesPath + @"\"bold-reports\data-visualization\ej2-circulargauge.min.js",
//Map component script
resourcesPath + @"\"bold-reports\data-visualization\ej2-maps.min.js",
resourcesPath + @"\"bold-reports\common\bold.reports.common.min.js",
resourcesPath + @"\"bold-reports\common\bold.reports.widgets.min.js",
//Chart Component Script
resourcesPath + @"\"bold-reports\data-visualization\ej.chart.min.js",
//Report Viewer Script
resourcesPath + @"\"bold-reports\bold.report-viewer.min.js"
};
reportOption.ReportModel.ExportResources.DependentScripts = new List<string>
{
resourcesPath + @"\"jquery-1.7.1.min.js"
};
}
`

```

The data visulization components will not export without the above script configurations.

NuGet Packages for ASP.NET MVC

Refer to the following steps to configure Bold Reporting NuGet packages for ASP.NET MVC application.

Configure NuGet feed URL

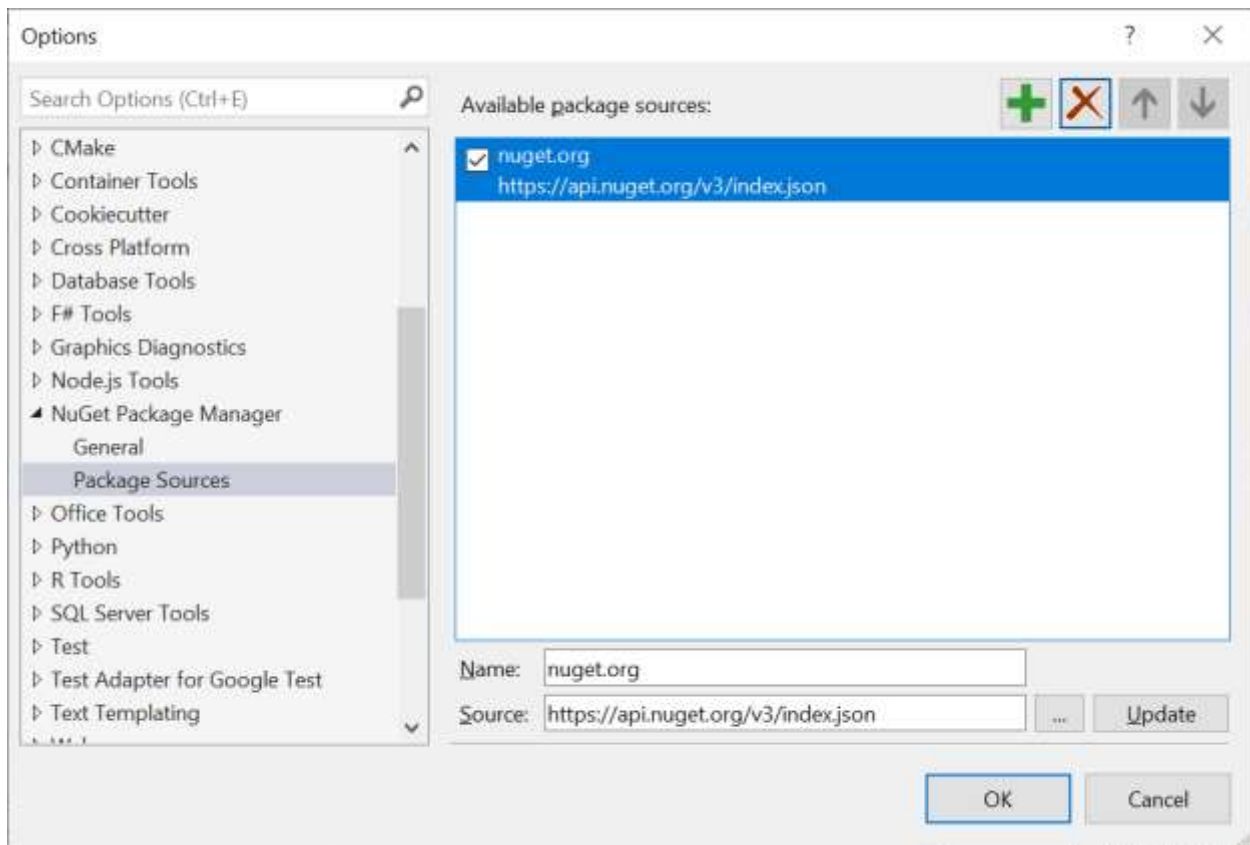
Online NuGet feed URL

The Bold Reporting NuGet packages are published in **Nuget.org**. To configure the online packages, use the following steps:

1. Open Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager** and select **Package Sources**.
4. Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: NuGet.org

Source: <https://api.nuget.org/v3/index.json>



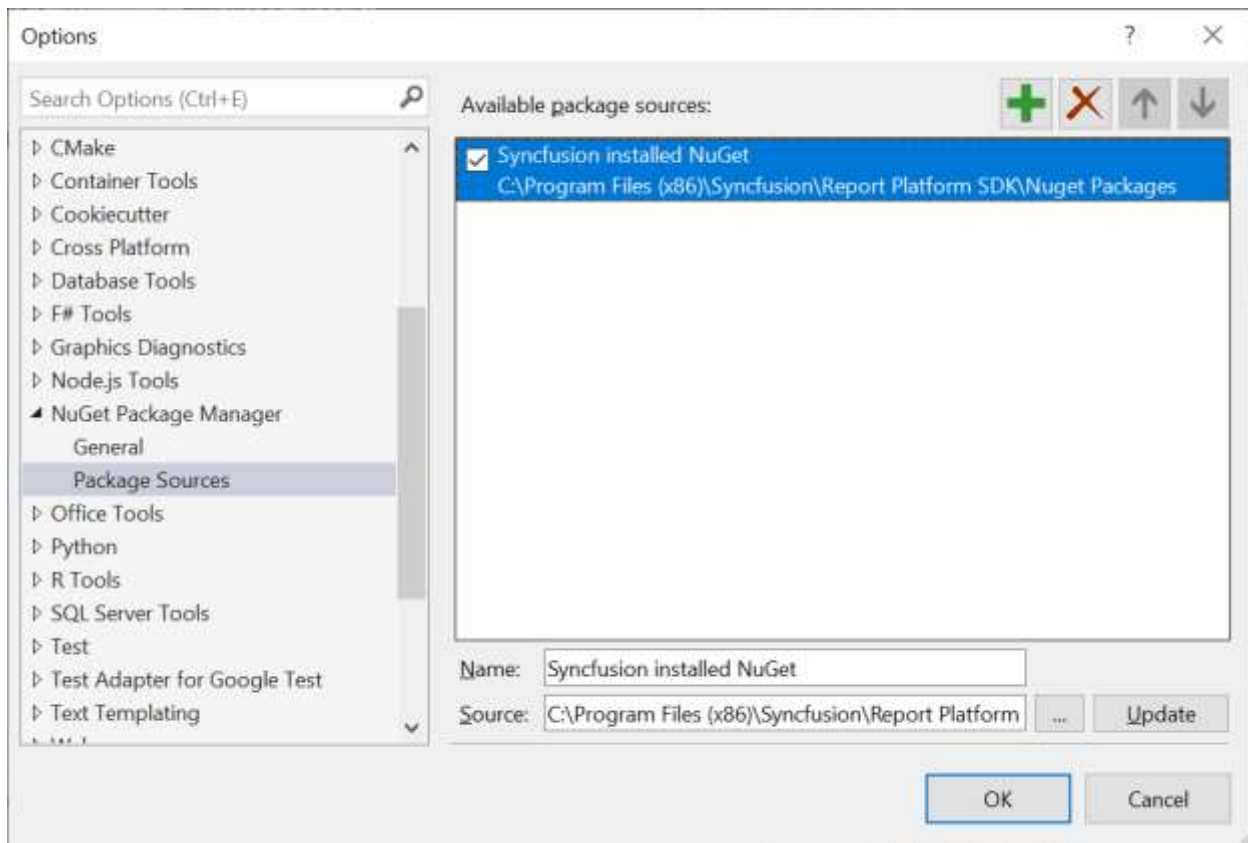
Offline NuGet feed URL

Bold Reporting NuGet packages are shipped into our Bold Reporting Tools build. To configure the packages from Bold Reports installed location, use the following steps:

1. Open your Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager**, and then select **Package Sources**.
4. Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: Bold Reports installed NuGet

Source: {System Drive}:\Program Files (x86)\Bold Reports\Reporting Tools\Nuget Packages.



The system drive varies based on the installed location in your machine.

Installing NuGet packages

Install using NuGet Package Manager

The NuGet Package Manager can be used to search and install NuGet packages in Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution**. Alternatively, right-click the project/solution in Solution Explorer tab, and choose **Manage NuGet Packages**.
2. By default, the **NuGet.org** package is selected in the **Package source** drop-down. Select package source, search for the packages (**BoldReports.Mvc5** or **BoldReports.Web**), and then click **Install** button.

Install using Package Manager Console

To install the Reporting component using the Package Manager Console as NuGet packages,

1. On the **Tools** menu, select **NuGet Package Manager**, and then click **Package Manager Console**.
2. Run the following NuGet installation commands:

```
`cmd
```

install specified package in default project

Install-Package <Package Name>

install specified package in default project with specified package source

Install-Package <Package Name> -Source <Source Location>

install specified package in specified project

Install-Package <Package Name> -ProjectName <Project Name>

,

For example:

`cmd

install specified package in default project

Install-Package BoldReports.Web

install specified package in default project with specified Package Source

Install-Package BoldReports.Web -Source "https://api.nuget.org/v3/index.json"

install specified package in specified project

Install-Package BoldReports.Web -ProjectName BoldReportsApplication

,

Upgrading NuGet packages

Upgrading using NuGet Package Manager

NuGet packages can be updated to their specific version or latest version available in the Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution....** Alternatively, right-click the project/solution in the Solution Explorer tab, and then choose **Manage NuGet Packages**.
2. Select the **Updates** tab to see the packages available for update from the desired package sources, select the required packages and specific version from the drop-down, and then click the **Update** button.

Upgrading using Package Manger Console

To update the installed Bold Reporting NuGet packages using the Package Manager Console:

1. On the **Tools** menu, select **NuGet Package Manager**, and then select **Package Manager Console**.
2. Run the following NuGet installation commands:

`cmd

Update specific NuGet package in default project

Update-Package <Package Name>

Update all the packages in default project

Update-Package

Update specified package in default project with specified package source

Update-Package <Package Name> -Source <Source Location>

Update specified package in specified project

Update-Package <Package Name> - ProjectName <Project Name>

,

For example:

`cmd

Update specified Bold Reporting NuGet package

Update-Package BoldReports.Web

Update specified package in default project with specified Package Source

Update-Package BoldReports.Web -Source "https://api.nuget.org/v3/index.json"

Update specified package in specified project

Update-Package BoldReports.Web -ProjectName BoldReportsApplication

,

Upgrading using NuGet CLI

Using the NuGet CLI, all the NuGet packages in the project can be updated to the available latest version:

1. Download the latest [NuGet CLI](#).

To update the existing `nuget.exe` to latest version use the following command:

`cmd

nuget update -self

,

2. Open the downloaded executable location in the command window. Run the following "update commands" to update the Bold Reporting NuGet packages.

```
`cmd
```

update all NuGet packages from config file

```
nuget update <configPath> [options]
```

update all NuGet packages from specified Packages Source

```
nuget update -Source <Source Location> [optional]
```

```
,
```

`configPath` is optional. It identifies the `package.config` or solutions file lists the packages utilized in the project.

For example:

```
`cmd
```

Update all NuGet packages from config file

```
nuget update "C:\Users\BoldReportsApplication\package.config"
```

Update all NuGet packages from specified Packages Source

```
nuget update -Source "https://api.nuget.org/v3/index.json"
```

```
,
```

The Update command is not working as expected in Mono (Mac and Linux) and projects using PackageReference format.

Frequently asked questions

This section helps to get the answer for the frequently asked questions in Bold Reports ASP.NET MVC Report Viewer.

1. [How can improve the performance and handle the large amounts of data with Report Viewer?](#)
2. [Is Report Viewer compatible with latest version of JQuery library?](#)
3. [Is the back action from drillthrough report will load the report again with Report Viewer?](#)
4. [Is possible to change the culture of the date time parameter?](#)
5. [Is it possible to hide report parameters?](#)
6. [Is it possible to hide the export options in Report Viewer?](#)
7. [Is it possible to load reports providing parameter values at runtime?](#)

Is possible to change the culture of the date time parameter

Yes, we can change the culture of the date time parameter for Report Viewer by changing the locale. You can make use the following reference to change the locale of Report Viewer.

[ReportViewer Localization](#)

In Report Viewer, we could not change the culture of specific parameter or UI. So, we have to achieve the requirements only by changing the culture.

Is it possible to hide the parameters in Report Viewer

Yes, it is possible to hide the parameters in Report Viewer. On hiding the parameters, the users will not be able to have the parameter block in the Report Viewer. Refer to this [Hide parameter block and toolbar items](#) section.

Is it possible to hide the export options in Report Viewer

Yes, it is possible to hide the export options in Report Viewer. The Report Viewer provides the `exportOptions` property to show or hide the default export types available in the component. Refer to this [Decide or Hide the export options](#) section.

Is it possible to load reports providing parameter values at runtime

Yes, it is possible to load reports with application inputs as parameters in Report Viewer. You need to provide the input values to the Report Viewer from client side at runtime using the `parameters` property, which accepts JSON array values. Refer to this [Set parameter at client](#) section.

How to Create RDL/RDLC Report

The following sections explain about how to create a new RDL/RDLC report using Bold Report Designer, Microsoft Report Builder and Visual Studio Report Server project template.

- [Create a RDL report](#)
- [Create a RDLC report](#)
- [Change the exporting document file name based on parameter](#)
- [Change the connection string datasource dynamically](#)
- [Disable the vertical scrollbar in parameter panel](#)
- [How to use Report Viewer in partial view?](#)
- [How to pass multiple values using custom data?](#)
- [How to clear cache when closing the Report Viewer?](#)

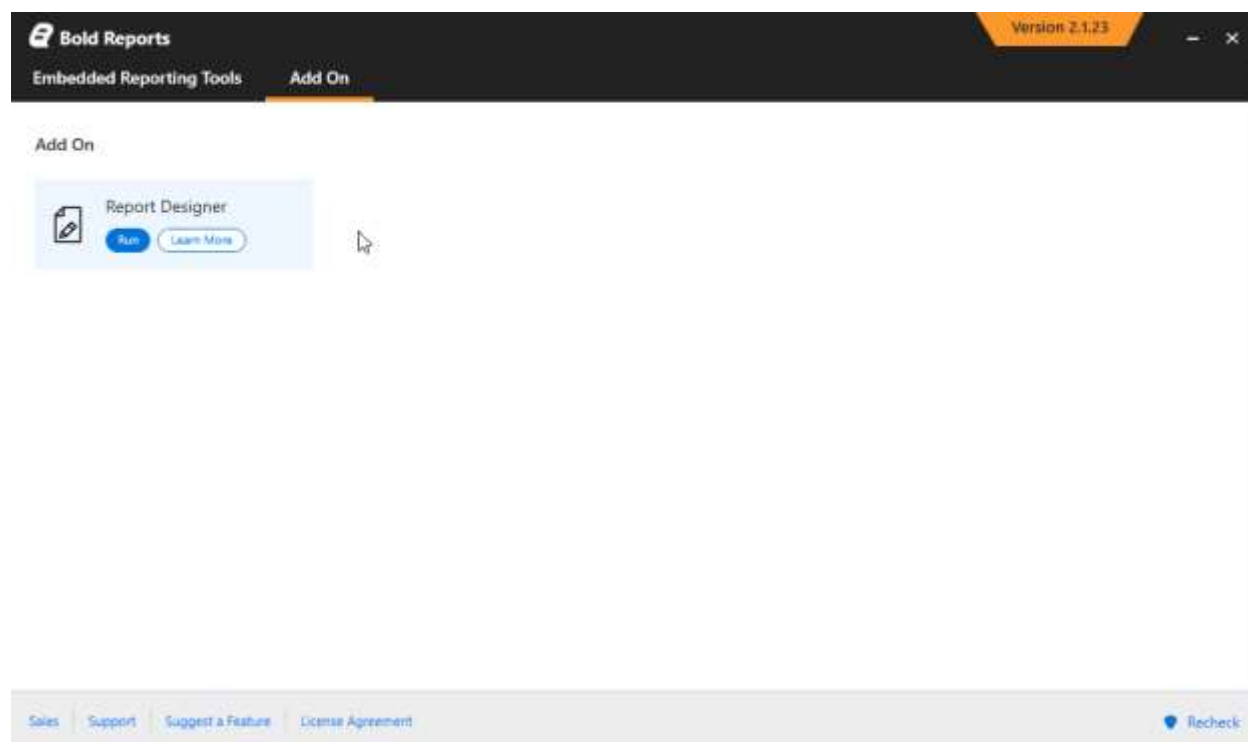
Create a SSRS RDL report

You can create an RDL report using any of the following reporting tools:

- Bold Reports Report Designer.
- Microsoft Report Builder.
- Visual Studio Report Server project template.

Bold Reports Report Designer

Bold Reports Report Designer provides the intuitive user interface to create and edit the RDL reports, which is available in Bold Embedded Reporting Tools Control Panel Add On.



Microsoft SQL Report Builder

You can create an RDL report using the Microsoft stand-alone Report Builder. For more details, refer to this [online documentation](#).

Visual Studio Report Server template

To create an RDL report in Visual Studio, a Report Server project is required where you can save your report definition (.rdl) file. For more details, refer to this [Visual Studio documentation](#).

If you do not have the Business Intelligence or Report Server Project options, you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create a RDLC report using business object data source

This section describes step by step procedure to create an RDLC report using Visual Studio Reporting project type.

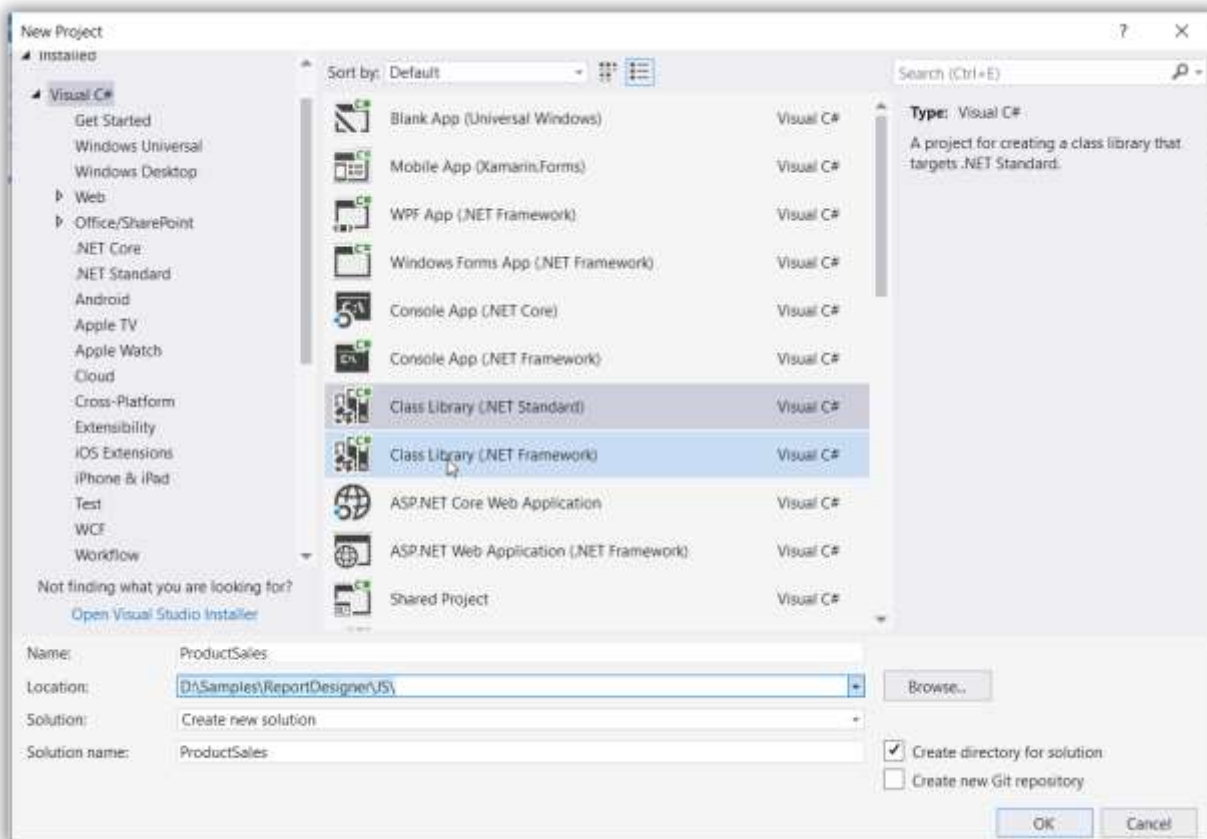
Prerequisites

- Microsoft Visual Studio 2017 or higher
- [Microsoft RDLC Report Designer](#)

If you are using Microsoft Visual Studio lower to 2017 version then you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create business object class

1. Open Visual Studio from the File menu and select **New Project**.
2. Create project with class library type from the project type list.



3. Create the class with necessary properties. You can find the reference below,

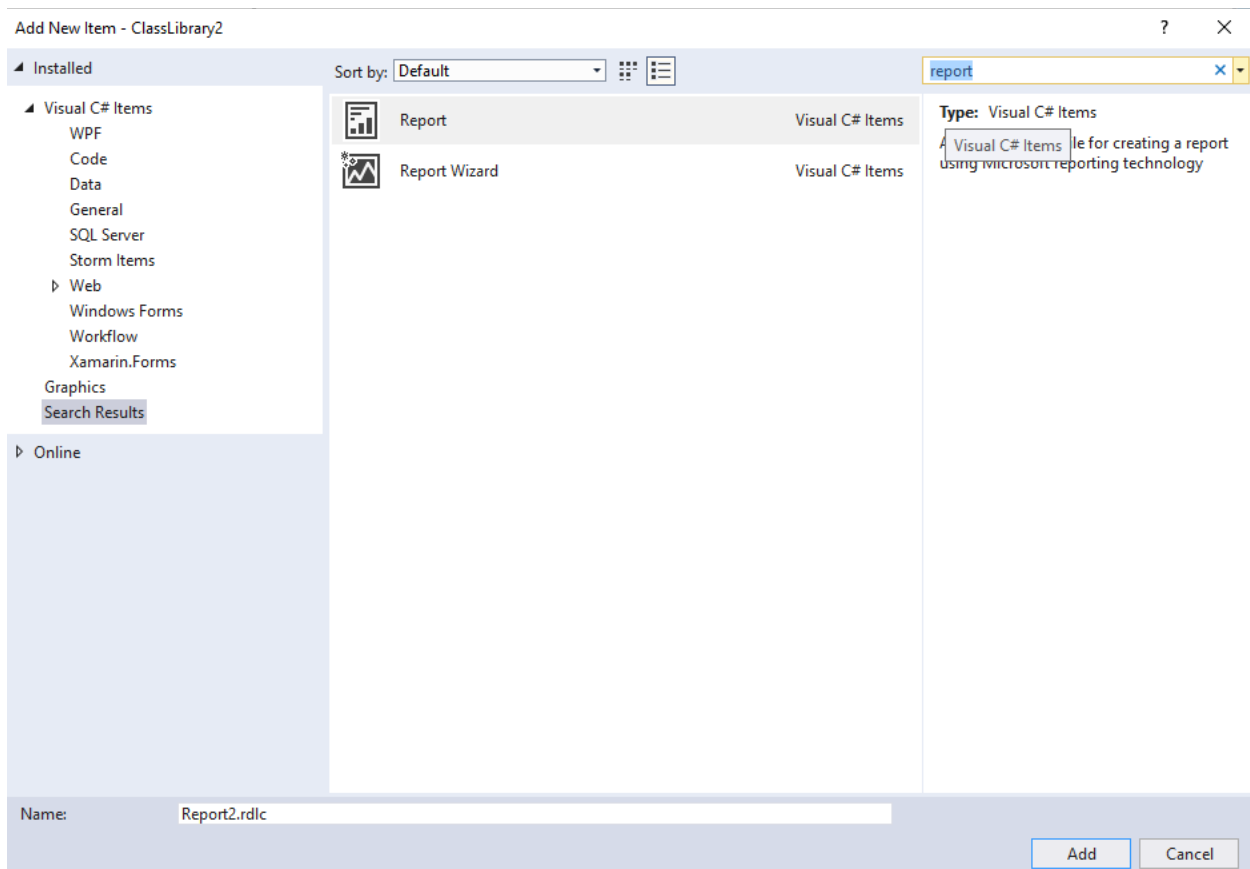
```
`csharp
public class ProductSales
{
    public string ProdCat { get; set; }
    public string SubCat { get; set; }
    public string OrderYear { get; set; }
    public string OrderQtr { get; set; }
    public double Sales { get; set; }
}
`
```

4. Clean and build the application.

Add an RDLC report

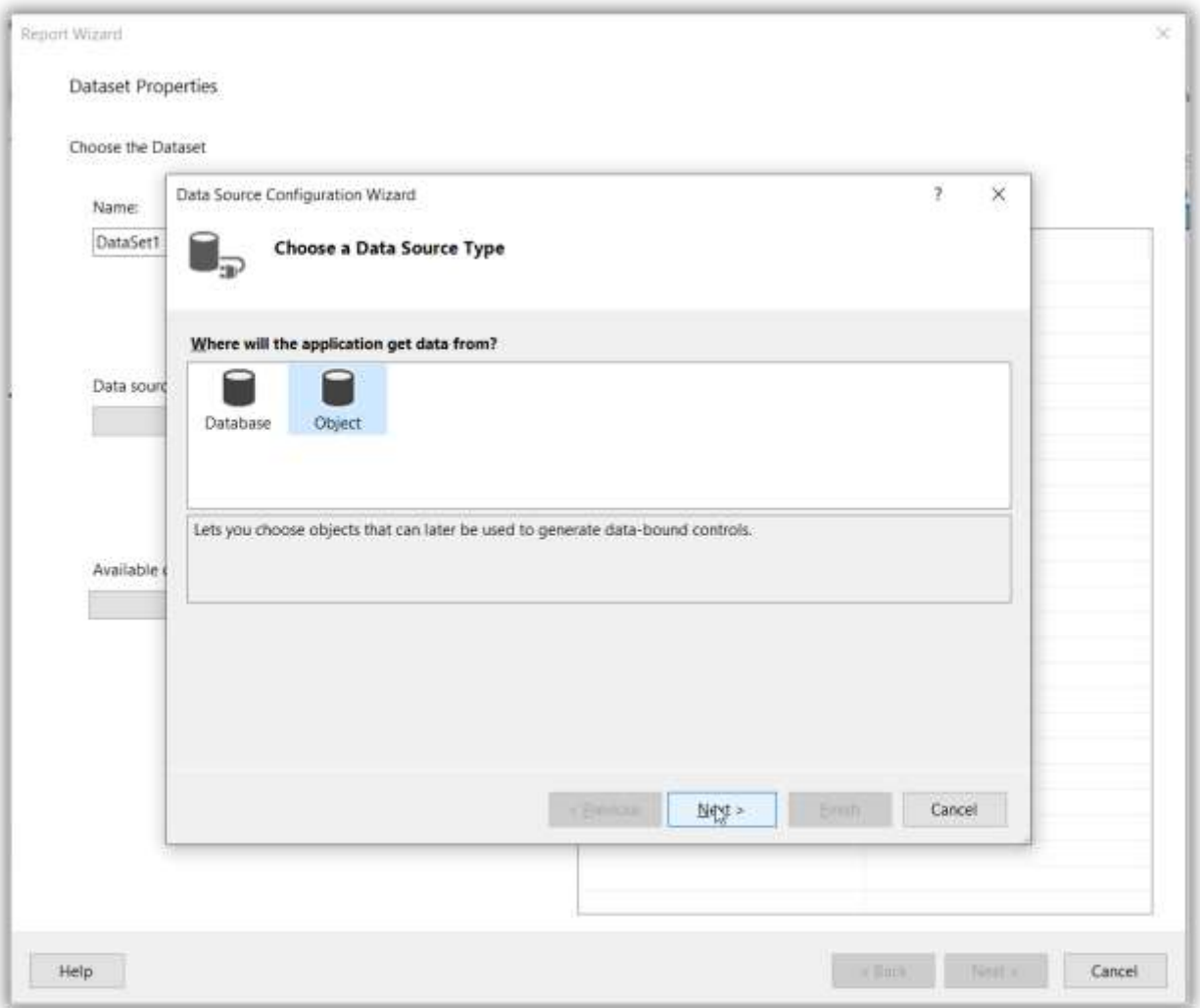
1. Right-click the project and click **Add > New Item**.

2. Search Report with new item and select **Report Wizard** to start the report creation with dataset selection.
3. Click **Add**.

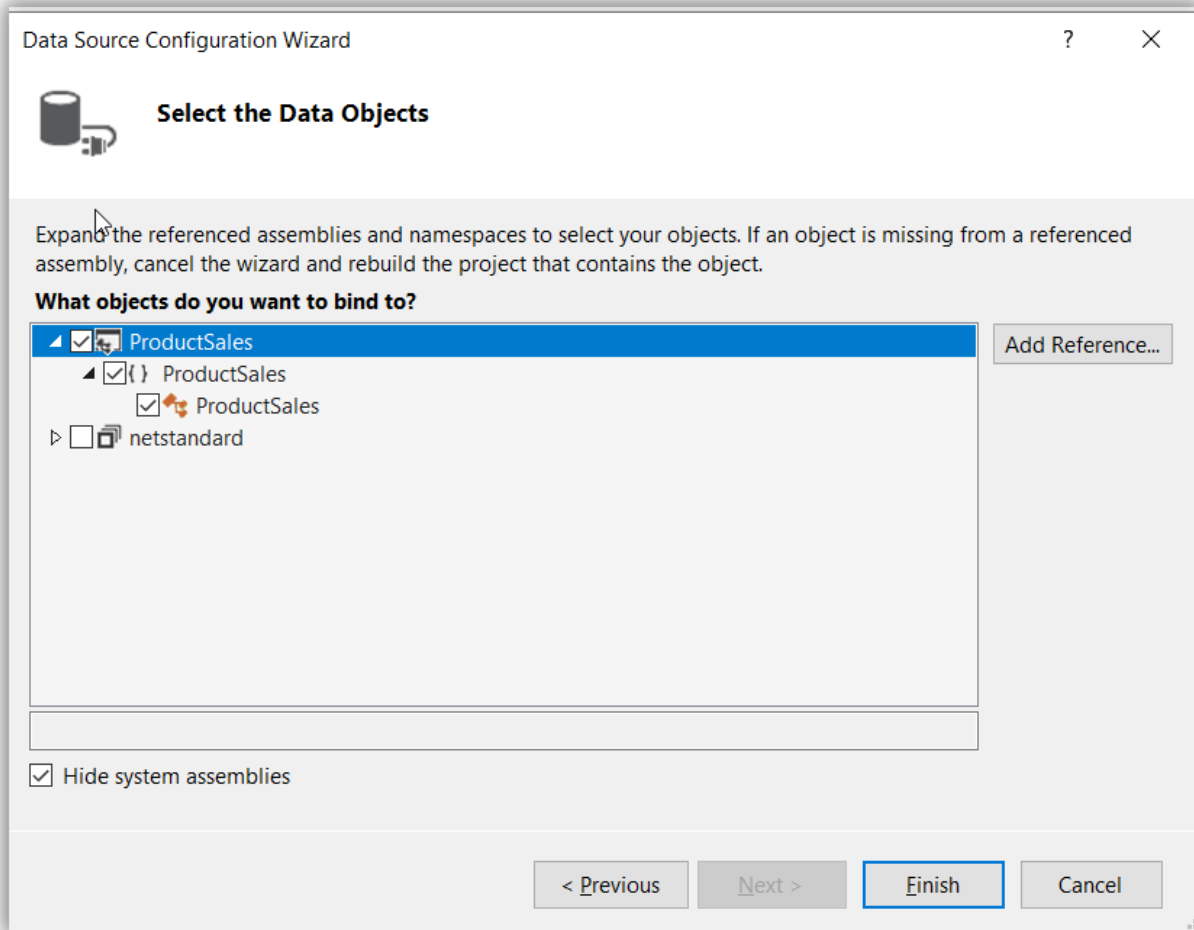


Data source and table configuration wizard

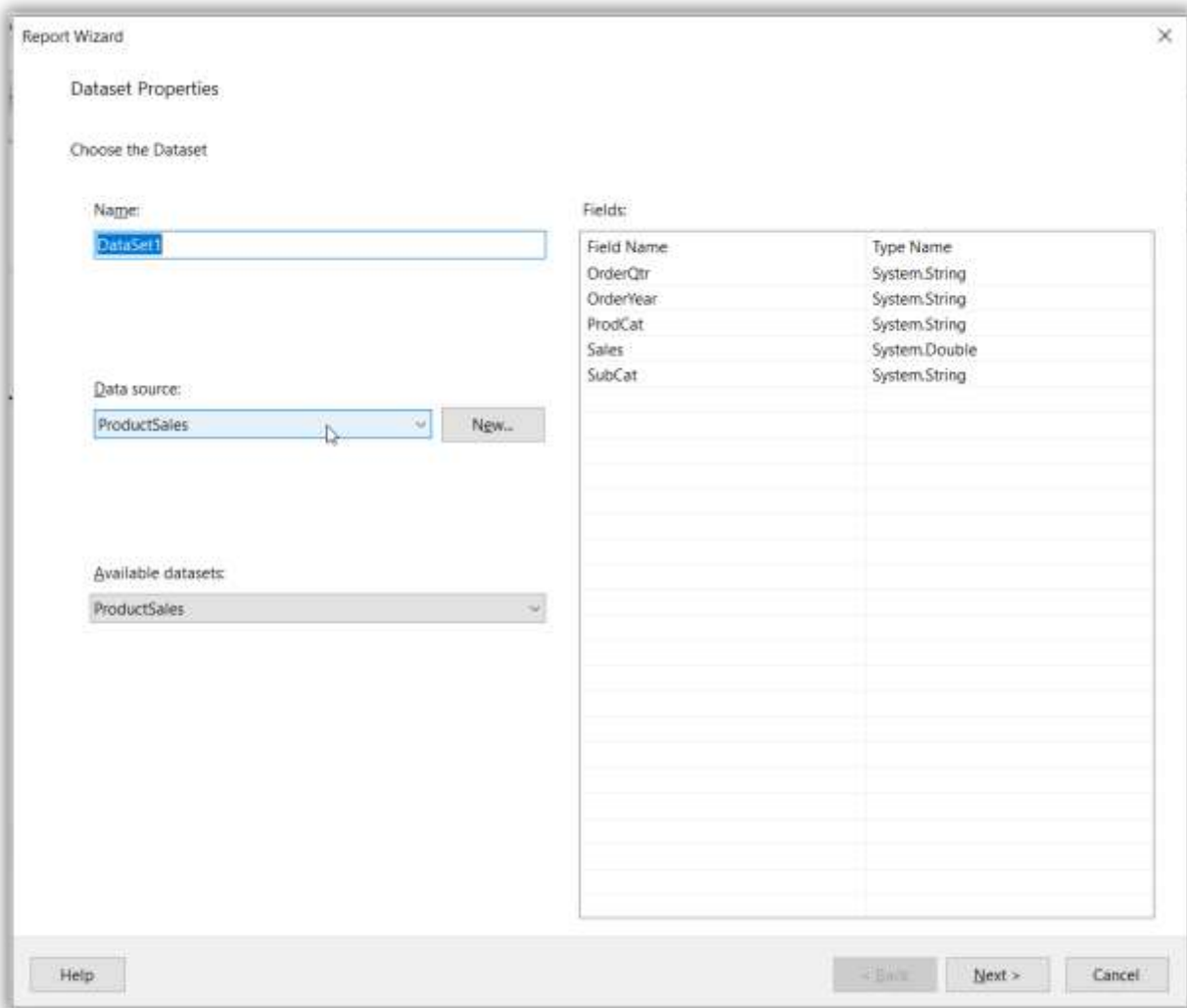
1. Choose object type from the Data Source Configuration wizard and click **Next**.



2. Expand the tree view and select **ProductSales**, and then click **Finish**.



3. In the DataSet Properties wizard, specify the dataset name as `SalesData`.

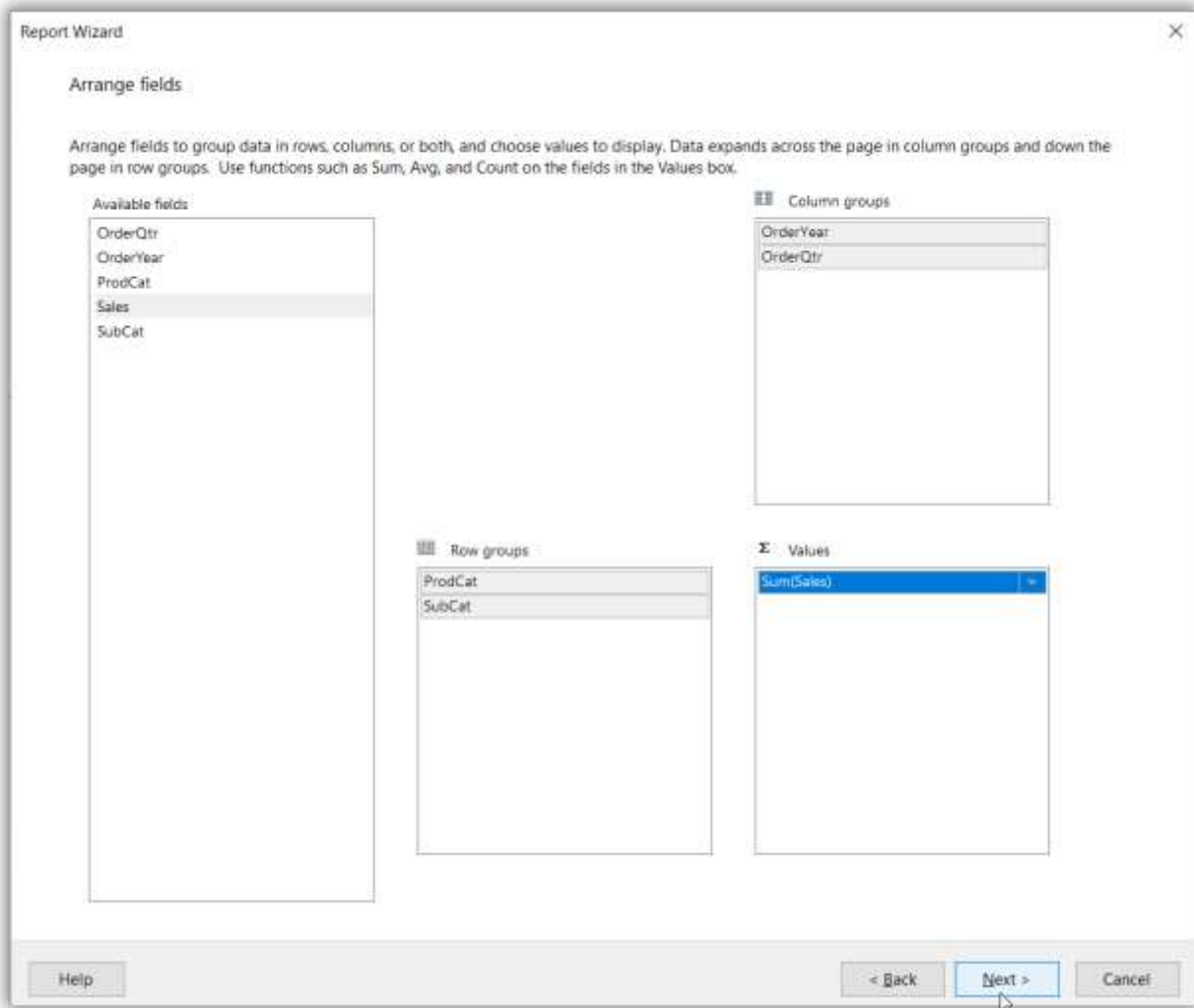


The image shows the 'Report Wizard' dialog box, specifically the 'Dataset Properties' tab. The 'Choose the Dataset' section is active. The 'Name' field is set to 'DataSet1'. The 'Data source' dropdown is set to 'ProductSales', with a 'New...' button next to it. The 'Available datasets' list also shows 'ProductSales'. On the right, the 'Fields' table lists the following fields and their types:

Field Name	Type Name
OrderQtr	System.String
OrderYear	System.String
ProdCat	System.String
Sales	System.Double
SubCat	System.String

At the bottom of the dialog, there are 'Help', '< Back', 'Next >', and 'Cancel' buttons.

4. Drag the fields into Values, Row, and Column groups, and then click **Next**.



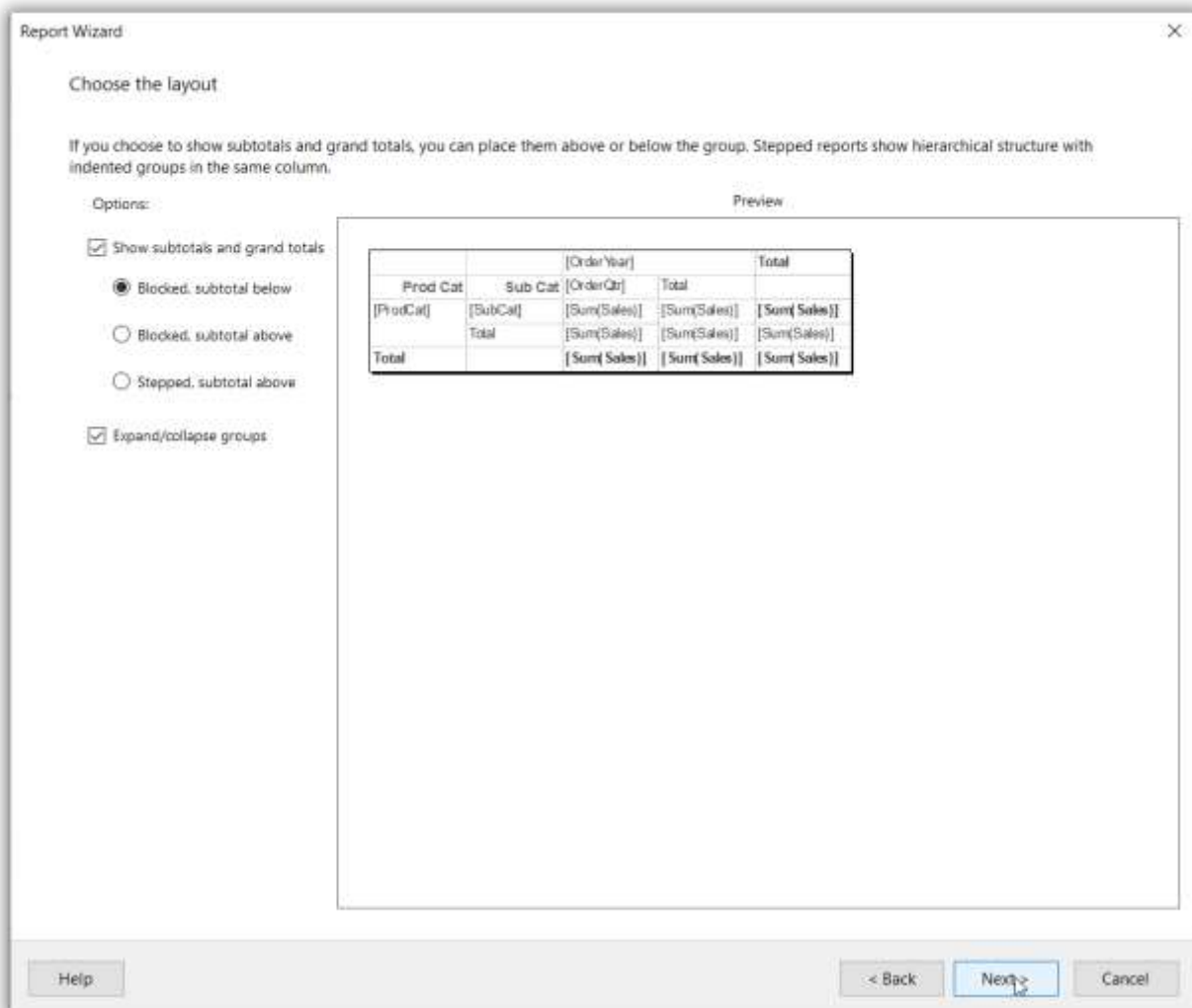
The image shows the 'Report Wizard' dialog box, specifically the 'Arrange fields' step. The dialog has a title bar with 'Report Wizard' and a close button. Below the title bar is the section 'Arrange fields' with a description: 'Arrange fields to group data in rows, columns, or both, and choose values to display. Data expands across the page in column groups and down the page in row groups. Use functions such as Sum, Avg, and Count on the fields in the Values box.'

The dialog is divided into four main sections:

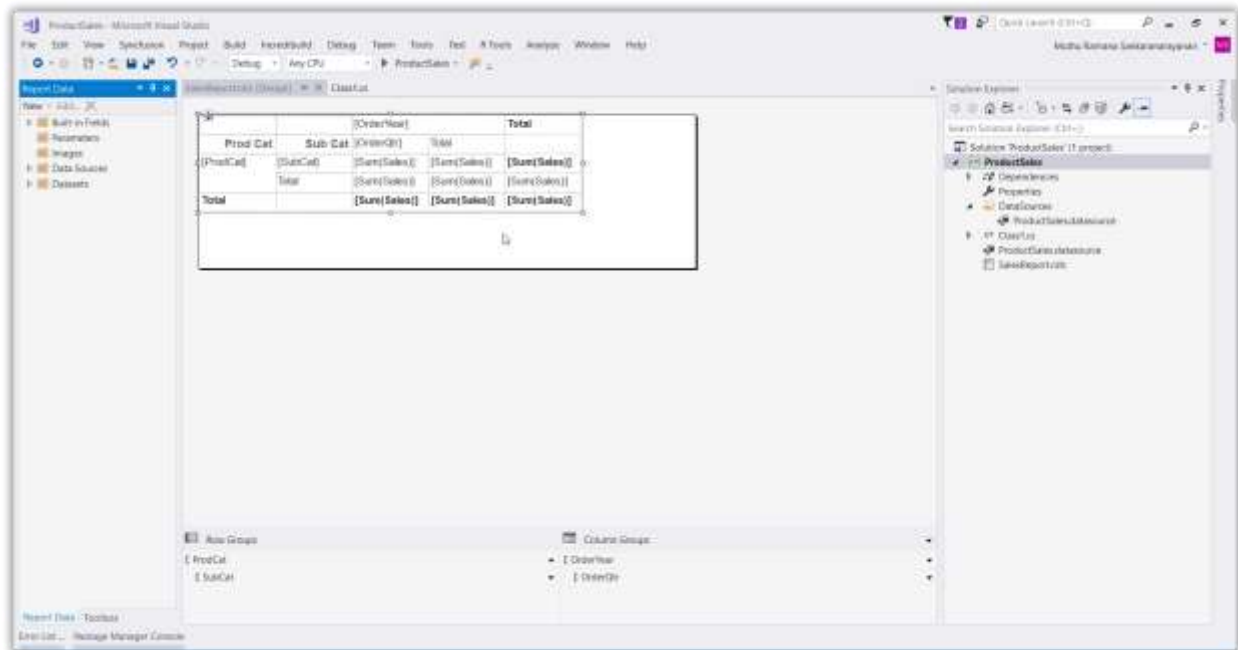
- Available fields:** A list box containing 'OrderQtr', 'OrderYear', 'ProdCat', 'Sales', and 'SubCat'. 'Sales' is currently selected.
- Column groups:** A list box containing 'OrderYear' and 'OrderQtr'.
- Row groups:** A list box containing 'ProdCat' and 'SubCat'.
- Values:** A list box containing 'Sum(Sales)'.

At the bottom of the dialog, there are four buttons: 'Help', '< Back', 'Next >', and 'Cancel'. The 'Next >' button is highlighted with a mouse cursor.

5. Choose the table layout and click **Next**.
6. Select table style and click **Finish**.



Now, the RDLC report is displayed in the Visual Studio as follows.



Adding data visualization scripts

To render the report with data visualization components such as chart, gauge and map items, must add scripts of the visualization element. The following table shows the script reference that need to be added in Report Viewer page for data visualization elements.

Visualization Item | Script File

Gauge | ej2-base.min.js, ej2-data.min.js, ej2-pdf-export.min.js, ej2-svg-base.min.js, ej2-lineargauge.min.js and ej2-circulargauge.min.js |

Map | ej2-maps.min.js

Chart | ej.chart.min.js

To render the chart report item, add chart control script `ej.chart.min.js` before the `bold.report-viewer.min.js` reference in `\Views\Shared_Layout.cshtml` page as in following code sample.

```
`html
<link href="~/Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="~/Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="~/Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<!--Used to render the chart item. Add this script, only if your report contains the chart report item.-->
<script src="~/Scripts/bold-reports/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="~/Scripts/bold-reports/bold.report-viewer.min.js"></script>
`
```

The following code can be used to render the chart, gauge and map report items in Report Viewer.

```
`html
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.reports.all.min.css"
rel="stylesheet" />
<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
<!--Render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
lineargauge.min.js"></script>
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-
circulargauge.min.js"></script>
<!--Render the map item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
<!--Render the chart item. Add this script only if your report contains the map report item.-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
`
```

How to change the exporting document file name based on parameter

Find the following steps to change the file based on parameter values in Report.

1. Create the file exporting file name using the parameters in **onRenderingComplete** event and store it in local variable.

```
`html
function onRenderingComplete(event) {
var parameters = event.reportParameters;
if(parameters){
```

```

for (var i = 0; i < parameters.length; i++) {
if(parameters[i].Name == "Department"){
this.exportFileName = "Sales for " + parameters[i].Value;
}
}
,

```

2. Use the file Name property with export, click event to change the file using the value stored in local variable used for having the file using the parameters.

```

`html
function onExportItemClick(event) {
event.fileName = this.exportFileName ;
}
,

```

How to change the data source dynamically

You have to use the `reportOption.ReportModel.DataSourceCredentials` available with the `OnInitReportOptions` method to dynamically change the data source in the web API controller. The following code sample shows how to change the connection string of the `<database>` data source in the report.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
DataSourceCredentials dataSourceCredentials = new DataSourceCredentials();
string connectionString = "Data Source = <instancename>; Initial Catalog = <database>; User ID =
'<username>'; Password = '<password>'";
//You have to provide the shared data source name used with the report or the data source name
available with the report.
dataSourceCredentials.Name = "<database>";
dataSourceCredentials.ConnectionString = connectionString;
reportOption.ReportModel.DataSourceCredentials = new List<DataSourceCredentials> {
dataSourceCredentials };
}
,

```

How to disable the vertical scrollbar in parameter panel

To disable the vertical scrollbar in parameter panel, set the `enableparameterblockscroller` property to false.

Example

```
`js
<div id="report viewer"></div>
<script>
$("#report viewer").boldReportViewer(
{
  enableParameterBlockScroller: false
});
</script>
`
```

How to generate the RDL and RDLC reports programmatically in the report viewer

The Bold Reports reporting library allows you to generate and preview RDL and RDLC reports programmatically in the report viewer. The `ReportDefinition` class is defined as a report object model. In the report definition instance, you can create, add, and modify the report properties, report sections such as header and footer, and report items.

The following steps illustrates how to create a basic report object model:

Initialize a report definition

The following code snippet guides you in initializing the report definition.

```
`csharp
ReportDefinition CreateReport()
{
  ReportDefinition report = new ReportDefinition();
  report.ReportSections = new ReportSections();
  var reportSection = new ReportSection();
  report.ReportSections.Add(reportSection);
  reportSection.Width = new BoldReports.RDL.DOM.Size("6in");
  report.ReportUnitType = "Inch";
  report.RDLType = RDLType.RDL2010;
  return report;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a Data source for report

```
BoldReports.RDL.DOM.Style AddStyle()
{
    BoldReports.RDL.DOM.Style style = new BoldReports.RDL.DOM.Style();
    style.Border = new BoldReports.RDL.DOM.Border();
    style.Border.Width = new BoldReports.RDL.DOM.Size("1pt");
    style.Border.Style = "Solid";
    style.Border.Color = "Black";
    return style;
}
```

Create a Data source for report

The following code snippet guides you in creating a **Datasource** for the report and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
    ...
    ...
    ...
    reportSection.Page = new BoldReports.RDL.DOM.Page();
    reportSection.Page.Style = AddStyle();
    reportSection.Page.PageHeight = "4in";
    reportSection.Page.PageWidth = "6in";
    var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
    <database>");
    report.DataSources = new DataSources();
    report.DataSources.Add(newDataSource);
    ...
    ...
    ...
}

//If you are using RDLC report then you can pass any string value as connection string
BoldReports.RDL.DOM.DataSource CreateDataSource(string name, string dataProvider, string
connectionString)
```


How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a Dataset for the report

```
{
var dataSource = new BoldReports.RDL.DOM.DataSource();
dataSource.Name = name;
dataSource.SecurityType = BoldReports.RDL.DOM.SecurityType.Integrated;
dataSource.ConnectionProperties = new BoldReports.RDL.DOM.ConnectionProperties();
dataSource.ConnectionProperties.DataProvider = dataProvider;
dataSource.ConnectionProperties.ConnectionString = connectionString;
dataSource.ConnectionProperties.IntegratedSecurity = true;
return dataSource;
}
`
```

Create a Dataset for the report

The following code snippet guides you in creating a **Dataset** for the report and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Page.PageWidth = "6in";
var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
<database>");
report.DataSources = new DataSources();
report.DataSources.Add(newDataSource);
var newDataSet = CreateDataSet("DataSource1", "DataSet1");
report.DataSets = new DataSets();
report.DataSets.Add(newDataSet);
...
...
...
}
BoldReports.RDL.DOM.DataSet CreateDataSet(string dataSourceName, string dataSetName)
{
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a Dataset for the report

```
var dataSet = new BoldReports.RDL.DOM.DataSet();
dataSet.Name = dataSetName;
dataSet.Query = new Query();
dataSet.Query.DataSourceName = dataSourceName;
dataSet.Query.CommandText = "SELECT
HumanResources.Department.DepartmentID,HumanResources.Department.Name,HumanResources.De
partment.GroupName,HumanResources.Department.ModifiedDate FROM
HumanResources.Department";
dataSet.Query.QueryDesignerState = new QueryDesignerState();
var table = CreateTable();
dataSet.Query.QueryDesignerState.Tables = new List<Table>();
dataSet.Query.QueryDesignerState.Tables.Add(table);
dataSet.Fields = new Fields();
dataSet.Fields.Add(CreateField("DepartmentID", "System.Int16"));
dataSet.Fields.Add(CreateField("Name", "System.String"));
dataSet.Fields.Add(CreateField("GroupName", "System.String"));
dataSet.Fields.Add(CreateField("ModifiedDate", "System.DateTime"));
return dataSet;
}

BoldReports.RDL.DOM.Table CreateTable()
{
var table = new Table();
table.Schema = "HumanResources";
table.Name = "Department";
table.Columns = new List<Column>();
table.Columns.Add(CreateColumn("DepartmentID"));
table.Columns.Add(CreateColumn("Name"));
table.Columns.Add(CreateColumn("GroupName"));
table.Columns.Add(CreateColumn("ModifiedDate"));
return table;
}

BoldReports.RDL.DOM.Column CreateColumn(string columnName)
{
var column = new Column();
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a report page header

```
column.Name = columnName;
return column;
}

BoldReports.RDL.DOM.Field CreateField(string fieldName, string type)
{
    var field = new Field();
    field.Name = fieldName;
    field.TypeName = type;
    field.DataField = fieldName;
    return field;
}
`
```

Create a report page header

The following code snippet guides you in creating a **PageHeader** report section and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
    ...
    ...
    ...
    reportSection.Page = new BoldReports.RDL.DOM.Page();
    reportSection.Page.Style = AddStyle();
    reportSection.Page.PageHeight = "4in";
    reportSection.Page.PageWidth = "6in";
    reportSection.Page.PageHeader = CreateHeader();
    ...
    ...
    ...
}

PageHeader CreateHeader()
{
    PageHeader pageHeader = new PageHeader();
    pageHeader.Height = new BoldReports.RDL.DOM.Size("0.59167in");
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a report page footer

```
pageHeader.Style = AddStyle();
pageHeader.PrintOnFirstPage = true;
pageHeader.PrintOnLastPage = true;
pageHeader.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
return pageHeader;
}
`
```

Create a report page footer

The following code snippet guides you in creating a **PageFooter** report section and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Page.PageHeader = CreateFooter();
...
...
...
}
PageFooter CreateFooter()
{
PageFooter pageFooter = new PageFooter();
pageFooter.Style = AddStyle();
pageFooter.Height = new BoldReports.RDL.DOM.Size("0.59167in");
pageFooter.ReportItems = new ReportItems();
pageFooter.PrintOnFirstPage = true;
pageFooter.PrintOnLastPage = true;
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
return pageFooter;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
}  
,
```

Initialize a report body section

- Initialize a report body object and set the values to their properties like height, styles, and report items as shown in the following code snippet.

```
`csharp  
ReportDefinition CreateReport()  
{  
...  
...  
...  
reportSection.Body = CreateBody();  
...  
...  
...  
}  
Body CreateBody()  
{  
var body = new Body();  
body.Height = new BoldReports.RDL.DOM.Size("2.03333in");  
body.Style = AddStyle();  
body.ReportItems = new ReportItems();  
var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");  
body.ReportItems.Add(textBox);  
return body;  
}  
,
```

- Using the following code snippets, you can create a **Textbox** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

```
`csharp  
PageHeader CreateHeader()  
{
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
...
...
...
pageHeader.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
...
...
...
}
PageFooter CreateFooter()
{
...
...
...
pageFooter.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
...
...
...
}
Body CreateBody()
{
...
...
...
body.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
body.ReportItems.Add(textBox);
...
...
...
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
}
```

```
BoldReports.RDL.DOM.TextBox CreateTextBox(string name, string text, string left, string top, string width, string height)
```

```
{
```

```
var textBox = new BoldReports.RDL.DOM.TextBox();
```

```
textBox.Name = name;
```

```
textBox.Height = new BoldReports.RDL.DOM.Size(height);
```

```
textBox.Width = new BoldReports.RDL.DOM.Size(width);
```

```
textBox.Left = new BoldReports.RDL.DOM.Size(left);
```

```
textBox.Top = new BoldReports.RDL.DOM.Size(top);
```

```
textBox.Style = new BoldReports.RDL.DOM.Style();
```

```
textBox.Style.TextAlign = "Center";
```

```
textBox.Style.VerticalAlign = "Top";
```

```
textBox.Paragraphs = new Paragraphs();
```

```
BoldReports.RDL.DOM.Paragraph paragraph = new BoldReports.RDL.DOM.Paragraph();
```

```
TextRuns runs = new TextRuns();
```

```
TextRun run = new TextRun();
```

```
run.Style = new BoldReports.RDL.DOM.Style();
```

```
run.Style.FontStyle = "Default";
```

```
run.Style.TextAlign = "Center";
```

```
run.Style.FontFamily = "Arial";
```

```
run.Style.FontSize = new BoldReports.RDL.DOM.Size("10pt");
```

```
run.Value = text;
```

```
runs.Add(run);
```

```
paragraph.Style = new BoldReports.RDL.DOM.Style();
```

```
paragraph.Style.VerticalAlign = "Top";
```

```
paragraph.Style.TextAlign = "Center";
```

```
paragraph.TextRuns = runs;
```

```
textBox.Paragraphs.Add(paragraph);
```

```
return textBox;
```

```
}
```

```
,
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

Create a Table for the report

- Using the following code snippets, you can create a **Table** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

```
`csharp
```

```
BoldReports.RDL.DOM.Tablix CreateTablix(string name, string text, string left, string top, string width, string height)
```

```
{
```

```
var tableItem = new BoldReports.RDL.DOM.Tablix();
```

```
tableItem.Name = name;
```

```
tableItem.Height = new BoldReports.RDL.DOM.Size(height);
```

```
tableItem.Width = new BoldReports.RDL.DOM.Size(width);
```

```
tableItem.Left = new BoldReports.RDL.DOM.Size(left);
```

```
tableItem.Top = new BoldReports.RDL.DOM.Size(top);
```

```
tableItem.Style = new BoldReports.RDL.DOM.Style();
```

```
tableItem.Style.Border = new BoldReports.RDL.DOM.Border();
```

```
tableItem.Style.Border.Style = "Solid";
```

```
tableItem.DataSetName = "DataSet1";
```

```
tableItem.TablixBody = new TablixBody();
```

```
tableItem.TablixBody.TablixColumns = new TablixColumns();
```

```
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
```

```
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
```

```
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
```

```
tableItem.TablixBody.TablixRows = new TablixRows();
```

```
var rowOne = new List<TablixRowValues>();
```

```
rowOne.Add(new TablixRowValues { TextBoxName = "TextBox1", TextBoxValue = "Department ID" });
```

```
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox3", TextBoxValue = "Name" });
```

```
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox5", TextBoxValue = "Group Name" });
```

```
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowOne));
```

```
var rowTwo = new List<TablixRowValues>();
```

```
rowTwo.Add(new TablixRowValues { TextBoxName = "DepartmentID", TextBoxValue =  
"=Fields!DepartmentID.Value" });
```

```
rowTwo.Add(new TablixRowValues { TextBoxName = "Name", TextBoxValue = "=Fields!Name.Value" });
```


How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
rowTwo.Add(new TablixRowValues { TextBoxName = "GroupName", TextBoxValue =
"=Fields!GroupName.Value" });
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowTwo));
tableItem.TablixColumnHierarchy = new TablixColumnHierarchy();
tableItem.TablixColumnHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixRowHierarchy = new TablixRowHierarchy();
tableItem.TablixRowHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixRowHierarchy.TablixMembers.Add(new TablixMember() { KeepWithGroup =
KeepWithGroup.After });
tableItem.TablixRowHierarchy.TablixMembers.Add(CreateTablixMember("Details"));
return tableItem;
}
```

BoldReports.RDL.DOM.TablixColumn CreateTablixColumn()

```
{
var tablixColumn = new TablixColumn();
tablixColumn.Width = "1in";
return tablixColumn;
}
```

BoldReports.RDL.DOM.TablixRow CreateTablixRow(List<TablixRowValues> values)

```
{
var tablixRow = new TablixRow();
tablixRow.Height = "0.25in";
tablixRow.TablixCells = new TablixCells();
for(int i = 0; i < values.Count; i++)
{
tablixRow.TablixCells.Add(CreateTablixCell(values[i].TextBoxName, values[i].TextBoxValue));
}
return tablixRow;
}
```

BoldReports.RDL.DOM.TablixCell CreateTablixCell(string textBoxName, string textBoxValue)

```
{
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
var tablixCell = new TablixCell();
tablixCell.CellContents = new CellContents();
tablixCell.CellContents.ReportItem = CreateTextBox(textBoxName, textBoxValue);
return tablixCell;
}

BoldReports.RDL.DOM.TablixMember CreateTablixMember(string groupName)
{
    var tablixMember = new TablixMember();
    tablixMember.Group = new Group();
    tablixMember.Group.Name = groupName;
    return tablixMember;
}

internal class TablixRowValues
{
    public string TextBoxName { get; set; }
    public string TextBoxValue { get; set; }
}
`
```

Create a Chart for the report

- Using the following code snippets, you can create a **Chart** report item and assign the values for their properties like name, styles, and dimension values.

```
`csharp
BoldReports.RDL.DOM.Chart CreateChart(string name, string left, string top, string width, string height)
{
    var chartItem = new Chart();
    chartItem.Name = name;
    chartItem.Height = new BoldReports.RDL.DOM.Size("10in");
    chartItem.Width = new BoldReports.RDL.DOM.Size("10in");
    chartItem.Left = new BoldReports.RDL.DOM.Size("5in");
    chartItem.Top = new BoldReports.RDL.DOM.Size("5in");
    chartItem.DataSetName = "DataSet1";
    chartItem.Style = new BoldReports.RDL.DOM.Style();
    chartItem.Bookmark= "=First(Fields!Name.Value, \"DataSet1\")";
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
chartItem.ChartCategoryHierarchy = new ChartCategoryHierarchy();
chartItem.ChartCategoryHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartCategoryHierarchy.ChartMembers.Add(ChartCategoryMember());
chartItem.ChartSeriesHierarchy = new ChartSeriesHierarchy();
chartItem.ChartSeriesHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartSeriesHierarchy.ChartMembers.Add(ChartSeriesMember());
chartItem.ChartData = new ChartData();
chartItem.ChartData.ChartDerivedSeriesCollection = new ChartDerivedSeriesCollection();
chartItem.ChartData.ChartSeriesCollection = CreateChartSeriesCollection();
chartItem.ChartLegends = new ChartLegends();
chartItem.ChartAreas = new ChartAreas();
chartItem.ChartAreas.Add(CreateChartArea());
chartItem.ChartTitles = new ChartTitles();
chartItem.Palette = "Pacific";
chartItem.ChartBorderSkin = new ChartBorderSkin();
chartItem.ChartNoDataMessage = new ChartNoDataMessage();
chartItem.ChartNoDataMessage.Caption = "No Data Available";
chartItem.ChartNoDataMessage.Name = "NoDataMessage";
return chartItem;
}

BoldReports.RDL.DOM.ChartMember ChartCategoryMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = new Group();
    ChartMember.Group = CreateChartGroup();
    ChartMember.Label="=Fields!DepartmentID.Value";
    ChartMember.SortExpressions = SortExpressions();
    return ChartMember;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
BoldReports.RDL.DOM.ChartMember ChartSeriesMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = null;
    ChartMember.Label = "Department ID";
    ChartMember.SortExpressions = new SortExpressions();
    return ChartMember;
}

BoldReports.RDL.DOM.SortExpressions SortExpressions()
{
    var SortExpressions = new SortExpressions();
    var SortExpression = new SortExpression();
    SortExpression.Direction = SortDirection.Ascending;
    SortExpression.Value = "=Fields!DepartmentID.Value";
    SortExpressions.Add(SortExpression);
    return SortExpressions;
}

BoldReports.RDL.DOM.Group CreateChartGroup()
{
    var ChartGroup = new Group();
    ChartGroup.DataElementName = null;
    ChartGroup.DataElementOutput = DataElementOutputs.Auto;
    ChartGroup.DocumentMapLabel = null;
    ChartGroup.DocumentMapLabelLocID = null;
    ChartGroup.DomainScope = null;
    ChartGroup.Filters = new Filters();
    ChartGroup.GroupExpressions = CreateGroupExpressions();
    ChartGroup.Name = "Chart2_CategoryGroup";
    ChartGroup.PageBreak = null;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
ChartGroup.PageName = null;
ChartGroup.Parent = null;
ChartGroup.Variables = new Variables();
return ChartGroup;
}

BoldReports.RDL.DOM.GroupExpressions CreateGroupExpressions()
{
    var GroupExpressions = new GroupExpressions();
    var GroupExpression = new GroupExpression();
    GroupExpression.Value = "=Fields!DepartmentID.Value";
    GroupExpressions.Add(GroupExpression);
    return GroupExpressions;
}

BoldReports.RDL.DOM.ChartSeriesCollection CreateChartSeriesCollection()
{
    var ChartSeriesCollection = new ChartSeriesCollection();
    ChartSeriesCollection.Add(CreateChartSeries());
    return ChartSeriesCollection;
}

BoldReports.RDL.DOM.ChartSeries CreateChartSeries()
{
    var ChartSeries = new ChartSeries();
    ChartSeries.CategoryAxisName = "Primary";
    ChartSeries.ChartAreaName = null;
    ChartSeries.ChartDataLabel = null;
    ChartSeries.ChartDataPoints = new ChartDataPoints();
    ChartSeries.ChartDataPoints.Add(ChartDataPoint());
    ChartSeries.ChartEmptyPoints = new ChartEmptyPoints();
    ChartSeries.ChartEmptyPoints.ChartDataLabel = new ChartDataLabel();
    ChartSeries.ChartEmptyPoints.ChartDataLabel.Style = new Style();
    ChartSeries.ChartEmptyPoints.ChartMarker = new ChartMarker();
    ChartSeries.ChartEmptyPoints.ChartMarker.Style = new Style();
    ChartSeries.ChartSmartLabel = new ChartSmartLabel();
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
ChartSeries.Name = "DepartmentID";
ChartSeries.Type = VisualizationType.Column;
ChartSeries.Subtype = VisualizationSubType.Plain;
ChartSeries.Style = new Style();
return ChartSeries;
}

BoldReports.RDL.DOM.ChartDataPoint ChartDataPoint()
{
var ChartDataPoint = new ChartDataPoint();
ChartDataPoint.ActionInfo = null;
ChartDataPoint.ChartDataLabel = null;
ChartDataPoint.ChartDataLabel = ChartDataLabel();
ChartDataPoint.ChartDataPointValues = new ChartDataPointValues();
ChartDataPoint.ChartDataPointValues.Y = "=Sum(Fields!DepartmentID.Value)";
ChartDataPoint.ChartItemInLegend = null;
ChartDataPoint.ChartMarker = new ChartMarker();
ChartDataPoint.ChartMarker.Size = null;
ChartDataPoint.ChartMarker.Type = null;
ChartDataPoint.ChartMarker.Style= chartStyle("Transparent", "Arial");
ChartDataPoint.CustomProperties = new CustomProperties();
ChartDataPoint.DataElementName = null;
ChartDataPoint.DataElementOutput = DataElementOutputs.Output;
ChartDataPoint.ToolTip = null;
ChartDataPoint.Style= chartStyle("Transparent", "Arial");
return ChartDataPoint;
}

BoldReports.RDL.DOM.ChartDataLabel ChartDataLabel()
{
var ChartDataLabel = new ChartDataLabel();
ChartDataLabel.ActionInfo = null;
ChartDataLabel.Label = null;
ChartDataLabel.Position = null;
ChartDataLabel.Rotation = "0";
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
ChartDataLabel.Style = new Style();
ChartDataLabel.Style = chartStyle("Transparent", "Arial");
ChartDataLabel.ToolTip = null;
return ChartDataLabel;
}

BoldReports.RDL.DOM.Style chartStyle(string BackgroundColor, string FontFamily)
{
    var style = new Style();
    style.BackgroundColor = BackgroundColor;
    style.FontFamily = FontFamily;
    return style;
}

BoldReports.RDL.DOM.ChartArea CreateChartArea()
{
    var chartArea = new ChartArea();
    chartArea.Style = new Style();
    chartArea.Style = chartStyle("Transparent", "Arial");
    chartArea.AlignOrientation = AlignOrientation.None;
    chartArea.AlignWithChartArea = null;
    chartArea.ChartAlignType = null;
    chartArea.ChartElementPosition = null;
    chartArea.ChartInnerPlotPosition = null;
    chartArea.ChartThreeDProperties = null;
    chartArea.Hidden = false;
    chartArea.EquallySizedAxesFont = false;
    chartArea.Name = "Default";
    chartArea.ChartCategoryAxes = new ChartCategoryAxes();
    chartArea.ChartCategoryAxes.Add(createChartAxis("Primary"));
    chartArea.ChartCategoryAxes.Add(createChartAxis("Secondary"));
    chartArea.ChartValueAxes = new ChartValueAxes();
    chartArea.ChartValueAxes.Add(createChartAxis("Primary"));
    chartArea.ChartValueAxes.Add(createChartAxis("Secondary"));
    return chartArea;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Render** the generated report in ReportViewer

```
}  
BoldReports.RDL.DOM.ChartAxis createChartAxis(string Name)  
{  
    var ChartAxis = new ChartAxis();  
    ChartAxis.Style = new Style();  
    ChartAxis.AllowLabelRotation = AllowLabelRotation.None;  
    ChartAxis.Angle = "0";  
    ChartAxis.Arrows = Arrows.None;  
    ChartAxis.ChartAxisScaleBreak = new ChartAxisScaleBreak();  
    ChartAxis.ChartAxisTitle = new ChartAxisTitle();  
    ChartAxis.ChartMajorGridLines = new ChartMajorGridLines();  
    ChartAxis.ChartMajorTickMarks = new ChartMajorTickMarks();  
    ChartAxis.ChartMinorGridLines = new ChartMinorGridLines();  
    ChartAxis.ChartMinorTickMarks = new ChartMinorTickMarks();  
    ChartAxis.ChartStripLines = new ChartStripLines();  
    ChartAxis.CrossAt = "NaN";  
    ChartAxis.CustomProperties = new CustomProperties();  
    ChartAxis.LineStyle = LineStyle.Solid;  
    ChartAxis.Name = Name;  
    return ChartAxis;  
}  
`
```

[Render the generated report in ReportViewer](#)

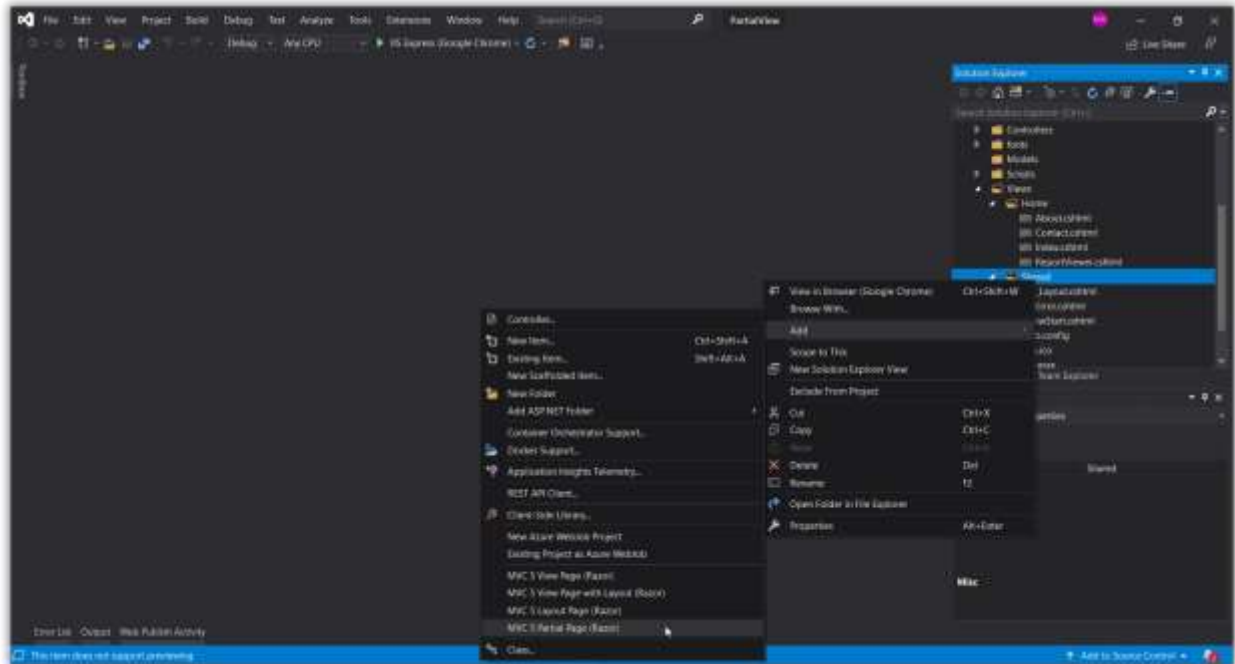
You can render the report using the ReportViewer, after the report definition is created. The following code snippet illustrates how to render the report using the ReportViewer by report definition.

```
`csharp  
[NonAction]  
public void OnInitReportOptions(ReportViewerOptions reportOption)  
{  
    var report = CreateReport();  
    reportOption.ReportModel.ReportDefinition = reportDefinition;  
}  
`
```


How to use Report Viewer in partial view

This section explains the steps required to use the Report Viewer in partial view in ASP.NET MVC reporting application.

1. To create your first ASP.NET MVC reporting application, refer to this [Getting-started](#) section.
2. Add a partial view in your project.



3. Add the Report Viewer component in the partial view page and it was added using the following code sample.

```
`js
@Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
)
@Html.Bold().ScriptManager()
`
```

4. You need to add a controller action that returns the partial view.

```
`csharp
public ActionResult ReportViewer(string reportName)
{
    ViewBag.Report = reportName;
}
```

```
return PartialView("ReportViewerPartial");
}
```

5. Set the `ReportPath` API value for the Report Viewer, based on the value sent from the main page.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/SSRSReports")
.ReportPath("~/Resources/" + ViewBag.Report + ".rdl")
)
```

6. You can now load the Bold Report Viewer in partial view with the ajax load functionality.

```
`js
function openReport() {
$('#partialView').load('@Url.Action("ReportViewer", "Home", new {reportName = "sales-order-detail"})');
}
```

The `@Url.Action()` should be in the format `@Url.Action("ActionName","ControllerName", new { value = "Value" })`

7. When you are closing the Report Viewer you need to destroy it to avoid creation of multiple instances of Report Viewer component. You can use the following code sample to destroy the Report Viewer in your application.

```
`js
function closeReport() {
$('#viewer').data('boldReportViewer').destroy();
}
```

How to pass multiple values using custom data

Pass the multiple data values as JSON in `ajaxBeforeLoad` event using the 'data' property, which needs to be serialized in the server side API using known class type and `JsonConvert.DeserializeObject`.

1. Map the [ajaxBeforeLoad](#) event with `onAjaxRequest` function in the script to pass custom data to the server.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.AjaxBeforeLoad("onAjaxRequest")
)
<script>
function onAjaxRequest(args) {
//Passing custom data to server
}
</script>
`
```

2. You need to pass the multiple custom data values as JSON and use the `data` property to send custom data to the server in the Ajax request.

```
`js
@(Html.Bold().ReportViewer("viewer")
.ReportServiceUrl("/api/ReportViewer")
.ReportPath("~/Resources/sales-order-detail.rdl")
.AjaxBeforeLoad("onAjaxRequest")
)
<script>
function onAjaxRequest(args){
//Passing custom data to server
var jsonData = {
customerID: "CI0021",
productID: "PO0022"
}
args.data = jsonData;
}
</script>
`
```

3. The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates deserialization of the JSON string and change the data source connection strings based on Customer ID and Product ID in the `OnInitReportOptions` method.

```
`csharp
public SampleData customDatas = new SampleData();
public class SampleData
{
    public string customerID { get; set; }
    public string productID { get; set; }
}
[HttpPost]
public object PostReportAction([FromBody]Dictionary<string, object> jsonResult)
{
    if (jsonResult != null)
    {
        if (jsonResult.ContainsKey("customData"))
        {
            //Get client side custom data and store in local variable.
            customDatas = JsonConvert.DeserializeObject<SampleData>(jsonResult["customData"].ToString());
        }
    }
    return ReportHelper.ProcessReport(jsonResult, this, this._cache);
}
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (customDatas != null)
    {
        if (customDatas.customerID == "CI0021" && customDatas.productID == "PO0022") {
            //If you are changing the connection string based on customer id then could you please change the
            connection string as below.

            //reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));
        }
    }
}
```

```
reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>"));
}
}
}
`
```

How to load the report from the database

You must load the report using the **Stream** option available with **reportOption.ReportModel**. To load the report from the database, please follow these steps:

From byte array

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
byte[] bytes = ""; // provide your byte array report data
MemoryStream reportStream = new MemoryStream(bytes);
reportOption.ReportModel.Stream = reportStream;
}
`
```

From base64String

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
string base64String = ""; // provide your base64 report data
byte[] bytes = System.Convert.FromBase64String(base64String);
MemoryStream reportStream = new MemoryStream(bytes);
reportOption.ReportModel.Stream = reportStream;
}
`
```

From string

```
`csharp
[NonAction]
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string reportData = ""; // provide your database report data
    byte[] bytes = System.Text.Encoding.ASCII.GetBytes(reportFile);
    MemoryStream reportStream = new MemoryStream(bytes);
    reportOption.ReportModel.Stream = reportStream;
}
`
```

How to clear cache when closing the Report Viewer

By using the `clearReportCache` and `destroy` method, you can clear the server side cache. You have to use this method when closing report viewer and switching to another report within your application.

CSHTML

```
`js
<div id="reportviewer"></div>
<script>
var isSubmit = true;
$(document.body).bind('submit', $.proxy(this.formSubmit, this));
function formSubmit(args)
{
    isSubmit = false;
}
window.onbeforeunload = function () {
    if (isSubmit) {
        var reportviewerObj = $("#reportviewer").data("boldReportViewer");
        reportviewerObj.clearReportCache();
        reportviewerObj.destroy();
    }
    isSubmit = true;
};
</script>
`
```

Web API

In the `PostReportAction` method, you have to collect the `GC` with `ClearCache` as shown in the following code sample.

```
`csharp
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
{
    bool isclearcache = false;

    if (jsonArray != null && jsonArray.ContainsKey("reportAction") && jsonArray["reportAction"].ToString()
    == "ClearCache")
    {
        isclearcache = true;
    }

    var reportresult = ReportHelper.ProcessReport(jsonArray, this, this._cache);
    if (isclearcache)
    {
        GC.Collect(); isclearcache = false;
    }

    return reportresult;
}
`
```

Migrate to Report Viewer v2.0 component

This section provides simple step-by-step instructions to update your existing Report Viewer application to our latest modern v2.0 scripts, styles and components.

1. Open the `\Views\Shared_Layout.cshtml` page that contains Report Viewer and its scripts.
2. Remove the following older scripts and CSS references in your `_Layout.cshtml` page.
 - o `bold.reports.all.min.css`
 - o `ej2-base.min.js`
 - o `ej2-data.min.js`
 - o `ej2-pdf-export.min.js`
 - o `ej2-svg-base.min.js`
 - o `ej2-lineargauge.min.js`
 - o `ej2-circulargauge.min.js`
 - o `ej2-maps.min.js`
 - o `ej.chart.min.js`
 - o `bold.reports.common.min.js`
 - o `bold.reports.widgets.min.js`
 - o `bold.report-viewer.min.js`
3. Add the following v2.0 scripts and CSS references in your `_Layout.cshtml` page.

```
`js
```

```
@Styles.Render("~/Content/bold-reports/v2.0/tailwind-light/bold.report-viewer.min.css")
@Scripts.Render("~/Scripts/bold-reports/v2.0/common/bold.reports.common.min.js")
@Scripts.Render("~/Scripts/bold-reports/v2.0/common/bold.reports.widgets.min.js")
@Scripts.Render("~/Scripts/bold-reports/v2.0/bold.report-viewer.min.js")
`
```

Bold Report Writer

Report Writer is a class library that enables the user to render reports defined in Microsoft's RDL format (2008 or 2008 R2) as PDF, Word, HTML, Excel or CSV documents.

The important features of MVC Report Writer are listed as follows:

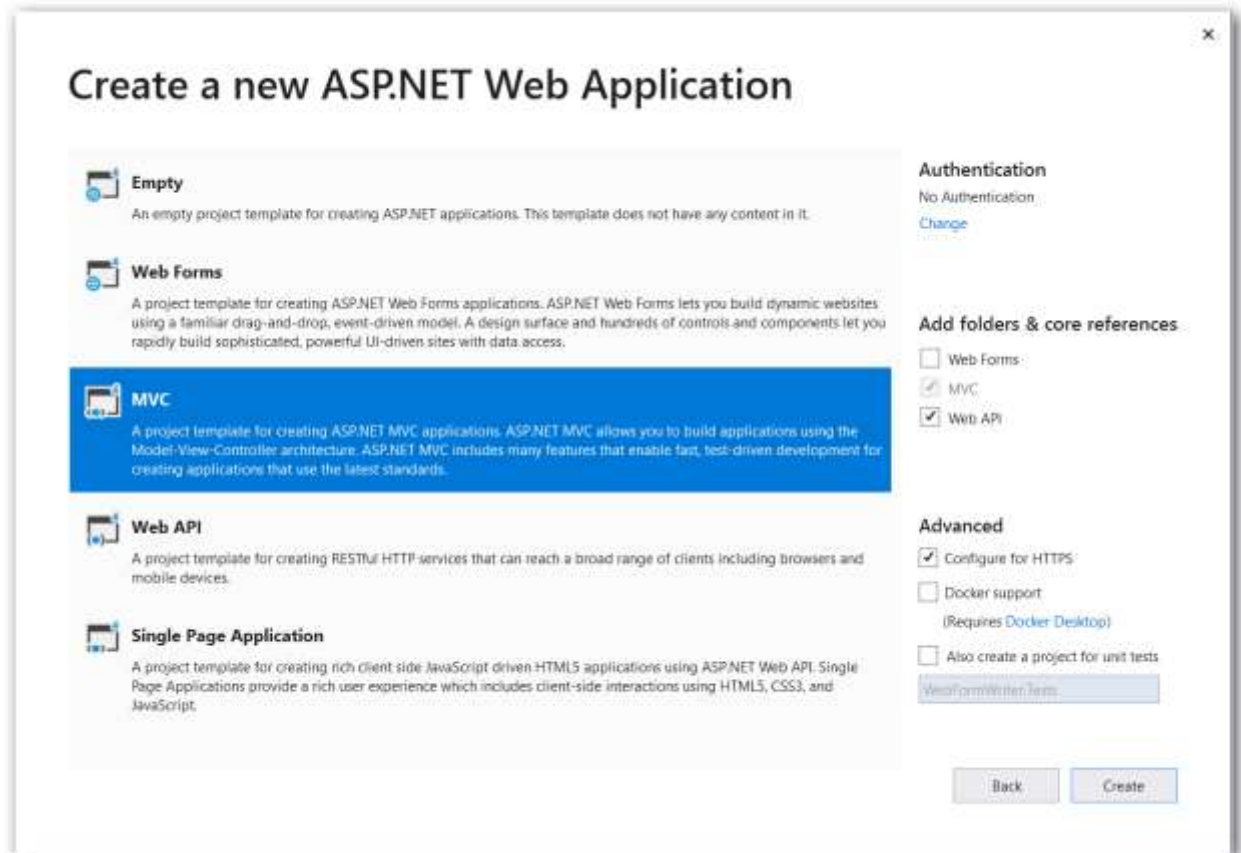
- RDL Specification - Supports RDL specification for the SQL Server 2008 and RDL specification for the SQL Server 2008 R2 only. List of available report definition formats: [msdn.microsoft.com/library/dd297486\(SQL.100\)](https://msdn.microsoft.com/library/dd297486(SQL.100)).
- Data sources - You can use advanced database servers Data Sources in Report Writer (SQL and Oracle).
- Charts - Show all basic types of charts that are available in Microsoft RDL reports.
- Tablix - Shows the summaries and simple tables.
- Gauge - Shows measurement values using the expression values.
- Textbox - Shows textbox data with expression support.
- Export - Export report as PDF, Word, Excel, HTML, and CSV.
- Report Parameter - Views the report based on the report parameter value.

Export SSRS RDL Report in Bold Reports ASP.NET MVC Report Writer

The Report Writer is a class library that is used to export the RDL report with popular file formats like PDF, Microsoft Word, Microsoft CSV, and Microsoft Excel without previewing the report on the webpage. This section describes how to export the RDL report to an ASP.NET MVC application using the Report Writer.

Create an ASP.NET MVC application

1. Start Visual Studio 2019 and click **Create new project**.
2. Choose **ASP.NET Web Application (.NET Framework)**, and then click **Next**.
3. Change the project name, and then click **Create**.
4. Choose the MVC and Web API, and then click **OK**.



List of dependency libraries

1. In the Solution Explorer tab, right-click the project or solution, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.Web** package and install this in your MVC application.

The following table provides details about the dependency packages and their usage.

Package | Purpose

Syncfusion.Pdf.AspNet | Exports the report to a PDF.

Syncfusion.DocIO.AspNet | Exports the report to a Word.

Syncfusion.XlsIO.AspNet | Exports the report to an Excel.

Syncfusion.Presentation.AspNet | Exports the report to a PowerPoint.

Newtonsoft.Json | Serializes and deserializes data for the Report Writer. It is a mandatory package for Report Writer, and the package version should be 10.0.1 or higher.

In this tutorial, the **sales-order-detail.rdl** report is used and it can be downloaded [here](#). You can get the reports from the Bold Reports installation location. For more information, refer to [samples and demos](#) section.

Server side Report Writer changes

1. Create a folder **Resources** in the **App_Data** folder in your application. Copy and paste the sample RDL reports into the **Resources** folder.
2. Open **HomeController.cs** and add the following using statement.

```
`csharp
using System.IO;
using BoldReports.Writer;
`
```

3. Add the **Export()** function to load the report as a stream. Refer to the following code snippet.

```
`csharp
[HttpPost]
public ActionResult Export(string writerFormat)
{
    // Here, we have loaded the sales-order-detail sample report from application the folder
    App_Data\Resources.
    FileStream reportStream = new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Resources\sales-order-
    detail.rdl"), FileMode.Open, FileAccess.Read);
    .....
}
```

4. Initialize the Report Writer instance with a report stream and set the specified export format and file name for the export document.

```
`csharp
public ActionResult Export(string writerFormat)
{
    .....
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    string fileName = null;
    WriterFormat format;
    string type = null;
    fileName = "sales-order-detail.pdf";
```

```

type = "pdf";
format = WriterFormat.PDF;
}
`

```

5. You can use the **Save** method in the Report Writer to generate the export document along with the information of the report stream, it will return the generated file as a Stream.

```

`csharp
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
`

```

6. To get the exported file stream, refer to the following complete code snippet.

```

`csharp
[HttpPost]
public ActionResult Export(string writerFormat)
{
// Here, we have loaded the sales-order-detail sample report from application the folder Resources.
FileStream reportStream = new
FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Resources\sales-order-
detail.rdl"), FileMode.Open, FileAccess.Read);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
string fileName = null;
WriterFormat format;
string type = null;
if (writerFormat == "PDF")
{
fileName = "sales-order-detail.pdf";
}
}

```

```
type = "pdf";
format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
fileName = "sales-order-detail.docx";
type = "docx";
format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
fileName = "sales-order-detail.csv";
type = "csv";
format = WriterFormat.CSV;
}
else if (writerFormat == "HTML")
{
fileName = "sales-order-detail.html";
type = "html";
format = WriterFormat.HTML;
}
else if (writerFormat == "PPT")
{
fileName = "sales-order-detail.ppt";
type = "ppt";
format = WriterFormat.PPT;
}
else
{
fileName = "sales-order-detail.xlsx";
type = "xlsx";
format = WriterFormat.Excel;
}
```

```

writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileName = fileName;
return fileStreamResult;
}

```

Client side changes

1. Use the following code snippet on the `Index.cshtml` home page to invoke the Web API from the client side.

```

`html
@{Html.BeginForm("Export", "Home", FormMethod.Post);
{
<div>
<input type="submit" value="Generate" style="width: 150px;" />
</div>
}
Html.EndForm();
}

```

2. You can add the export file types to the home page to choose the format you want to export to the Report Writer. Copy and paste the following code snippet into your application.

```

`html
@{Html.BeginForm("Export", "Home", FormMethod.Post);
{
<div class="Common">
<div class="tablediv">
<div id="description_Pane" style="text-align: justify;">
<h3>Description</h3>

```

```
<span>
Bold ReportWriter is a powerful control for exporting RDL and RDLC files into specified
format files.
</span>
</div>
<div class="rowdiv">
<div class="celldiv" style="padding:10px">
<label>
<strong> Save As :</strong>
</label>
<input id="rbtnPDF" type="radio" name="writerFormat" value="PDF" checked="checked" style="margin-
left: 15px" />
<label for="rbtnPDF" style="padding:0px 5px 0px 2px">
PDF
</label>
<input id="rbtnWord" type="radio" name="writerFormat" value="Word" style="margin-left: 15px" />
<label for="rbtnWord" style="padding:0px 5px 0px 2px">
Word
</label>
<input id="rbtnxls" type="radio" name="writerFormat" value="xls" style="margin-left: 15px" />
<label for="rbtnxls" style="padding:0px 5px 0px 2px">
Excel
</label>
<input id="rbtnCSV" type="radio" name="writerFormat" value="CSV" style="margin-left: 15px" />
<label for="rbtnCSV" style="padding:0px 25px 0px 2px ">
CSV
</label>
<input class="buttonStyle" type="submit" name="button" value="Generate" style="width:150px;" />
</div>
</div>
</div>
</div>
Html.EndForm();
```

```
}}
,
```

3. Now, run and export the report with the specified export format in your Report Writer application.

Congratulations! You have completed your first MVC Writer application! Click [here](#) to download the already created MVC Report Writer application.

Note: You can refer to our feature tour page for the [ASP.NET MVC Report Writer](#) to see its innovative features. Additionally, you can view our [ASP.NET MVC Report Writer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

Export RDLC Report

You can export the RDLC reports that exist on the local file system with custom business object data collection. The following steps demonstrates how to export a RDLC report using Report Writer.

This section requires an ASP.NET MVC Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

Bind data source in Web API controller

The following steps help you to configure the Web API to render the RDLC report with business object data collection.

1. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = new ProductList()
        {
            ProductName = "Baked Chicken and Cheese",
```

```
OrderId = "323B60",
Price = 55,
Category = "Non-Veg",
Ingredients = "grilled chicken, corn and olives.",
ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Delite",
    OrderId = "323B61",
    Price = 100,
    Category = "Non-Veg",
    Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
```


2. In this tutorial, the **Product List.rdlc** report is used, and it can be downloaded [here](#). Copy and paste the sample RDLC reports into the **App_Data/Resources** folder. For more information, see [Samples and demos](#).
3. Open the **HomeController.cs** file in your application and add the **Export()** function to load the report as stream. Refer to the following code snippet.

```
`csharp
public class HomeController : Controller
{
    [HttpPost]
    public ActionResult Export(string writerFormat)
    {
        // Here, we have loaded the Product List.rdlc sample report from application the Resources folder.
        FileStream reportStream = new
        FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Resources\Product
        List.rdlc"), FileMode.Open, FileAccess.Read);
        .....
    }
}
```

4. Initialize the Report Writer instance with created **reportStream** and Set the value of the **ReportProcessingMode** property value **ProcessingMode.Local** to provide the dataset collection for that RDLC report.

```
`csharp
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.ReportProcessingMode = ProcessingMode.Local;
`
```

5. Bind the business object data values collection by adding a new item to the **DataSources** as in the following code snippet.

```
`csharp
//Pass the dataset collection for report
writer.DataSources.Clear();

writer.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list", Value =
ProductList.GetData() });
`
```

Data source **Name** is case sensitive and it should be same as in the data source name in the report definition and the **Value** accepts IList, DataSet, and DataTable inputs.

6. You can use the **Save** method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as Stream.

```
`csharp
[HttpPost]
public ActionResult Export(string writerFormat)
{
    // Here, we have loaded the Product List.rdlc sample report from application the Resources folder.
    FileStream reportStream = new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Resources\Product
    List.rdlc"), FileMode.Open, FileAccess.Read);

    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Local;
    // Pass the dataset collection for report
    writer.DataSources.Clear();

    writer.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list", Value =
    ProductList.GetData() });
    string fileName = null;
    WriterFormat format;
    string type = null;
    if (writerFormat == "PDF")
    {
        fileName = "Product List.pdf";
        type = "pdf";
        format = WriterFormat.PDF;
    }
    else if (writerFormat == "Word")
    {
        fileName = "Product List.docx";
        type = "docx";
        format = WriterFormat.Word;
    }
    else if (writerFormat == "CSV")
```

```

{
    fileName = "Product List.csv";
    type = "csv";
    format = WriterFormat.CSV;
}
else
{
    fileName = "Product List.xlsx";
    type = "xlsx";
    format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileName = fileName;
return fileStreamResult;
}

```

7. Now, the RDLC report exported successfully in your application.

Export SSRS Report Server Report

Report Writer has support to Export RDL reports into popular file formats such as PDF, Microsoft Word, Microsoft Excel, and CSV from SSRS Report Server.

To export the SSRS Reports, set the **ReportProcessingMode**, **ReportServerURL**, and **ReportPath** properties in Web API as shown in the following steps.

1. To create your first ASP.NET MVC Report Writer application, refer to the [Getting-started](#) section.
2. Set the **reportProcessignMode** API for Bold Report Writer as shown in the following code snippet.

```

`csharp
[HttpPost]
public ActionResult Export(string writerFormat)

```

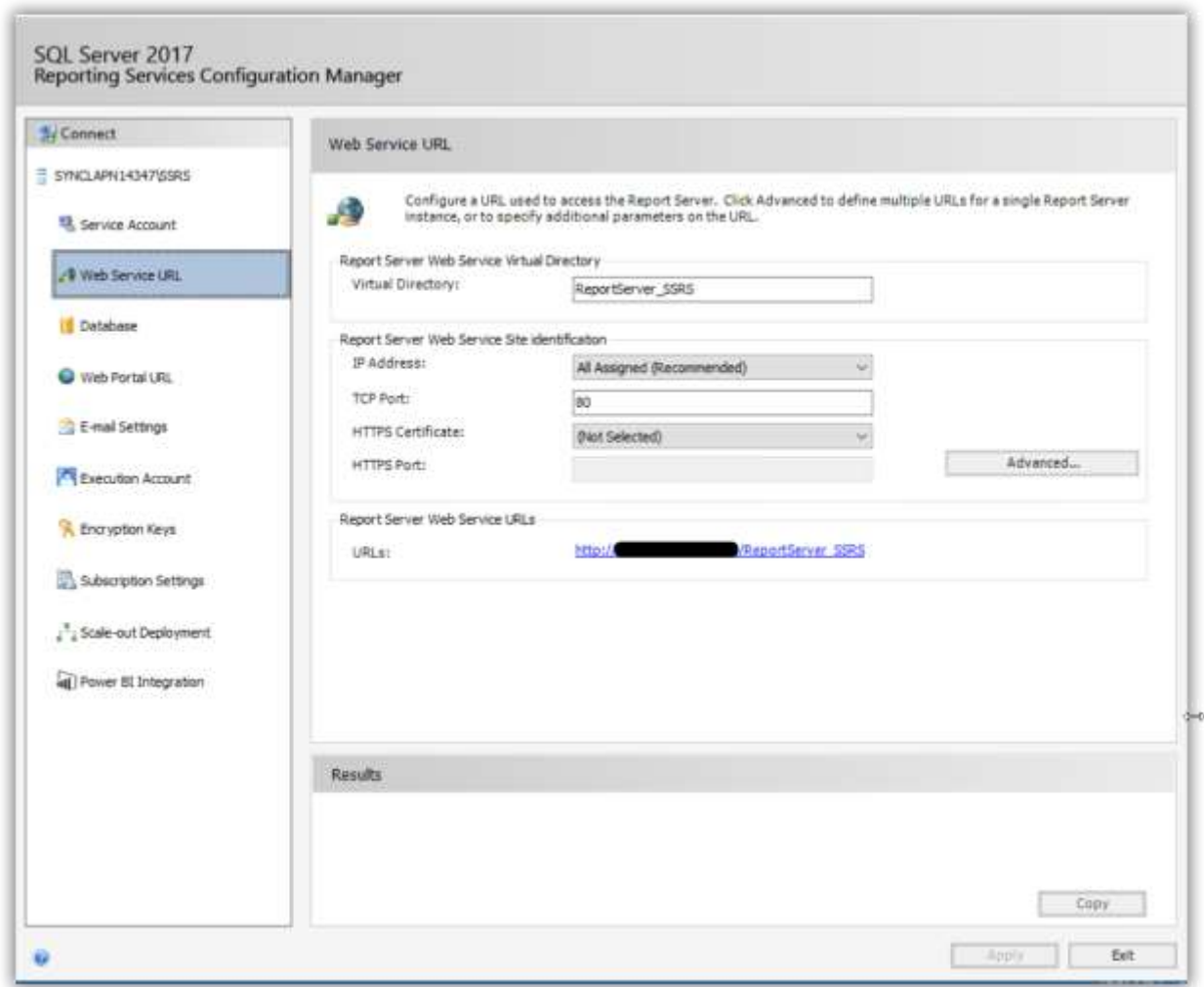
```
{  
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();  
writer.ReportProcessingMode = ProcessingMode.Remote;  
}  
`
```

3. Set the `reportServerUrl` API for Bold Report Writer with `Web Service URL` in the WebAPI as shown in the following code snippet.

```
`csharp  
[HttpPost]  
public ActionResult Export(string writerFormat)  
{  
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();  
writer.ReportProcessingMode = ProcessingMode.Remote;  
writer.ReportServerUrl = "http://<servername>/Reports_SSRS";  
}  
`
```

The Web Service URL should be set as `reportServerUrl` when using ASP.NET MVC service. The Web Service URL can be found from the Reporting Services Configuration manager under the `Web Service`

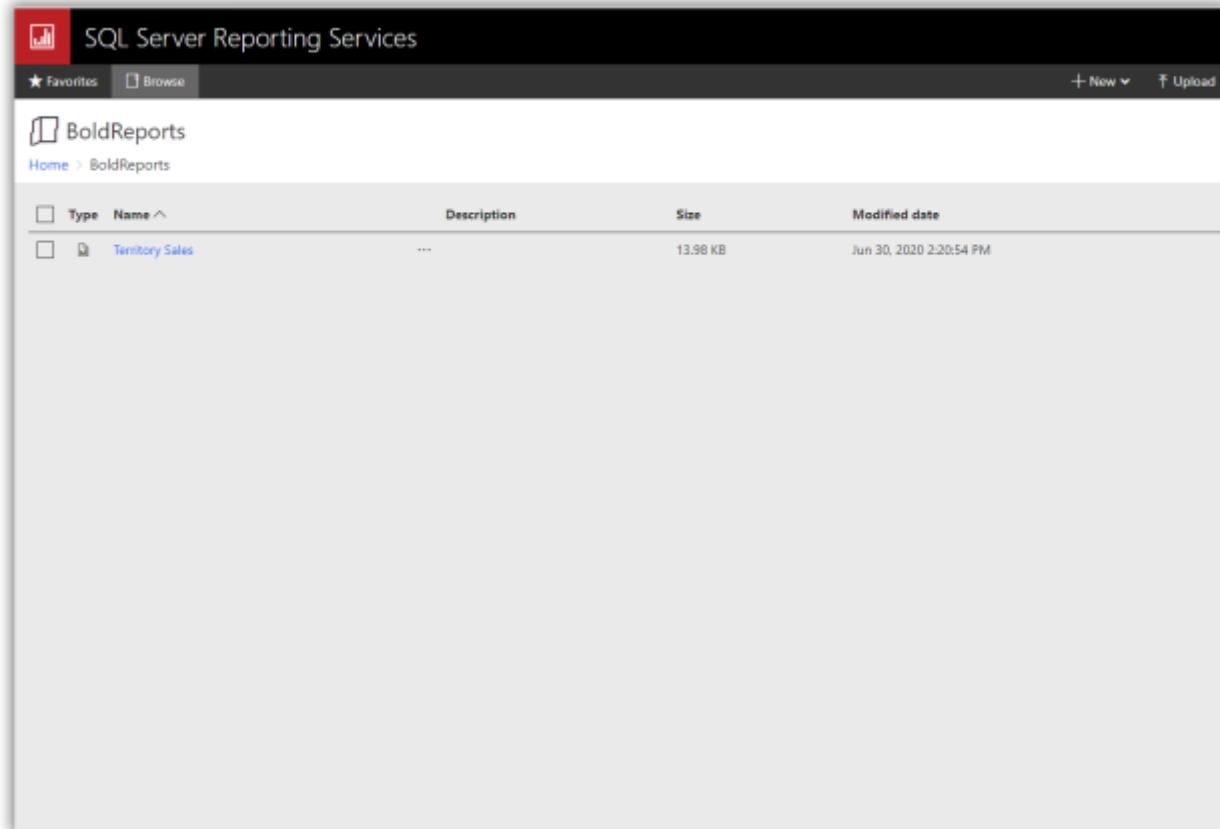
URL section, as shown in the following image.



4. Set the report path for loading the reports from the SSRS Report Server. The report path should be in the format of `/folder name/report name`.

```
`csharp
[HttpPost]
public ActionResult Export(string writerFormat)
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
    writer.ReportServerUrl = "http://<servername>/Reports_SSRS";
    writer.ReportPath = "/SSRSSamples2/Territory Sales";
}
`
```

The report path can be found from the SSRS Report Server by navigating to the path of the report to be loaded, as shown in the following image.



Network credentials for SSRS

The network credentials are required to load specified SSRS report from the specified SSRS Report Server using the Report Writer. Specify the `ReportServerCredential` property in writer instance.

```
`csharp
```

```
writer.ReportServerCredential = new System.Net.NetworkCredential("username", "password");
```

```
,
```

If you are facing problem to access the SSRS Report server reports, you can refer [How to provide the permission for user to access the SSRS Report Server reports](#).

Set data source credential to shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS Server. If the report has any data source that uses credentials to connect with the database, then you should specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp
```

```
List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new  
List<BoldReports.Web.DataSourceCredentials>();
```

```
dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",  
"username", "password"));  
writer.SetDataSourceCredentials(dataSourceCredentialsList);  
`
```

Data source credentials should be added to the shared data sources that do not have credentials in the connection strings.

Refer to the following final code snippet example to export the SSRS report server reports.

```
`csharp  
[HttpPost]  
public ActionResult Export(string writerFormat)  
{  
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();  
    writer.ReportProcessingMode = ProcessingMode.Remote;  
    writer.ReportServerUrl = "http://<servername>/reportserver$instanceName";  
    writer.ReportPath = "/SSRSSamples2/Territory Sales";  
    writer.ReportServerCredential = new System.Net.NetworkCredential("username", "password");  
    List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new  
    List<BoldReports.Web.DataSourceCredentials>();  
    dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("Datasource name",  
    "username", "password"));  
    writer.SetDataSourceCredentials(dataSourceCredentialsList);  
    string fileName = null;  
    WriterFormat format;  
    string type = null;  
    if (writerFormat == "PDF")  
    {  
        fileName = "sales-order-detail.pdf";  
        type = "pdf";  
        format = WriterFormat.PDF;  
    }  
    else if (writerFormat == "Word")  
    {  
        fileName = "sales-order-detail.docx";  
        type = "docx";  
    }  
}
```

```

format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
fileName = "sales-order-detail.csv";
type = "csv";
format = WriterFormat.CSV;
}
else
{
fileName = "sales-order-detail.xlsx";
type = "xlsx";
format = WriterFormat.Excel;
}
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}

```

Export SharePoint Server Report

Report Writer has support to Export RDL reports into popular file formats like PDF, Microsoft Word, Microsoft Excel, and CSV from SharePoint server reports.

This section requires an ASP.NET MVC Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

To export the SharePoint server reports, Initialize the Report Writer and set the `ReportProcessingMode`, `ReportServerURL`, and `ReportPath` properties in Web API as shown in the following code snippet.

```

`csharp
[HttpPost]
public ActionResult Export(string writerFormat)

```



```
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
    writer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
    writer.ReportPath = "http://<servername>/reportserver$instanceName/Report_Path";
}
```

In SharePoint server, the **ReportServerUrl** will be same as your site URL. The **ReportPath** is relative to the Report Server URL with the file extension.

Forms credential for SharePoint Server

The forms credentials are required to load the SharePoint integrated SSRS report from the specified SharePoint integrated SSRS Report Server using the Report Viewer. Specify the **ReportServerFormsCredential** property to access the share point reports.

```
`csharp
writer.ReportServerFormsCredential = new
    BoldReports.Web.ReportServerFormsCredential("username", "password");
```

Set data source credential to shared data sources

The shared data source credentials can be added to the **DataSourceCredentials** property to connect with the database.

```
`csharp
List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new
    List<BoldReports.Web.DataSourceCredentials>();
dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",
    "username", "password"));
writer.SetDataSourceCredentials(dataSourceCredentialsList);
```

Data source credentials should be added to shared data sources that do not have credentials in the connection strings.

Refer to the following final code snippet example to export the SharePoint server reports.

```
`csharp
[HttpPost]
public ActionResult Export(string writerFormat)
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
```

```
writer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
writer.ReportPath = "http://<servername>/reportserver$instanceName/Report_Path";
writer.ReportServerFormsCredential = new
BoldReports.Web.ReportServerFormsCredential("username", "password");
List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new
List<BoldReports.Web.DataSourceCredentials>();
dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",
"username", "password"));
writer.SetDataSourceCredentials(dataSourceCredentialsList);
string fileName = null;
WriterFormat format;
string type = null;
if (writerFormat == "PDF")
{
    fileName = "sales-order-detail.pdf";
    type = "pdf";
    format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
    fileName = "sales-order-detail.docx";
    type = "docx";
    format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
    fileName = "sales-order-detail.csv";
    type = "csv";
    format = WriterFormat.CSV;
}
else
{
    fileName = "sales-order-detail.xlsx";
    type = "xlsx";
```

```

format = WriterFormat.Excel;
}
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}

```

Export Subreport

You can export a Main report with subreport with popular file formats such as PDF, Microsoft Word, and Microsoft Excel without previewing the report in webpage using ASP.NET Report Writer application.

This section requires an ASP.NET MVC Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

Export RDL Subreport

In this tutorial, the `SideBySideMainReport.rdl`, `SideBySideSubReport.rdl` reports is used, and it can be downloaded [here](#). You can get the report from Bold Reports installation location. The reports used from installed location requires `NorthwindIO_Reports.sdf` database to run, so add the database to your application. For more information, refer to [Samples and demos](#).

Refer to the following steps to export the RDL sub report with specified format.

1. Create a folder `Resources` on `App_Data` folder in your application. Copy and paste the sample RDL reports into the `Resources` folder.
2. Open the `HomeController.cs` file in your application and add the `Export()` function to load the report as stream. Refer to the following code snippet.

```

`csharp
public class HomeController : Controller
{
    [HttpPost]
    public ActionResult Export(string writerFormat)
    {
        // Here, we have loaded the sample reports from application the App_Data/Resources folder.
    }
}

```

```

FileStream mainReportStream = new
FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\AppData\Resources\SideBy_SideMainReport.rdl"), FileMode.Open, FileAccess.Read);

FileStream subReportStream = new
FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\AppData\Resources\SideBy_SideSubReport.rdl"), FileMode.Open, FileAccess.Read);

.....
}
}
`

```

3. Initialize the Report Writer instance and pass the subreport name and stream to the `LoadSubreport()` method in Report Writer instance.

```

`csharp
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadSubreport("SideBySideSubReport", subReportStream);
`

```

4. After loading the main report stream using Report Writer instance. Refer to the following code snippet.

```

`csharp
writer.LoadReport(mainReportStream);
`

```

5. You can use the `Save` method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as Stream.

```

`csharp
[HttpPost]
public ActionResult Export(string writerFormat)
{
    FileStream mainReportStream = new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\AppData\Resources\SideBy_SideMainReport.rdl"), FileMode.Open, FileAccess.Read);

    FileStream subReportStream = new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\AppData\Resources\SideBy_SideSubReport.rdl"), FileMode.Open, FileAccess.Read);

    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();

```

```
writer.LoadSubreport("SideBySideSubReport", subReportStream);
writer.LoadReport(mainReportStream);
string fileName = null;
WriterFormat format;
string type = null;
if (writerFormat == "PDF")
{
    fileName = "SideBySideMainReport.pdf";
    type = "pdf";
    format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
    fileName = "SideBySideMainReport.docx";
    type = "docx";
    format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
    fileName = "SideBySideMainReport.csv";
    type = "csv";
    format = WriterFormat.CSV;
}
else
{
    fileName = "SideBySideMainReport.xlsx";
    type = "xlsx";
    format = WriterFormat.Excel;
}
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
```

```

FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}
`

```

6. Now, the RDL report exported successfully in your application.

Export RDLC subreport

To export the RDLC report along with subreport, we need to load the subreport streams before loading a main report to the Report Writer as in the following code example, then only subreport processing event will work.

```

`csharp
[HttpPost]
public ActionResult Pdf()
{
    // Here, we have loaded the sample report from application the folder App_Data.
    FileStream mainReportStream= new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Reports\MainReport.rdlc
    "), FileMode.Open, FileAccess.Read);

    FileStream subreport1Stream = new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Reports\SubReport1.rdlc
    "), FileMode.Open, FileAccess.Read);

    FileStream subreport2Stream = new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Reports\SubReport2.rdlc
    "), FileMode.Open, FileAccess.Read);

    ReportWriter writer = new ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Local;
    writer.SubreportProcessing += ReportWriter_SubreportProcessing;
    //Adding sub report stream going to be used with exporting report.
    writer.LoadSubreport("SubReport1", subreport1Stream);
    writer.LoadSubreport("SubReport2", subreport2Stream);
    //Loading the report going to export as PDF.
    writer.LoadReport(mainReportStream);
    writer.DataSources.Clear();
    writer.DataSources.Add(new ReportDataSource { Name = "DataSet", Value = MainReport.GetData() });
    // Steps to generate PDF report using Report Writer.
}

```

```

MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, BoldReports.Writer.WriterFormat.PDF);
// Download the generated from client.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/pdf");
fileStreamResult.FileNameDownload = "Invoice.pdf";
return fileStreamResult;
}
private void ReportWriter_SubreportProcessing(object sender, SubreportProcessingEventArgs e)
{
    if (e.ReportPath == "SubReport1")
    {
        //Pass the dataset collection for subreport
        e.DataSources.Clear();
        e.DataSources.Add(new ReportDataSource { Name = "DataSet1", Value = SubReport1.GetData() });
    }
    else if (e.ReportPath == "SubReport2")
    {
        //Pass the dataset collection for subreport
        e.DataSources.Clear();
        e.DataSources.Add(new ReportDataSource { Name = "DataSet2", Value = SubReport2.GetData() });
    }
}

```

Export Parameter Report

You can set report parameter default values or modify the values using the `SetParameters()` method of Report Writer instance. This section describes how to modify the exported document report parameter values.

This section requires an ASP.NET MVC Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

In this tutorial, the `sales-order-detail.rdl` report is used and it can be downloaded [here](#). You can get the reports from Bold Reports installation location. For more information, refer to [samples and demos](#) section.

Set report parameters to export file

1. Create a folder **Resources** into the **App_Data** folder in your application. Copy and paste the sample RDL reports into the **Resources** folder.
2. To load the report as stream from the application **Resources** folder using the **FileStream** class.

```
`csharp
public class HomeController : Controller
{
    [HttpPost]
    public ActionResult Export(string writerFormat)
    {
        // Here, we have loaded the sample reports from application the App_Data\Resources folder.
        FileStream reportStream = new
        FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Resources\sales-order-
        detail.rdl"), FileMode.Open, FileAccess.Read);
        BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
        writer.ReportProcessingMode = ProcessingMode.Remote;
        .....
    }
}
```

3. You can add collection of report parameters programmatically using the **ReportParameter** property that is the list type of Report Parameter class. Refer to the following code snippet to set the report parameter name and value.

```
`csharp
List<BoldReports.Web.ReportParameter> userParameters = new
List<BoldReports.Web.ReportParameter>();
userParameters.Add(new BoldReports.Web.ReportParameter()
{
    Name = "SalesOrderNumber",
    Values = new List<string>() { "SO50756" }
});
writer.SetParameters(userParameters);
`
```


4. You can use the **Save** method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as Stream.

```
`csharp
[HttpPost]
public ActionResult Export(string writerFormat)
{
    // Here, we have loaded the sample reports from application the App_Data\Resources folder.
    FileStream reportStream = new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Resources\sales-order-
    detail.rdl"), FileMode.Open, FileAccess.Read);

    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;

    List<BoldReports.Web.ReportParameter> userParameters = new
    List<BoldReports.Web.ReportParameter>();
    userParameters.Add(new BoldReports.Web.ReportParameter()
    {
        Name = "SalesOrderNumber",
        Values = new List<string>() { "SO50756" }
    });
    writer.SetParameters(userParameters);
    string fileName = null;
    WriterFormat format;
    string type = null;
    if (writerFormat == "PDF")
    {
        fileName = "sales-order-detail.pdf";
        type = "pdf";
        format = WriterFormat.PDF;
    }
    else if (writerFormat == "Word")
    {
        fileName = "sales-order-detail.docx";
        type = "docx";
        format = WriterFormat.Word;
    }
}
```

```

}
else if (writerFormat == "CSV")
{
fileName = "sales-order-detail.csv";
type = "csv";
format = WriterFormat.CSV;
}
else
{
fileName = "sales-order-detail.xlsx";
type = "xlsx";
format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated export document to the client side.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}

```

5. Now, the parameter report exported successfully in your application.

Report Writer Events

You can handle the Report Writer events with reports using the following events.

- SubreportProcessing
- ReportErrorOccurred

Subreport Processing

The **SubreportProcessing** event occurs when subreport is loaded and it is ready to start the processing. You can handle the event and specify the data source, parameters, and subreport loading. The following sample code loads a subreport by assigning the report data source credentials and parameter values in the **SubreportProcessing** event.

1. Report Writer instance used to register the **SubreportProcessing** event in your Report Writer application.

```
`csharp
writer.SubreportProcessing += (sen, arg) =>
{
    // Assign the data source and parameter values to the sub report.
};
`
```

2. Set the subreport data source details to the subreport processing event argument.

```
`csharp
writer.SubreportProcessing += (sen, arg) =>
{
    arg.DataSources.Add(new BoldReports.Web.ReportDataSource {Name= "Datasource name", Value =
    value });
}
`
```

Data source **Name** is case sensitive and it should be same as in the data source name in the report definition. The **Value** accepts IList, DataSet, and DataTable inputs.

3. Set the subreport parameters value to the subreport processing event argument.

```
`csharp
writer.SubreportProcessing += (sen, arg) =>
{
    arg.Parameters.Add(new BoldReports.Web.ReportParameterInfo { Name = "Parameter name", Values =
    values});
}
`
```

Report Error

The **ReportErrorOccurred** event raises when an error occurs in the report processing. You can handle the event and get the report error details from the event arguments. Refer to the following sample code to handle the report errors in the **ReportErrorOccurred** event.

```
`csharp
writer.ReportErrorOccurred += (sen, arg) =>
{
```

// You can handle the report errors using event arguments.

```
};  
`
```

Error logging in ASP.NET MVC Report Writer

If an error occurred in report processing, ASP.NET MVC Report writer `ReportErrorOccurred` event raised. You can register the event and get the report error details from the event arguments to save all logs, stack trace, and error information into a physical file location.

This section explains how to log the detailed error information to your ASP.NET MVC application.

This section requires an ASP.NET MVC Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

1. In Solution Explorer, open the `HomeController` file.
2. Add the following sample code to register the report errors in the `ReportErrorOccurred` event.

```
`csharp  
[HttpPost]  
public ActionResult Export(string writerFormat)  
{  
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();  
    writer.ReportErrorOccurred += Writer_ReportErrorOccurred;  
}  
private void Writer_ReportErrorOccurred(object sender, ReportErrorOccurredEventArgs e)  
{  
    // You can register the report errors using event arguments.  
}  
`
```

3. Create a method in `HomeController` to write the error text into application folder.

```
`csharp  
internal void WriteLogs(string errorMessage)  
{  
    string filePath = Path.Combine(System.Web.Hosting.HostingEnvironment.MapPath, "ErrorDetails.txt");  
    using (StreamWriter writer = new StreamWriter(filePath, true))  
    {  
        writer.AutoFlush = true;  
    }  
}
```

```
writer.WriteLine(errorMessage);
}
}
`
```

4. Invoke the newly created function in `ReportErrorOccurred` as follows.

```
`csharp
private void Writer_ReportErrorOccurred(object sender, ReportErrorOccurredEventArgs e)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2}", e.ClassName,
        e.MethodName, e.Message));
}
`
```

In cases of any issues faced in the report rendering, share the log file to our technical support team to get assistance on that.

5. The final controller is given as follows, you can replace it in your application.

```
`csharp
[HttpPost]
public ActionResult Export(string writerFormat)
{
    // Here, we have loaded the sales-order-detail sample report from application the folder Resources.
    FileStream reportStream = new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Resources\sales-order-
    detail.rdl"), FileMode.Open, FileAccess.Read);
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportErrorOccurred += Writer_ReportErrorOccurred;
    string fileName = null;
    WriterFormat format;
    string type = null;
    if (writerFormat == "PDF")
    {
        fileName = "sales-order-detail.pdf";
        type = "pdf";
        format = WriterFormat.PDF;
    }
}
```

```
}  
else if (writerFormat == "Word")  
{  
    fileName = "sales-order-detail.docx";  
    type = "docx";  
    format = WriterFormat.Word;  
}  
else if (writerFormat == "CSV")  
{  
    fileName = "sales-order-detail.csv";  
    type = "csv";  
    format = WriterFormat.CSV;  
}  
else  
{  
    fileName = "sales-order-detail.xlsx";  
    type = "xlsx";  
    format = WriterFormat.Excel;  
}  
writer.LoadReport(reportStream);  
MemoryStream memoryStream = new MemoryStream();  
writer.Save(memoryStream, format);  
// Download the generated export document to the client side.  
memoryStream.Position = 0;  
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/" + type);  
fileStreamResult.FileNameDownload = fileName;  
return fileStreamResult;  
}  
  
private void Writer_ReportErrorOccurred(object sender, ReportErrorOccurredEventArgs e)  
{  
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2}", e.ClassName,  
        e.MethodName, e.Message));  
}
```

```
private void WriteLogs(string errorMessage)
{
    string filePath =
    Path.Combine(System.Web.Hosting.HostingEnvironment.MapPath(@"\App_Data\Resources\ErrorDetails.txt"));
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
        writer.WriteLine(errorMessage);
    }
}
```

Encrypt and Secure Documents

Encrypt and Secure Documents option allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password. The following code snippet explains how to encrypt the exported document with the user defined password.

Password Protected PDF document

You can protect the exported PDF document using the following code snippet.

```
`csharp
writer.PDFOptions = new BoldReports.Writer.PDFOptions();
writer.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity
{
    UserPassword = "Password"
};
```

Password Protected Word document

You can protect the exported Word document using the following code snippet.

```
`csharp
writer.WordOptions = new BoldReports.Writer.WordOptions()
{
    EncryptionPassword = "password"
};
```

Password Protected Excel document

You can protect the exported Excel document using the following code snippet.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    PasswordToModify = "password",
    PasswordToOpen = "password"
};
`
```

Advance Layouts For Merged Cells

Report Writer supports additional export layouts in Word and Excel export formats to eliminate the tiny columns, rows, and merged cells. The benefits of the layout are:

- Overcomes the merging problem with tiny cells, rows, and columns in SSRS Excel and Word exporting.
- Provides clear readability by eliminating the tiny columns, rows, and merge cells.
- Allows you to perform data manipulations such as applying sorting, filters, and grouping in Excel.

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
writer.WordOptions.LayoutOption = WordLayoutOptions.TopLevel;
writer.WordOptions.ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
    Bottom = 0.5f,
    Top = 0.5f
};
`
```

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior. The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, add an empty paragraph between two tables.

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` as `IgnoreCellMerge`.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
```



```
{  
LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge  
};  
`
```

Change the Default File Format

Users can change the default file format to any other file format that is supported by the selected file type. It includes APIs to set the formats before exporting the document. Refer to the following section to change the default file formats.

Change the Word document type

You can save the report to the required Word document version by setting the `FormatType` property.

```
`csharp  
writer.WordOptions = new BoldReports.Writer.WordOptions()  
{  
FormatType = WordFormatType.Docx  
};  
`
```

Change the Excel document type

You can save the report to the required Excel document version by setting the `ExcelSaveType` property.

```
`csharp  
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()  
{  
ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013  
};  
`
```

Change the CSV document type

You can save the report to the required CSV file extension by setting the `FileExtension` property.

```
`csharp  
writer.CsvOptions = new BoldReports.Writer.CsvOptions()  
{  
FileExtension = ".txt"  
};  
`
```

PDF Settings

The PDF export options provides properties to manage PDF export behaviors. You can set the customization properties in the `PDFOptions`. Refer to the following code snippet to initialize the `PDFOptions` property.

```
`csharp
writer.PDFOptions = new PDFOptions();
`
```

Export with complex scripts

To export reports with the complex script language texts, set the `ComplexScript` property of `PDFOptions` instance to `true`.

```
`csharp
writer.PDFOptions.EnableComplexScript = true;
`
```

PDF conformance

You can export the report as a PDF/A-1b document by specifying the `PdfConformanceLevel.Pdf_A1B` conformance level in the `PdfConformanceLevel` property.

```
`csharp
writer.PDFOptions.PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B;
`
```

Embedding Custom PDF fonts

You can add custom fonts to the PDF exported document by adding the font streams to `Fonts` collection in `PDFOptions` instance.

To add custom fonts to the PDF exported document, follow these steps:

1. Add the font `.ttf` files to your application `App_Data/Resources` folder.
2. In the Solution Explorer, open the properties of the font file and set the `Copy` property to Output Directory as Copy always.
3. Initialize the `Font` collection and add the font stream to it.

The key value provided in the font collection should be same as in the report item font property.

```
`csharp
//Load Missing font stream to the pdf document
writer.PDFOptions.Fonts = new Dictionary<string, System.IO.Stream>
{
    { "Segoe UI", new
    FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"\AppData\Resources\Fontfile_name.
    ttf"), FileMode.Open, FileAccess.Read) }
}
```

```
};
`
```

If any fonts used in the report definition is not installed or available in the local system, then you should load the font stream like above code snippet.

Password Protected PDF document

Allows you to protect the exported PDF document from unauthorized users by encrypting the document using encryption password. The following code snippet explains how to encrypt the exported document with user-defined password.

```
`csharp
writer.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity
{
    UserPassword = "Password"
};
`
```

Excel Settings

The Excel export options provides properties to manage Excel document export behaviors. You can set the customization properties in the `ExcelOptions`. Refer to the following code snippet to initialize the `ExcelOptions` property.

```
`csharp
writer.ExcelOptions = new ExcelOptions();
`
```

Excel document type

You can save the report to the required Excel version by setting the `ExcelSaveType` property.

```
`csharp
writer.ExcelOptions.ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013;
`
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` as `IgnoreCellMerge`.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
};
`
```

Protecting Excel document from editing

You can restrict the Excel document from editing by providing the `ExcelSheetProtection` or enabling the `ReadOnlyRecommended` properties.

```
`csharp
writer.ExcelOptions.ReadOnlyRecommended = true;

writer.ExcelOptions.ExcelSheetProtection = Syncfusion.XlsIO.ExcelSheetProtection.DeletingColumns;
`
```

Change Excel export format

Allows you to change the default file format to any other file format using the `ExcelSaveType` properties.

```
`csharp
writer.ExcelOptions.ExcelSaveType = ExcelVersion.Excel2013;
`
```

Password Protected Excel document

Allows you to protect the exported Excel document from unauthorized users by encrypting the document using the encryption password. The following code snippet explains how to encrypt the exported document with user-defined password.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    PasswordToModify = "password",
    PasswordToOpen = "password"
};
`
```

Word Settings

The Word export options provides properties to manage Word document export behaviors. You can set the customization properties in the `WordOptions`. Refer to the following code snippet to initialize the `WordOptions` property.

```
`csharp
writer.WordOptions = new WordOptions();
`
```

Word document type

You can save the report to the required document version by setting the `FormatType` property.

```
`csharp
writer.WordOptions.FormatType = WordFormatType.Docx;
`
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
writer.WordOptions.LayoutOption = WordLayoutOptions.TopLevel;
writer.WordOptions.ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
    Bottom = 0.5f,
    Top = 0.5f
};
```

A paragraph element is inserted between two tables in the exported document to overcome the Word document auto merging behavior. The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, add an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without password. The following are the types of protection:

- `AllowOnlyComments`: Adds or modifies only the comments in the Word document.
- `AllowOnlyFormFields`: Modifies the form field values in the Word document.
- `AllowOnlyRevisions`: Accepts or rejects the revisions in the Word document.
- `AllowOnlyReading`: Views the content only in the Word document.
- `NoProtection`: Accesses or edits the Word document contents as normally.

```
`csharp
writer.WordOptions.ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading;
```

Password Protected Word document

Allows you protect the exported Word document from unauthorized users by encrypting the document using encryption password. The following code snippet explains how to encrypt the exported document with user-defined password.

```
`csharp
writer.WordOptions = new BoldReports.Writer.WordOptions()
{
    EncryptionPassword = "password"
```

```
};  
`
```

CSV Export Settings

The `CsvOptions` allows you to change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document. You should set the customization properties in the `CsvOptions` property of Report Writer instance.

```
`csharp  
writer.CsvOptions = writer.CsvOptions = new BoldReports.Writer.CsvOptions()  
{  
    Encoding = System.Text.Encoding.Default,  
    FieldDelimiter = ",",  
    UseFormattedValues = false,  
    Qualifier = "#",  
    RecordDelimiter = "@",  
    SuppressLineBreaks = true,  
    FileExtension = ".txt"  
};  
`
```

Export Data Visualization in MVC Environment

we have provided the below option to export the data visualization report items

1. [External browser - Puppeteer](#).
2. [PhantomJS - Classic Tool](#)

`External Browser - Puppeteer`

1. It supports `ASP.NET Core 2.1` or above version only.
2. Open the Report Writer application and install the `PuppeteerSharp` package from [NuGet Gallery](#).
3. Create a folder like `wwwroot/ResourcesTemp` in your Report Writer application.
4. Open the Report Writer application Web API file.
5. Set the `BrowserTypes` Enum property to `External`.
6. Create and initialize `CustomBrowserType` by inheriting the `ExportSettings` class.
7. Set the `ResourcePath`, `Scripts` and `DependentScripts` property in Report Writer instance.
The following code example demonstrates how to configure `Puppeteer` in your ASP.NET Core application.

In this tutorial, the `product-line-sales.rdl` report is used, and it can be downloaded in this [link](#).

```
`csharp
public class HomeController : Controller
{
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // IWebHostEnvironment initialized with controller to get the data from application data folder.
    public HomeController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _hostingEnvironment = hostingEnvironment;
    }
    public IActionResult Index()
    {
        return View();
    }
    [HttpPost]
    public IActionResult Export(string writerFormat)
    {
        string fileName = null;
        WriterFormat format;
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sample report from application the folder wwwroot.
        FileStream inputStream = new FileStream(basePath + @"\Resources\product-line-sales.rdl",
        FileMode.Open, FileAccess.Read);
        MemoryStream reportStream = new MemoryStream();
        inputStream.CopyTo(reportStream);
        reportStream.Position = 0;
        inputStream.Close();
        BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
        // Initialize the class for puppeteer
        writer.ExportSettings = new CustomBrowserType();
        // Puppeteer Option to export the data visualization report items.
        writer.ExportResources.BrowserType = ExportResources.BrowserTypes.External;
        writer.ExportResources.ResourcePath = basePath + @"/ResourcesTemp/";
```

```
writer.ExportResources.Scripts = new List<string>
{
    //Gauge component scripts
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-lineargauge.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-circulargauge.min.js",
    //Map component script
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js",
    //Chart component script
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js",
    //Report Viewer Script
    "https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js",
};
writer.ExportResources.DependentScripts = new List<string>
{
    "https://code.jquery.com/jquery-1.10.2.min.js"
};
if (writerFormat == "PDF")
{
    fileName = "ProductLineSales.pdf";
    format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
    fileName = "ProductLineSales.docx";
    format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
```



```
{
fileName = "ProductLineSales.CSV";
format = WriterFormat.CSV;
}
else
{
fileName = "ProductLineSales.xlsx";
format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated document from client.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/pdf");
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}

// Add the class to override the method for puppeteer
public class CustomBrowserType : ExportSettings
{
// Can be used to convert the image from external browser
public override string GetImageFromHTML(string url)
{
return LaunchPuppeteer(url).Result;
}

// Can be used to convert the image from Puppeteer
public async Task<string> LaunchPuppeteer(string url)
{
var browserFetcher = new BrowserFetcher();
await browserFetcher.DownloadAsync();
await using var browser = await Puppeteer.LaunchAsync(new LaunchOptions { Headless = true });
await using var page = await browser.NewPageAsync();
```

```

await page.GoToAsync(url);

return await
page.WaitForSelectorAsync("#imagejsonData").Result.GetPropertyAsync("innerText").Result.JsonValueA
sync<string>();
}
}
}
`

```

The **Scripts** and **Dependent** scripts must be added to export the data visualization report items.

`PhantomJS - Classic Tool`

To download **PhantomJS** application and deploy it on your machine, you should accept its license terms on [LICENSE](#) and [Third-Party](#) document.

1. Download **PhantomJS** for windows [here](#) and for linux [here](#) and extract the download file.
2. Copy the **PhantomJS** file from the extracted bin folder and paste into **AppData/PhantomJS** folder in your application.
3. Open the Report Writer application Web API file.
4. Set the **UsePhantomJS** property to true.
5. Set the **PhantomJSPath**, **Scripts** and **DependentScripts** property in **Report Writer** instance. The following code example demonstrates how to configure **PhantomJS** in your ASP.NET MVC application.

In this tutorial, the **sales-order-detail.rdl** report is used, and it can be downloaded in this [link](#).

```

`csharp
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    [HttpPost]
    public ActionResult Export(string writerFormat)
    {
        // Here, we have loaded the sample report from application the folder App_Data.
        FileStream reportStream = new FileStream(Server.MapPath(@"\App_Data\Resources\sales-order-
        details.rdl"), FileMode.Open, FileAccess.Read);

        BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();

```

```
string fileName = null;
WriterFormat format;
string type = null;
// PhantomJS Option to export the data visualization report items.
writer.ExportResources.UsePhantomJS = true;
// Set the IncludeText property to true, when text not displayed in the data visualization images.
writer.ExportResources.IncludeText = true;
writer.ExportResources.PhantomJSPath = Server.MapPath(@"\App_Data\PhantomJS\");
writer.ExportResources.Scripts = new List<string>
{
    //Gauge component scripts
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-lineargauge.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-circulargauge.min.js",
    //Map component script
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js",
    //Chart component script
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js",
    //Report Viewer Script
    "https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js",
};
writer.ExportResources.DependentScripts = new List<string>
{
    "https://code.jquery.com/jquery-1.10.2.min.js"
};
if (writerFormat == "PDF")
{
    fileName = "sales-order-detail.pdf";
}
```

```
format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
{
fileName = "sales-order-detail.docx";
format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
fileName = "sales-order-detail.CSV";
format = WriterFormat.CSV;
}
else
{
fileName = "sales-order-detail.xlsx";
format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated document from client.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/pdf");
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}
}
、
```

The **Scripts** and **Dependent** scripts must be added to export the data visualization report items.

Limitations

RDL Specification support

ReportWriter control does not support RDL Specification for SQL Server 2000 and RDL Specification for SQL Server 2005.

Layout Process

Syncfusion ReportWriter control has some limitations in Tablix Cell split Layout process in comparison with MS ReportWriter. When table cell width value exceeds the page width, the entire cell moves to the next page in order to display the complete cell items.

Unsupported expression

- Object function and VB function do not have complete support in ReportWriter.
- VB Code functions are not supported in Silverlight and WinRT ReportWriter.

HTML Formatted Data with Report

Report viewer supports showing the HTML formatted data with Textbox report items along with the inline CSS. This section provides the information about supported tags and limitations.

Supported HTML Tags

- Hyperlinks:
- Fonts:
- Header, style and block elements: `

`,`

,

,

- , `
- Text format: , , ,

Limitations of Cascading Style Sheet Attributes

The following is a list of attributes that are supported:

- text-align, text-indent
- font-family
- font-size
- Only valid RDL size values are supported in absolute CSS length units. Supported units are: in, cm, mm, pt, pc, ex, px and em.
- Relative CSS length units are ignored and are not supported. Unsupported units include percentage (%), and rem.
- color
- padding, padding-bottom, padding-top, padding-right, and padding-left
- font-weight

How to section for Report Writer component

1. [Generate and export RDL and RDLC reports programmatically](#)

How to generate and export RDL and RDLC reports programmatically

The Bold Reports reporting library allows you to generate and export RDL and RDLC reports programmatically. The `ReportDefinition` class is defined as an object model of a report. You can create, add, and modify the report properties, report sections like header and footer and report items in the report definition instance.

The following steps illustrates how to create a basic report object model:

Initialize a report definition

The following code snippet guides you in initializing the report definition.

```
`csharp
ReportDefinition CreateReport()
{
    ReportDefinition report = new ReportDefinition();
    report.ReportSections = new ReportSections();
    var reportSection = new ReportSection();
    report.ReportSections.Add(reportSection);
    reportSection.Width = new BoldReports.RDL.DOM.Size("6in");
    report.ReportUnitType = "Inch";
    report.RDLType = RDLType.RDL2010;
    return report;
}

BoldReports.RDL.DOM.Style AddStyle()
{
    BoldReports.RDL.DOM.Style style = new BoldReports.RDL.DOM.Style();
    style.Border = new BoldReports.RDL.DOM.Border();
    style.Border.Width = new BoldReports.RDL.DOM.Size("1pt");
    style.Border.Style = "Solid";
    style.Border.Color = "Black";
    return style;
}
`
```

Create a Data source for report

The following code snippet guides you in creating a `Datasource` for the report and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
```

```
{
...
...
...
reportSection.Page = new BoldReports.RDL.DOM.Page();
reportSection.Page.Style = AddStyle();
reportSection.Page.PageHeight = "4in";
reportSection.Page.PageWidth = "6in";
var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
<database>");
report.DataSources = new DataSources();
report.DataSources.Add(newDataSource);
...
...
...
}

//If you are using RDLC report then you can pass any string value as connection string
BoldReports.RDL.DOM.DataSource CreateDataSource(string name, string dataProvider, string
connectionString)
{
var dataSource = new BoldReports.RDL.DOM.DataSource();
dataSource.Name = name;
dataSource.SecurityType = BoldReports.RDL.DOM.SecurityType.Integrated;
dataSource.ConnectionProperties = new BoldReports.RDL.DOM.ConnectionProperties();
dataSource.ConnectionProperties.DataProvider = dataProvider;
dataSource.ConnectionProperties.ConnectionString = connectionString;
dataSource.ConnectionProperties.IntegratedSecurity = true;
return dataSource;
}
`
```

Create a Dataset for the report

The following code snippet guides you in creating a **Dataset** for the report and setting values to their properties.

`csharp

```

ReportDefinition CreateReport()
{
...
...
...
reportSection.Page.PageWidth = "6in";
var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
<database>");
report.DataSources = new DataSources();
report.DataSources.Add(newDataSource);
var newDataSet = CreateDataSet("DataSource1", "DataSet1");
report.DataSets = new DataSets();
report.DataSets.Add(newDataSet);
...
...
...
}

BoldReports.RDL.DOM.DataSet CreateDataSet(string dataSourceName, string dataSetName)
{
var dataSet = new BoldReports.RDL.DOM.DataSet();
dataSet.Name = dataSetName;
dataSet.Query = new Query();
dataSet.Query.DataSourceName = dataSourceName;
dataSet.Query.CommandText = "SELECT
HumanResources.Department.DepartmentID,HumanResources.Department.Name,HumanResources.De
partment.GroupName,HumanResources.Department.ModifiedDate FROM
HumanResources.Department";
dataSet.Query.QueryDesignerState = new QueryDesignerState();
var table = CreateTable();
dataSet.Query.QueryDesignerState.Tables = new List<Table>();
dataSet.Query.QueryDesignerState.Tables.Add(table);
dataSet.Fields = new Fields();
dataSet.Fields.Add(CreateField("DepartmentID", "System.Int16"));
dataSet.Fields.Add(CreateField("Name", "System.String"));

```



```
dataSet.Fields.Add(CreateField("GroupName", "System.String"));
dataSet.Fields.Add(CreateField("ModifiedDate", "System.DateTime"));
return dataSet;
}

BoldReports.RDL.DOM.Table CreateTable()
{
    var table = new Table();
    table.Schema = "HumanResources";
    table.Name = "Department";
    table.Columns = new List<Column>();
    table.Columns.Add(CreateColumn("DepartmentID"));
    table.Columns.Add(CreateColumn("Name"));
    table.Columns.Add(CreateColumn("GroupName"));
    table.Columns.Add(CreateColumn("ModifiedDate"));
    return table;
}

BoldReports.RDL.DOM.Column CreateColumn(string columnName)
{
    var column = new Column();
    column.Name = columnName;
    return column;
}

BoldReports.RDL.DOM.Field CreateField(string fieldName, string type)
{
    var field = new Field();
    field.Name = fieldName;
    field.TypeName = type;
    field.DataField = fieldName;
    return field;
}
、
```

Create a report page header

The following code snippet guides you in creating a **PageHeader** report section and setting values to their properties.

```

`csharp
ReportDefinition CreateReport()
{
    ...
    ...
    ...
    reportSection.Page = new BoldReports.RDL.DOM.Page();
    reportSection.Page.Style = AddStyle();
    reportSection.Page.PageHeight = "4in";
    reportSection.Page.PageWidth = "6in";
    reportSection.Page.PageHeader = CreateHeader();
    ...
    ...
    ...
}
PageHeader CreateHeader()
{
    PageHeader pageHeader = new PageHeader();
    pageHeader.Height = new BoldReports.RDL.DOM.Size("0.59167in");
    pageHeader.Style = AddStyle();
    pageHeader.PrintOnFirstPage = true;
    pageHeader.PrintOnLastPage = true;
    pageHeader.ReportItems = new ReportItems();
    var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
    pageHeader.ReportItems.Add(textBox);
    return pageHeader;
}
`

```

Create a report page footer

The following code snippet guides you in creating a **PageFooter** report section and setting values to their properties.

```

`csharp
ReportDefinition CreateReport()
{

```

```

...
...
...
reportSection.Page.PageHeader = CreateFooter();
...
...
...
}
PageFooter CreateFooter()
{
PageFooter pageFooter = new PageFooter();
pageFooter.Style = AddStyle();
pageFooter.Height = new BoldReports.RDL.DOM.Size("0.59167in");
pageFooter.ReportItems = new ReportItems();
pageFooter.PrintOnFirstPage = true;
pageFooter.PrintOnLastPage = true;
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
return pageFooter;
}
`

```

Initialize a report body section

- Initialize a report body object and set the values to their properties like height, styles, and report items as shown in the following code snippet.

```

`csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Body = CreateBody();
...
...
}

```

```
...
}
Body CreateBody()
{
    var body = new Body();
    body.Height = new BoldReports.RDL.DOM.Size("2.03333in");
    body.Style = AddStyle();
    body.ReportItems = new ReportItems();
    var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
    body.ReportItems.Add(textBox);
    return body;
}
`
```

- Using the following code snippets, you can create a **Textbox** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

```
`csharp
PageHeader CreateHeader()
{
    ...
    ...
    ...
    pageHeader.ReportItems = new ReportItems();
    var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
    pageHeader.ReportItems.Add(textBox);
    ...
    ...
    ...
}
PageFooter CreateFooter()
{
    ...
    ...
    ...
}
```

```
pageFooter.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
...
...
...
}
Body CreateBody()
{
...
...
...
body.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
body.ReportItems.Add(textBox);
...
...
...
}
BoldReports.RDL.DOM.TextBox CreateTextBox(string name, string text, string left, string top, string
width, string height)
{
var textBox = new BoldReports.RDL.DOM.TextBox();
textBox.Name = name;
textBox.Height = new BoldReports.RDL.DOM.Size(height);
textBox.Width = new BoldReports.RDL.DOM.Size(width);
textBox.Left = new BoldReports.RDL.DOM.Size(left);
textBox.Top = new BoldReports.RDL.DOM.Size(top);
textBox.Style = new BoldReports.RDL.DOM.Style();
textBox.Style.TextAlign = "Center";
textBox.Style.VerticalAlign = "Top";
textBox.Paragraphs = new Paragraphs();
BoldReports.RDL.DOM.Paragraph paragraph = new BoldReports.RDL.DOM.Paragraph();
```

```

TextRuns runs = new TextRuns();
TextRun run = new TextRun();
run.Style = new BoldReports.RDL.DOM.Style();
run.Style.FontStyle = "Default";
run.Style.TextAlign = "Center";
run.Style.FontFamily = "Arial";
run.Style.FontSize = new BoldReports.RDL.DOM.Size("10pt");
run.Value = text;
runs.Add(run);
paragraph.Style = new BoldReports.RDL.DOM.Style();
paragraph.Style.VerticalAlign = "Top";
paragraph.Style.TextAlign = "Center";
paragraph.TextRuns = runs;
textBox.Paragraphs.Add(paragraph);
return textBox;
}
`

```

Create a Table for the report

- Using the following code snippets, you can create a **Table** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

```
`csharp
```

```
BoldReports.RDL.DOM.Tablix CreateTablix(string name, string text, string left, string top, string width,
string height)
```

```

{
var tableItem = new BoldReports.RDL.DOM.Tablix();
tableItem.Name = name;
tableItem.Height = new BoldReports.RDL.DOM.Size(height);
tableItem.Width = new BoldReports.RDL.DOM.Size(width);
tableItem.Left = new BoldReports.RDL.DOM.Size(left);
tableItem.Top = new BoldReports.RDL.DOM.Size(top);
tableItem.Style = new BoldReports.RDL.DOM.Style();
tableItem.Style.Border = new BoldReports.RDL.DOM.Border();
tableItem.Style.Border.Style = "Solid";

```

```
tableItem.DataSetName = "DataSet1";
tableItem.TablixBody = new TablixBody();
tableItem.TablixBody.TablixColumns = new TablixColumns();
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
tableItem.TablixBody.TablixRows = new TablixRows();
var rowOne = new List<TablixRowValues>();
rowOne.Add(new TablixRowValues { TextBoxName = "TextBox1", TextBoxValue = "Department ID" });
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox3", TextBoxValue = "Name" });
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox5", TextBoxValue = "Group Name" });
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowOne));
var rowTwo = new List<TablixRowValues>();
rowTwo.Add(new TablixRowValues { TextBoxName = "DepartmentID", TextBoxValue =
    "=Fields!DepartmentID.Value" });
rowTwo.Add(new TablixRowValues { TextBoxName = "Name", TextBoxValue = "=Fields!Name.Value" });
rowTwo.Add(new TablixRowValues { TextBoxName = "GroupName", TextBoxValue =
    "=Fields!GroupName.Value" });
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowTwo));
tableItem.TablixColumnHierarchy = new TablixColumnHierarchy();
tableItem.TablixColumnHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixRowHierarchy = new TablixRowHierarchy();
tableItem.TablixRowHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixRowHierarchy.TablixMembers.Add(new TablixMember() { KeepWithGroup =
    KeepWithGroup.After });
tableItem.TablixRowHierarchy.TablixMembers.Add(CreateTablixMember("Details"));
return tableItem;
}

BoldReports.RDL.DOM.TablixColumn CreateTablixColumn()
{
    var tablixColumn = new TablixColumn();
```

```
tablixColumn.Width = "1in";
return tablixColumn;
}

BoldReports.RDL.DOM.TablixRow CreateTablixRow(List<TablixRowValues> values)
{
    var tablixRow = new TablixRow();
    tablixRow.Height = "0.25in";
    tablixRow.TablixCells = new TablixCells();
    for(int i = 0; i < values.Count; i++)
    {
        tablixRow.TablixCells.Add(CreateTablixCell(values[i].TextBoxName, values[i].TextBoxValue));
    }
    return tablixRow;
}

BoldReports.RDL.DOM.TablixCell CreateTablixCell(string textBoxName, string textBoxValue)
{
    var tablixCell = new TablixCell();
    tablixCell.CellContents = new CellContents();
    tablixCell.CellContents.ReportItem = CreateTextBox(textBoxName, textBoxValue);
    return tablixCell;
}

BoldReports.RDL.DOM.TablixMember CreateTablixMember(string groupName)
{
    var tablixMember = new TablixMember();
    tablixMember.Group = new Group();
    tablixMember.Group.Name = groupName;
    return tablixMember;
}

internal class TablixRowValues
{
    public string TextBoxName { get; set; }
    public string TextBoxValue { get; set; }
}
```


Create a Chart for the report

- Using the following code snippets, you can create a **Chart** report item and assign the values for their properties like name, styles, and dimension values.

```
`csharp
BoldReports.RDL.DOM.Chart CreateChart(string name, string left, string top, string width, string height)
{
var chartItem = new Chart();
chartItem.Name = name;
chartItem.Height = new BoldReports.RDL.DOM.Size("10in");
chartItem.Width = new BoldReports.RDL.DOM.Size("10in");
chartItem.Left = new BoldReports.RDL.DOM.Size("5in");
chartItem.Top = new BoldReports.RDL.DOM.Size("5in");
chartItem.DataSetName = "DataSet1";
chartItem.Style = new BoldReports.RDL.DOM.Style();
chartItem.Bookmark= "=First(Fields!Name.Value, \"DataSet1\")";
chartItem.ChartCategoryHierarchy = new ChartCategoryHierarchy();
chartItem.ChartCategoryHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartCategoryHierarchy.ChartMembers.Add(ChartCategoryMember());
chartItem.ChartSeriesHierarchy = new ChartSeriesHierarchy();
chartItem.ChartSeriesHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartSeriesHierarchy.ChartMembers.Add(ChartSeriesMember());
chartItem.ChartData = new ChartData();
chartItem.ChartData.ChartDerivedSeriesCollection = new ChartDerivedSeriesCollection();
chartItem.ChartData.ChartSeriesCollection = CreateChartSeriesCollection();
chartItem.ChartLegends = new ChartLegends();
chartItem.ChartAreas = new ChartAreas();
chartItem.ChartAreas.Add(CreateChartArea());
chartItem.ChartTitles = new ChartTitles();
chartItem.Palette = "Pacific";
chartItem.ChartBorderSkin = new ChartBorderSkin();
chartItem.ChartNoDataMessage = new ChartNoDataMessage();
chartItem.ChartNoDataMessage.Caption = "No Data Available";
```

```
chartItem.ChartNoDataMessage.Name = "NoDataMessage";
return chartItem;
}

BoldReports.RDL.DOM.ChartMember ChartCategoryMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = new Group();
    ChartMember.Group = CreateChartGroup();
    ChartMember.Label="=Fields!DepartmentID.Value";
    ChartMember.SortExpressions = SortExpressions();
    return ChartMember;
}

BoldReports.RDL.DOM.ChartMember ChartSeriesMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = null;
    ChartMember.Label = "Department ID";
    ChartMember.SortExpressions = new SortExpressions();
    return ChartMember;
}

BoldReports.RDL.DOM.SortExpressions SortExpressions()
{
    var SortExpressions = new SortExpressions();
    var SortExpression = new SortExpression();
    SortExpression.Direction = SortDirection.Ascending;
```

```
SortExpression.Value = "=Fields!DepartmentID.Value";
SortExpressions.Add(SortExpression);
return SortExpressions;
}

BoldReports.RDL.DOM.Group CreateChartGroup()
{
    var ChartGroup = new Group();
    ChartGroup.DataElementName = null;
    ChartGroup.DataElementOutput = DataElementOutputs.Auto;
    ChartGroup.DocumentMapLabel = null;
    ChartGroup.DocumentMapLabelLocID = null;
    ChartGroup.DomainScope = null;
    ChartGroup.Filters = new Filters();
    ChartGroup.GroupExpressions = CreateGroupExpressions();
    ChartGroup.Name = "Chart2_CategoryGroup";
    ChartGroup.PageBreak = null;
    ChartGroup.PageName = null;
    ChartGroup.Parent = null;
    ChartGroup.Variables = new Variables();
    return ChartGroup;
}

BoldReports.RDL.DOM.GroupExpressions CreateGroupExpressions()
{
    var GroupExpressions = new GroupExpressions();
    var GroupExpression = new GroupExpression();
    GroupExpression.Value = "=Fields!DepartmentID.Value";
    GroupExpressions.Add(GroupExpression);
    return GroupExpressions;
}

BoldReports.RDL.DOM.ChartSeriesCollection CreateChartSeriesCollection()
{
    var ChartSeriesCollection = new ChartSeriesCollection();
    ChartSeriesCollection.Add(CreateChartSeries());
}
```

```
return ChartSeriesCollection;
}

BoldReports.RDL.DOM.ChartSeries CreateChartSeries()
{
    var ChartSeries = new ChartSeries();
    ChartSeries.CategoryAxisName = "Primary";
    ChartSeries.ChartAreaName = null;
    ChartSeries.ChartDataLabel = null;
    ChartSeries.ChartDataPoints = new ChartDataPoints();
    ChartSeries.ChartDataPoints.Add(ChartDataPoint());
    ChartSeries.ChartEmptyPoints = new ChartEmptyPoints();
    ChartSeries.ChartEmptyPoints.ChartDataLabel = new ChartDataLabel();
    ChartSeries.ChartEmptyPoints.ChartDataLabel.Style = new Style();
    ChartSeries.ChartEmptyPoints.ChartMarker = new ChartMarker();
    ChartSeries.ChartEmptyPoints.ChartMarker.Style = new Style();
    ChartSeries.ChartSmartLabel = new ChartSmartLabel();
    ChartSeries.Name = "DepartmentID";
    ChartSeries.Type = VisualizationType.Column;
    ChartSeries.Subtype = VisualizationSubType.Plain;
    ChartSeries.Style = new Style();
    return ChartSeries;
}

BoldReports.RDL.DOM.ChartDataPoint ChartDataPoint()
{
    var ChartDataPoint = new ChartDataPoint();
    ChartDataPoint.ActionInfo = null;
    ChartDataPoint.ChartDataLabel = null;
    ChartDataPoint.ChartDataLabel = ChartDataLabel();
    ChartDataPoint.ChartDataPointValues = new ChartDataPointValues();
    ChartDataPoint.ChartDataPointValues.Y = "=Sum(Fields!DepartmentID.Value)";
    ChartDataPoint.ChartItemInLegend = null;
    ChartDataPoint.ChartMarker = new ChartMarker();
    ChartDataPoint.ChartMarker.Size = null;
}
```

```
ChartDataPoint.ChartMarker.Type = null;
ChartDataPoint.ChartMarker.Style= chartStyle("Transparent", "Arial");
ChartDataPoint.CustomProperties = new CustomProperties();
ChartDataPoint.DataElementName = null;
ChartDataPoint.DataElementOutput = DataElementOutputs.Output;
ChartDataPoint.ToolTip = null;
ChartDataPoint.Style= chartStyle("Transparent", "Arial");
return ChartDataPoint;
}

BoldReports.RDL.DOM.ChartDataLabel ChartDataLabel()
{
    var ChartDataLabel = new ChartDataLabel();
    ChartDataLabel.ActionInfo = null;
    ChartDataLabel.Label = null;
    ChartDataLabel.Position = null;
    ChartDataLabel.Rotation = "0";
    ChartDataLabel.Style = new Style();
    ChartDataLabel.Style = chartStyle("Transparent", "Arial");
    ChartDataLabel.ToolTip = null;
    return ChartDataLabel;
}

BoldReports.RDL.DOM.Style chartStyle(string BackgroundColor, string FontFamily)
{
    var style = new Style();
    style.BackgroundColor = BackgroundColor;
    style.FontFamily = FontFamily;
    return style;
}

BoldReports.RDL.DOM.ChartArea CreateChartArea()
{
    var chartArea = new ChartArea();
    chartArea.Style = new Style();
    chartArea.Style = chartStyle("Transparent", "Arial");
```

```
chartArea.AlignOrientation = AlignOrientation.None;
chartArea.AlignWithChartArea = null;
chartArea.ChartAlignType = null;
chartArea.ChartElementPosition = null;
chartArea.ChartInnerPlotPosition = null;
chartArea.ChartThreeDProperties = null;
chartArea.Hidden = false;
chartArea.EquallySizedAxesFont = false;
chartArea.Name = "Default";
chartArea.ChartCategoryAxes = new ChartCategoryAxes();
chartArea.ChartCategoryAxes.Add(createChartAxis("Primary"));
chartArea.ChartCategoryAxes.Add(createChartAxis("Secondary"));
chartArea.ChartValueAxes = new ChartValueAxes();
chartArea.ChartValueAxes.Add(createChartAxis("Primary"));
chartArea.ChartValueAxes.Add(createChartAxis("Secondary"));
return chartArea;
}

BoldReports.RDL.DOM.ChartAxis createChartAxis(string Name)
{
    var ChartAxis = new ChartAxis();
    ChartAxis.Style = new Style();
    ChartAxis.AllowLabelRotation = AllowLabelRotation.None;
    ChartAxis.Angle = "0";
    ChartAxis.Arrows = Arrows.None;
    ChartAxis.ChartAxisScaleBreak = new ChartAxisScaleBreak();
    ChartAxis.ChartAxisTitle = new ChartAxisTitle();
    ChartAxis.ChartMajorGridLines = new ChartMajorGridLines();
    ChartAxis.ChartMajorTickMarks = new ChartMajorTickMarks();
    ChartAxis.ChartMinorGridLines = new ChartMinorGridLines();
    ChartAxis.ChartMinorTickMarks = new ChartMinorTickMarks();
    ChartAxis.ChartStripLines = new ChartStripLines();
    ChartAxis.CrossAt = "NaN";
    ChartAxis.CustomProperties = new CustomProperties();
}
```

```
ChartAxis.LineStyle = LineStyle.Solid;
ChartAxis.Name = Name;
return ChartAxis;
}
```

Export the generated report in ReportWriter

You can export the report using the ReportWriter, after the report definition or stream is created. The following code snippet illustrates how to export the report into PDF format using the Report Writer by report definition.

```
`csharp
ReportWriter reportWriter = new ReportWriter();
reportWriter.LoadReport(reportDefinition);
reportWriter.Save(fileName, WriterFormat.PDF);
```

How to load the report from the database

You must load the report using the **Stream** option available with **writer.LoadReport()** method. To load the report from the database, please follow these steps:

From byte array

```
`csharp
byte[] bytes = ""; // provide your byte array report data
MemoryStream reportStream = new MemoryStream(bytes);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadReport(reportStream);
```

From base64String

```
`csharp
string base64String = ""; // provide your base64 report data
byte[] bytes = System.Convert.FromBase64String(base64String);
MemoryStream reportStream = new MemoryStream(bytes);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadReport(reportStream);
```

From string

```
`csharp
```

```
string reportData = ""; // provide your Report data
byte[] bytes = System.Text.Encoding.ASCII.GetBytes(reportData);
MemoryStream reportStream = new MemoryStream(bytes);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadReport(reportStream);
,
```

Reporting tools for ASP.NET Web Forms

Enterprise-class reporting tools for ASP.NET Web Forms development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your web applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

Reporting tools for ASP.NET Web Forms

Enterprise-class reporting tools for ASP.NET Web Forms development to embed reporting functionalities such as designing, viewing, exporting, and printing reports in your web applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application. A good starting point would be to refer to the code snippets in the online [sample browser](#) which contains code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

System Requirements

This topic describes the software and hardware requirements for setting up the development environment of Bold Reports ASP.NET Webforms.

Supported Operating Systems

- Windows 7+, 8+
- Windows Server 2008 R2+

Hardware Environments

The following hardware environments are necessary for setting up the Bold Reports ASP.NET Web Forms.

- 1 GHZ or faster, 32 bit or 64 bit processor.
- 1 GB RAM for 32 bit or 2 GB RAM for 64 bit.

Development Environments

By using the following IDEs, you can develop the Bold Reports ASP.NET Web Forms.

- [Microsoft Visual Studio 2010](#) or [later](#)
- Internet Information Services (IIS) 7.0+

Framework

The below tool is required for Bold Reports ASP.NET Web Forms.

.NET Framework 4.5 or higher

Browser Compatibility

- IE 9+
- Microsoft Edge
- Mozilla Firefox 22+
- Chrome 17+
- Opera 12+
- Safari 5+

See Also

- [Licensing procedure for deployment](#)

Overview

The Report Viewer is a visualization control used to display SSRS, RDL, RDLC, and Bold Report Server reports within web applications. It allows you to view RDL/RDLC reports with or without using SSRS or Bold Report Server. You can bind data sources, parameters, and render reports with all major capabilities of RDL reporting and export the report to PDF, Microsoft Excel, CSV, Microsoft Word, and HTML formats. Some of the key features are,

- Renders interactive reports with drill down, drill through, hyperlinks, and interactive sorting.
- Easily customize each element of Report Viewer and provide events for report processing customization.
- Supports jQuery, Angular, React, Blazor, ASP.NET Core, ASP.NET MVC, ASP.NET WebForms, WPF, and UWP.

Display SSRS RDL report in Bold Reports ASP.NET Web Forms Report Viewer

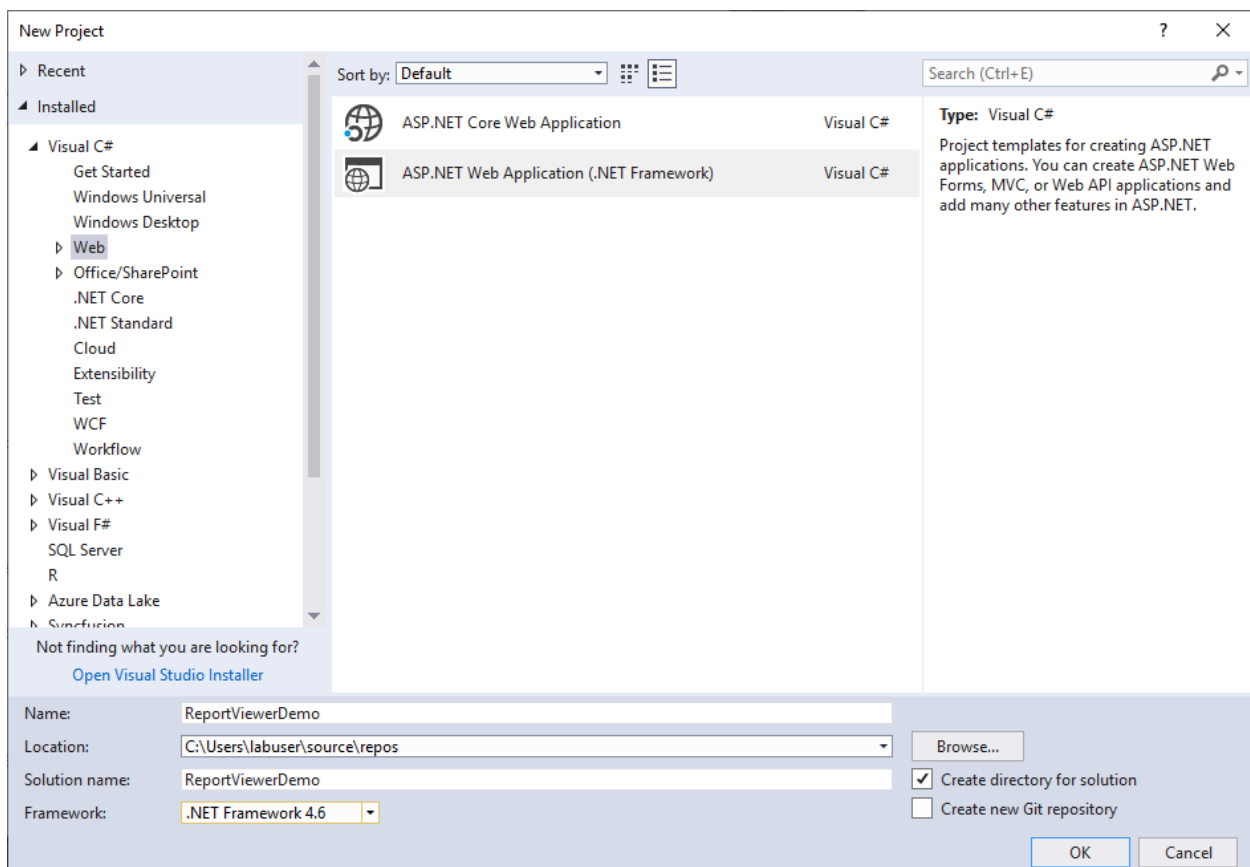
This section explains you the steps required to create your first ASP.NET Web Forms reporting application to display an already created SSRS RDL report in Bold Reports ASP.NET Web Forms Report Viewer without using a Report Server.

To get start quickly with Report Viewer, you can check on this video:

youtube: <https://youtu.be/D3pEoqkTC34>

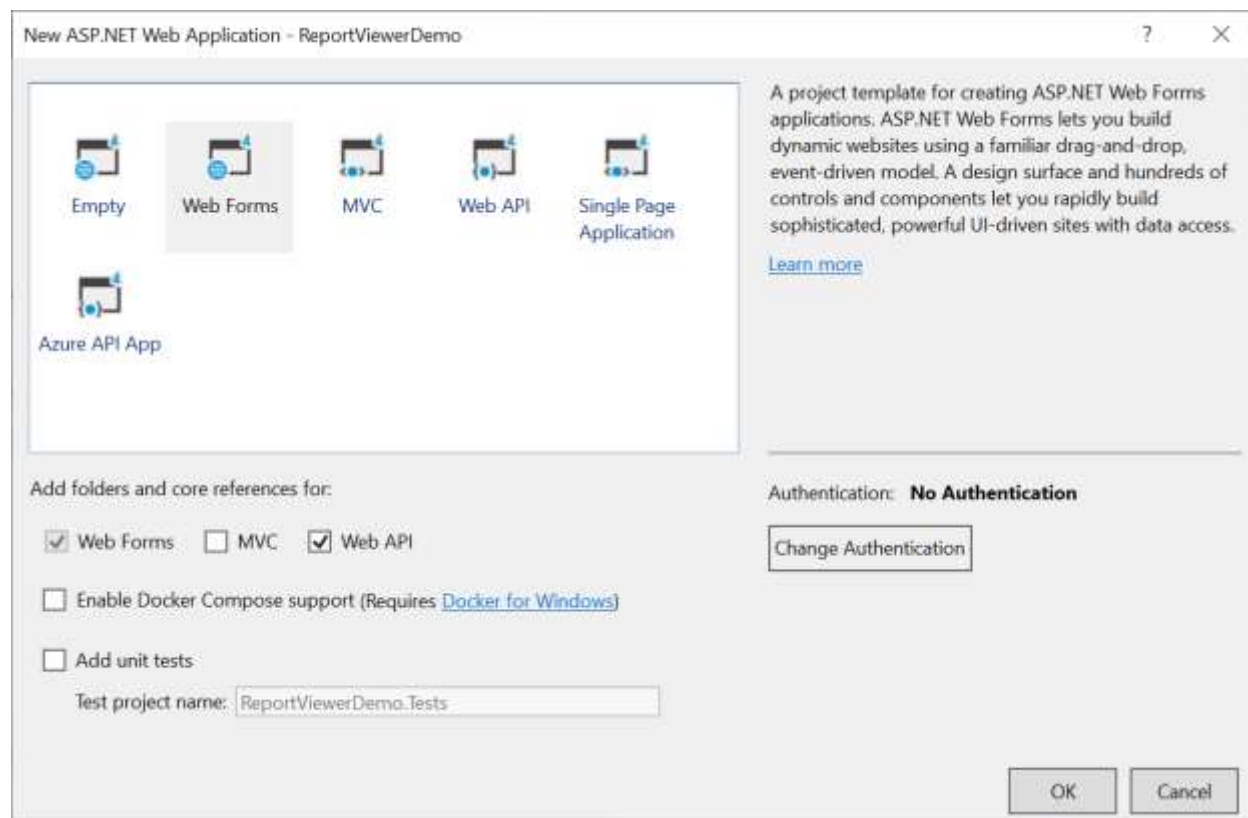
Create an ASP.NET Web Forms application

1. Open Visual Studio 2017, click the **File** menu, go to **New**, and then select **Project**.
2. Go to **Installed > Visual C# > Web**, and then select the required **.NET Framework** in the dropdown.
3. Select **ASP.NET Web Application (.NET Framework)**, change the application name, and then click **OK**.



Display SSRS RDL report in Bold Reports ASP.NET Web Forms Report Viewer **Configure** Report Viewer in an application

4. Choose **Web Forms**, **Web API** and then click **OK**.



Configure Report Viewer in an application

1. Right-click the project or solution on the Solution Explorer tab and choose **Manage NuGet Packages**. Alternatively, select **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring Report Viewer NuGet packages.

2. Search for **BoldReports.Web** and **BoldReports.WebForms** NuGet packages and install them in your Web Forms application. The following table provides details about the packages and their usage.

Package | Purpose

PostReportAction | Action (HttpPost) method for posting the request in the report process.

OnInitReportOptions | Report initialization method occurs when the report is about to be processed.

OnReportLoaded | Report loaded method occurs when the report and sub report start loading.

GetResource | Action (HttpGet) method to get resources for the report.

Display SSRS RDL report in Bold Reports ASP.NET Web Forms Report Viewer **Configure** Report Viewer in an application

ReportHelper

The class **ReportHelper** contains helper methods that help to process a Post or Get request from the Report Viewer control and return the response to the Report Viewer control. It has the following methods:

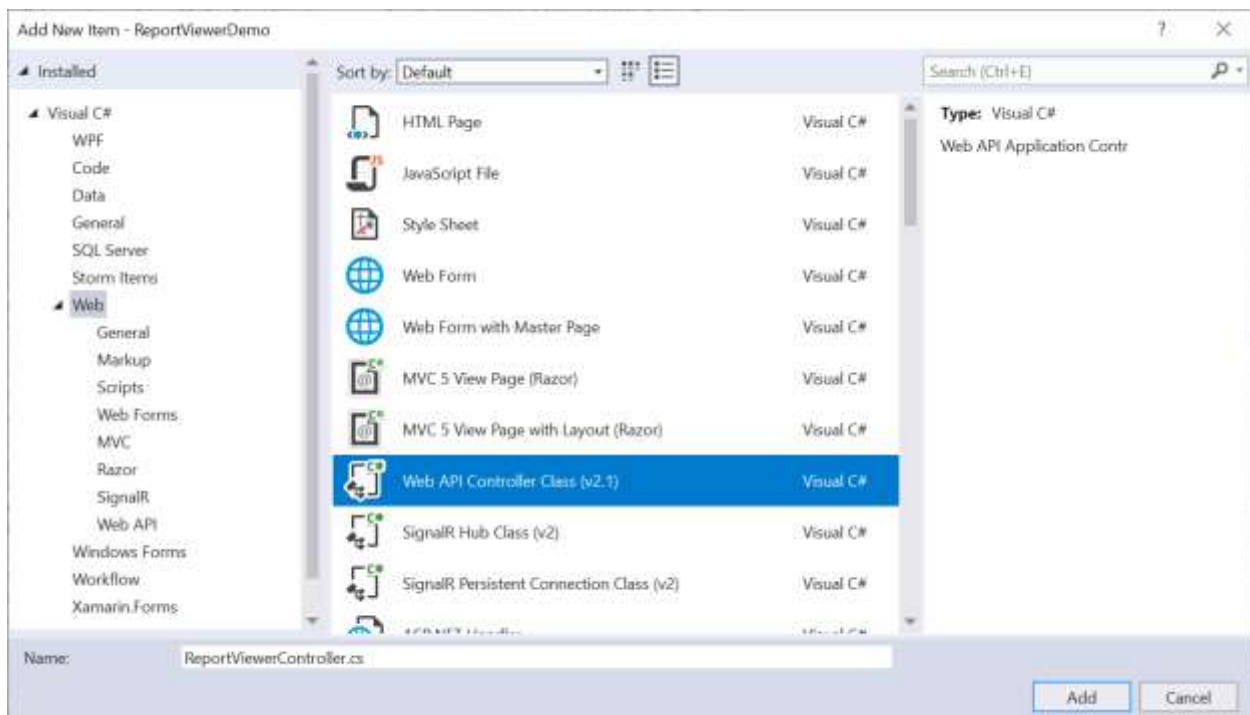
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click the **Controller** folder in your project and select **Add > New Item** from the context menu.
2. Select **Web API Controller Class** from the listed templates and name it as **ReportViewerController.cs**.



3. Click **Add**.

While adding the Web API Controller class, naming it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
```

```
using BoldReports.Web.ReportViewer;
```

```
`
```

Display SSRS RDL report in Bold Reports ASP.NET Web Forms Report Viewer **Configure** Report Viewer in an application

5. Inherit the `IReportController` interface, and implement its methods (replace the following code in the newly created Web API controller).

```
`csharp
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
    [System.Web.Http.ActionName("GetResource")]
    [AcceptVerbs("GET")]
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }
    // Method that will be called when initialize the report options before start processing the report
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
    // Method that will be called when reported is loaded
    [NonAction]
    public void OnReportLoaded(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
}
```

Display SSRS RDL report in Bold Reports ASP.NET Web Forms Report Viewer **Set** report path and service URL

Add routing information

To configure routing to include an action name in the URI, open the `WebApiConfig.cs` file and change the `routeTemplate` in the **Register** method as follows,

```
`csharp
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services
        // Web API routes
        config.MapHttpAttributeRoutes();
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{action}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

If you are looking to load the report directly from the SQL Server Reporting Services (SSRS), then you can skip the following steps and move to the [SSRS Report](#).

Set report path and service URL

To render the reports available in the application, set the `ReportPath` and `ReportServiceUrl` properties of the Report Viewer. You can replace the following code on your Report Viewer page.

```
`js
<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
<link href="Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<script src="Scripts/bold-reports/bold.report-viewer.min.js"></script>
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer" ReportPath="~/Resources/sales-order-detail.rdl"
ReportServiceUrl="/api/ReportViewer">
```

```
</Bold:ReportViewer>
```

```
</div>
```

```
</asp:Content>
```

The report path property is set for the RDL report that is added to the project **Resources** folder.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

The screenshot shows a web application interface for viewing a sales order report. At the top, there is a toolbar with icons for print, copy, save, zoom, and other report viewer functions. Below the toolbar, a 'Sales Order No.' dropdown menu is set to 'SO50750', and a 'View Report' button is visible on the right. The main content area displays the report for 'REVENUE RIDE cycles'. The report includes a header with the company logo and name, followed by a 'Sales Order' section containing key details: Order ID (#SO50750), Billing Date (22-04-2019), Order Date (01-06-2003), Purchase Order (POT192170677), and Shipment Method (CARGO TRANSPORT 5). Below this, the Billing and Shipping addresses for 'Jean Handley' at 'Central Discount Store' are listed. The core of the report is a table with 7 columns: Line, Qty, Item Number, Description, Tracking No., Unit Price, and Item Total. It lists various items including caps, gloves, jerseys, and bicycles. At the bottom, there is a total row showing a total of \$2,802.57. A footer section contains a certification statement and a line for the 'Signature of Authorized Person'.

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0162-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.64	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	B6-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	B6-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

Note: You can refer to our feature tour page for the [ASP.NET Web Forms Report Viewer](#) to see its innovative features. Additionally, you can view our [ASP.NET Web Forms Report Viewer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

See Also

[Render report with data visualization report items](#)

[Create RDLC report](#)

[List of SSRS server versions are supported in Bold Reports](#)

Load SSRS Report Server reports

Report Viewer has support to load RDL reports from SSRS Report Server. To render SSRS Reports, set the `reportServerUrl`, `reportPath`, and `reportServiceUrl` properties as shown in the following steps.

1. To create your first ASP.NET WebForms reporting application, refer to the [Getting-started](#) section.

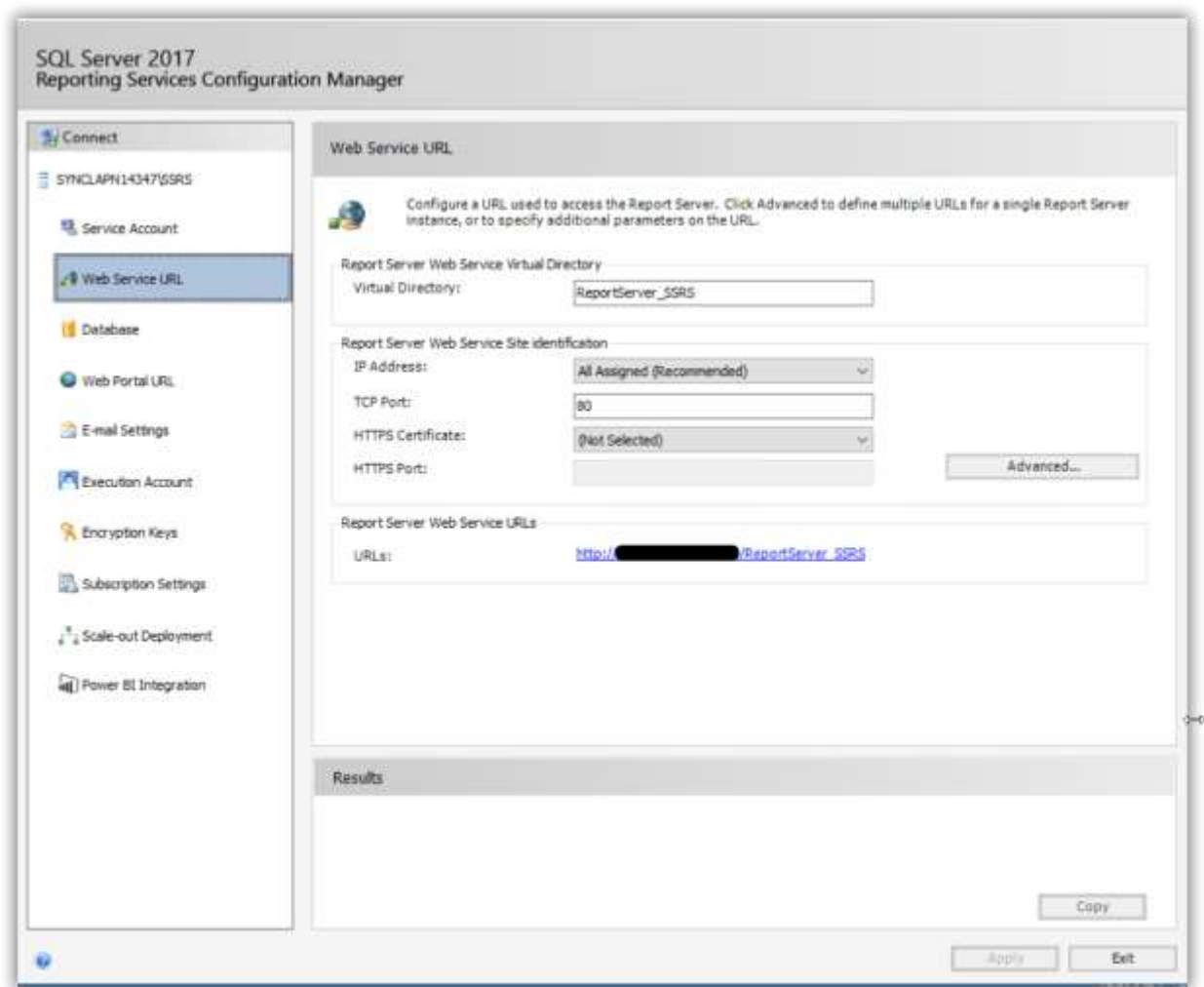
If you need to know about the difference between `reportServiceUrl` and `reportServerUrl`, then refer to [Difference between Report Service URL and Report Server URL](#).

2. Set the `reportServerUrl` API on Bold Report Viewer with `WebServiceURL`. Open the `App.js` or `app.component.ts` and replace the following code example.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="/BoldReports/Territory Sales"
ReportServerUrl="http://<servername>/Reports_SSRS">
</Bold:ReportViewer>
</div>
`
```

The Web Service URL should be set as `reportServerUrl` in the report viewer configuration. The Web Service URL can be found from the Reporting Services Configuration manager under the `Web Service`

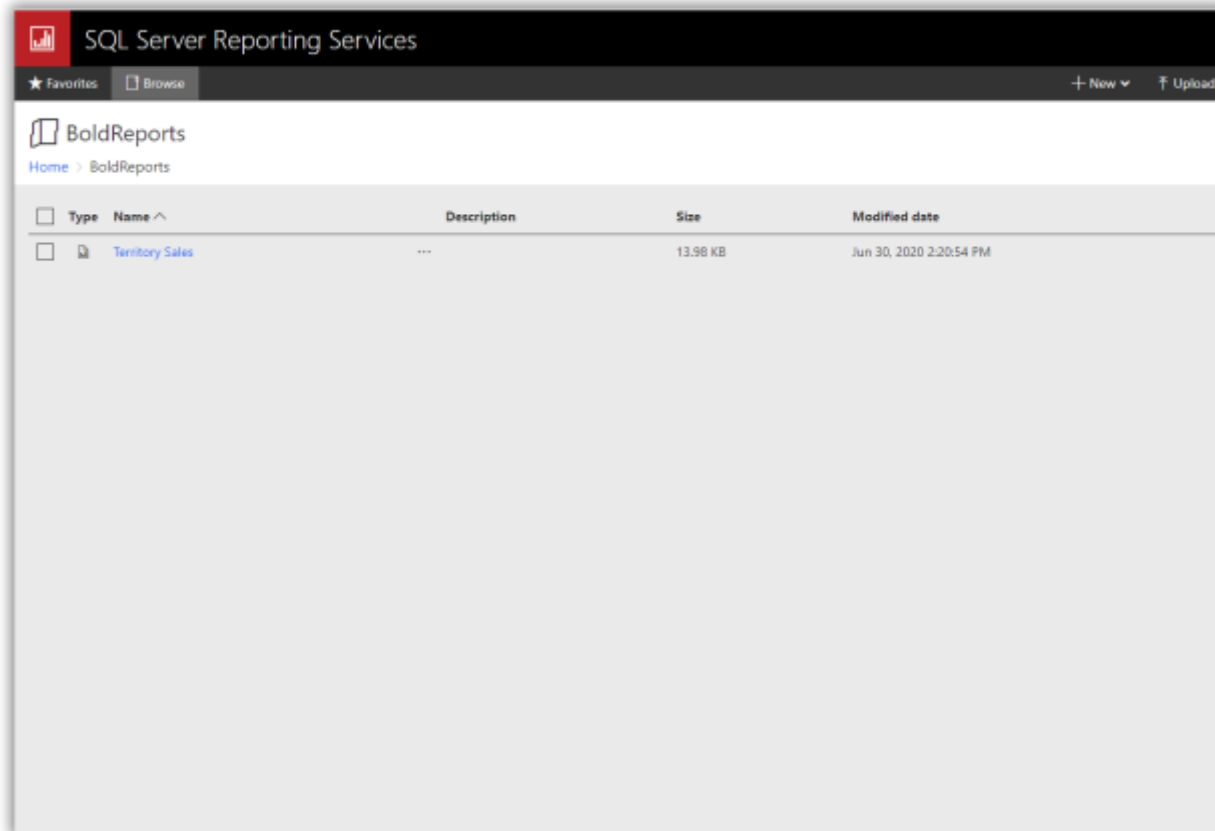
URL section, as shown in the following image.



3. Set the report path for loading the reports from the SSRS Report Server. The report path should be in the format of `/folder name/report name`. Open the `Index.cshtml` and replace the following code example.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="/BoldReports/Territory Sales"
ReportServerUrl="http://<servername>/Reports_SSRS">
</Bold:ReportViewer>
</div>
`
```

The report path can be found from the SSRS Report Server by navigating to the path of the report to be loaded, as shown in the following image.



Network credentials for SSRS

The network credentials are required to load specified SSRS report from the specified SSRS Report Server using the Report Viewer. Specify the `ReportServerCredential` property in the Web API Controller `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server credential
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
    "RDLReport1");
}
`
```

If you are facing problem to access the SSRS Report server reports, you can refer [How to provide the permission for user to access the SSRS Report Server reports](#).

Set data source credential to shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS Server. If the report has any data source that uses credentials to connect with the database, then you should specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server and data source credentials
    reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
    "RDLReport1");
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
}
`
```

Data source credentials should be added to the shared data sources that do not have credentials in the connection strings.

Change data source connection string

You can change the connection string of a report data source before it is loaded in the Report Viewer. The `DataSourceCredentials` class provides the option to set and update the modified connection string as in the following code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "<username>", "<password>", "Data
    Source=<instancename>;Initial Catalog=<database>;"));
}
`
```

The previous code shows an option to change the connection string only, but the class provides multiple options to change data source information. To learn more about this, refer to this `DataSourceCredentials` class.

See also

[Does Bold Report Viewer use SSRS Report processing?](#)

Load SharePoint Server reports

To render SharePoint server reports, set the `reportServerUrl`, `reportPath`, and `reportServiceUrl` properties as shown in the following code snippet.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/SharePointReports"
ReportServerUrl="http://<servername>/reportserver$instanceName"
ReportPath="http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl">
</Bold:ReportViewer>
</div>
`
```

In SharePoint integrated mode, the `reportServerUrl` will be same as your site URL. The `reportPath` is relative to the Report Server URL with the file extension.

Forms credential for SharePoint Server

The forms credentials are required to load the SharePoint integrated SSRS report from the specified SharePoint integrated SSRS Report Server using the Report Viewer. Specify the `ReportServerFormsCredential` property in the Web API Controller `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
//Add ReportServerFormsCredential for server
reportOption.ReportModel.ReportServerFormsCredential = new
BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");
}
`
```

Set data source credential to shared data sources

The shared data source credentials can be added to the `DataSourceCredentials` property to connect with the database.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
//Add ReportServerFormsCredential and data source credentials
}
```

```
reportOption.ReportModel.ReportServerFormsCredential = new
BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
}
,
```

Data source credentials should be added to shared data sources that do not have credentials in the connection strings.

Render RDLC report

The data binding support allows you to view RDLC reports that exist on the local file system with JSON array and custom business object data collection. The following steps demonstrates how to render an RDLC report with JSON array and custom business object data collection.

Add the RDLC report `product-list.rdlc` from Bold Reports installation location to your application `Resources` folder. For more information, see [Samples and demos](#).

Bind data source at client

1. Set the `ReportPath` and `ReportServiceUrl` to the report viewer initialization.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/product-list.rdlc">
</Bold:ReportViewer>
</div>
,
```

2. The following code is used to set the report datasource client side in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
ProductList productList = new ProductList();

List<BoldReports.Models.ReportViewer.ReportDataSource> dataSources = new
List<BoldReports.Models.ReportViewer.ReportDataSource>();
```

```
BoldReports.Models.ReportViewer.ReportDataSource dataSource = new
BoldReports.Models.ReportViewer.ReportDataSource();
dataSource.Name = "list";
dataSource.Value = productList.GetData();
dataSources.Add(dataSource);
this.viewer.DataSources = dataSources;
}
`
```

Bind data source in Web API controller

The following steps help you to configure the Web API to render the RDLC report with business object data collection.

1. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```
`csharp
[Serializable]
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = null;
        data = new ProductList()
        {
            ProductName = "Baked Chicken and Cheese",
            OrderId = "323B60",
            Price = 55,
            Category = "Non-Veg",

```

```

Ingredients = "grilled chicken, corn and olives.",
ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Delite",
    OrderId = "323B61",
    Price = 100,
    Category = "Non-Veg",
    Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
,

```

2. Set the value of the `ProcessingMode` property to `ProcessingMode.Local` in the RDLC report location.
3. To set the `ReportPath` of the report by using the `MapPath`.
4. Bind the business object data values collection by adding a new item to the `DataSources` as in the following code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ProcessingMode = ProcessingMode.Local;
    reportOption.ReportModel.ReportPath =
        System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/product-list.rdlc");
    reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list",
        Value = ProductList.GetData() });
}
`
```

Here, the **Name** is case sensitive and it should be same as in the data source name in the report definition.

The **Value** accepts *ICollection*, *DataSet*, and *DataTable* inputs.

In the previous code, the **product-list.rdlc** report is loaded from the **Resources** folder location.

Render subreport

You can display another report inside the body of a main report using the Report Viewer. The following steps help you to customize the subreport properties such as data source, report path, and parameters.

1. Add the sub report and main reports to the application **Resources** folder. In this tutorial, the already created reports are used. Refer to the [Create RDL Report section](#) or [Create RDLC Report section](#).

Download the **side-by-side-main-report.rdl**, **side-by-side-sub-report.rdl** reports from [here](#). You can add a report from the Syncfusion installation location. For more information, refer to [Samples and demos](#).

- Set the **ReportPath** and **ReportServiceUrl** properties of the Report Viewer as in the following code snippet.

The following code example demonstrates how to load a subreport in the Report Viewer at client side.

```
`js
<div style="height: 650px; width: 950px; min-height: 404px;">
<Bold:ReportViewer runat="server" ID="viewer"
    ReportServiceUrl="/api/ReportViewer">
    ReportPath="~/Resources/side-by-side-main-report.rdl"
</Bold:ReportViewer>
</div>
```


,

Change subreport path

To change the subreport file path, set the **Stream** property of SubReportModel in the **OnInitReportOptions** method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        FileStream SubStream = new
        FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/" +
        reportOption.SubReportModel.ReportPath), FileMode.Open, FileAccess.Read);
        reportOption.SubReportModel.Stream = SubStream;
    }
}
,
```

Set subreport parameter

You can change the parameter default values of a subreport in the **OnReportLoaded** method of the Web API Controller as given in the following code snippet.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.Parameters = new BoldReports.Web.ReportParameterInfoCollection();
        reportOption.SubReportModel.Parameters.Add(new BoldReports.Web.ReportParameterInfo()
        {
            Name = "SalesPersonID",
            Values = new List<string>() { "2" }
        });
    }
}
,
```

Modify subreport data source connection string

You can change the credential and connection information of the data sources used in the subreport using the SubReportModel in the `OnInitReportOptions` method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSourceCredentials = new
        List<BoldReports.Web.DataSourceCredentials>();
        reportOption.SubReportModel.DataSourceCredentials.Add(new
        BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
        Catalog=<database>;user id=<username>;password=<password>"));
    }
}
```

Set subreport data source

You can bind local business object data source collection only for RDLC reports. To specify data source of a RDLC subreport, set the `ReportDataSource` property in the `OnReportLoaded` method.

The RDL report has the connection information in report definition itself, so no need to bind data source.

1. Add the RDLC sub report and main reports to your application `Resources` folder. You can download it from [here](#).
2. Set the `ReportPath` and `ReportServiceUrl` properties of the Report Viewer as shown in following code snippet.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer">
ReportPath="~/Resources/product-list-main.rdlc"
ProcessingMode="Local"
</Bold:ReportViewer>
</div>
```

3. Create a class and methods that returns business object data collection. Use the following code in the application Web API Service.

```
`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = null;
        data = new ProductList()
        {
            ProductName = "Baked Chicken and Cheese",
            OrderId = "323B60",
            Price = 55,
            Category = "Non-Veg",
            Ingredients = "grilled chicken, corn and olives.",
            ProductImage = ""
        };
        datas.Add(data);
        data = new ProductList()
        {
            ProductName = "Chicken Delite",
            OrderId = "323B61",
            Price = 100,
            Category = "Non-Veg",
            Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
            ProductImage = ""
        };
    }
}
```

```
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
`
```

4. Bind the business object data values collection to the subreport by adding a new item to the **DataSources** as shown in the following code snippet.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'product-list.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
        "list", Value = ProductList.GetData() });
    }
}
`
```

The data source name is case sensitive, it should be same as in the report definition.

Load subreport stream

To load subreport as stream, set the **Stream** property in the **OnInitReportOptions** method.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (reportOption.SubReportModel != null)
    {
        FileStream reportStream = new
        FileStream(System.Web.Hosting.HostingEnvironment.MapPath(@"~/Resources/" +
        reportOption.SubReportModel.ReportPath), FileMode.Open, FileAccess.Read);
        reportOption.SubReportModel.Stream = reportStream;
    }
}
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Assigning the data source for 'product-list.rdlc'
    if (reportOption.SubReportModel != null)
    {
        reportOption.SubReportModel.DataSources = new BoldReports.Web.ReportDataSourceCollection();
        reportOption.SubReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource { Name =
        "list", Value = ProductList.GetData() });
    }
}
`
```

Report parameters

Provides property options to pass or set report parameters default values at run time using the **parameters** property. You can set the report parameters while creating the Report Viewer control in a script or in the Web API Controller.

In this tutorial, the **sales-order-detail.rdl** report is used, and it can be downloaded from [here](#).

Set parameter at client

1. Set the **ReportPath** and **ReportServiceUrl** to the report viewer initialization.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`
```

2. The following code is used to set the report parameter client side in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.Parameters.Add(new BoldReports.Models.ReportViewer.ReportParameter()
{
Name = "SalesOrderNumber",
Values = new List<string>() { "SO50756" }
});
}
`
```

Set parameters in Web API Controller

To set parameter default value in Web API Controller, use the following code in the `OnReportLoaded` method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
List<BoldReports.Web.ReportParameter> userParameters = new
List<BoldReports.Web.ReportParameter>();
userParameters.Add(new BoldReports.Web.ReportParameter()
{
Name = "SalesOrderNumber",
Values = new List<string>() { "SO50756" }
}
```

```
});
reportOption.ReportModel.Parameters = userParameters;
}
`
```

The Report Parameters name should be case sensitive

Get report parameter

The **ReportHelper** class provides methods to get the report parameters used in the report. The following helper methods used to get parameter with or without values.

Methods | Description

MaxDateTime | Specify minimum range value of a date parameter

MinDateTime | Specify maximum range value of a date parameter

Refer to the following code sample to set data range to the report viewer initialization.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/product-line-sales.rdl">
</Bold:ReportViewer>
</div>
`
```

Refer to the following code sample to set data range using **parameterSettings** in Report Viewer controller.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.ParameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
this.viewer.ParameterSettings.MaxDateTime = new DateTime(2003, 6, 22);
this.viewer.ParameterSettings.MinDateTime = new DateTime(2003, 5, 20);
}
`
```

The above code sets date range for all the date parameters used in the report.

To set different date range for each date parameter used in the report, register the event **beforeParameterAdd** and specify range based on parameter name as in below code sample.

```
`js
```

```

<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/product-line-sales.rdl"
OnClientBeforeParameterAdd="beforeParameterAdd" >
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function beforeParameterAdd(args) {
if (args.parameterModel.Name === "StartDate") {
args.parameterSettings.minDateTime = new Date("4/5/2003 5:00:00 AM");
args.parameterSettings.maxDateTime = new Date("4/15/2003 5:00:00 AM");
}
if (args.parameterModel.Name === "EndDate") {
args.parameterSettings.minDateTime = new Date("5/10/2003 5:00:00 AM");
args.parameterSettings.maxDateTime = new Date("5/20/2003 5:00:00 AM");
}
}
}
</script>
`

```

Set date time display format for date parameter

The properties `dateTimeFormat` and `timeDisplayFormat` in the `parameterSettings` are used to set date and time format to be displayed in the `DateTimePicker` control in a report.

Format | Display in `DateTimePicker`

Short Date and Time - d/M/yy h:mm tt | 9/12/2014 2:04 PM

Medium Date - d MMM yy h:mm tt | 12 Sep 14 2:04: PM

Full Date and short time - dddd, MMMM dd, yyyy HH:mm tt | Friday, September 12,2014 2:04 PM

Full Date and Long Time - dddd, MMMM dd, yyyy HH:mm:ss tt | Friday, September 12,2014 2:04:00 PM

UTC - yyyy-MM-dThh:mm:ssz | 2014-09-12T2:04:00+5

Refer to the following code sample to set date and time format to be displayed in the report viewer initialization.

```
`js
```



```
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/product-line-sales.rdl">
</Bold:ReportViewer>
</div>
`
```

Refer to the following code sample to set data range using **parameterSettings** in Report Viewer controller.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.ParameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
this.viewer.ParameterSettings.DateTimeFormat = "d/M/yyyy h:mm tt";
this.viewer.ParameterSettings.TimeDisplayFormat = "HH:mm";
this.viewer.ParameterSettings.TimeInterval = 60;
}
`
```

The above code sets date and time value to be display for all the date parameters used in the report.

To set different date and time value to be display for each date parameter used in the report, register the event **beforeParameterAdd** and specify date and time value based on parameter name as in below code sample.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/product-line-sales.rdl"
OnClientBeforeParameterAdd="beforeParameterAdd" >
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function beforeParameterAdd(args) {
if (args.parameterModel.Name === "StartDate") {
args.parameterSettings.dateTimeFormat = "d/M/yyyy h:mm tt";
```

```

args.parameterSettings.timeDisplayFormat = "HH:mm";
args.parameterSettings.timeInterval = 60;
}
if (args.parameterModel.Name === "EndDate") {
args.parameterSettings.dateTimeFormat = "d/M/yyyy h:mm tt";
args.parameterSettings.timeDisplayFormat = "HH:mm";
args.parameterSettings.timeInterval = 60;}
}
}
</script>

```

Change the Parameter drop-down height and width

The `parameterSettings` helps you to change the height and width of the parameter available in parameter panel.

```

`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.ParameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
this.viewer.ParameterSettings.PopupHeight = "100px";
this.viewer.ParameterSettings.PopupWidth = "100px";
}

```

Hide a Parameter scroller

The `enableparameterblockscroller` helps you to hide the scrollbar in parameter panel.

```

`html
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
EnableParameterBlockScroller="false" >
</Bold:ReportViewer>

```

Hide a Parameter Pane on load

The `parameterSettings` helps you to hide and show the parameter block.

```

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

```

```
{
this.viewer.ParameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
this.viewer.ParameterSettings.HideParameterBlock = true;
}
`
```

Change the Parameter Item Width and Label Width

The `parameterSettings` helps you to change the parameter Item width and label width.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.ParameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
this.viewer.ParameterSettings.ItemWidth = "200px";
this.viewer.ParameterSettings.LabelWidth = "auto";
}
`
```

Access the hidden or internal parameter information

The `accessInternalValue` property in the `parameterSettings` helps you to expose the `hidden` or `internal` report parameter information used in report to the user.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.ParameterSettings = new BoldReports.Models.ReportViewer.ParameterSettings();
this.viewer.ParameterSettings.AccessInternalValue = true;
}
`
```

Set the report parameter visibility in Web API Controller

The `Hidden` property of `ReportParameter` allows you to show or hide the parameter at the top of the report viewer panel. The following code example shows hiding a report parameter in the Web API controller's `OnReportLoaded` method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
var reportParameters = ReportHelper.GetParameters(jsonArray, this);
}
```

```

List<BoldReports.Web.ReportParameter> modifiedParameters = new
List<BoldReports.Web.ReportParameter>();
if (reportParameters != null)
{
    foreach (var rptParameter in reportParameters)
    {
        modifiedParameters.Add(new BoldReports.Web.ReportParameter()
        {
            Name = rptParameter.Name,
            Hidden = true
        });
    }
    reportOption.ReportModel.Parameters = modifiedParameters;
}
}
,

```

Extensions

See Also

- [Custom Data Extension](#)

Custom Data Extension

This section explains the steps required to create and load data extensions in Web ASP.NET Web Forms Application

See Also

- [Configure Data Extension](#)

Configure a new data source extension in ASP.NET Web Forms Report Viewer

The ASP.NET Web Forms Report Viewer provides SQL, ODBC and OLEDB data sources as built in support data sources. Other data source like Web API, JSON, XML, and OData are provided as extension data sources.

This documentation provides step by step procedure to register and connect with new extension data sources in Report Viewer Application.

Create ASP.NET Web Forms Report Viewer Application

Refer [Getting Started](#) and create a ASP.NET Web Forms Report Viewer Application.

Configure a new data source extension in ASP.NET Web Forms Report Viewer extension from NuGet

Install data source

Install data source extension from NuGet

Based on the required data connector install the respective NuGet package to the application. The NuGet packages name for each data connectors are provided in below table,

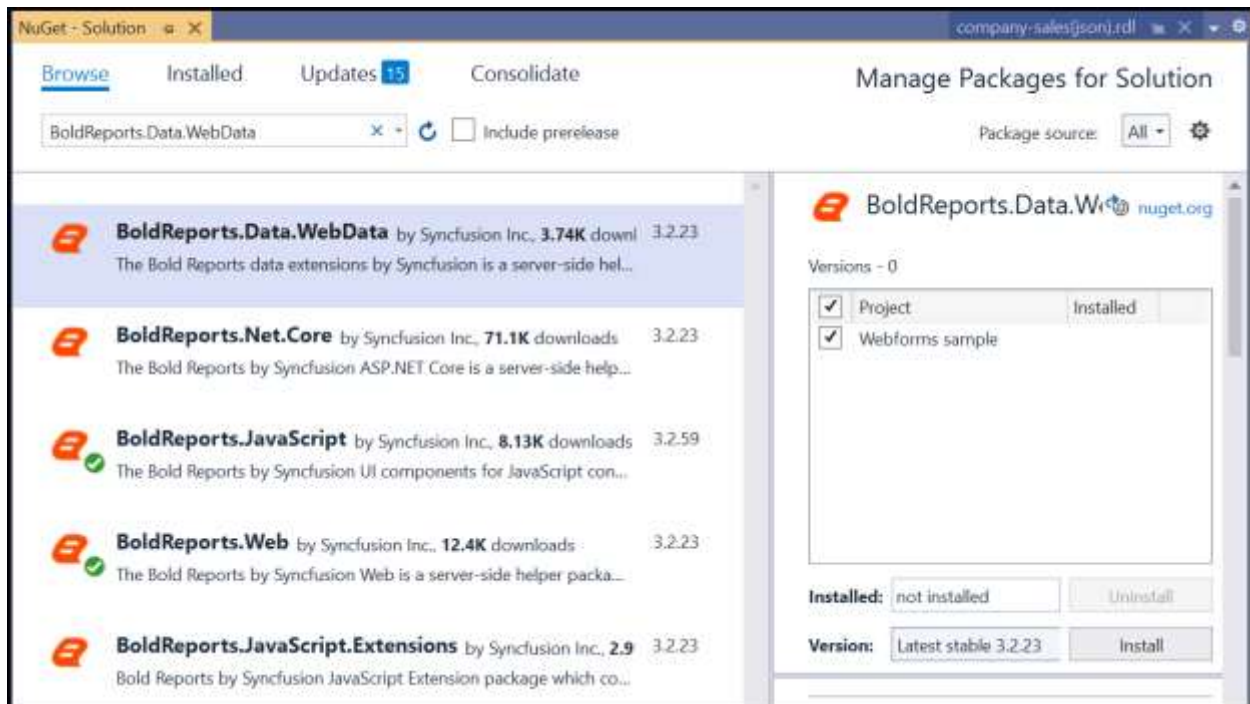
Data source	Package Name	Assembly Name
-----	-----	-----
Web data sources(WebAPI, JSON, XML, and OData)	BoldReports.Data.WebData	BoldReports.Data.WebData.dll
PostgreSQL data sources	BoldReports.Data.PostgreSQL	BoldReports.Data.PostgreSQL.dll
CSV data sources	BoldReports.Data.Csv	BoldReports.Data.Csv.dll
Excel data sources	BoldReports.Data.Excel	BoldReports.Data.Excel.dll
MySQL data sources	BoldReports.Data.MySQL	BoldReports.Data.MySQL.dll
Oracle data sources	BoldReports.Data.Oracle	BoldReports.Data.Oracle.dll

For example, to register and load web data sources in the application install *BoldReports.Data.WebData* package.

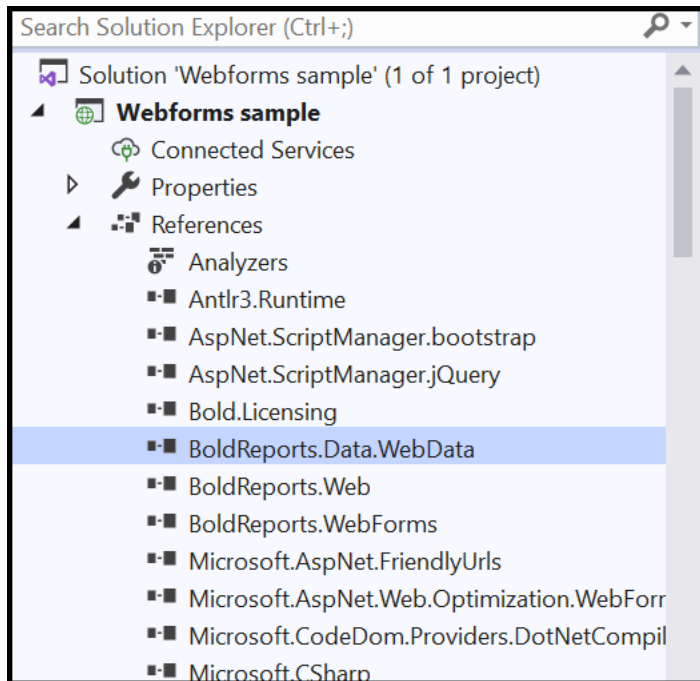
Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring NuGet packages.

Search for **BoldReports.Data.WebData** NuGet package, and install it in your application.



BoldReports.Data.WebData will install into your application. Click OK. Now, the assembly will be added in the respective project references.



Register data source extension

1. Open the code-behind file `WebApiConfig.cs` and add the following using statement.

```
`csharp
using BoldReports.Web;
`
```

2. Then add the following code to register extension assembly in `Register` method.

```
`csharp
public static void Register(HttpConfiguration config)
{
    //Use the below code to register extensions assembly into report viewer
    ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {
        "BoldReports.Data.WebData" });

    //To register multiple data extensions, provide the assembly name's as list of strings. For example:
    "ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {
        "BoldReports.Data.WebData", "BoldReports.Data.Excel"});"

    //Incase the data source extensions fails to register or any error occurs replace the code as below,
    "ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string>
    {System.IO.Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory) +
    "BoldReports.Data.WebData.dll" });"

    ...
}
```

```
...
}
`
```

Run the application

1. Run the ASP.NET Web Forms Report Viewer Application
2. Now we can able to see the company sales report rendered with **JSON** data

Application name Home About Contact

1 / 2

Company Sales

		2002				2003		
		Q1	Q2	Q3	Q4	Q1	Q2	Q3
Components	Bottom Brackets							\$17,454
	Brakes							\$26,059
	Chains							\$3,476
	Cranksets							\$80,244
	Derailleurs							\$25,385
	Forks			\$26,167	\$23,543	\$9,914	\$18,345	
	Handlebars			\$35,341	\$18,309	\$6,275	\$17,194	\$43,824
	Headsets			\$19,702	\$16,382	\$10,950	\$14,102	
	Mountain Frames	\$127,558	\$220,935	\$808,353	\$443,599	\$236,070	\$440,261	\$827,268
	Pedals							\$94,185
Road Frames	\$47,486	\$155,311	\$957,715	\$456,089	\$132,692	\$457,689	\$755,821	
Saddles							\$24,998	
Touring Frames							\$666,977	
Clothing	Wheels			\$266,626	\$163,922	\$63,186	\$163,630	\$1,676
	Bik-Shorts			\$66,860	\$35,323	\$21,544	\$43,458	\$351
	Caps	\$921	\$1,479	\$3,991	\$3,070	\$1,782	\$2,940	\$8,676

Download the custom data extension configured Report Viewer sample from the link [Sample](#)

Report interaction events

You can handle the report interaction events with reports using the following events.

- **ReportLoaded**
- **ReportError**
- **ShowError**
- **Drill through**
- **Hyperlink**

Report loaded

The **OnClientReportLoaded** event occurs after the report is loaded and it is ready to start the processing. You can handle the event and specify the data source and parameters at client side. The following sample code loads a report by assigning the report data source input in the **OnClientReportLoaded** event.

In this tutorial, the **product-list.rdlc** report is used. You can add the report from the Bold Reports installation location. For more information, refer to [Samples and demos](#).

```
`js
```

```

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
<link href="Content/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="Scripts/jquery-3.3.1.min.js"></script>
<script src="Scripts/common/bold.reports.common.min.js"></script>
<script src="Scripts/common/bold.reports.widgets.min.js"></script>
<script src="Scripts/data-visualization/ej.chart.min.js"></script>
<script src="Scripts/bold.report-viewer.min.js"></script>
<div style="height: 600px; width: 100%; min-height: 404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ProcessingMode="Local"
ReportPath="~/Resources/product-list.rdlc"
OnClientReportLoaded="onReportLoaded">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onReportLoaded(args) {
var dataSource = [
{
ProductName: "Baked Chicken and Cheese", OrderId: "323B60", Price: 55, Category: "Non-Veg",
Ingredients: "Grilled chicken, Corn and Olives.", ProductImage: ""
},
{
ProductName: "Chicken Delite", OrderId: "323B61", Price: 100, Category: "Non-Veg", Ingredients:
"Cheese, Chicken chunks, Onions & Pineapple chunks.", ProductImage: ""
},
{
ProductName: "Chicken Tikka", OrderId: "323B62", Price: 64, Category: "Non-Veg", Ingredients: "Onions,
Grilled chicken, Chicken salami & Tomatoes.", ProductImage: ""
}
];
var reportObj = $('#MainContent_viewer').data("boldReportViewer");
reportObj.model.dataSources = [{
value: ej.DataManager(dataSource).executeLocal(ej.Query()),
name: "list"
}

```



```

});
}
</script>
</asp:Content>
`

```

Report error

When an error occurs in the report processing, it raises the `OnClientReportError` event. You can handle the event and show the details in your custom dialog instead of component default error detail interface.

```

`js
<div style="height: 600px; width: 100%; min-height: 404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ProcessingMode="Local"
<!--ReportPath="~/Resources/product-list.rdlc"--%>
OnClientReportError="onReportError">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onReportError(args) {
alert(args.errmsg);
args.cancel = true;
}
</script>
`

```

To suppress the default error dialog, set the cancel argument to true.

Show error

The `OnClientShowError` event is invoked whenever users click a report item that contains an error in processing. It provides detailed information about the cause of error. You can hide the default dialog and show your customized dialog.

```

`js
<div style="height: 600px; width: 100%; min-height: 404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ProcessingMode="Local"

```

```

<!--ReportPath=~\Resources/product-list.rdlc-->
OnClientShowError="onShowError">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onShowError(args) {
alert("Error code : " + args.errorCode + "\n" +
"Error Detail : " + args.errorDetail + "\n" +
"Error Message : " + args.errorMessage);
args.cancel = true;
}
</script>
`

```

Drill through

When a drill through item is selected in a report, it invokes the **OnClientDrillThrough** event. You can change the drill through arguments such as report parameter and data sources. The following sample code can be used to change the drill through report name and set the parameter value before the drill through report is rendered.

```

`js
<div style="height: 600px; width: 100%; min-height: 404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath=~\Resources/sales-person-details.rdl"
OnClientDrillThrough="onDrillThrough">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onDrillThrough(args) {
args.actionInfo.ReportName = "personal-details";
args.actionInfo.Parameters = [{ name: 'EmployeeID', value: ['3'] }];
}
</script>
`

```

Hyperlink

The **OnClientHyperlink** event occurs when users click a hyperlink in a report, before the hyperlink is followed. The following sample code redirects to a new custom URL and cancels the component default action.

```
`js
<div style="height: 600px; width: 100%; min-height: 404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/customer-support-analysis.rdl"
OnClientHyperlink="onHyperlink">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onHyperlink(args) {
args.cancel = true;
//You can modify the URL here
window.open(args.actionInfo.Hyperlink);
}
</script>
`
```

IReportController

The **IReportController** interface has the declaration of action methods that is defined in Web API Controller to process the RDL, RDLC, SSRS report and handling request from Report Viewer control. The IReportController has the following action methods declaration.

Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

```
`csharp
```

```
public class ReportsWebApiController: ApiController,IReportController
{
/// <summary>
/// Action (HttpGet) method for getting resource for report.
/// </summary>
/// <param name="key">The unique key to get the required resource.</param>
```

```
/// <param name="resourceType">The type of the requested resource.</param>
/// <param name="isPrinting">If set to <see langword="true"/>, then the resource is generated for
printing.</param>
/// <returns>The object data.</returns>
public object GetResource(string key, string resourceType, bool isPrinting)
{
    //Returns the report resource for the requested key.
    return ReportHelper.GetResource(key, resourceType, isPrinting);
}
/// <summary>
/// Report Initialization method that is triggered when report begin processed.
/// </summary>
/// <param name="reportOptions">The Report Viewer options.</param>
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOptions)
{
    //You can update report options here
}
/// <summary>
/// Report loaded method that is triggered when report and sub report begins to be loaded.
/// </summary>
/// <param name="reportOptions">The Report Viewer options.</param>
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOptions)
{
    //You can update report options here
}
/// <summary>
/// Action (HttpPost) method for posting the request for report process.
/// </summary>
/// <param name="jsonData">The JSON data posted for processing report.</param>
/// <returns>The object data.</returns>
public object PostReportAction(Dictionary < string, object > jsonData)
```

```
{
//Processes the report request and returns the result.
return ReportHelper.ProcessReport(jsonData, this);
}
}
```

Handle post actions

Report processing actions are sent in an Ajax request to exchange data with the Web API service. You can handle post actions event to customize the Ajax requests.

- AjaxBeforeLoad
- AjaxSuccess
- AjaxError

AjaxBeforeLoad

This event can be triggered before an Ajax request is sent to the Report Viewer Web API service. It allows you to set additional headers and custom data in the Ajax request. The following code sample demonstrates adding custom authorization header and passing default parameter values to service.

Add custom header to Ajax request

Initialize the `OnClientAjaxBeforeLoad` event in the script and add the authorization token to the `headers` property.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
OnClientAjaxBeforeLoad="onAjaxRequest">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
}
</script>
`
```

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded from [here](#).

Get the custom header value from the `HttpContext` header collection using the key name specified at client side.

```
`csharp
string authenticationHeader;
public object PostReportAction(Dictionary<string, object> jsonResult)
{
    if (jsonResult != null)
    {
        //Get client side custom ajax header and store in local variable
        authenticationHeader = HttpContext.Current.Request.Headers["Authorization"];
        //Perform your custom validation here
        if (authenticationHeader == "")
        {
            return new Exception("Authentication failed!!!");
        }
        else
        {
            return ReportHelper.ProcessReport(jsonResult, this);
        }
    }
    return null;
}
```

Perform your own action to validate the header values.

Pass custom data in Ajax request

Use the `data` property to set custom data to the server in the Ajax request. In the following code sample, parameter values are passed to the server side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
OnClientAjaxBeforeLoad="onAjaxRequest">
</Bold:ReportViewer>
```

```
</div>
<script type="text/javascript">
function onAjaxRequest(args) {
//Passing custom data to server
var customerID = "CI0021";
args.data = customerID;
}
</script>
`
```

The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates to change the datasource connection strings based on Customer ID in the `OnInitReportOptions` method.

```
`csharp
string customerID = null;
public object PostReportAction(Dictionary<string, object> jsonResult)
{
if (jsonResult != null)
{
if (jsonResult.ContainsKey("customData"))
{
//Get client side custom data and store in local variable
customerID = jsonResult["customData"].ToString();
}
}
return ReportHelper.ProcessReport(jsonResult, this);
}
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
if (customerID != null)
{
if(customerID.Contains("CI0021"))
{
```

//If you are changing the connection string based on customer id then could you please change the connection string as below.

```
//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>;"));

}
```

```
else if(customerID.Contains("CI0022"))
```

```
{
```

//If you are changing the connection string based on customer id then could you please change the connection string as below.

```
//reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
Catalog=<database>;"));

}
```

```
}
```

```
}
```

```
,
```

AjaxSuccess

To perform custom action or show user defined message, use the `OnClientAjaxSuccess` event on the successful Ajax request.

```
`js
```

```
<div style="height: 650px;width: 950px;min-height:404px;">
```

```
<Bold:ReportViewer runat="server" ID="viewer"
```

```
ReportServiceUrl="/api/ReportViewer"
```

```
ReportPath="~/Resources/sales-order-detail.rdl"
```

```
OnClientAjaxBeforeLoad="onAjaxRequest"
```

```
OnClientAjaxSuccess="onAjaxSuccess">
```

```
</Bold:ReportViewer>
```

```
</div>
```

```
<script type="text/javascript">
```

```
function onAjaxRequest(args) {
```

```
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
```



```
//Passing custom parameter data to server
args.data = [{ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'] }];
}
function onAjaxSuccess(args) {
//Perform your custom success message here
alert("Ajax request success!!!");
}
</script>
`
```

AjaxError

The `OnClientAjaxError` event is called, if an error occurred with the request, you can display the customized error detail in the event method.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
OnClientAjaxBeforeLoad="onAjaxRequest"
OnClientAjaxError="onAjaxFailure">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onAjaxRequest(args) {
args.headers.push({ Key: 'Authorization', Value: btoa('guest', 'demo@123') });
//Passing custom parameter data to server
args.data = [{ name: 'SalesOrderNumber', labels: ['SO50756'], values: ['SO50756'] }];
}
function onAjaxFailure(args) {
alert("Status: " + args.status + "\n" +
"Error: " + args.responseText);
}
</script>
`
```

You can never have both an error and a success callback with a request.

Change Report Viewer default WEB API action with custom endpoint

The `actionName` event argument allows to change the methods of `IReportController` to custom WEB API action endpoints.

Change report processing endpoint

Create a new Web Api action method in Report Viewer API controller as in the following code snippet,

```
`csharp
public object PostReportCustomAction(Dictionary<string, object> jsonResult)
{
    return ReportHelper.ProcessReport(jsonResult, this);
}
`
```

The custom method must have the `Dictionary<TKey, TValue>` argument and add code `ReportHelper.ProcessReport` for processing the report.

Register the event `ajaxBeforeLoad` in your html page and set the newly created name to `actionName` property as in the below code snippet.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
OnClientAjaxBeforeLoad="onAjaxRequest">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onAjaxRequest(args) {
args.actionName = "PostReportCustomAction";
}
</script>
`
```

Change export action endpoint

Create a new Web Api action method in Report Viewer API controller as in the following code snippet,

```
`csharp
public object ExportReportCustomAction()
{

```

Error logging in ASP.NET Web Forms Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

```
return ReportHelper.ProcessReport(null, this);  
}  
`
```

The custom method must have the code `ReportHelper.ProcessReport` for exporting the report.

Register the event `onExportProgressChanged` in your html page and set the newly created name to `actionName` property as in the below code snippet.

```
`js  
<div style="height: 650px;width: 950px;min-height:404px;">  
<Bold:ReportViewer runat="server" ID="viewer"  
ReportServiceUrl="/api/ReportViewer"  
ReportPath="~/Resources/sales-order-detail.rdl"  
OnClientExportProgressChanged="onExportProgressChanged">  
</Bold:ReportViewer>  
</div>  
<script type="text/javascript">  
function onExportProgressChanged(args) {  
if(args.stage === "exportStarted"){  
args.actionName = "ExportReportCustomAction";  
}  
}  
</script>  
`
```

Error logging in ASP.NET Web Forms Report Viewer

If an error occurred in report processing, ASP.NET Web Forms Report Viewer displays short messages about the error in view. You need to configure the `ApiController` to save all logs, stack trace, and error information.

This section explains how to log the detailed error information to your ASP.NET Web Forms application.

This section requires a ASP.NET Web Forms Report Viewer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

1. In Solution Explorer, open the Report Viewer Controller file.
2. Inherit the `IReportLogger` interface and implement the interface methods.

```
`csharp  
public class ReportViewerController : ApiController, IReportController, IReportLogger
```

Error logging in ASP.NET Web Forms Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

```
{  
    public void LogError(string message, Exception exception, MethodBase methodType, ErrorType  
        errorType)  
    {  
        throw new NotImplementedException();  
    }  
  
    public void LogError(string errorCode, string message, Exception exception, string errorDetail, string  
        methodName, string className)  
    {  
        throw new NotImplementedException();  
    }  
}
```

3. Create a method in **ReportViewerController** to write the error text into application folder.

```
`csharp  
internal void WriteLogs(string errorMessage)  
{  
    string filePath = HttpContext.Current.Server.MapPath("/App_Data/ErrorDetails.txt");  
    using (StreamWriter writer = new StreamWriter(filePath, true))  
    {  
        writer.AutoFlush = true;  
        writer.WriteLine(errorMessage);  
    }  
}
```

4. Invoke the newly created function in **LogError** as follows.

```
`csharp  
public void LogError(string message, Exception exception, MethodBase methodType, ErrorType  
    errorType)  
{  
    WriteLogs(string.Format("Error Message: {0} \n Stack Trace: {1}", message, exception.StackTrace));  
}
```

Error logging in ASP.NET Web Forms Report Viewer **Change** Report Viewer default WEB API action with custom endpoint

```
public void LogError(string errorCode, string message, Exception exception, string errorDetail, string
methodName, string className)
```

```
{
```

```
WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2} \n Stack Trace:
{3}", className, methodName, errorDetail, exception.StackTrace));
```

```
}
```

```
,
```

In cases of any issues faced in the report rendering, share the log file to our technical support team to get assistance on that.

5. The final controller is given as follows, you can replace it in your application.

```
`csharp
```

```
public class ReportViewerController : ApiController, IReportController, IReportLogger
```

```
{
```

```
// Post action for processing the RDL/RDLC report
```

```
public object PostReportAction(Dictionary<string, object> jsonResult)
```

```
{
```

```
return ReportHelper.ProcessReport(jsonResult, this);
```

```
}
```

```
// Get action for getting resources from the report
```

```
[System.Web.Http.ActionName("GetResource")]
```

```
[AcceptVerbs("GET")]
```

```
public object GetResource(string key, string resourcetype, bool isPrint)
```

```
{
```

```
return ReportHelper.GetResource(key, resourcetype, isPrint);
```

```
}
```

```
// Method that will be called when initialize the report options before start processing the report
```

```
[NonAction]
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
// You can update report options here
```

```
}
```

```
// Method that will be called when reported is loaded
```

```
[NonAction]
```

```

public void OnReportLoaded(ReportViewerOptions reportOption)
{
    // You can update report options here
}

public void LogError(string message, Exception exception, MethodBase methodType, ErrorType
errorType)
{
    WriteLogs(string.Format("Error Message: {0} \n Stack Trace: {1}", message, exception.StackTrace));
}

public void LogError(string errorCode, string message, Exception exception, string errorDetail, string
methodName, string className)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2} \n Stack Trace:
{3}", className, methodName, errorDetail, exception.StackTrace));
}

internal void WriteLogs(string errorMessage)
{
    // Error details text file path location
    string filePath = HttpContext.Current.Server.MapPath("/App_Data/Errordetails.txt");
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
        writer.WriteLine(errorMessage);
    }
}

```

Print report

The Report Viewer provides print report option in the toolbar to print a copy of the report. The page setup dialog allows you to set the paper size or other page setup properties. To see print margins, click **Print Layout** on the toolbar.

You can set values in the Page Setup dialog box for current session only. When you close the report and reopen it, it will have the default values again. The default values of the Page Setup dialog is based on the report properties set in the design view.

View report in print mode

Print margins are displayed in the print layout only. To view report in print mode by default, set the `PrintMode` property to true.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
PrintMode="true">
</Bold:ReportViewer>
</div>
`
```

By default, the Report Viewer renders report in normal layout in which the print margins are not displayed.

Print in new page

To open the print in a new tab of the current browser, set the `PrintOption` property to `NewTab`. By default, it shows the print dialog in the same page.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
PrintMode="true"
PrintOption="NewTab">
</Bold:ReportViewer>
</div>
`
```

The pop-up blocker should be enabled for the page to open the print view in new tab.

Set page orientation and paper size

You can specify the print page paper size and orientation at client-side to change the page setup properties by setting the `page-settings` property.

1. The following code example illustrates how to set page orientation and paper size in the Report Viewer for client side.

```
`js
```

```
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`
```

2. The following code example illustrates how to set page orientation and paper size in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.PageSettings = new BoldReports.Models.ReportViewer.PageSettings();
this.viewer.PageSettings.Orientation = BoldReports.ReportViewerEnums.Orientation.Landscape;
this.viewer.PageSettings.PaperSize = BoldReports.ReportViewerEnums.PaperSize.Letter;
}
`
```

Set report margin

To set margin values to the report page setup, use the `Margins` property and specify the value to top, right, bottom, and left.

1. The following code example illustrates how to set report margin in the Report Viewer for client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`
```

2. The following code example illustrates how to set report margin in your application's `Page_Load()` function.


```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
    this.viewer.PageSettings = new BoldReports.Models.ReportViewer.PageSettings();
    this.viewer.PageSettings.Margins = new BoldReports.Models.ReportViewer.Margins();
    this.viewer.PageSettings.Margins.Top = 0.5;
    this.viewer.PageSettings.Margins.Bottom = 0.5;
    this.viewer.PageSettings.Margins.Right = 0.5;
    this.viewer.PageSettings.Margins.Left = 0.5;
}
`
```

The values set in the margin property is considered as inches input.

Set page height and width

To set height and width values to the report page setup, use the **Height** and **Width** properties.

1. The following code example illustrates how to set page height and width in the Report Viewer for client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`
```

2. The following code example illustrates how to set page height and width in your application's **Page_Load()** function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
    this.viewer.PageSettings = new BoldReports.Models.ReportViewer.PageSettings();
    this.viewer.PageSettings.Height = 11.69;
    this.viewer.PageSettings.Width = 8.27;
}
```

```
}
,
```

The values set in the height and width properties are considered as inches input.

Print report with images

When the report has more images, the browser will send the report stream to the print dialog before the images are completely loaded. To load the print report stream with complete images, you should set the `EmbedImageData` property to true in `OnInitReportOptions` as shown in the following code.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.EmbedImageData = true;
}
,
```

Replace the following code sample in the client-side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/product-details.rdl">
</Bold:ReportViewer>
</div>
,
```

In this tutorial, the `product-details.rdl` report is used, and it can be downloaded from [here](#).

External styles in report printing

While printing report, the external styles are used in the application overrides printable page style and prints output with incorrect alignments. To avoid the external script overriding, set the `isStyleLoad` property to false, which will print the page using only the Report Viewer styles.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/product-details.rdl"
OnClientReportPrint="onReportPrint">
</Bold:ReportViewer>
```

```

</div>
<script type="text/javascript">
function onReportPrint(args) {
args.isStyleLoad = false;
}
</script>
`

```

Show print progress

Report Viewer provides events to show the progress information when the printing takes long time to complete.

To show print progress, follow these steps:

1. Set the `OnClientPrintProgressChanged` in Report Viewer initialization.
2. Implement the function and add code samples to show a custom message based on the print progress status as shown in the following code snippet.

```

`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/product-details.rdl"
OnClientPrintProgressChanged="onPrintProgressChanged">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onPrintProgressChanged(args) {
if (args.stage == "beginPrint") {
$('#MainContent_viewer').ejWaitingPopup({ showOnInit: true, cssClass: "customStyle", text: "Preparing
print data.. Please wait..." });
}
if (args.stage == "printStarted") {
var popupObj = $('#MainContent_viewer').data('ejWaitingPopup');
popupObj.hide();
}
else if (args.stage == "preparation") {
console.log(args.stage);
}
}

```

```

if (args.preparationStage == "dataPreparation") {
  console.log(args.preparationStage);
  console.log(args.totalPages);
  console.log(args.currentPage);
  if (args.totalPages > 1 && args.currentPage > 1) {
    var progressPercentage = Math.floor((args.currentPage / args.totalPages) * 100);
    if (progressPercentage > 0) {
      var popupObj = $('#MainContent_viewer').data('ejWaitingPopup');
      popupObj.setModel({ text: "Preparing print data.." + progressPercentage + " % completed.. Please wait..." });
    }
  }
}
args.handled = true;
}
</script>
`

```

Remove empty spaces in printing

The extra blank page is created when the body of your report is too wide for your page. To make the report appear on a single page, all the content within the report body must fit on the physical page, and the body width should be as the following formula.

Body Width <= Page Width - (Left Margin + Right Margin)

For more details about removing the empty pages in the report while designing, refer to the knowledge base article of [report page sizing](#).

Export report

The Report Viewer provides events and properties to control and customize the report exporting functionality.

Export event handling

You can show the progress information, when the exporting process takes long time to complete using the `OnClientExportProgressChanged` event.

1. Set the `OnClientExportProgressChanged` event in Report Viewer initialization.
2. Implement the function and replace the following code samples to show a custom message based on the progress stage.

The following code example demonstrates how to export event handling in the Report Viewer at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
OnClientExportProgressChanged="onExportProgressChanged">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onExportProgressChanged(args) {
if (args.stage === "beginExport") {
console.log(args.stage);
args.format =
$('#MainContent_viewer').ejWaitingPopup({ showOnInit: true, cssClass: "customStyle", text: "Preparing
exporting document.. Please wait..." });
}
else if (args.stage === "exportStarted") {
console.log(args.stage);
var popupObj1 = $('#MainContent_viewer').data('ejWaitingPopup');
popupObj1.hide();
}
else if (args.stage === "preparation") {
console.log(args.stage);
console.log(args.format);
console.log(args.preparationStage);
if (args.format === "PDF" && args.preparationStage === "documentPreparation") {
console.log(args.totalPages);
console.log(args.currentPage);
if (args.totalPages > 1 && args.currentPage > 1) {
var progressPercentage = Math.floor((args.currentPage / args.totalPages) * 100);
if (progressPercentage > 0) {
var popupObj2 = $('#MainContent_viewer').data('ejWaitingPopup');
```

```

popupObj2.setModel({ text: "Preparing exporting document.." + progressPercentage + " % completed..
Please wait..." });
}
}
}
}
args.handled = true;
}
</script>
`

```

Change Excel and Word export format

Allows you change the default file format to any other file format using the **ExcelFormat** and **WordFormat** properties.

1. The following code example demonstrates how to change Excel and Word export format in the Report Viewer at client side.

```

`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`

```

2. The following code example demonstrates how to change Excel and Word export format in your application's **Page_Load()** function.

```

`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.ExportSettings = new BoldReports.Models.ReportViewer.ExportSettings();
this.viewer.ExportSettings.ExcelFormat = BoldReports.ReportViewerEnums.ExcelFormats.Excel2013;
this.viewer.ExportSettings.WordFormat = BoldReports.ReportViewerEnums.WordFormats.Word2013;
}
`

```

Hide specific export type for report

Show or hide the default export types available in the component using the `ExportOptions` property.

1. The following code example demonstrates how to hide specific export type to the report in the Report Viewer at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`
```

2. The following code example demonstrates how to hide specific export type to the report in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.ExportSettings = new BoldReports.Models.ReportViewer.ExportSettings();
this.viewer.ExportSettings.ExportOptions = BoldReports.ReportViewerEnums.ExportOptions.All &
~BoldReports.ReportViewerEnums.ExportOptions.Html;
}
`
```

PDF export options

The `PDFOptions` provides properties to manage PDF export behaviors. You should set the properties in the `OnInitReportOptions` method of the Web API service.

Export with complex scripts

To export reports with the complex scripts, set the `EnableComplexScript` property of `PDFOptions` instance to true.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
{

```

```
EnableComplexScript = true
```

```
};
```

```
}
```

```
,
```

PDF conformance

You can export the report as a PDF/A-1b document by specifying the PdfConformanceLevel.Pdf_A1B conformance level in the PdfConformanceLevel property.

```
`csharp
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
```

```
{
```

```
PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B
```

```
};
```

```
}
```

```
,
```

Add custom PDF fonts

You can add custom fonts to the PDF exported document by adding the font streams to Fonts collection in PDFOptions instance.

To add custom fonts to the PDF exported document, follow these steps:

1. Add the font .ttf files into your application Resources folder.
2. In the Solution Explorer, open the properties of the font file and set the Copy property to Output Directory as Copy always.
3. Initialize the Font collection and add the font stream to it.

The key value provided in the font collection should be same as in the report item font property.

```
`csharp
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
string basePath = _hostingEnvironment.WebRootPath;
```

```
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions()
```

```
{
```

```
//Load Missing font stream
```

```
Fonts = new Dictionary<string, System.IO.Stream>
```

```
{
```



```
{ "Segoe UI", new FileStream(basePath + @"\Resources\font_symbols.ttf", FileMode.Open,
FileAccess.Read) }
}
};
}
,
```

If any fonts used in the report definition is not installed or available in the local system, then you should load the font stream. In the above code, `font_symbols` font stream is loaded to export the `sales-order-detail.rdl` report as symbols.

Word export options

The `WordOptions` provides properties to manage Word document export behaviors.

Word document type

You can save the report to the required document version by setting the `FormatType` property.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
    {
        FormatType = BoldReports.Writer.WordFormatType.Docx
    };
}
,
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
    {
        LayoutOption = BoldReports.Writer.WordLayoutOptions.TopLevel,
        ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
        {
            Bottom = 0.5f,

```

Top = 0.5f

}

};

}

,

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior.

The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, add an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without password. The following are the types of protection:

- **AllowOnlyComments**: Adds or modifies only the comments in the Word document.
- **AllowOnlyFormFields**: Modifies the form field values in the Word document.
- **AllowOnlyRevisions**: Accepts or rejects the revisions in the Word document.
- **AllowOnlyReading**: Views the content only in the Word document.
- **NoProtection**: Accesses or edits the Word document contents as normally.

```
`csharp
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
    {
        ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading
    };
}
```

Excel export options

The **ExcelOptions** provides properties to manage Excel document export behaviors.

Excel document type

You can save the report to the required excel version by setting the **ExcelSaveType** property.

```
`csharp
```

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
```

```
{
ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013
};
}
`
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` to `IgnoreCellMerge`.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
};
}
`
```

Protecting Excel document from editing

You can restrict the Excel document from editing either by providing the `ExcelSheetProtection` or enabling the `ReadOnlyRecommended` properties.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
ReadOnlyRecommended = true,
ExcelSheetProtection = Syncfusion.XlsIO.ExcelSheetProtection.DeletingColumns
};
}
`
```

CSV export options

The `CsvOptions` allows you to change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
reportOption.ReportModel.CsvOptions = new BoldReports.Writer.CsvOptions()
{
Encoding = System.Text.Encoding.Default,
FieldDelimiter = ",",
UseFormattedValues = false,
Qualifier = "#",
RecordDelimiter = "@",
SuppressLineBreaks = true,
FileExtension = ".txt"
};
}
```

[Password protect exported document](#)

Allows you protect the exported document such as PDF, Word, Excel, and PowerPoint from unauthorized users by encrypting the document using encryption password. The following code snippet illustrates how to encrypt the exported document with user-defined password.

```
`csharp
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
//PDF encryption
reportOption.ReportModel.PDFOptions = new BoldReports.Writer.PDFOptions();
reportOption.ReportModel.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity()
{
UserPassword = "password"
};
//Word encryption
reportOption.ReportModel.WordOptions = new BoldReports.Writer.WordOptions()
{
EncryptionPassword = "password"
};
//Excel encryption
reportOption.ReportModel.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
```

```

PasswordToModify = "password",
PasswordToOpen = "password"
};
//PPT encryption
reportOption.ReportModel.PPTOptions = new BoldReports.Writer.PPTOptions()
{
EncryptionPassword = "password"
};
}
`

```

Password protection is not supported for HTML export format.

Toolbar customization

You can hide the component toolbar to show customized user interface or to customize the toolbar icons and element's appearances using the templates and Report Viewer toolbar customization properties.

In this tutorial, the `sales-order-detail.rdl` report is used and it can be downloaded from [here](#). You can add the reports from the Syncfusion installation location. For more information, refer to [Samples and demos](#).

Hide toolbar items

To hide toolbar items, set the `ToolbarSettings` property. The following code can be used to remove the parameter option from the toolbar and hide the parameter block.

Similarly, you can show or hide all other toolbar options with the help of [toolbarSettings.items](#) enum.

1. The following code example demonstrates how to hide the parameter block in the Report Viewer at client side.

```

`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`

```

2. The following code example demonstrates how to hide the parameter block in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
    this.viewer.ToolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
    this.viewer.ToolbarSettings.Items = BoldReports.ReportViewerEnums.ToolbarItems.All &
    ~BoldReports.ReportViewerEnums.ToolbarItems.Parameters;
}
`
```

Enable stop option in toolbar

To enable stop option in toolbar, set the `ToolbarSettings.Items` property to `BoldReports.ReportViewerEnums.ToolbarItems.All`. The following code can be used to enable stop option in toolbar.

1. The following code example demonstrates how to enable stop option in the Report Viewer toolbar at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`
```

2. The following code example demonstrates how to enable stop option in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
    this.viewer.ToolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
    this.viewer.ToolbarSettings.Items = BoldReports.ReportViewerEnums.ToolbarItems.All;
}
`
```

Enable export setup option in toolbar

To enable export setup option in toolbar, set the `ToolbarSettings.Items` property to `BoldReports.ReportViewerEnums.ToolbarItems.All`. The following code can be used to enable export setup option in toolbar.

1. The following code example demonstrates how to enable export setup option in the Report Viewer toolbar at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`
```

2. The following code example demonstrates how to enable export setup option in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.ToolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
this.viewer.ToolbarSettings.Items = BoldReports.ReportViewerEnums.ToolbarItems.All;
}
`
```

Enable search text option in toolbar

To enable search text option in toolbar, set the `ToolbarSettings.Items` property to `BoldReports.ReportViewerEnums.ToolbarItems.All`. The following code can be used to enable search text option in toolbar.

1. The following code example demonstrates how to enable search text option in the Report Viewer toolbar at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
```

```
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`
```

2. The following code example demonstrates how to enable search text option in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
    this.viewer.ToolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
    this.viewer.ToolbarSettings.Items = BoldReports.ReportViewerEnums.ToolbarItems.All;
}
`
```

Hide toolbar

To hide the Report Viewer toolbar, set the `ShowToolbar` property to false.

1. The following code example demonstrates how to hide the toolbar in the Report Viewer at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
`
```

2. The following code example demonstrates how to hide the toolbar in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
    this.viewer.ToolbarSettings = new BoldReports.Models.ReportViewer.ToolbarSettings();
}
```



```
this.viewer.ToolbarSettings.ShowToolbar = false;
}
```

Decide or hide the export option

The Report Viewer provides the `ExportOptions` property to show or hide the default export types available in the component. The following code hides the HTML export type from the default export options.

1. The following code example demonstrates how to decide or hide the export option in the Report Viewer at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl">
</Bold:ReportViewer>
</div>
```

2. The following code example demonstrates how to decide or hide the export option in your application's `Page_Load()` function.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
this.viewer.ExportSettings = new BoldReports.Models.ReportViewer.ExportSettings();
this.viewer.ExportSettings.ExportOptions = BoldReports.ReportViewerEnums.ExportOptions.All &
~BoldReports.ReportViewerEnums.ExportOptions.Html;
}
```

Add custom items to the export drop-down

To add custom items to the export drop-down available in the Report Viewer toolbar, use the property `CustomItems` and provide the JSON array of collection input with the `Index`, `CssClass` name, and `Value` properties. Register the `OnClientExportItemClick` event to handle the custom item actions as given in following code snippet.

You can use the following codes to add custom items to the export drop-down in your application's `Page_Load()` function using `ExportSettings` property.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
    ExportSettings exportSettings = new ExportSettings();
    exportSettings.CustomItems = new List<CustomExportItem>();
    var exportItem1 = new CustomExportItem() { Index = 2, CssClass = "", Value = "Text File" };
    var exportItem2 = new CustomExportItem() { Index = 4, CssClass = "", Value = "DOT" };
    exportSettings.CustomItems.Add(exportItem1);
    exportSettings.CustomItems.Add(exportItem2);
    this.viewer.ExportSettings = exportSettings;
}
`
```

You can use the following codes to create an **OnClientExportItemClick** event at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl" OnClientExportItemClick="onExportItemClick">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
//Export click event handler
function onExportItemClick(args) {
if (args.value === "Text File") {
//Implement the code to export report as Text
alert("Text File export option clicked");
} else if (args.value === "DOT") {
//Implement the code to export report as DOT
alert("DOT export option clicked");
}
}
</script>
`
```

Add custom toolbar item

You can add custom items to Report Viewer toolbar using the `ToolbarSettings` property. You must register the `OnClientToolBarItemClick` event to handle the newly added custom items action.

You can use the following codes to add new toolbar group in your application's `Page_Load()` function using `ToolbarSettings` property.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
    ToolbarSettings toolbarSettings = new ToolbarSettings();
    toolbarSettings.CustomItems = new List<CustomItem>();
    var customItem = new CustomItem()
    {
        GroupIndex = 1,
        Index = 1,
        CssClass = "e-icon e-mail e-reportviewer-icon",
        Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
        Id = "E-Mail",
        Tooltip = new ToolTip() { Header = "E-Mail", Content = "Send rendered report as mail attachment" }
    };
    toolbarSettings.CustomItems.Add(customItem);
    this.viewer.ToolbarSettings = toolbarSettings;
}
`
```

You can use the following codes to create an `OnClientToolBarItemClick` event at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl" OnClientToolBarItemClick="onToolBarItemClick">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onToolBarItemClick(args) {
if (args.value == "E-Mail") {
```

```

alert('Action Triggered');
}
}
</script>
`

```

Add custom item to exiting toolbar group

To add a custom item to existing toolbar group use the property **CustomGroup** in **ToolbarSettings** and provide the JSON array of collection input with the **GroupIndex**, **Items**, **Type**, **CssClass** name, and **Tooltip** properties as given in following code snippet.

You can use the following codes to add custom item to exiting toolbar group in your application's **Page_Load()** function using **ToolbarSettings** property.

```

`csharp
protected void Page_Load(object sender, EventArgs e)
{
    ToolbarSettings toolbarSettings = new ToolbarSettings();
    var groupItem = new List<CustomItem>();
    groupItem.Add(new CustomItem()
    {
        CssClass = "e-icon e-mail e-reportviewer-icon CustomGroup",
        Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
        Id = "CustomGroup",
        Tooltip = new ToolTip() { Header = "CustomGroup", Content = "toolbargroups" }
    });
    groupItem.Add(new CustomItem()
    {
        CssClass = "e-icon e-mail e-reportviewer-icon subCustomGroup",
        Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
        Id = "subCustomGroup",
        Tooltip = new ToolTip() { Header = "subCustomGroup", Content = "subtoolbargroups" }
    });
    toolbarSettings.CustomGroups.Add(new CustomGroup() { Items = groupItem, GroupIndex = 4 });
    this.viewer.ToolbarSettings = toolbarSettings;
}
`

```

You can use the following codes to create an `OnClientToolBarItemClick` event at client side.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl" OnClientToolBarItemClick="onToolBarItemClick">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onToolBarItemClick(args) {
if (args.value === "CustomGroup") {
//Implement the code to CustomGroup toolbar option
alert("CustomGroup toolbar option clicked");
}
if (args.value === "subCustomGroup") {
//Implement the code to subCustomGroup toolbar option
alert("SubCustomGroup toolbar option clicked");
}
}
</script>
`
```

Add new toolbar group

To add new toolbar group and custom items to it, use the property `CustomItems` in `ToolBarSettings` and provide the JSON array of collection input with the `GroupIndex`, `Index` properties. The `CustomItems` must have the properties `Type`, `CssClass` and `Tooltip` as given in following code snippet.

You can use the following codes to add new toolbar group in your application's `Page_Load()` function using `ToolBarSettings` property.

```
`csharp
protected void Page_Load(object sender, EventArgs e)
{
ToolBarSettings toolbarSettings = new ToolBarSettings();
toolbarSettings.CustomItems = new List<CustomItem>();
var customItem = new CustomItem()
{
```

```

GroupIndex = 1,
Index = 1,
CssClass = "e-icon e-mail e-reportviewer-icon",
Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
Id = "E-Mail",
Tooltip = new ToolTip() { Header = "E-Mail", Content = "Send rendered report as mail attachment" }
};
toolbarSettings.CustomItems.Add(customItem);
this.viewer.ToolbarSettings = toolbarSettings;
}
,

```

Custom Actions

Add user defined buttons to the toolbar and invoke custom actions using the **Report Viewer** property. You can create a custom email button to send an email with the rendered report to users.

Add email button

1. Create an email button in the toolbar using the **CustomItems** property with the values such as **groupIndex**, **index**, **itemType**, **cssClass**, and **tooltip**. The **OnClientToolBarItemClick** event triggers when you click the email button.
2. Access the Report Viewer model and create a JSON array for sending requests to the Web API Server. You can use the following codes to create an event with custom action.
3. You can use the following codes to add email button in your application's **Page_Load()** function using **ToolbarSettings** property.

```

`csharp
protected void Page_Load(object sender, EventArgs e)
{
    ToolbarSettings toolbarSettings = new ToolbarSettings();
    toolbarSettings.CustomItems = new List<CustomItem>();
    var customItem = new CustomItem()
    {
        GroupIndex = 1,
        Index = 1,
        CssClass = "e-icon e-mail e-reportviewer-icon",
        Type = BoldReports.ReportViewerEnums.ToolBarItemType.Default,
        Id = "E-Mail",

```

```

Tooltip = new ToolTip() { Header = "E-Mail", Content = "Send rendered report as mail attachment" }
};
toolbarSettings.CustomItems.Add(customItem);
this.viewer.ToolbarSettings = toolbarSettings;
}
,

```

4. You can use the following codes to create an **OnClientToolBarItemClick** event at client side.

```

`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl" OnClientToolBarItemClick="onToolBarItemClick">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onToolBarItemClick(args) {
alert('Action Triggered');
}
</script>
,

```

5. You can use the following codes to invoke custom actions using custom email button at client side.

```

`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl" OnClientToolBarItemClick="onToolBarItemClick">
</Bold:ReportViewer>
</div>
<script type="text/javascript">
function onToolBarItemClick(args) {
if (args.value == "E-Mail") {

```

```

var proxy = $('#viewer').data('boldReportViewer');
var Report = proxy.model.reportPath;
var lastIndex = Report.lastIndexOf("/");
var reportName = Report.substring(lastIndex + 1);
var requrl = proxy.model.reportServiceUrl + '/SendEmail';
var _json = {
  exportType: "PDF", reportViewerToken: proxy._reportViewerToken, ReportName: reportName
};
$.ajax({
  type: "POST",
  contentType: "application/json; charset=utf-8",
  url: requrl,
  data: JSON.stringify(_json),
  dataType: "json",
  crossDomain: true
})
}
}
</script>
`

```

Create custom email action

1. Create a new action method `SendEmail` in the Web API service.
2. Export the report to the required type using the `ReportHelper.GetReport` method to send a report stream as an attachment.
3. The following code sample exports the report to stream and send it as an attachment to a specified mail address. In the code, the `SmtpClient` method is used to send the report as an email attachment.

```

`csharp
public object SendEmail(Dictionary<string, object> jsonResult)
{
  string _token = jsonResult["reportViewerToken"].ToString();
  var stream = ReportHelper.GetReport(_token, jsonResult["exportType"].ToString());
  stream.Position = 0;
  if (!ComposeEmail(stream, jsonResult["reportName"].ToString()))

```



```
{
return "Mail not sent !!!";
}
return "Mail Sent !!!";
}
public bool ComposeEmail(Stream stream, string reportName)
{
try
{
MailMessage mail = new MailMessage();
SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
mail.IsBodyHtml = true;
mail.From = new MailAddress("xx@gmail.com");
mail.To.Add("xx@gmail.com");
mail.Subject = "Report Name : " + reportName;
stream.Position = 0;
if (stream != null)
{
ContentType ct = new ContentType();
ct.Name = reportName + DateTime.Now.ToString() + ".pdf";
System.Net.Mail.Attachment attachment = new System.Net.Mail.Attachment(stream, ct);
mail.Attachments.Add(attachment);
}
SmtpServer.Port = 587;
SmtpServer.Credentials = new System.Net.NetworkCredential("xx@gmail.com", "xx");
SmtpServer.EnableSsl = true;
SmtpServer.Send(mail);
return true;
}
catch (Exception ex)
{
throw ex;
}
```

```
return false;  
}  
`
```

In the above code sample, the report is exported to PDF format and send to users using the `SmptClient` method.

Custom properties

The custom properties helps you to include additional features that are not natively supported in the RDL reporting. This topic explains the list of custom properties supported in ASP.NET Web Forms Report Viewer.

See Also

- [Textbox Custom Properties](#)
- [Table Custom Properties](#)
- [Image Custom Properties](#)
- [Chart Custom Properties](#)
- [Report Custom Properties](#)
- [Parameter Custom Properties](#)
- [Export Custom Properties](#)

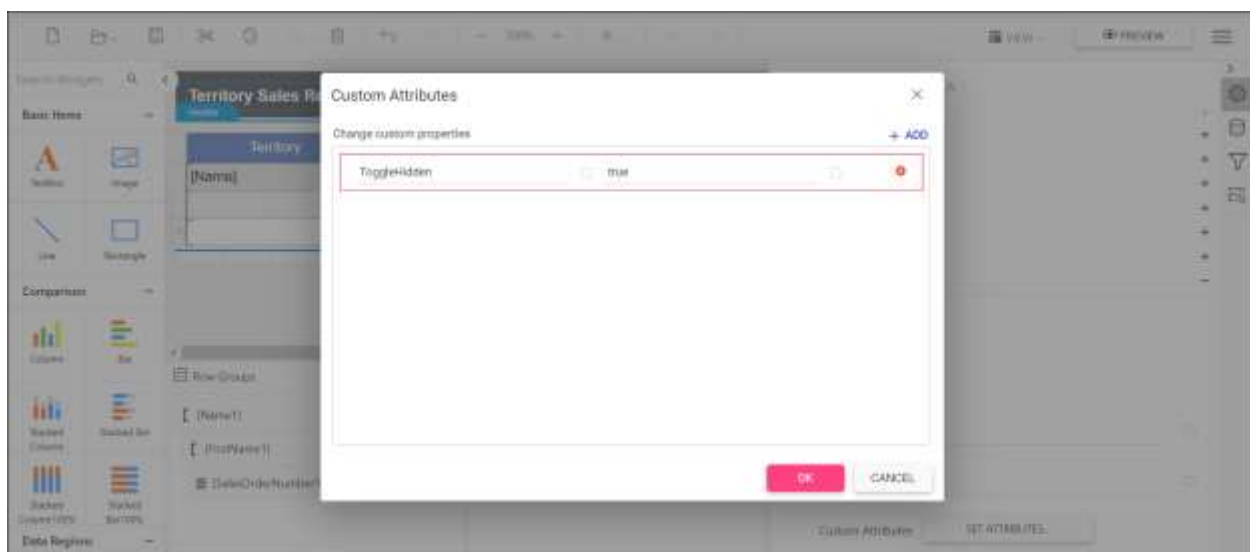
Textbox custom properties

This topic explains about the list of textbox report item custom properties that are supported to render in ASP.NET Web Forms Report Viewer.

Show or hide toggle icon in text box report item

The `ToggleHidden` custom property is used to show or hide the toggle icon in the textbox.

You can set the `ToggleHidden` property value, as shown in the below.



Before setting the toggle hidden property, the default value will be displayed as below.



Territory	Sales Person	Order Number	Total Sales
Australia			\$1,943,016
	Lynn Tsotilas		\$1,943,016
Canada			\$12,868,458
	Garrett Vargas		\$4,840,689
	José Saraiva		\$7,967,769
Central			\$13,434,510
	Jillian Carson		\$13,434,510
France			\$6,083,691
	Ranjit Varkey Chudukatil		\$6,083,691
Germany			\$2,476,530
	Rachel Valdez		\$2,476,530
Northeast			\$12,433,503
	Michael Blythe		\$12,433,503
Northwest			\$6,305,407

Preview the report and the see the toggle icon is hidden in the report.



Territory	Sales Person	Order Number	Total Sales
Australia			\$1,943,016
	Lynn Tsotilas		\$1,943,016
Canada			\$12,868,458
	Garrett Vargas		\$4,840,689
	José Saraiva		\$7,967,769
Central			\$13,434,510
	Jillian Carson		\$13,434,510
France			\$6,083,691
	Ranjit Varkey Chudukatil		\$6,083,691
Germany			\$2,476,530
	Rachel Valdez		\$2,476,530
Northeast			\$12,433,503
	Michael Blythe		\$12,433,503
Northwest			\$6,305,407

Table custom properties

This topic explains about the list of table report item custom properties that are supported to render in ASP.NET Web Forms Report Viewer.

Limit number of table records on each page

The **RowsPerPage** custom property is used to specify the number of table records to display on each page. It supports integer data value greater than zero.

This property is ignored when table rows heights higher than current page size. Increase the report page height or reduce **RowsPerPage** count that fits within the page.

You can set the **RowsPerPage** property value, as shown in the below.

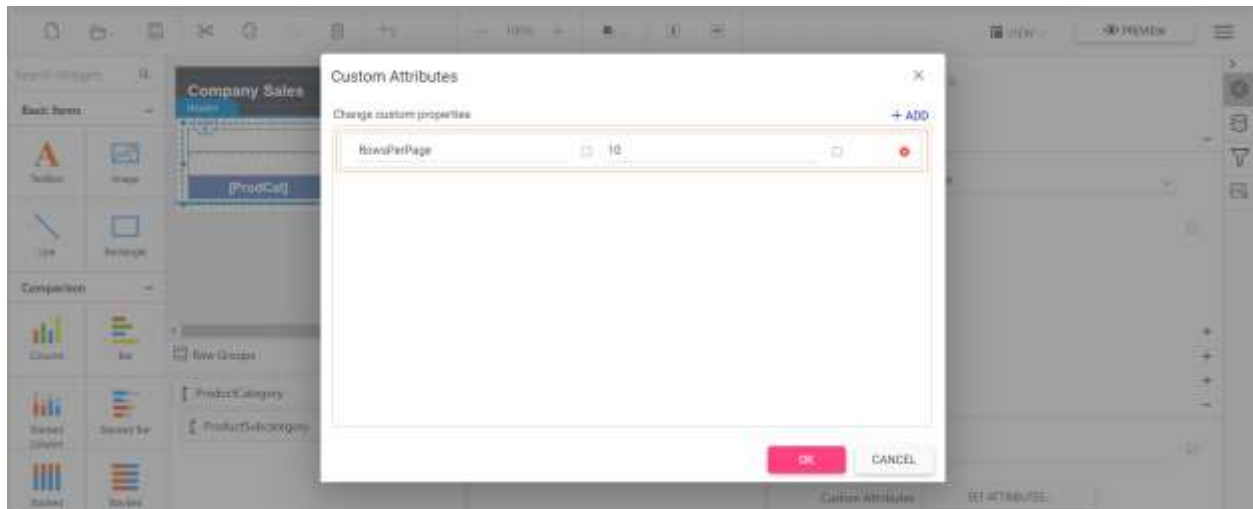


Image custom properties

This topic explains about the list of image custom properties that are supported to render in the ASP.NET web forms Report Viewer.

Angle

Set **Angle** custom property to image report item to rotate the image on a specified angle. It supports the angle values 0, 90, 180, and 270. You can set the property value, as shown in the below.

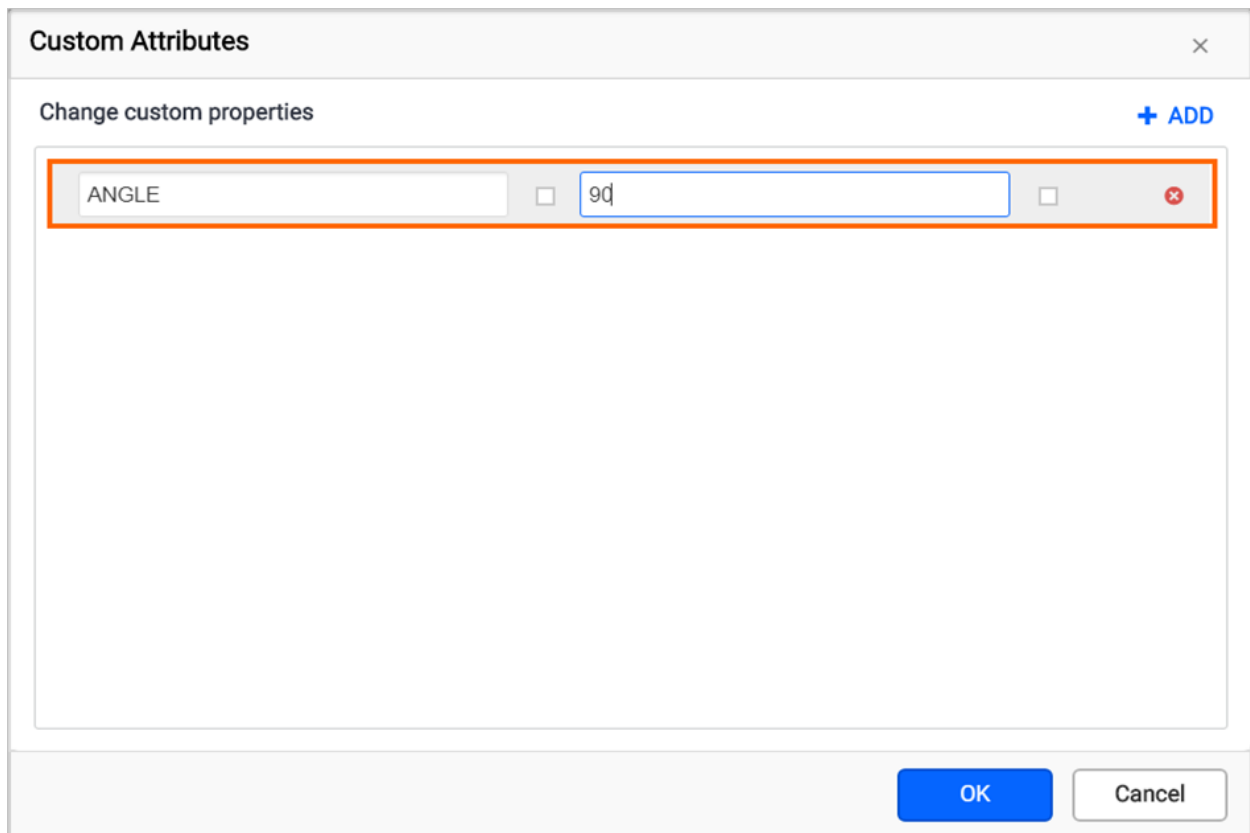
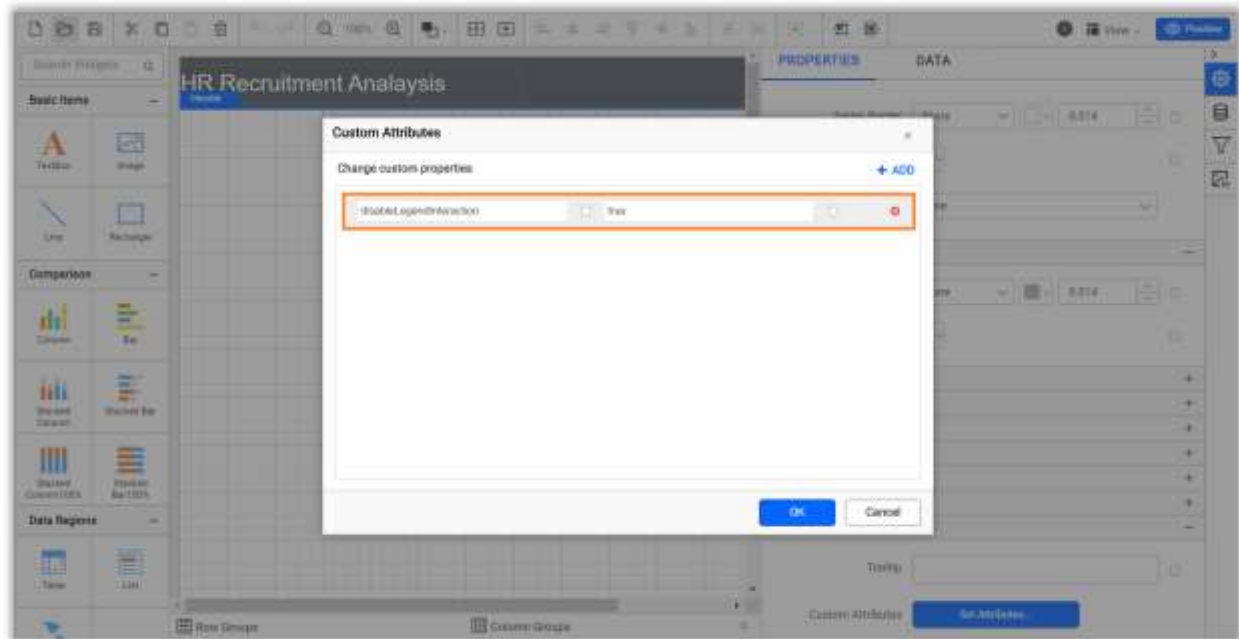


Chart custom properties

This topic explains about the list of chart custom properties that are supported to render in the ASP.NET Web Forms Report Viewer.

Disable legend item interaction

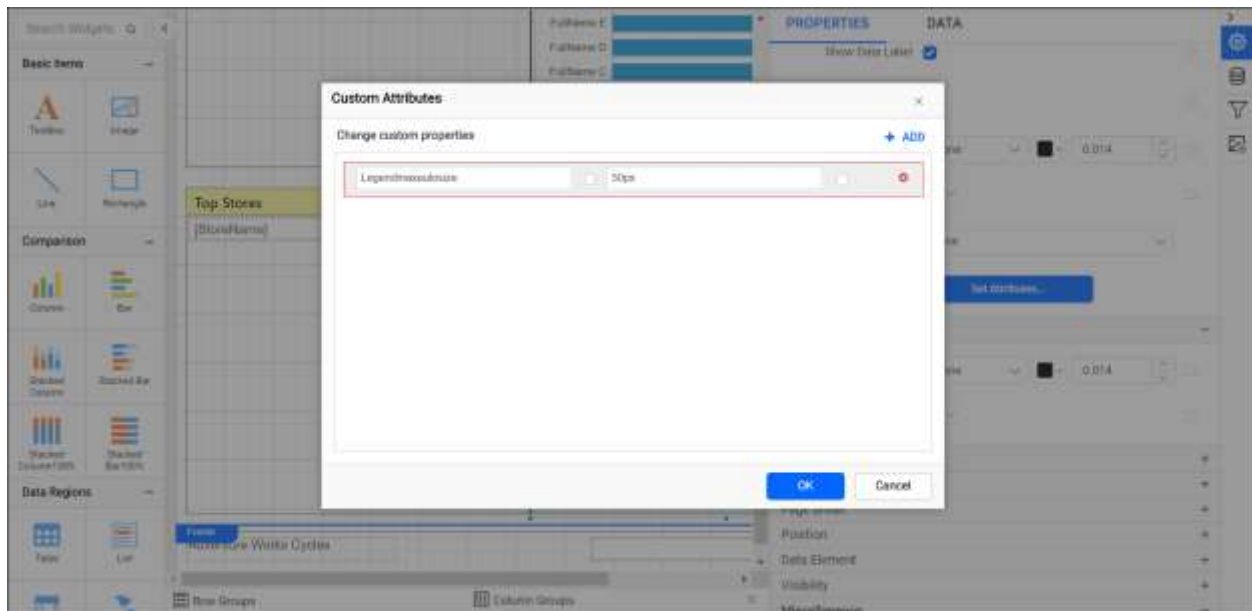
Set `DisableLegendInteraction` custom property value as `true` to stop the legend item interaction. The property value should be boolean. You can set the property value, as shown in the below.



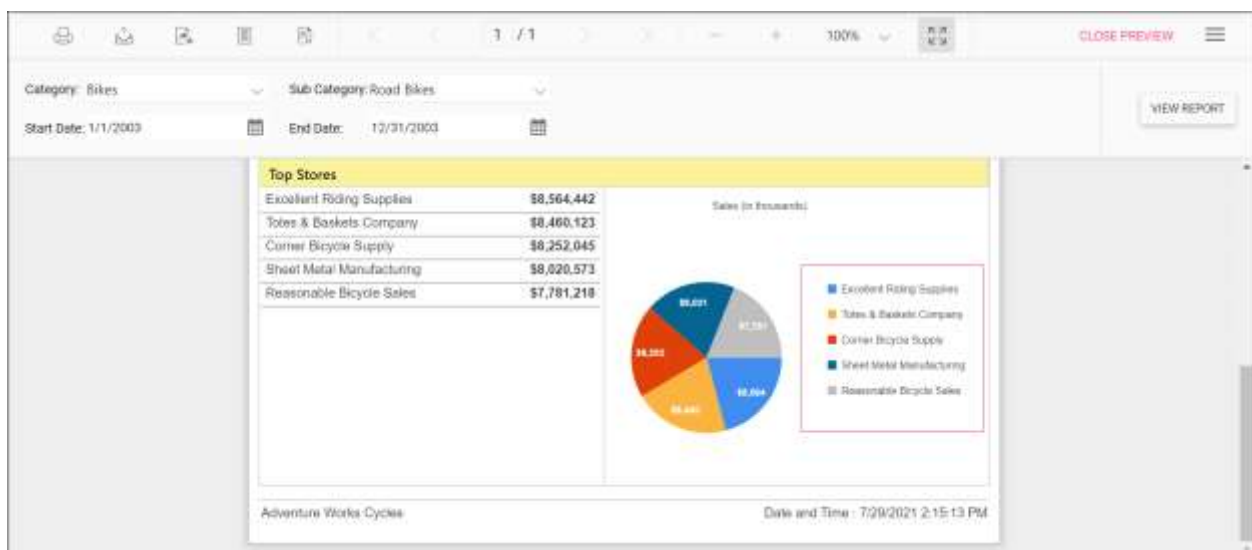
Set maximum size for chart legend

The `LegendMaxAutoSize` custom property specifies the maximum size of the legend container in the report.

You can set the `LegendMaxAutoSize` property value, as shown in the below.



Preview the report and see legend container size in chart report.

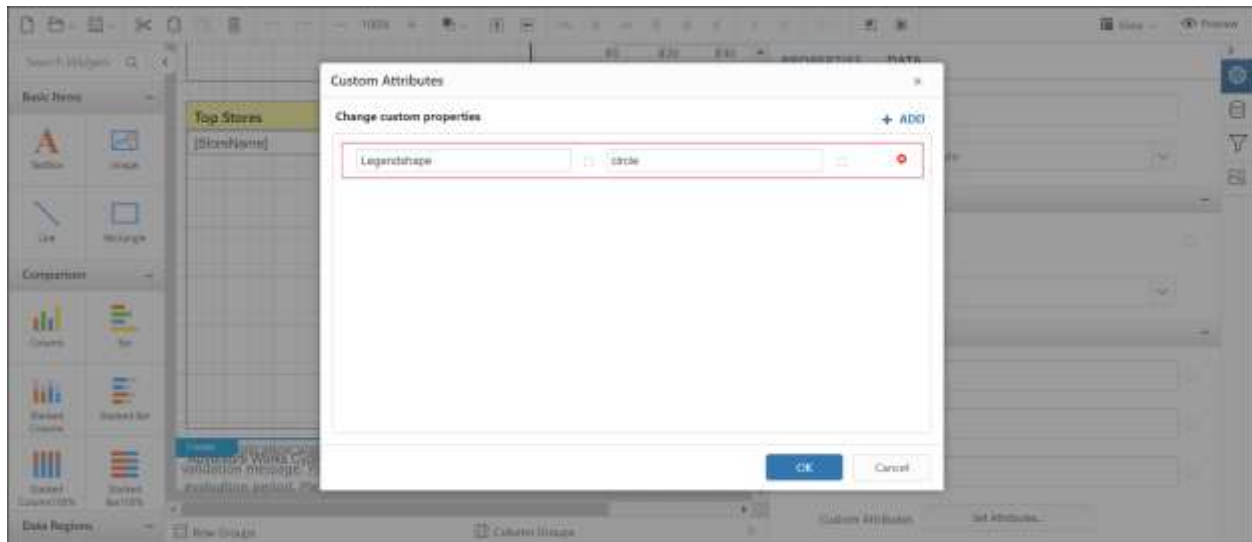


Change chart legend shape

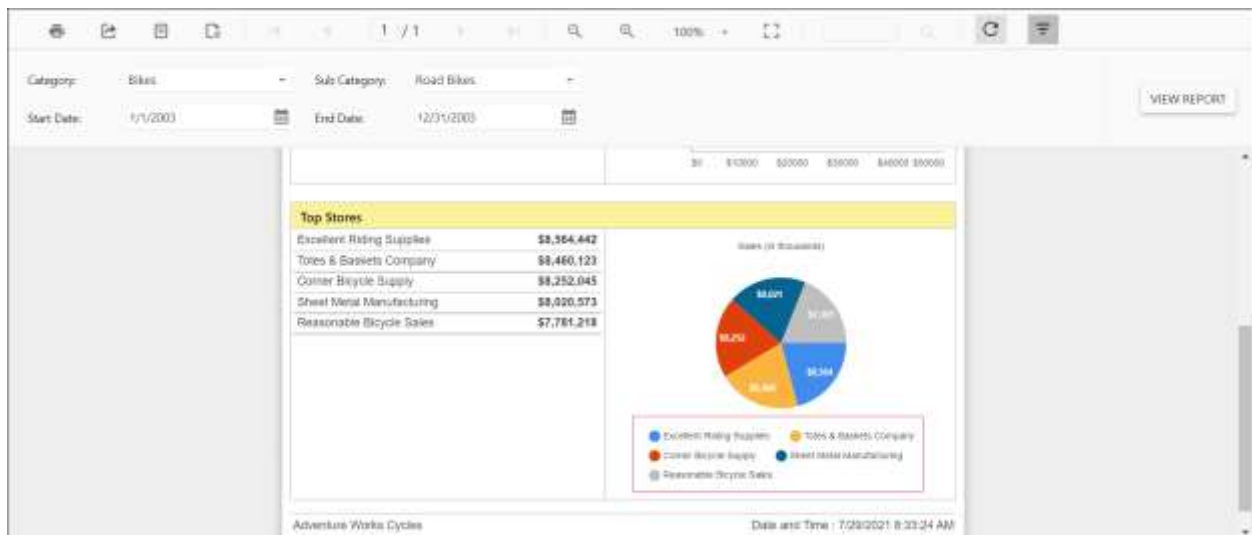
The `LegendShape` custom property allows changing the shape of the legend in the chart report item. By default, the `LegendShape` value is `rectangle`.

- Rectangle - legend shapes displayed in `rectangle`.
- Circle - legends are displayed in `circle`.
- Thumbnail - legends are displayed in `seriestype`.

You can set the `LegendShape` property value, as shown in the below.



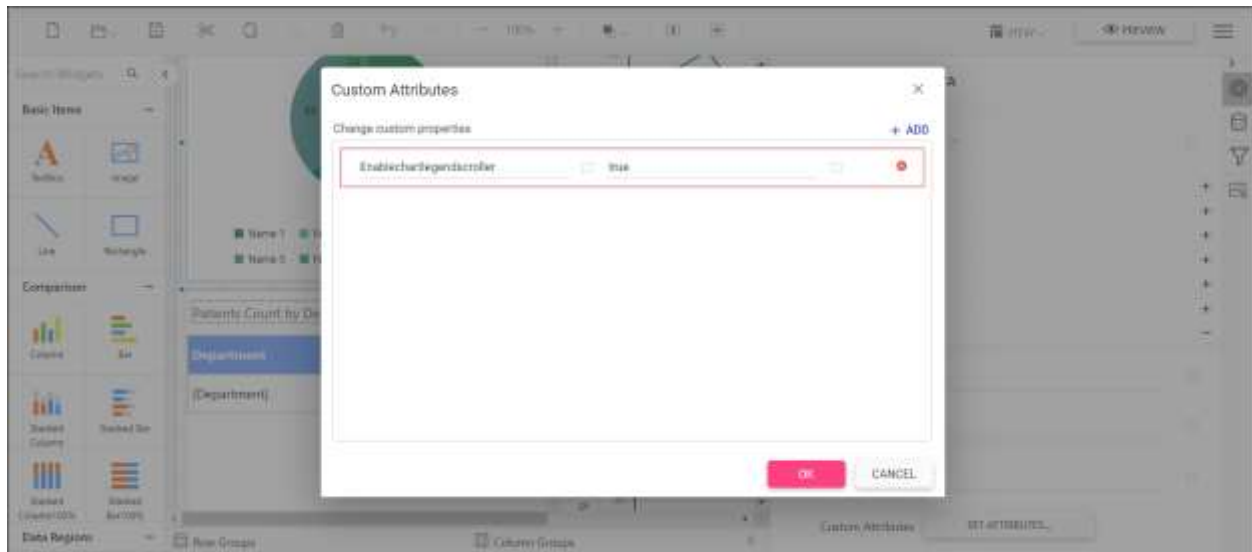
Preview the report and the see legend shape in chart report.



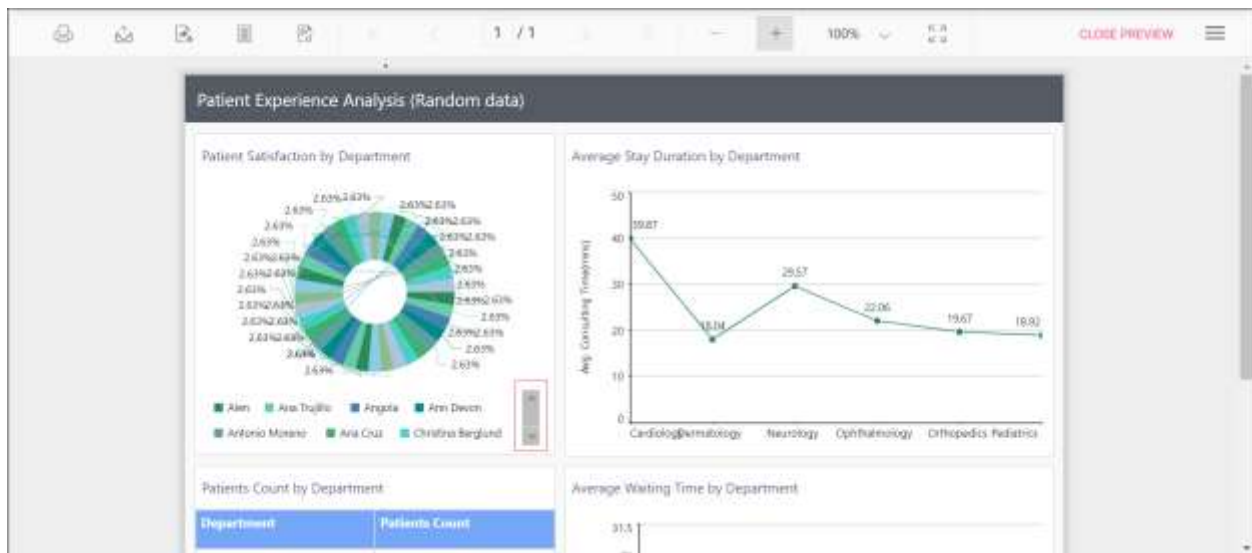
Show or hide chart legend scroller

The `EnableChartLegendScroller` custom property controls whether legend has to use scrollbar or not. The scrollbar appears depending upon size and position properties of legend. By default, the `EnableChartLegendScroller` value is `false`.

You can set the `EnableChartLegendScroller` property value, as shown in the below.



Preview the report and the see the legend scrollbar in chart report.



Set range padding for X-axis and Y-axis

Padding can be applied to the minimum and maximum extremes of the axis range by using the `XAxisRangePadding` and `YAxisRangePadding` property. The default value is `none`.

Numeric axis supports the following types of padding.

Name | Description

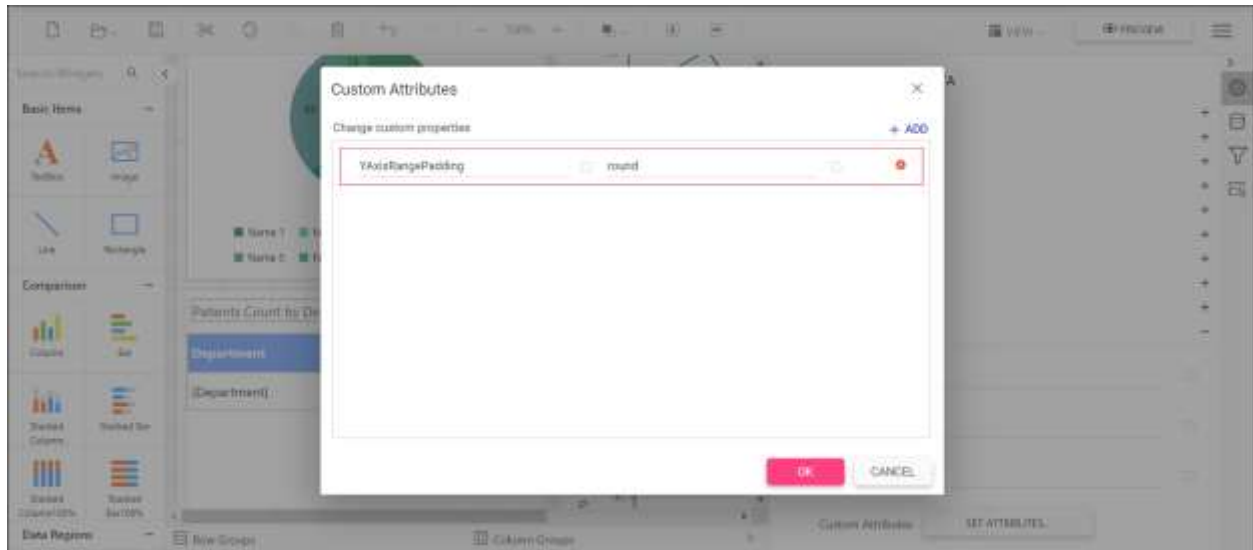
Additional | Interval of the axis is added as padding to the minimum and maximum values of the range

Normal | Padding is applied to the axis based on the range calculation

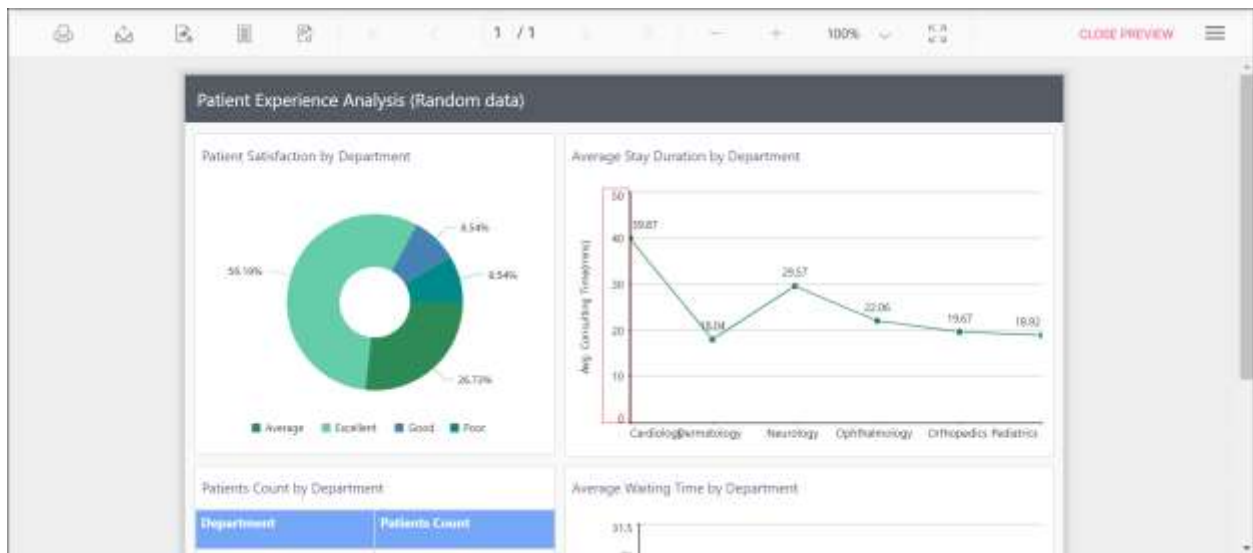
None | Padding cannot be applied to the axis

Round | Axis range is rounded to the nearest possible value divided by the interval

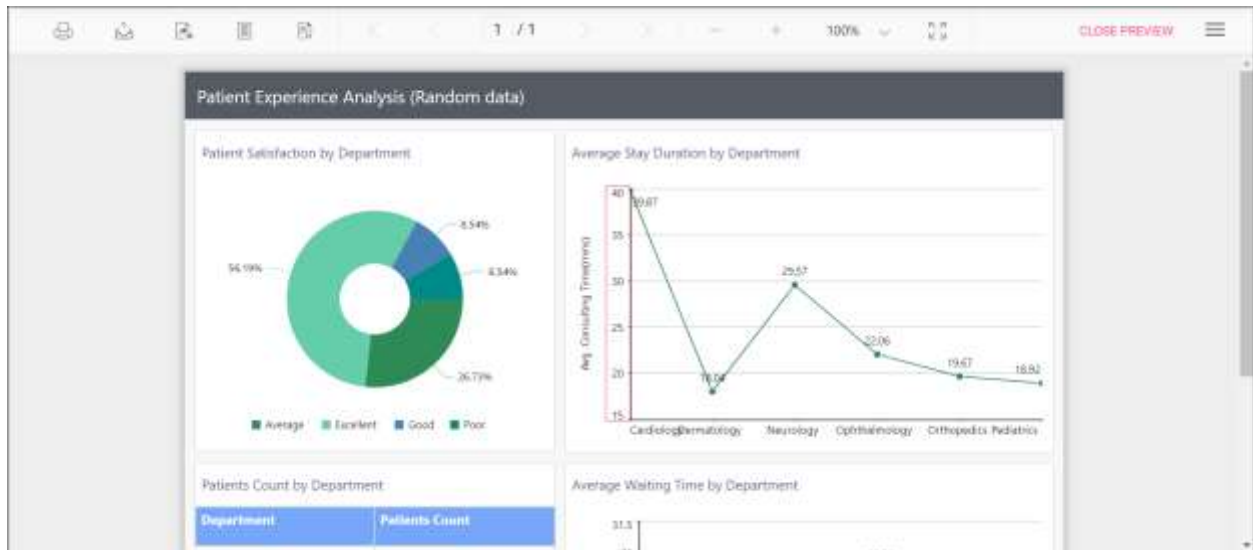
You can set the `XAxisRangePadding` and `YAxisRangePadding` property value, as shown in the below.



Before setting the range padding, the default value will be displayed as below.



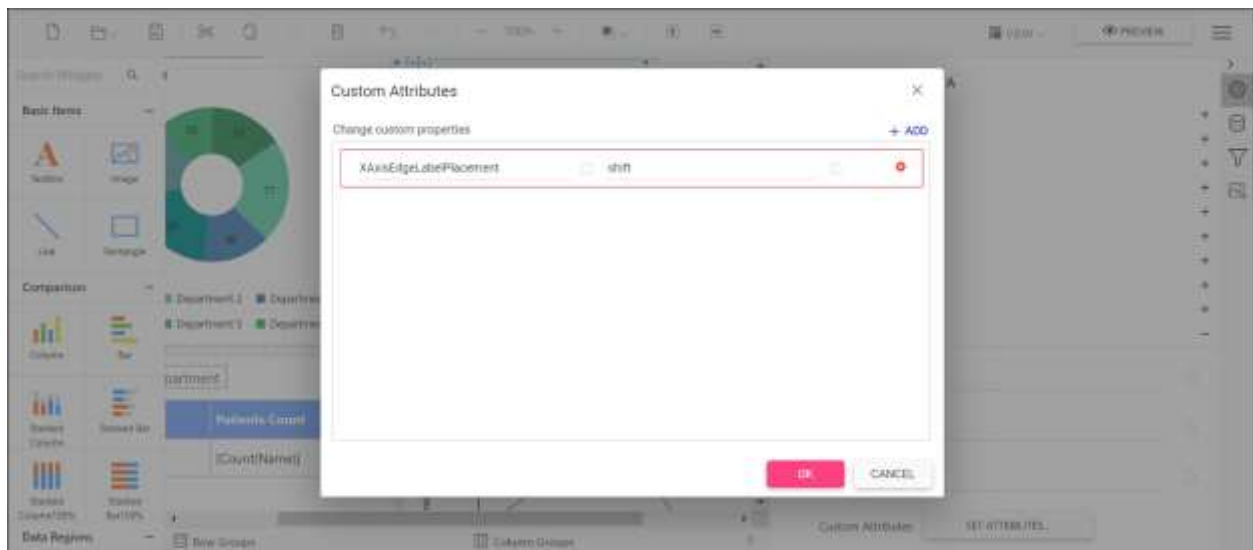
Set the padding range and see the changes in chart report as below.



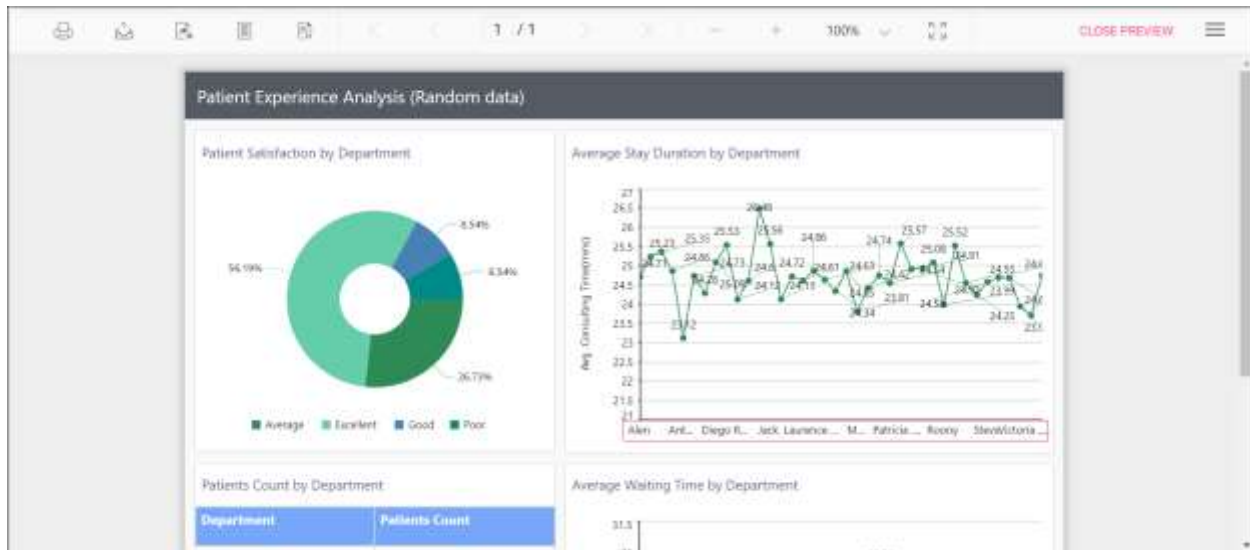
Control edge label placement in chart axis

Labels with long text at the edges of an axis may appear partially outside the chart. The `XAxisEdgeLabelPlacement` and `YAxisEdgeLabelPlacement` custom property can be used to avoid the partial appearance of the labels at the corners. The default value is `none`.

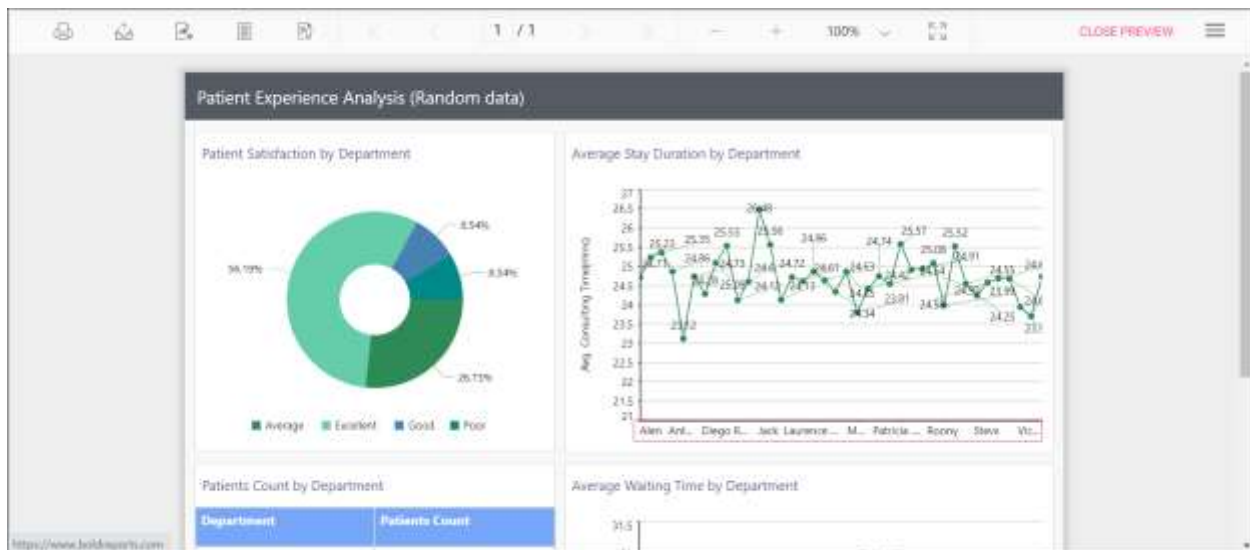
You can set the `XAxisEdgeLabelPlacement` property value, as shown in the below.



Before setting the edge label placement, the default value will be displayed as below.



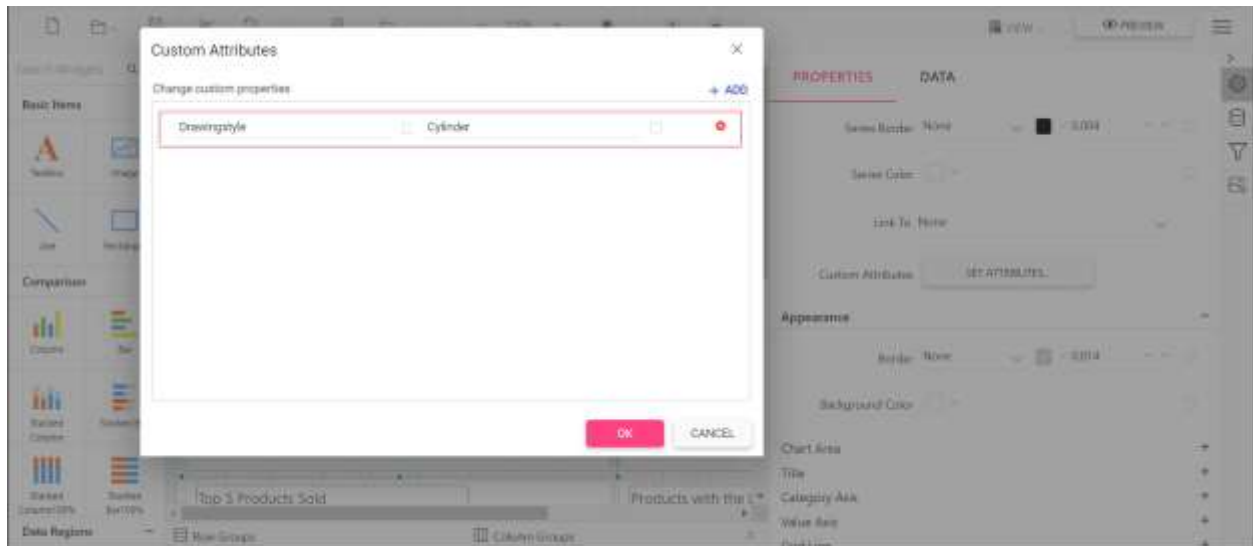
Set the edge label placement and see the changes in chart report as below.



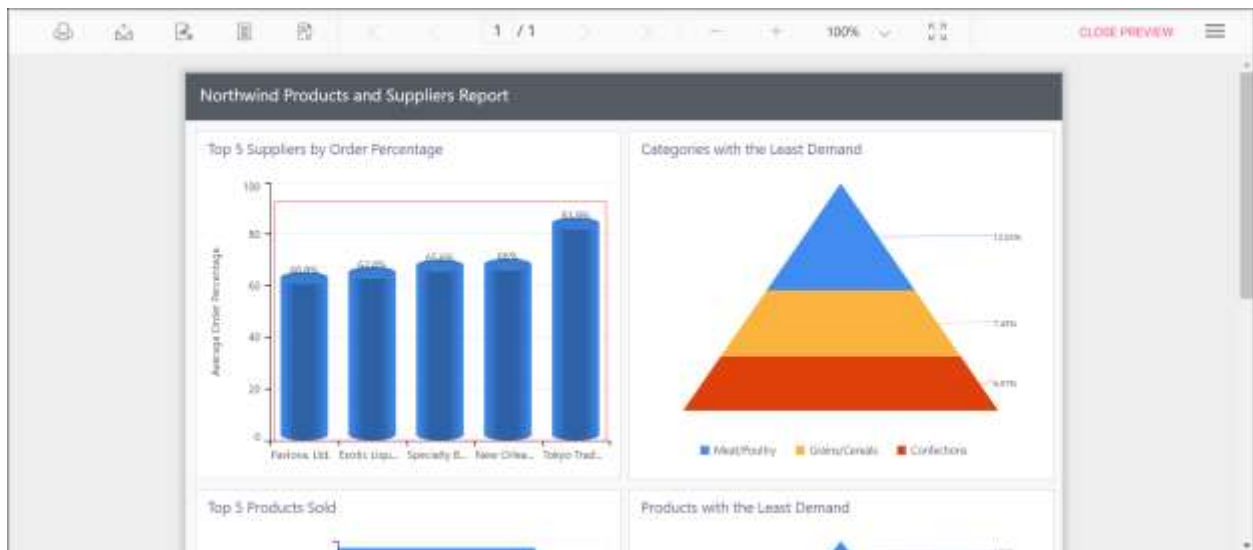
Change the drawing style of a chart column or bar series

The shape of the chart column or bar can be changed using this **Drawingstyle** custom property. By default, the **Drawingstyle** property value is **rectangle**.

You can set the **Drawingstyle** property value, as shown in the below.



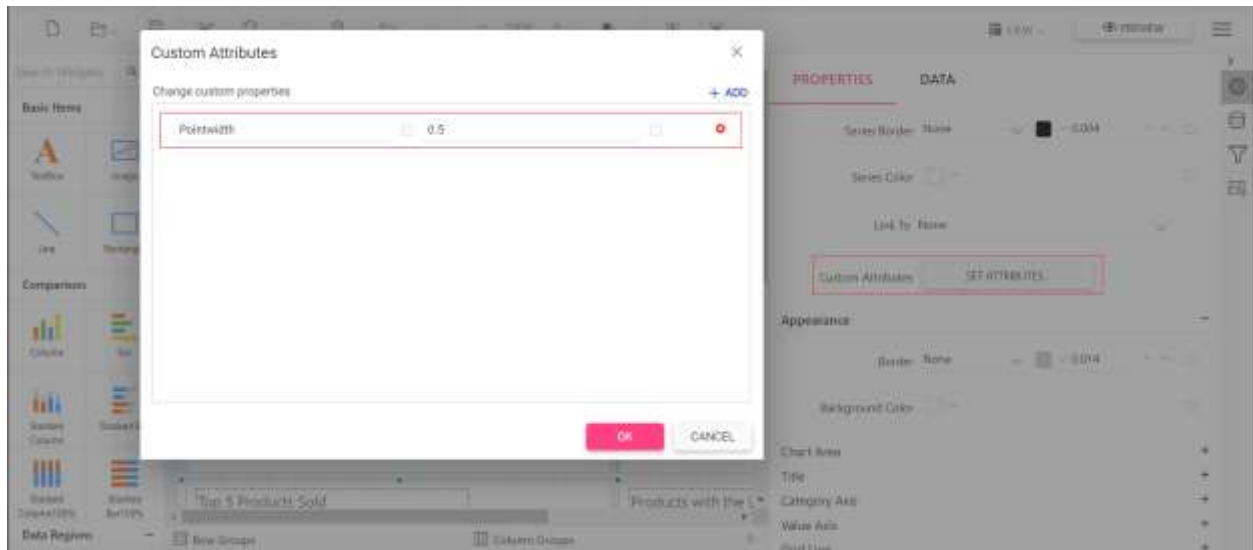
Preview the report and the see the column or bar shape in chart report.



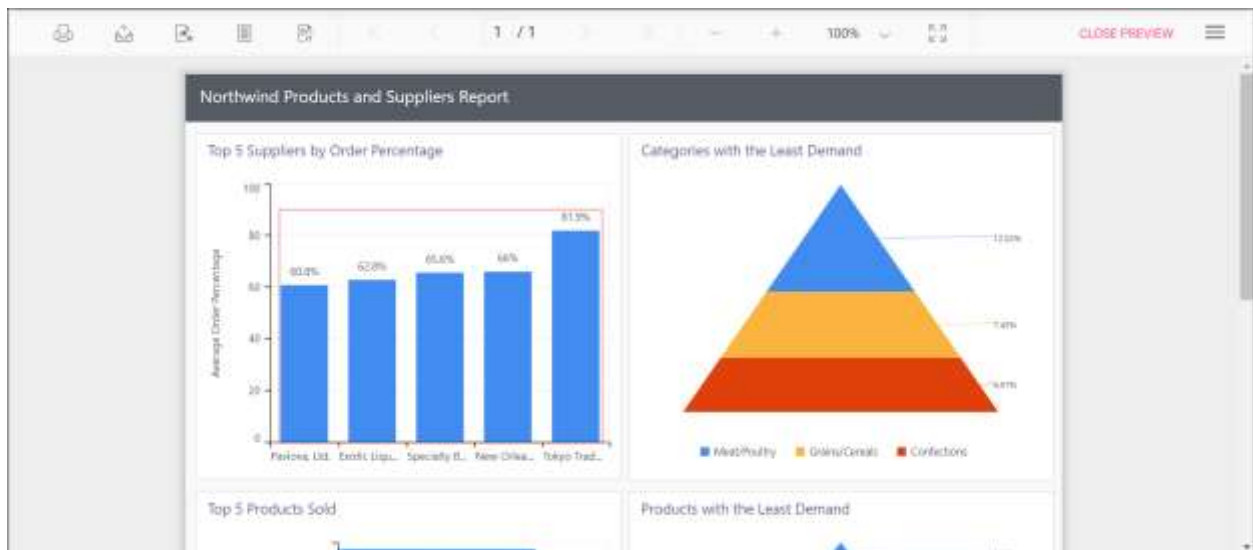
Change width of the column type series in chart

Width of the column type series can be customized by using the **Pointwidth** property. Default value of **Pointwidth** is 0.7. Value ranges from 0 to 1. Here 1 corresponds to 100% of available width and 0 corresponds to 0% of available width.

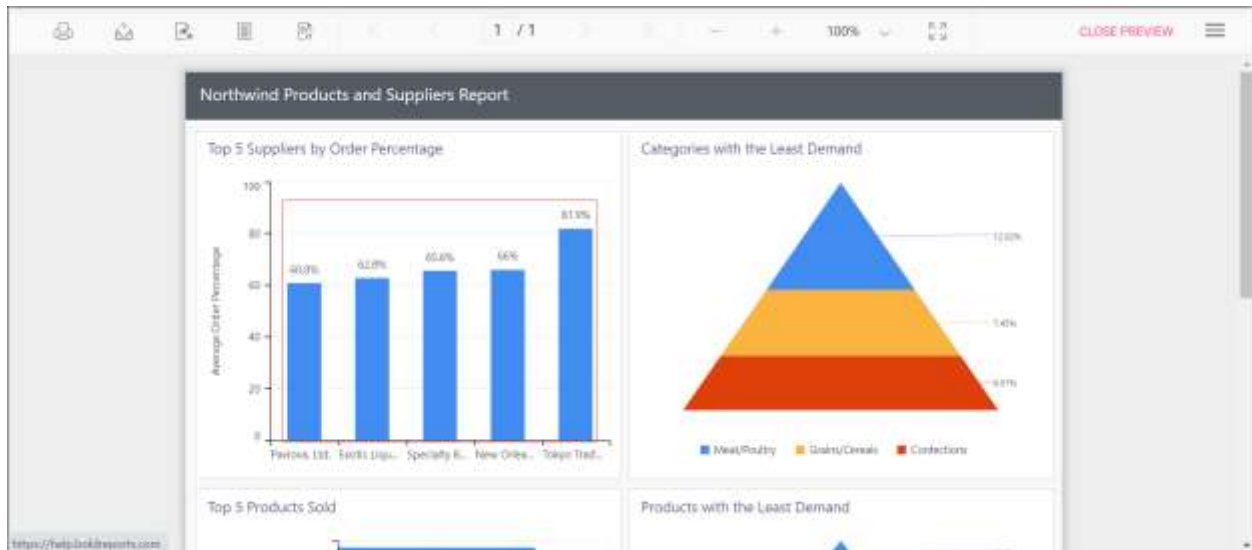
You can set the **Pointwidth** property value, as shown in the below.



Before setting the point width, the default value will be displayed as below.



Preview the report and the see the column width in chart report.



Report custom properties

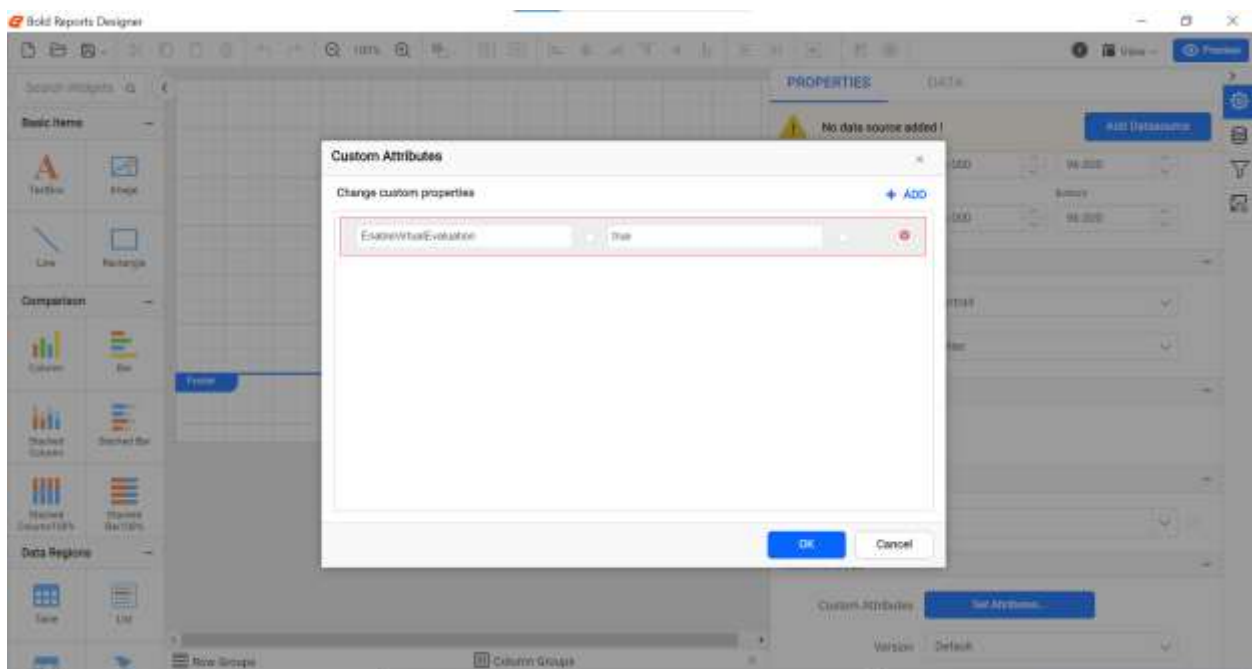
This topic explains about the list of report custom properties that are supported to render in ASP.NET Web Forms Report Viewer.

Improve performance and handle large amount of data

Set the `EnableVirtualEvaluation` and `DisablePageSplitting` custom properties in a report to improve performance and handle the larger amount of data with less memory footprint.

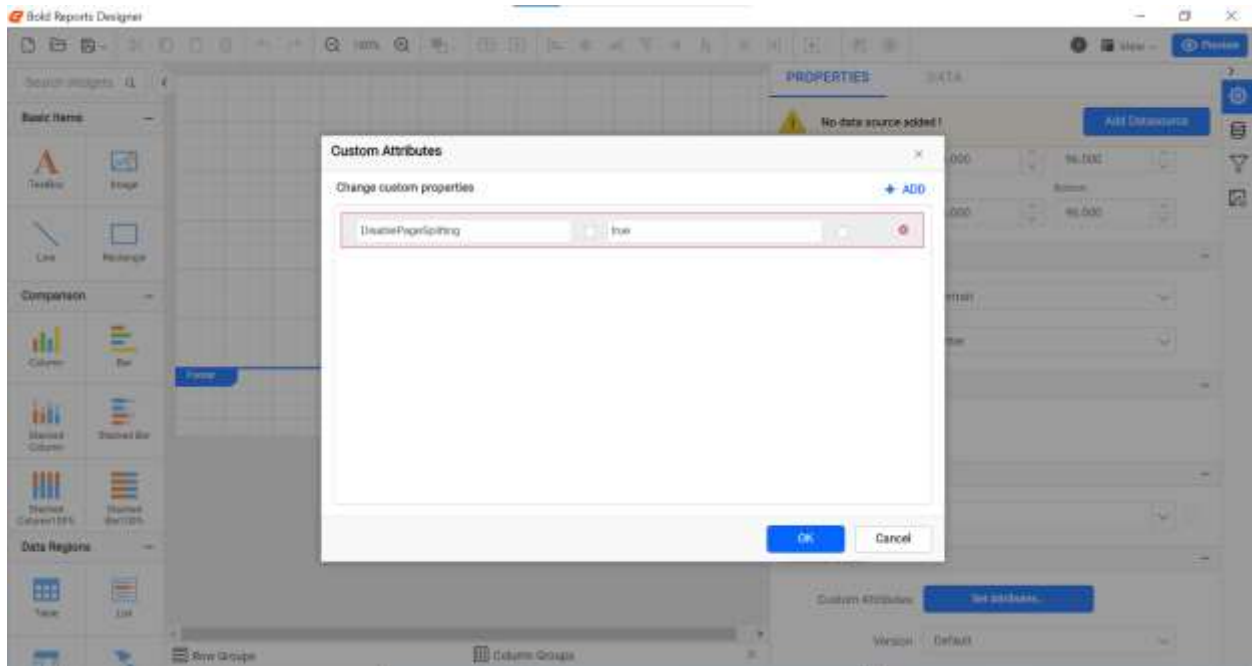
Render large data report faster

The `EnableVirtualEvaluation` custom property is used to render the large data report faster. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Reduce memory footprint for large data report

The `DisablePageSplitting` custom property is used to reduce the memory footprint for large data report. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Improve report items layout and avoid extra blank pages

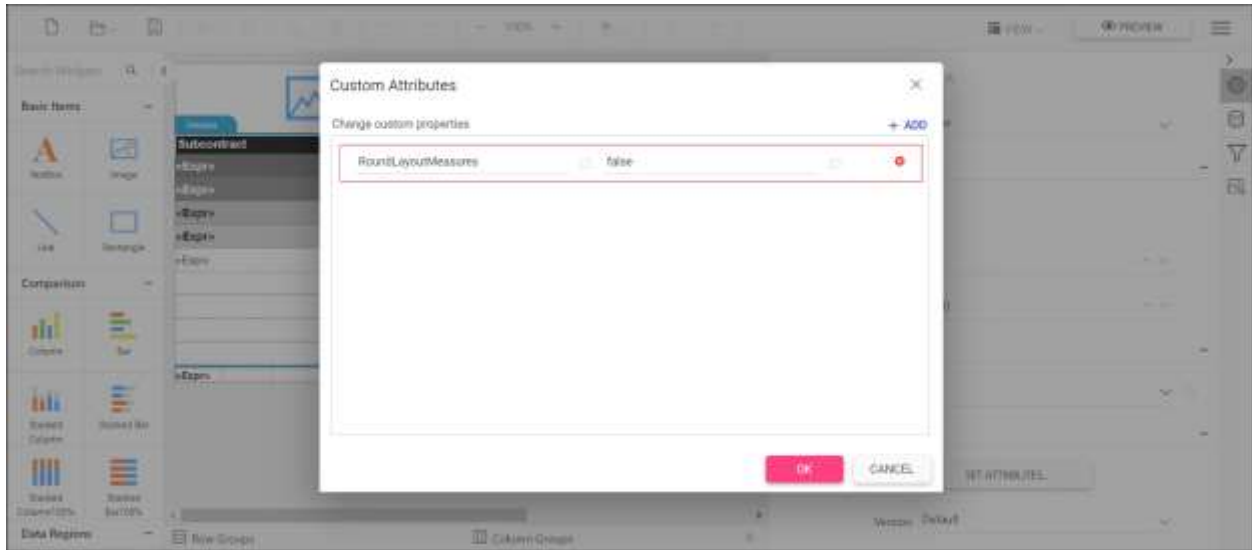
When the `RoundLayoutMeasures` property is false, all non-integral values that are calculated during the report processing are rounded to whole pixel values. It provides following improvements,

- Report text rendered without cuts.
- Eliminates extra blank pages.
- Removes item and text overlaps.
- Eliminates the blur semi-transparent edges that are produced by anti-aliasing.
- Produces identical look in report view and export output.

You can set the property value, as shown in the below.

Parameter custom properties
timeouts

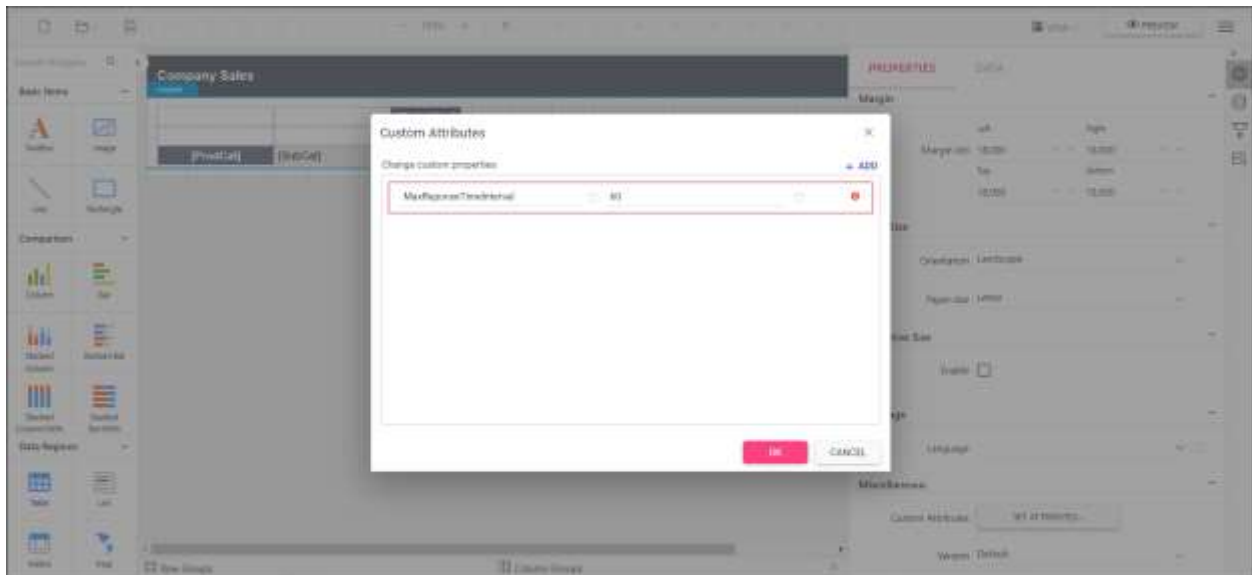
Handling failure in long-running HTTP requests, report processes, and



Handling failure in long-running HTTP requests, report processes, and timeouts

When a report process for a long time due to huge records or a HTTP request takes too long to respond then it results in Gateway Timeout or report rendering errors. The `MaxResponseTimeInterval` allows to specify the seconds to process an HTTP request and respond back. It helps to keep the client and server connection live by avoiding the timeout.

You can set the property value as shown in the below.



Parameter custom properties

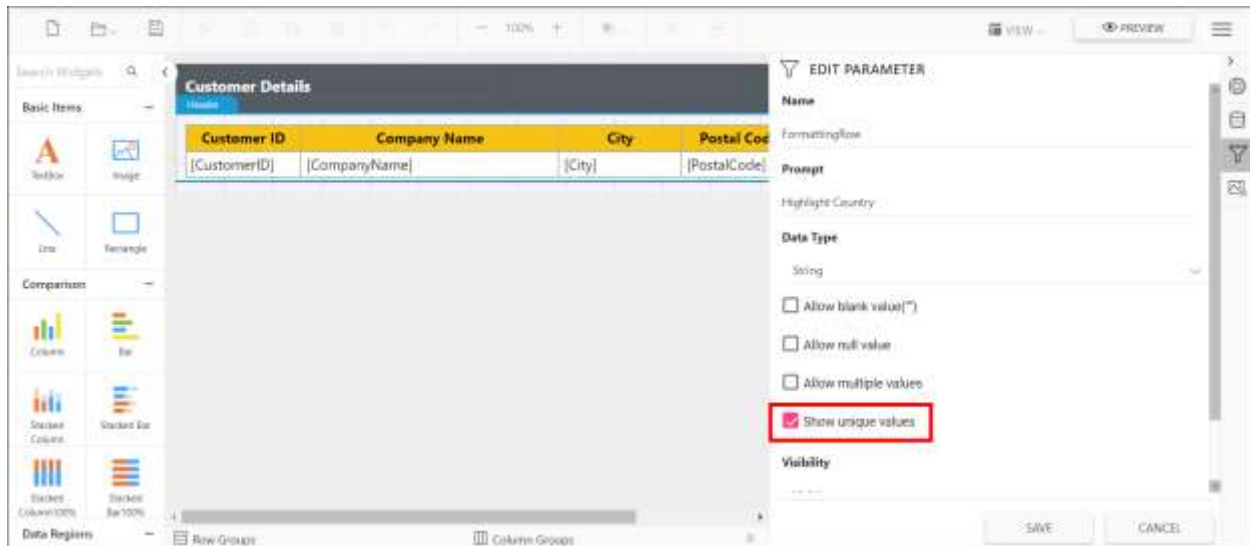
This topic explains about the list of parameter custom properties that are supported to customize the reports preview in Report Viewer.

Show only distinct values in parameter

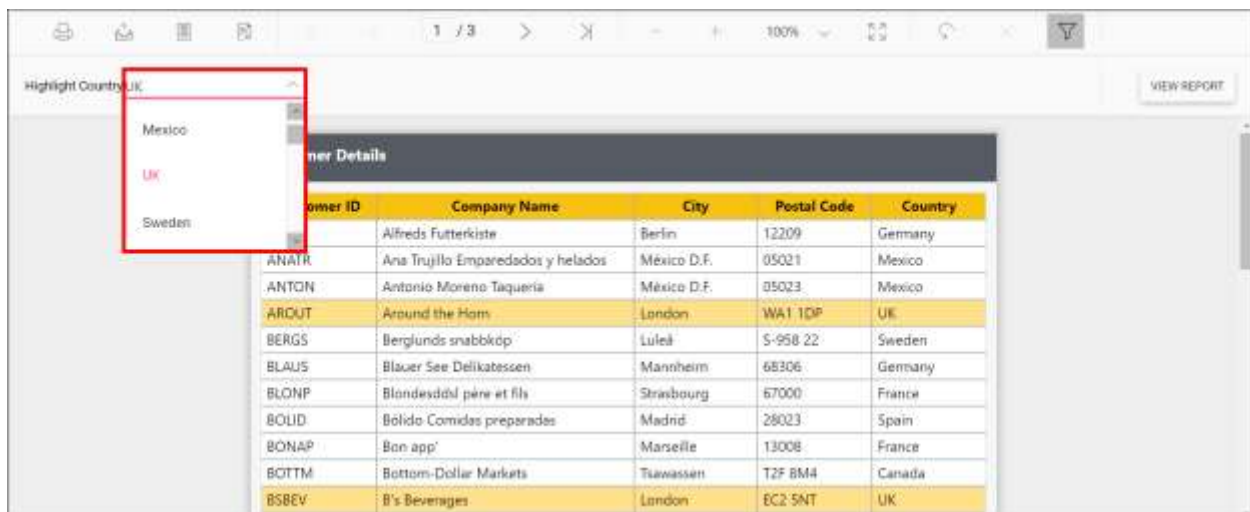
A parameter created with data set query values may contain duplicate values. The Report Viewer supports `UniqueValueParameters` custom property that helps show only unique values in the

parameter drop-down while viewing the report, without creating new a data set. Refer to the below image for property setting option.

You can also provide the parameter names with comma (,) separator to set for multiple parameters.



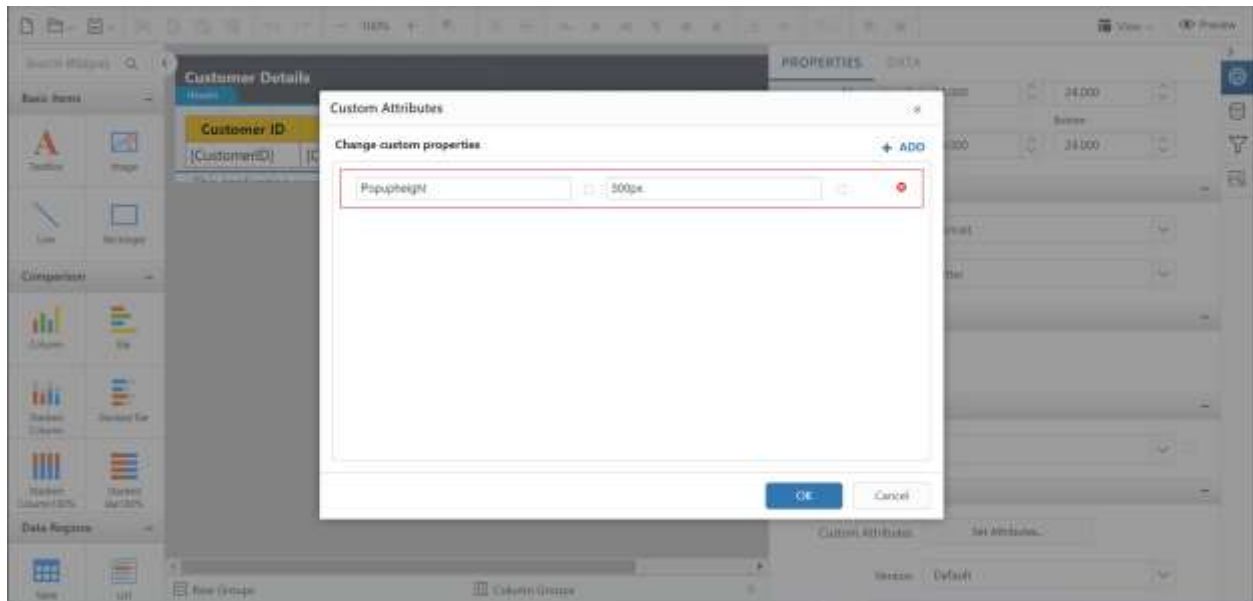
Preview the report and the unique values showed in the parameter drop down.



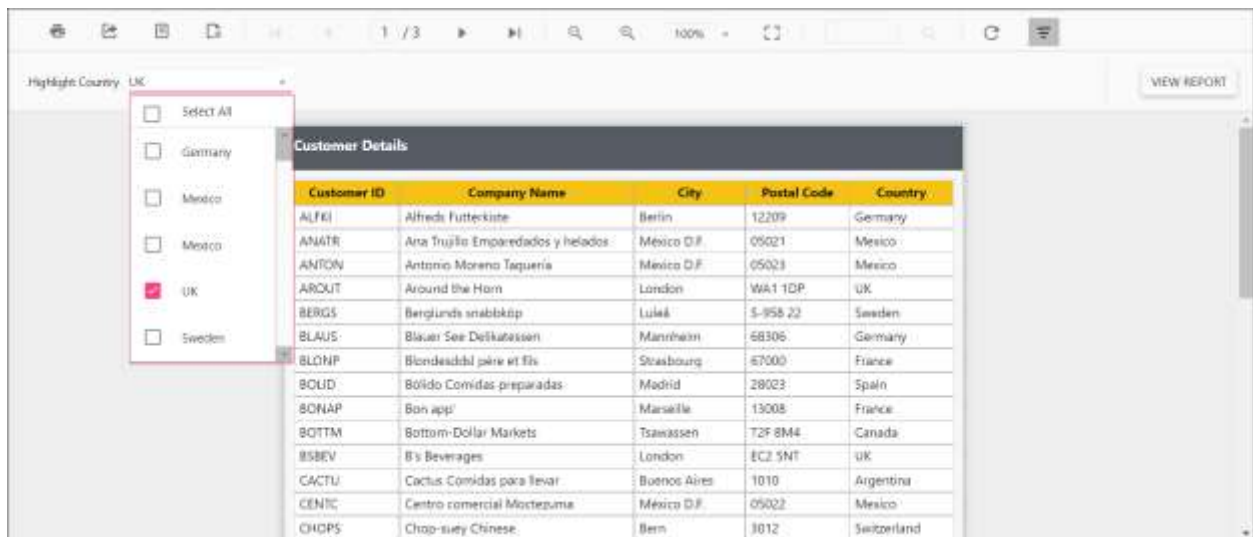
Change the dropdown parameter pop up height value

The **PopupHeight** custom property specifies the height of the parameter combobox popup list in the report. By default, the **PopupHeight** value is **152px**.

You can set the **PopupHeight** property value, as shown in the below.



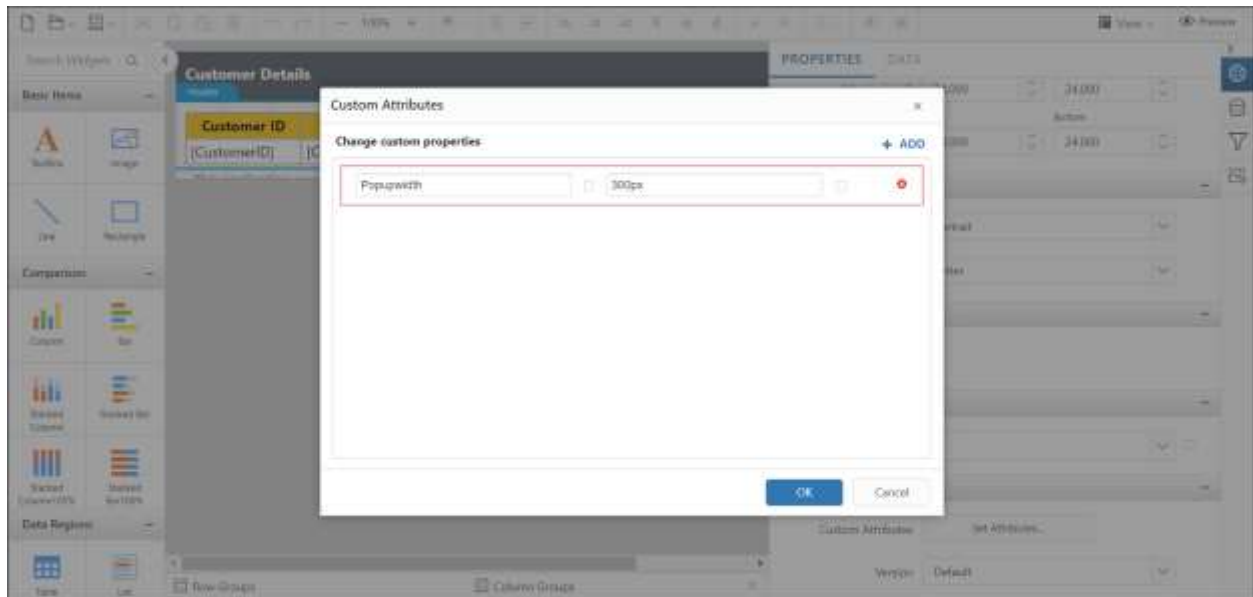
Preview the report and the pop up height showed in the parameter drop down.



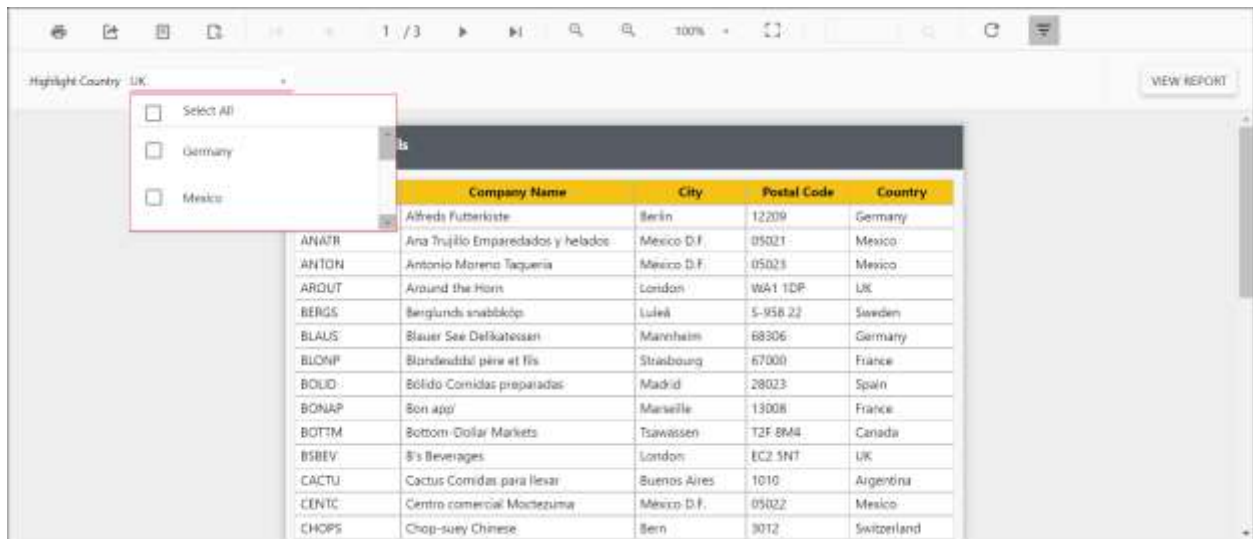
Change the dropdown parameter pop up width value

The **PopupWidth** custom property specifies the width of the parameter combobox popup list in the report. By default, the popup width sets based on the width of the component.

You can set the **PopupWidth** property value, as shown in the below.



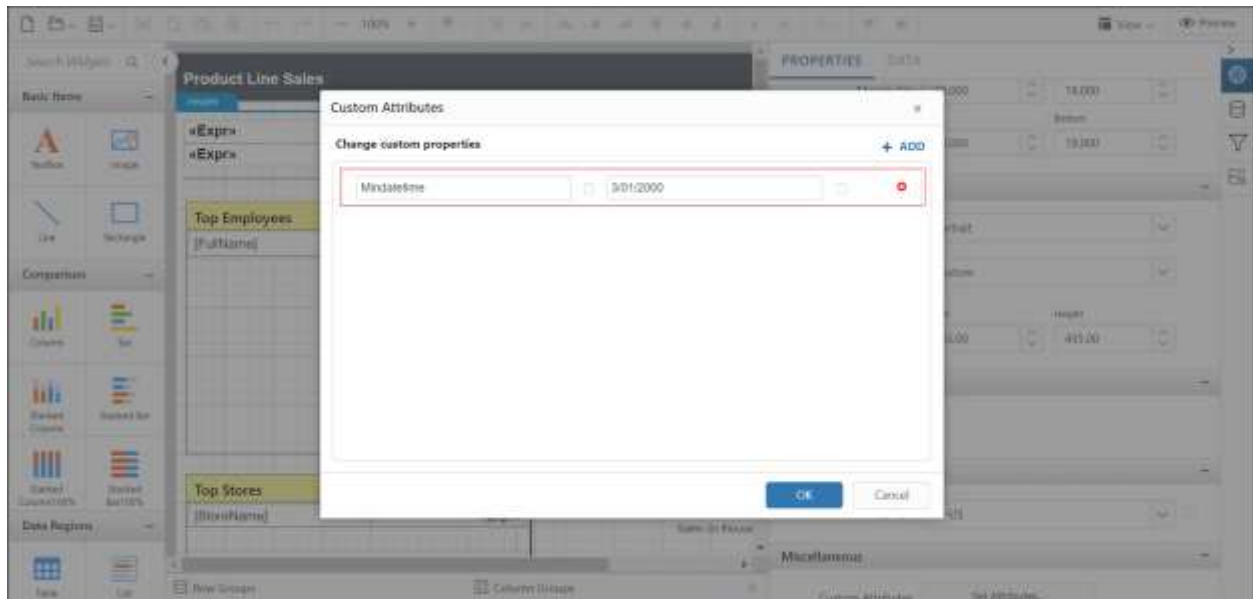
Preview the report and the pop up width showed in the parameter drop down.



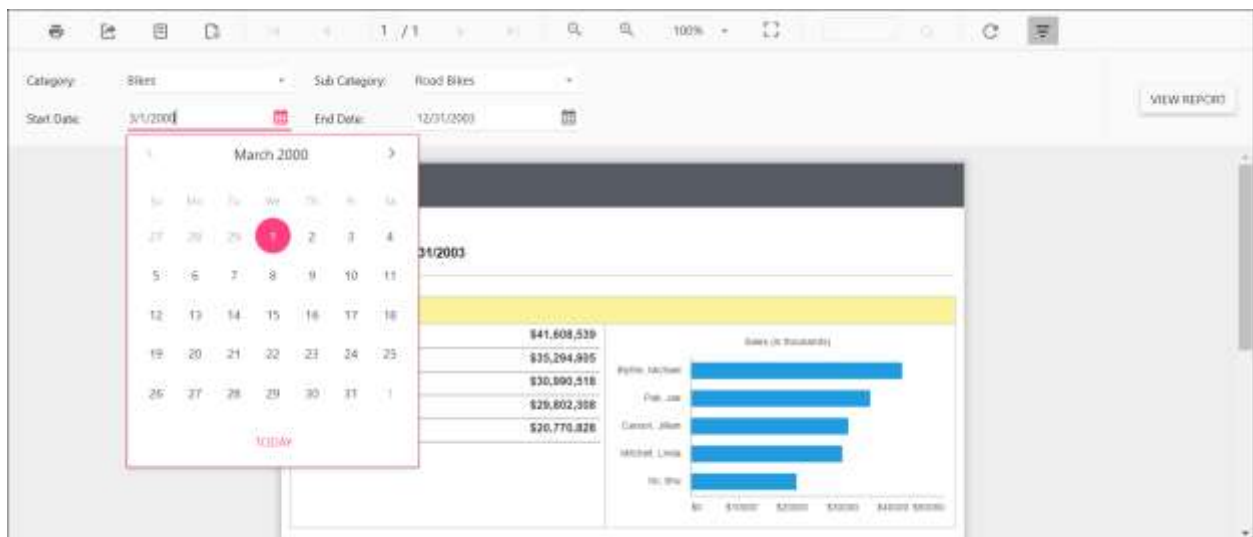
Set the minimum date range for the date report parameter

The **MinDateTime** custom property specifies the minimum date in the datetime parameter item. By default, the **MinDateTime** value is set as **null**.

You can set the **MinDateTime** property value, as shown in the below.



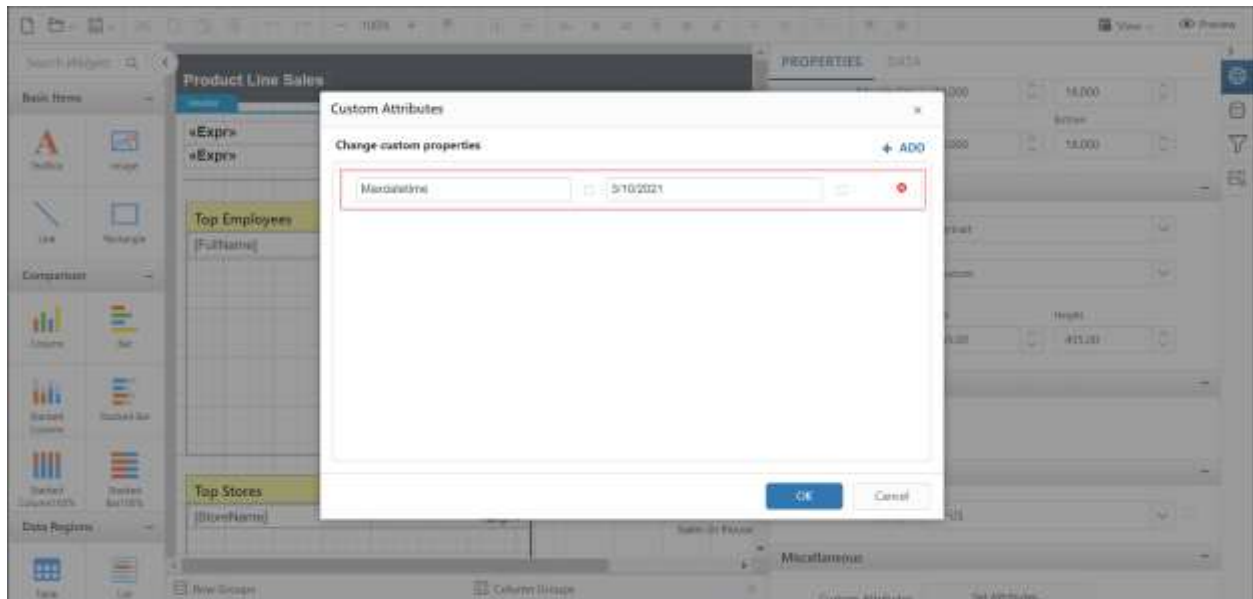
Preview the report and the minimum date showed in the datetime parameter drop down.



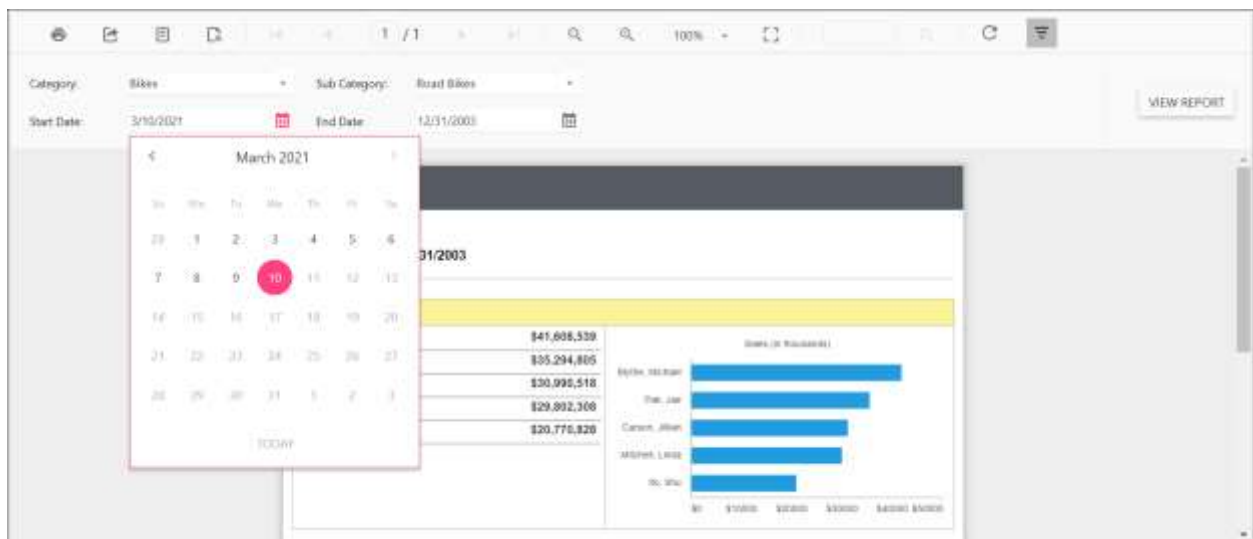
Set the maximum date range for date report parameter

The `MaxDateTime` custom property specifies the maximum date in the datetime parameter item. By default, the `MaxDateTime` value is set as `null`.

You can set the `MaxDateTime` property value, as shown in the below.



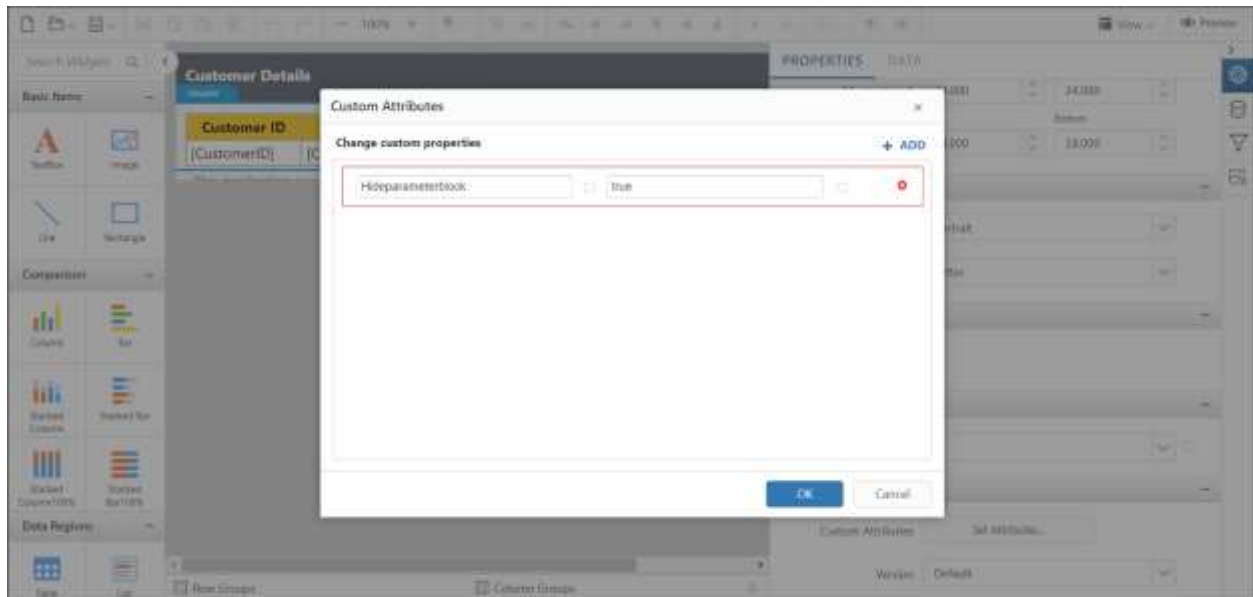
Preview the report and the maximum date showed in the datetime parameter drop down.



Hide the report parameter block

The `HideParameterBlock` custom property is used to hide the parameter block on report initial rendering. The property value should be boolean. By default, the `HideParameterBlock` value is `false`.

You can set the `HideParameterBlock` property value, as shown in the below.



Preview the report and the see the parameter block is hidden.

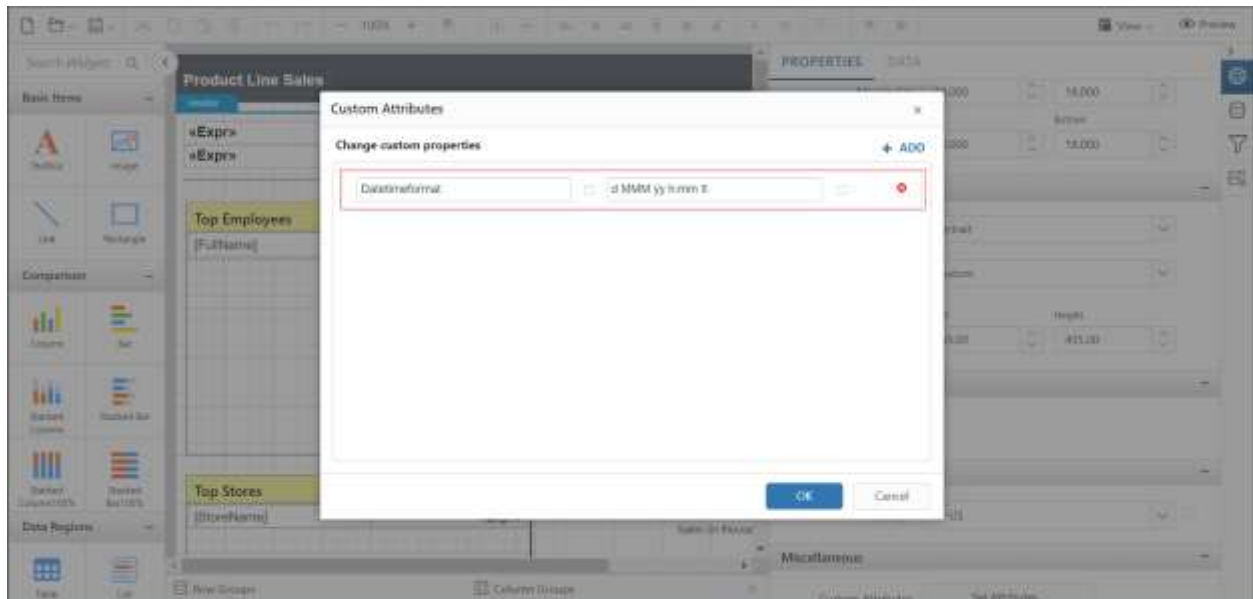
The screenshot shows a report preview titled 'Customer Details'. It displays a table with the following data:

Customer ID	Company Name	City	Postal Code	Country
ALFKI	Alfreds Futterkiste	Berlin	12209	Germany
ANATR	Ana Trujillo Emparedados y helados	México D.F.	05021	Mexico
ANTON	Antonio Moreno Taquería	México D.F.	05023	Mexico
AROUT	Around the Horn	London	WA1 1DP	UK
BERGS	Berglunds snabbköp	Luleå	S-958 22	Sweden
BLAUS	Blaauw Sea Delicatessen	Mannheim	68306	Germany
BONAP	Bonaparte's pates en file	Strasbourg	67000	France
BOLID	Bólido Comidas preparadas	Madrid	28023	Spain
BONAP	Bon app'	Marseille	13008	France
BOITM	Bottom-Dollar Markets	Toronto	T2F 8M4	Canada
BSBEV	B's Beverages	London	EC2 5NT	UK
CACTU	Cactus Comidas para llevar	Buenos Aires	1010	Argentina
CENTC	Centro comercial Motezuma	México D.F.	05022	Mexico
CHOPS	Chop-uey Chinese	Bern	3012	Switzerland
COMM1	Comércio Mineiro	São Paulo	05432-043	Brazil
CONSH	Consolidated Holdings	London	WX1 6LT	UK
DRACD	Drachendorff & Pfeiffer	Dresden	81068	Germany

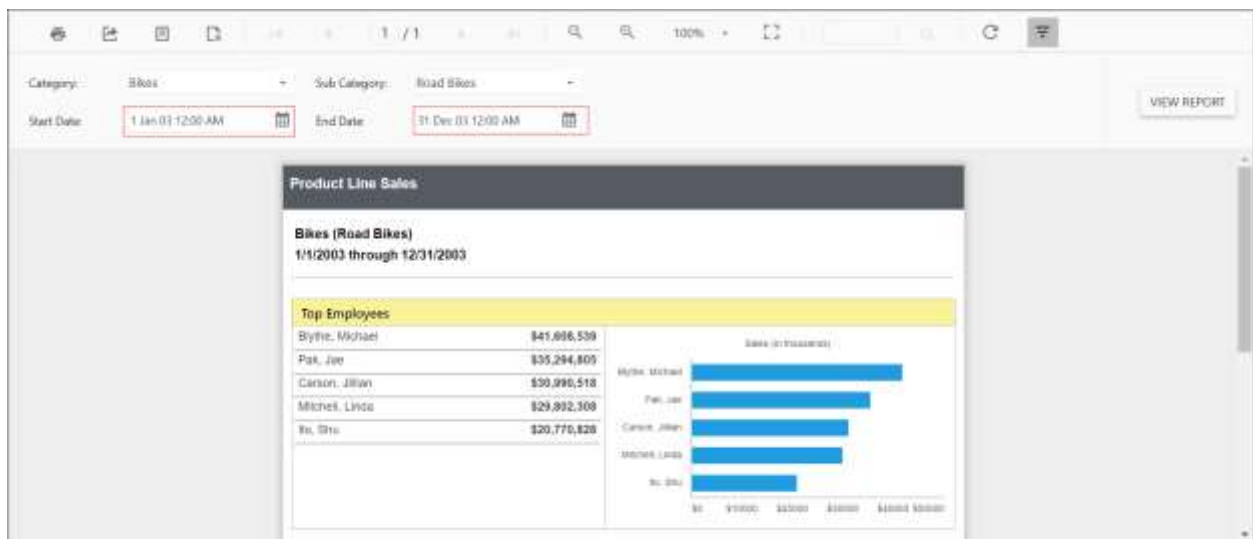
Set date and time format for parameter

The `DateTimeFormat` custom property defines the date time format to be displayed in the `DateTimePicker` popup. By default, the `DateTimeFormat` value is set as `empty`.

You can set the `DateTimeFormat` property value, as shown in the below.



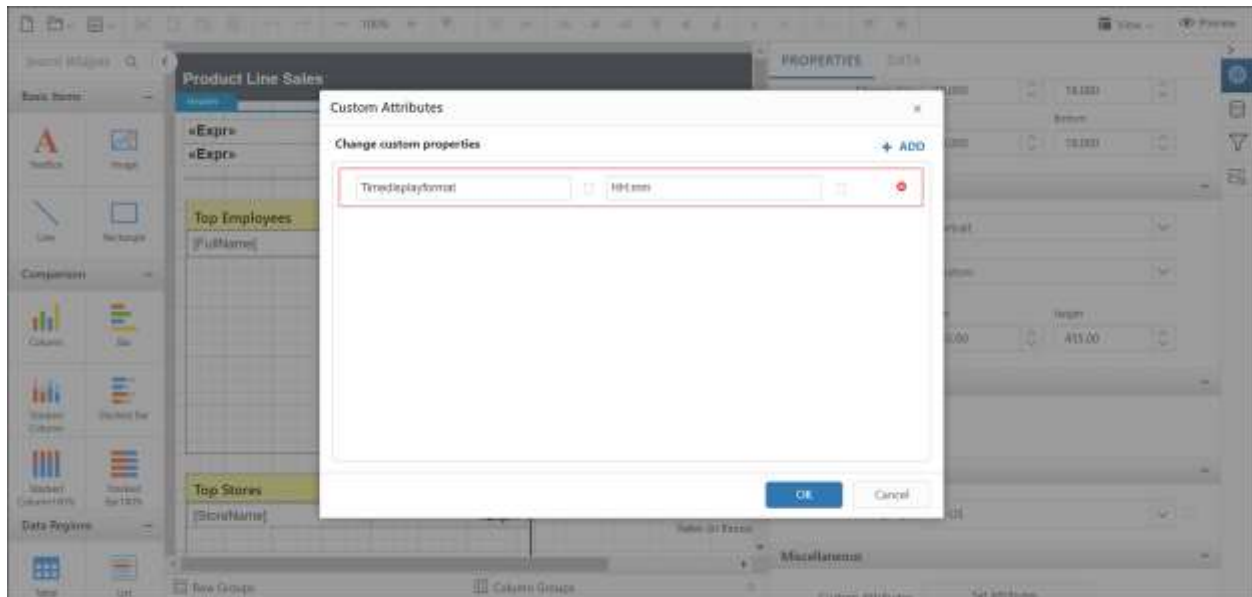
Preview the report and the see date time format in datetime parameter.



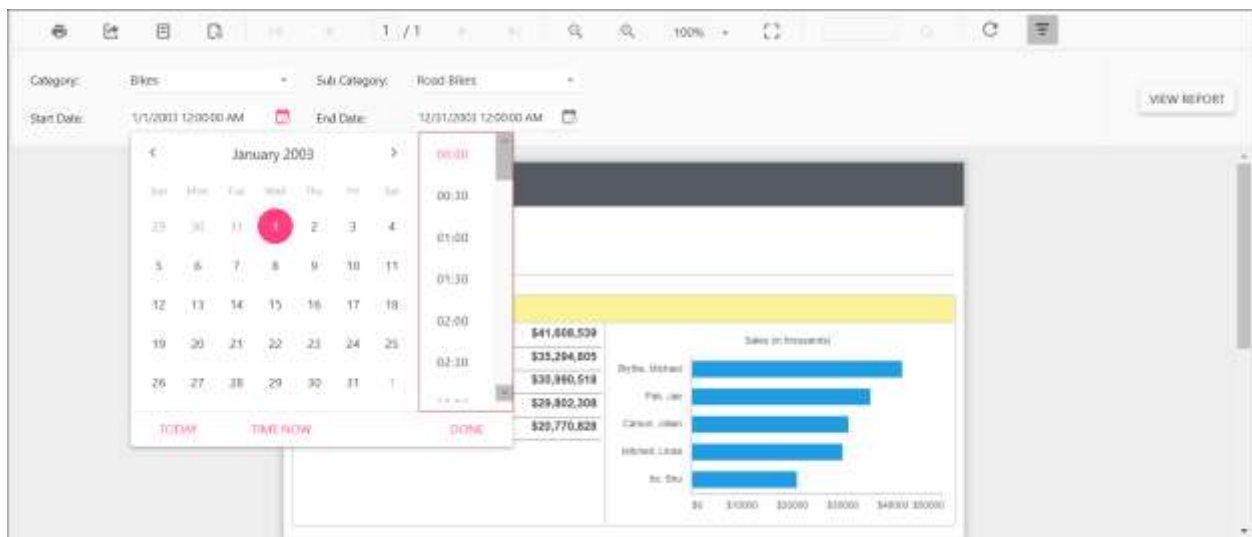
Change time display format for parameter

The `TimeDisplayFormat` custom property defines the time format to be displayed in the time dropdown inside the `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeDisplayFormat`. By default, the `TimeDisplayFormat` value is set as empty.

You can set the `TimeDisplayFormat` property value, as shown in the below.



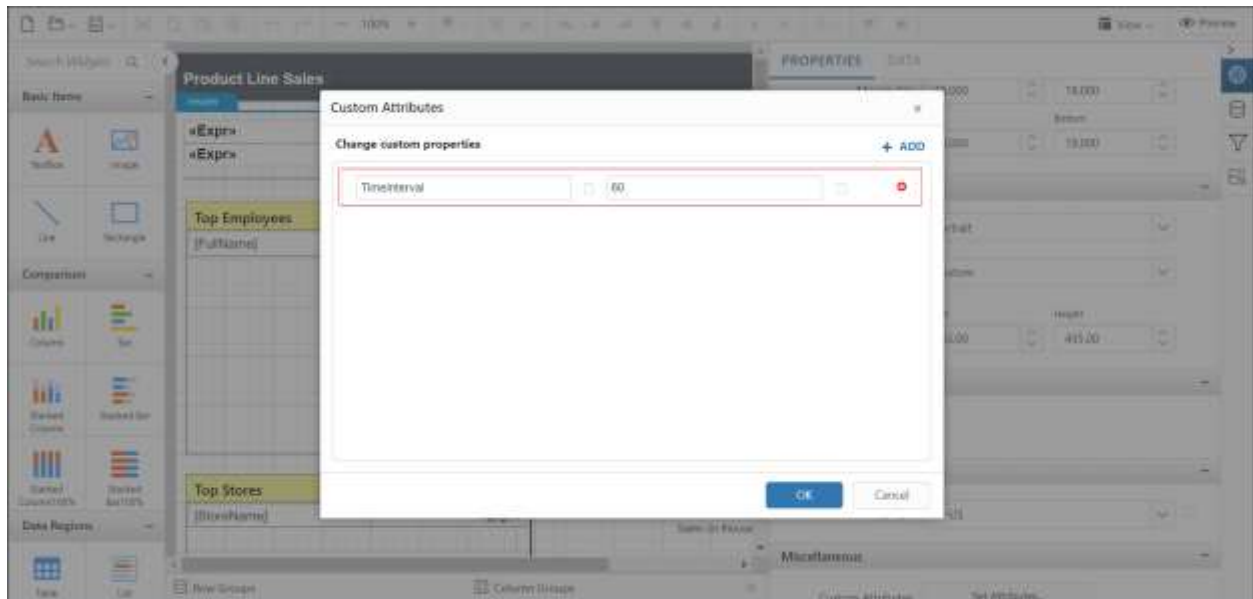
Preview the report and the see time format in datetime parameter.



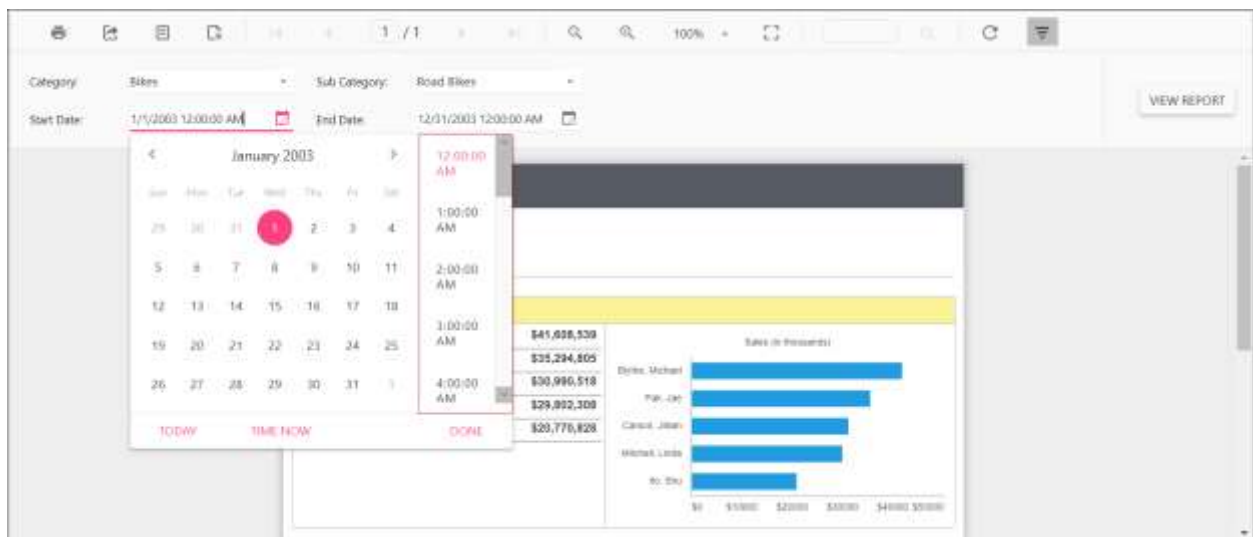
Set time interval in datetime parameter

The `TimeInterval` custom property is used to set the interval between the two adjacent time values in `DateTimePicker` popup. You must set the `DateTimePickerType` custom property as `DateTime` before setting the `TimeInterval`. By default, the `TimeInterval` value is set as 30.

You can set the `TimeInterval` property value, as shown in the below.



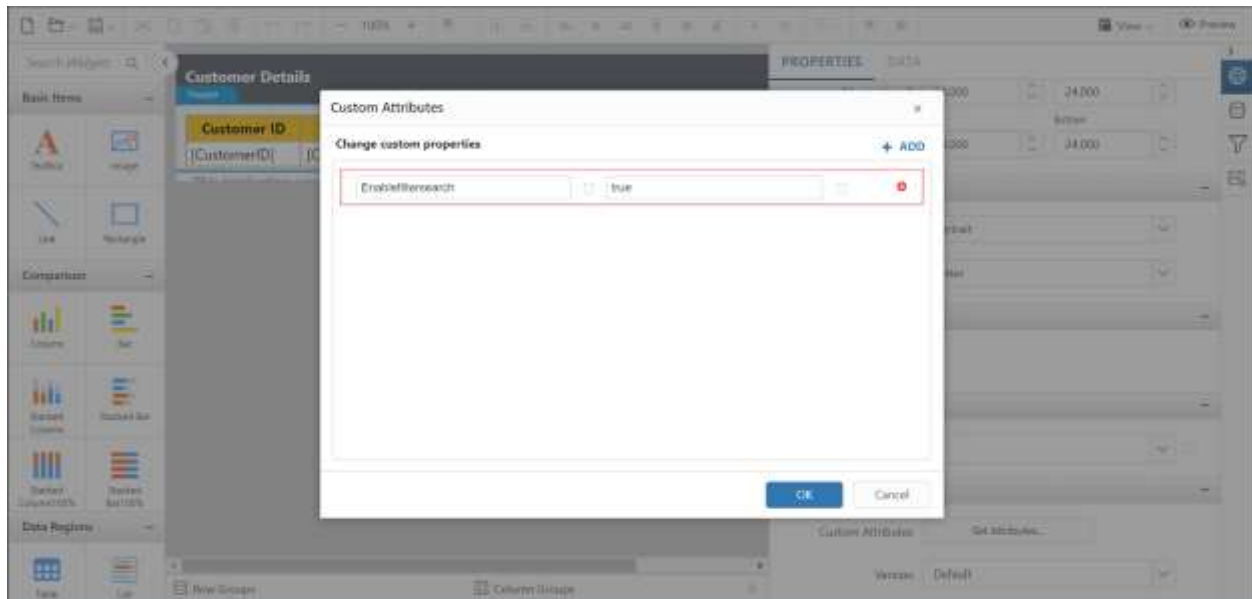
Preview the report and the see time interval in datetime parameter.



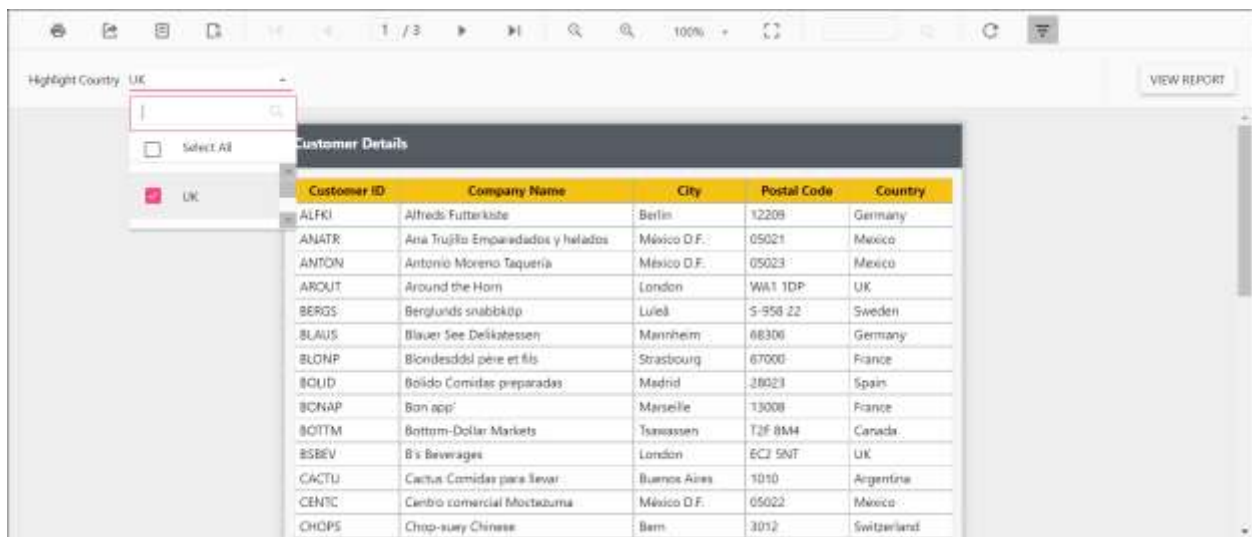
Enable filtering and searching in drop-down report parameter

Setting `EnableFilterSearch` custom property enables search and filtering option in dropdown parameter to easily find the values. The property value should be boolean. By default, the `EnableFilterSearch` value is `false`.

You can set the `EnableFilterSearch` property value, as shown in the below.



Preview the report and the see search filter in parameter.



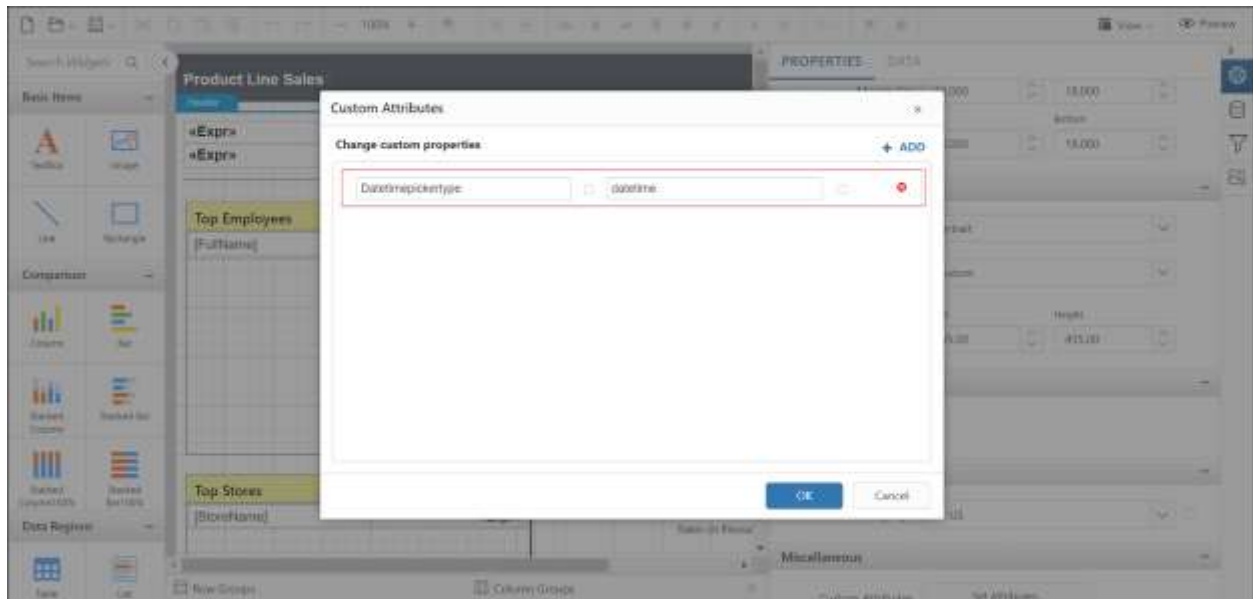
Change date report parameter to display date time picker

You can set `DateTimePickerType` custom property value as `DateTime` to change date parameter item to display `DateTimePicker` for value selection as shown the below.

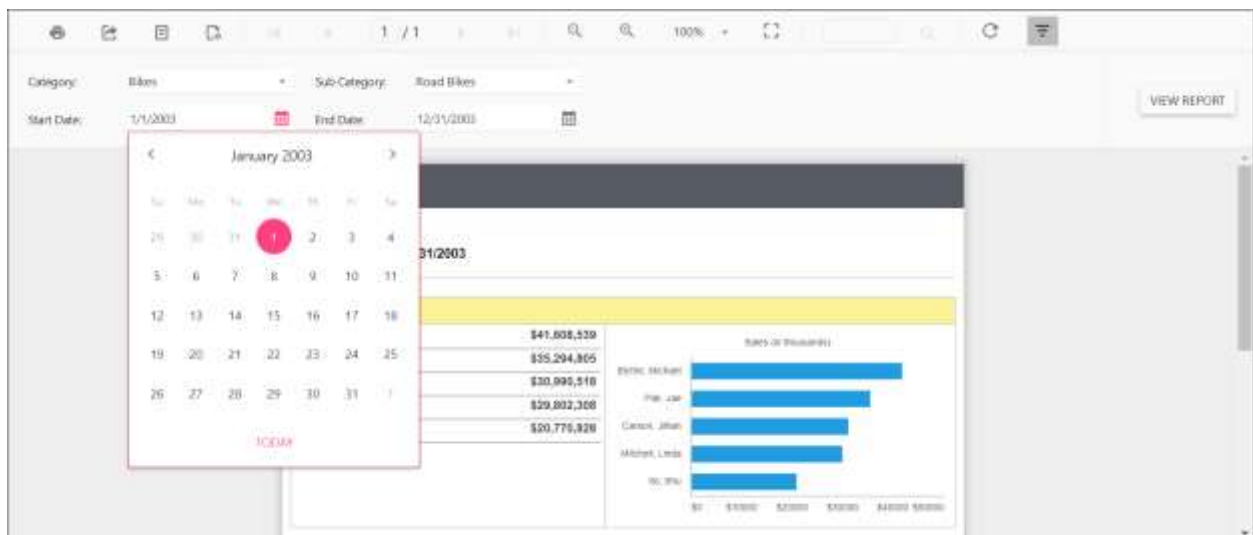
You can set the `DateTimePickerType` property value, as shown in the below.

Parameter custom properties

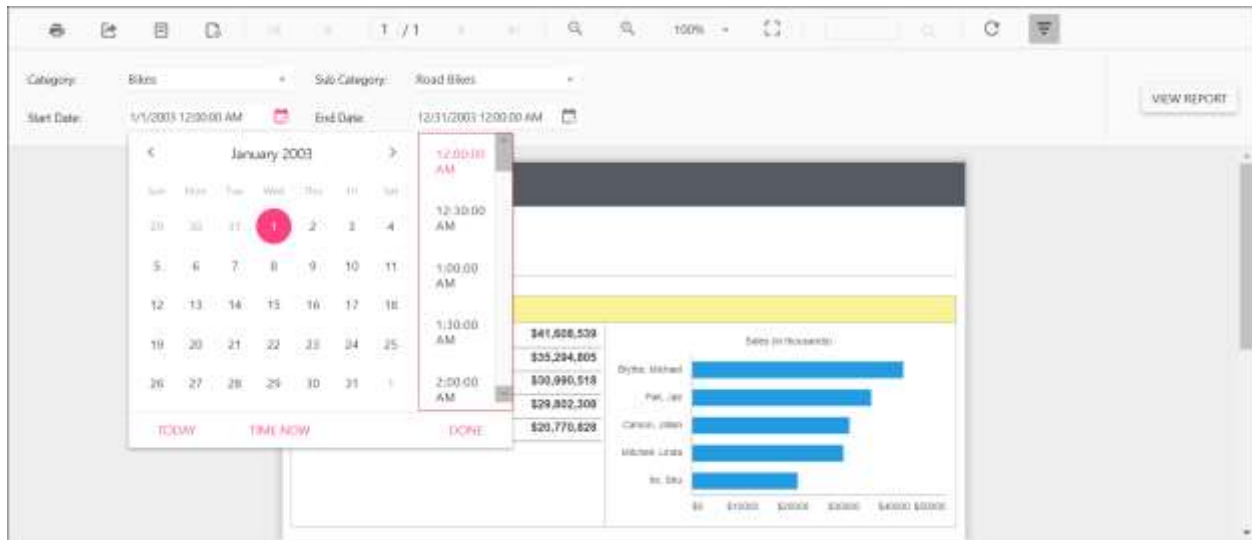
Change date report parameter to display date time picker



Before enabling date time picker type, the default value will be displayed as below..



Enable date time picker type and see the time in `DateTimePicker` as in below output.



Export custom properties

This topic explains the list of custom properties that are supported at the report level to control the export behaviour in ASP.NET Web Forms Report Viewer.

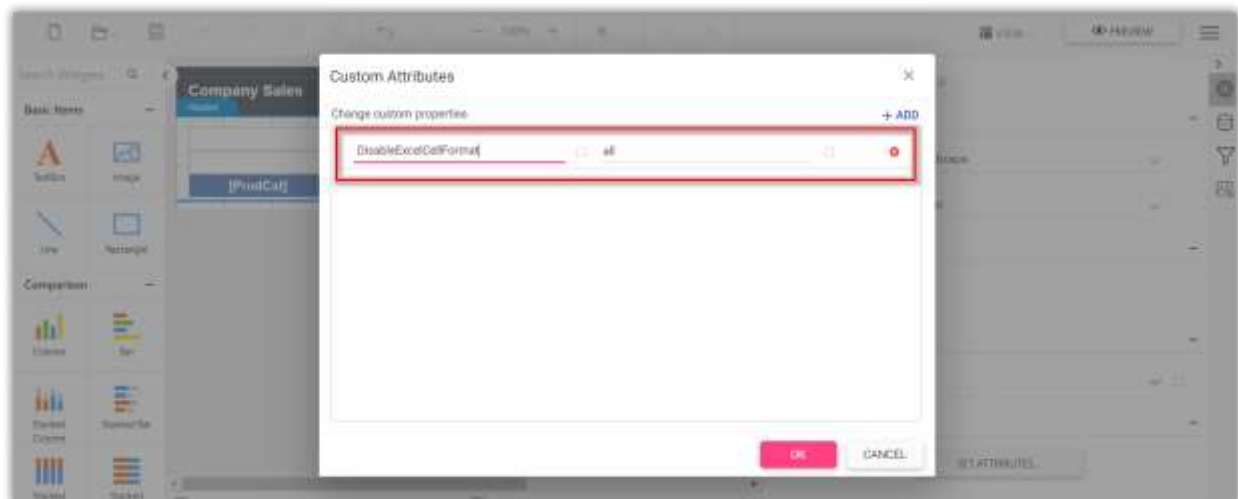
Improve excel export performance

The custom property `DisableExcelCellFormat` helps to improve the excel export performance by ignoring the rendering of report item styles. It allows property value from anyone of the following values.

- **Style** - Disables rendering of the cell styles like padding, background, color, and text style
- **Border** - Disables rendering of the cell border
- **All** - Disables rendering of the cell border, padding, background, color, and text style

Set the property value as **All** to improve maximum excel export performance.

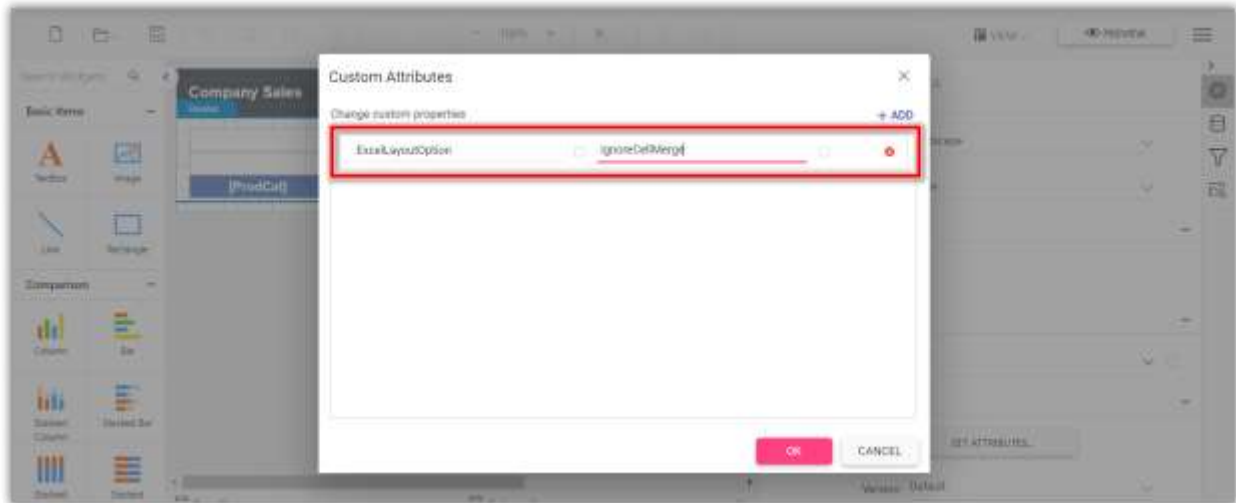
You can set the `DisableExcelCellFormat` custom property as shown below,



The `DisableExcelCellFormat` property must be added to report properties.

Improve excel export readability

Set `ExcelLayoutOption` custom property value as `IgnoreCellMerge` to improve the readability of the excel document by eliminating the tiny columns, rows, and merged cells. You can set the property value, as shown below,



The `ExcelLayoutOption` property must be added to report properties.

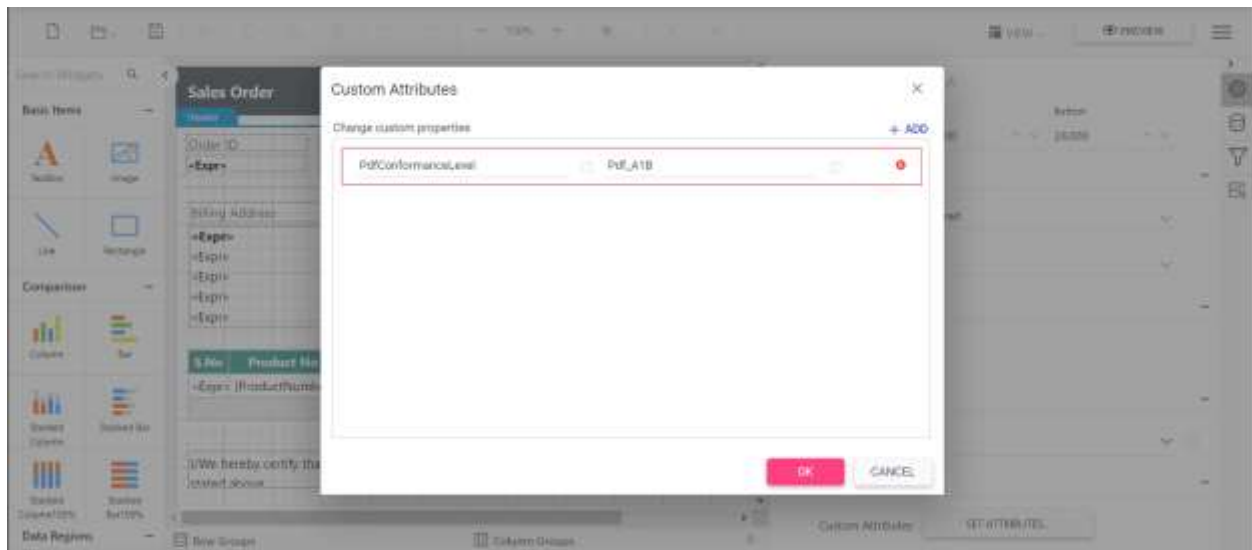
Set pdf conformance level

The `PdfConformanceLevel` allows you set PDF document versions.

You can set anyone of the following PDF conformance option.

- **PdfA1B** - You can create a PdfA1B document by specifying the conformance level as Pdf_A1B through PdfConformanceLevel Enum when creating the new PDF document.
- **PdfA2B** - You can create a PdfA2B document by specifying the conformance level as Pdf_A2B through PdfConformanceLevel Enum when creating the new PDF document
- **PdfA3B** - The PDF/A3B conformance supports the external files as attachment to the PDF document, so you can attach any document format such as Excel, Word, HTML, CAD, or XML files.
- **PdfA1A** - This conformance includes all PdfA1B requirements in addition to the features intended to improve a document's accessibility. PDF/A-1a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2A** - This conformance includes all PDF/A2B requirements in addition to the features intended to improve a document's accessibility. PDF/A-2a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2U** - This conformance includes all PdfA2U requirements, and additionally Unicode mapping for all text in the document.
- **PdfX1A2001** - You can create a PDF/X-1a document by specifying the conformance level as PdfX1A2001 through PdfConformanceLevel Enum when creating the new PDF document.

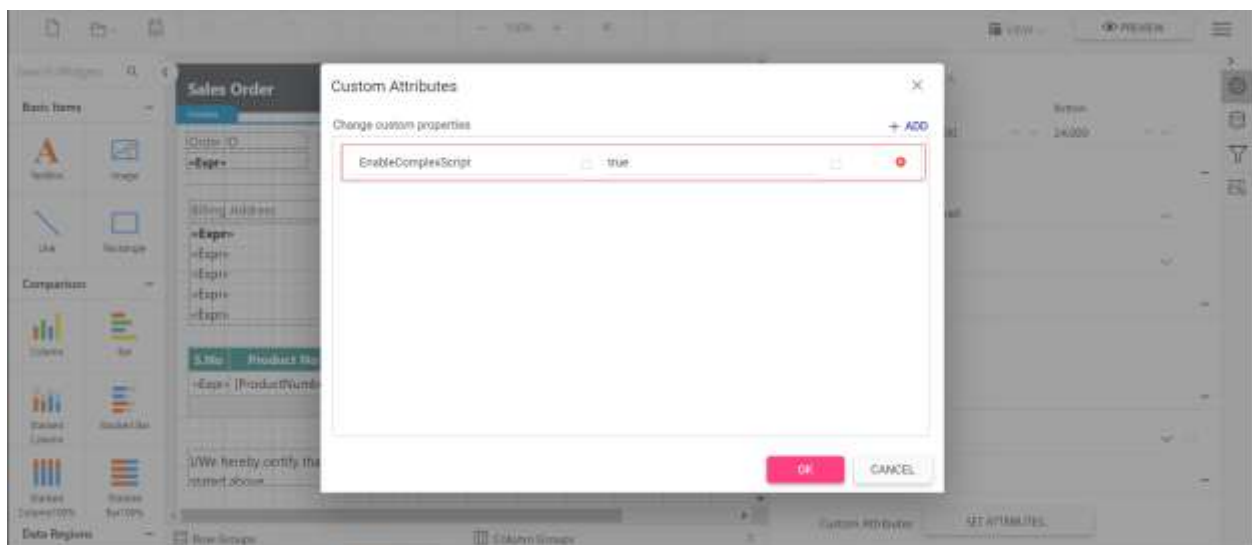
You can set the `PdfConformanceLevel` custom property as shown below,



Export pdf document with complex script

The `EnableComplexScript` custom property allows you to export pdf documents with complex script language texts.

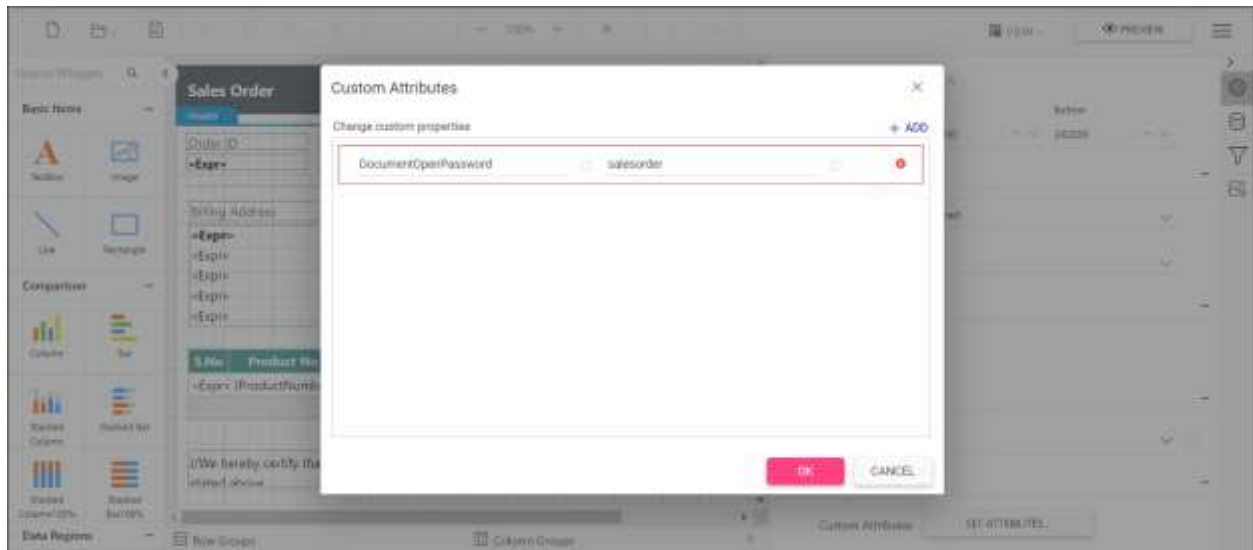
You can set the property value, as shown below,



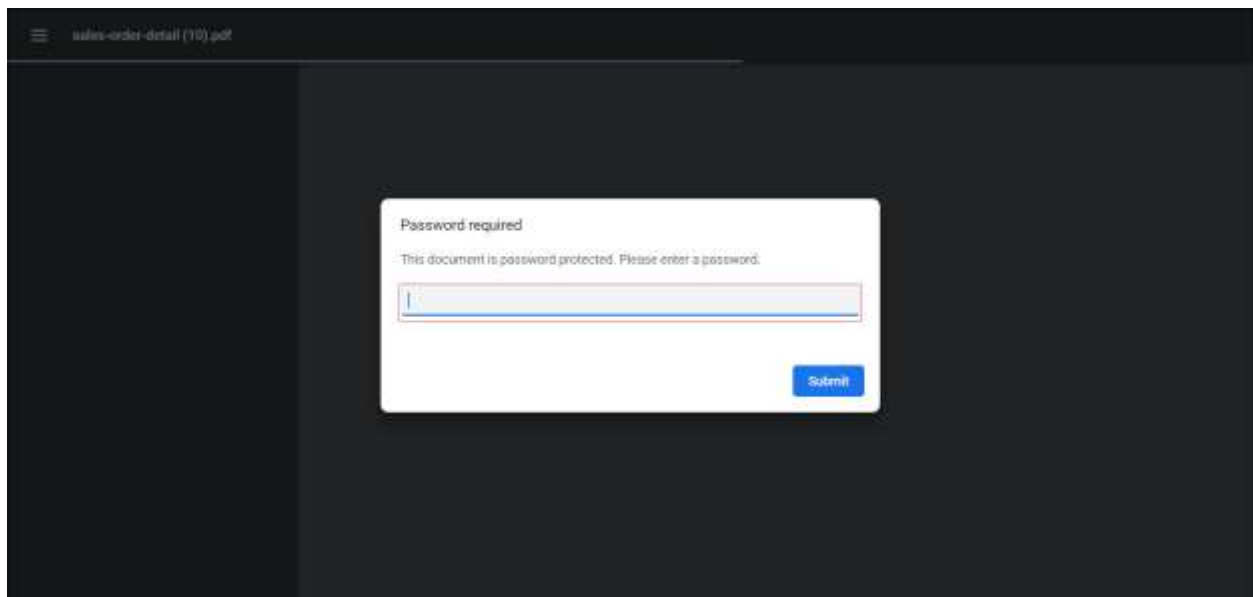
Encrypt and secure export documents

The custom property `DocumentOpenPassword` allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password.

You can set the property value, as shown below,



open the pdf document and see the below output.

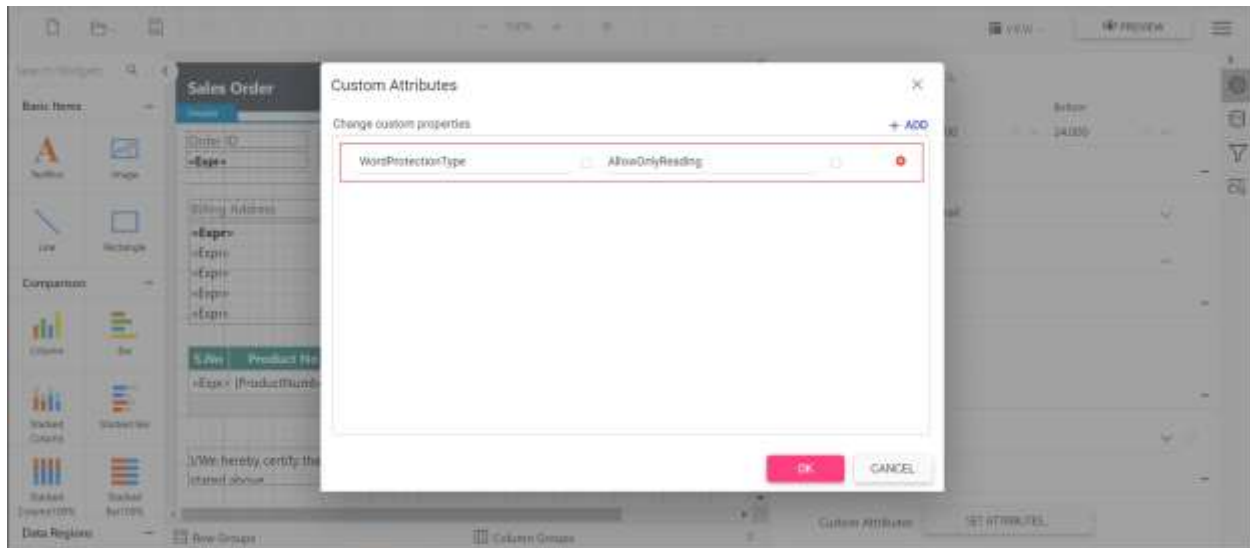


Protecting Word document from editing

The custom property `WordProtectionType` allows you to restrict a Word document from editing either by providing a password or without a password by using following types of protection.

- `AllowOnlyComments`- You can add or modify only the comments in the Word document.
- `AllowOnlyFormFields`- You can modify the form field values in the Word document.
- `AllowOnlyRevisions`- You can accept or reject the revisions in the Word document.
- `AllowOnlyReading`- You can only view the content in the Word document.
- `NoProtection`- You can access or edit the Word document contents as normally.

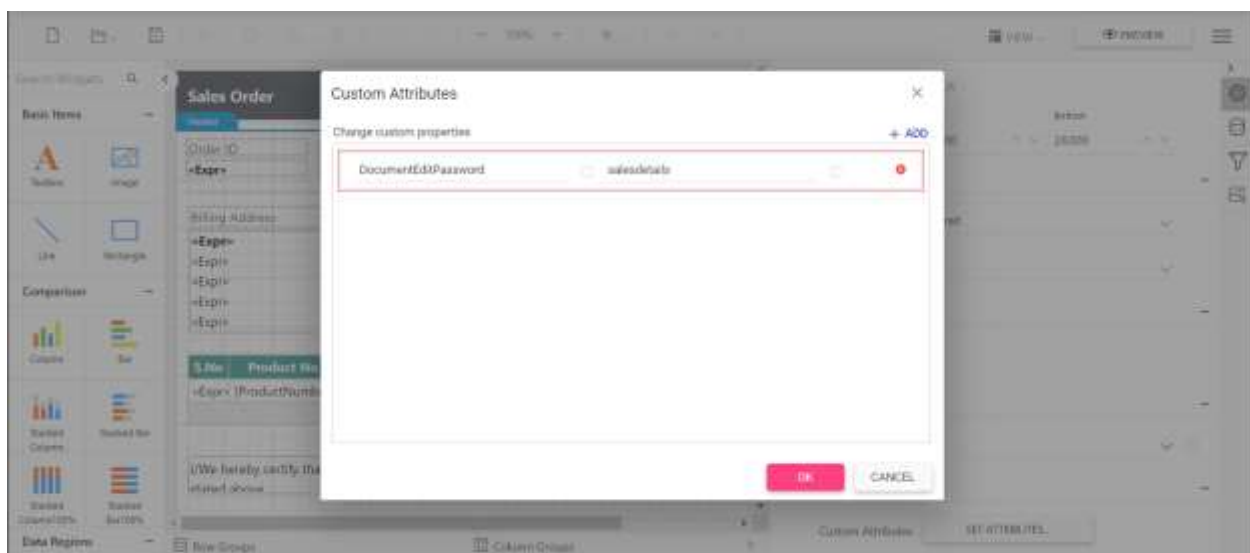
You can set the `WordProtectionType` custom property as shown below,



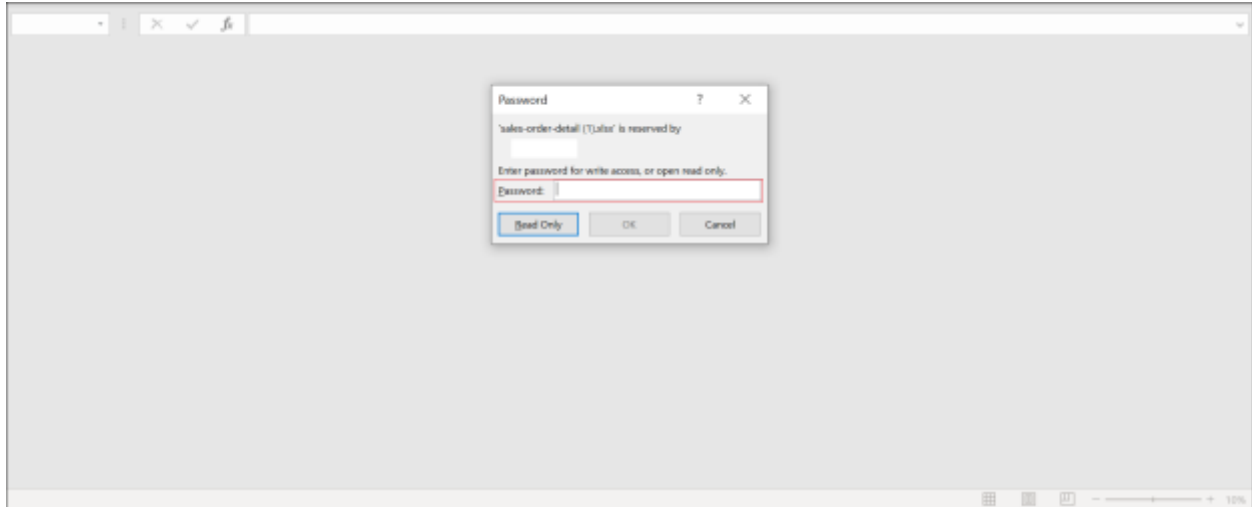
Set excel document edit password

The custom property `DocumentEditPassword` helps to allow specific users permission to modify the workbook data and save changes to the file in the excel document.

You can set the property value, as shown below,



open the excel document and see the below output.

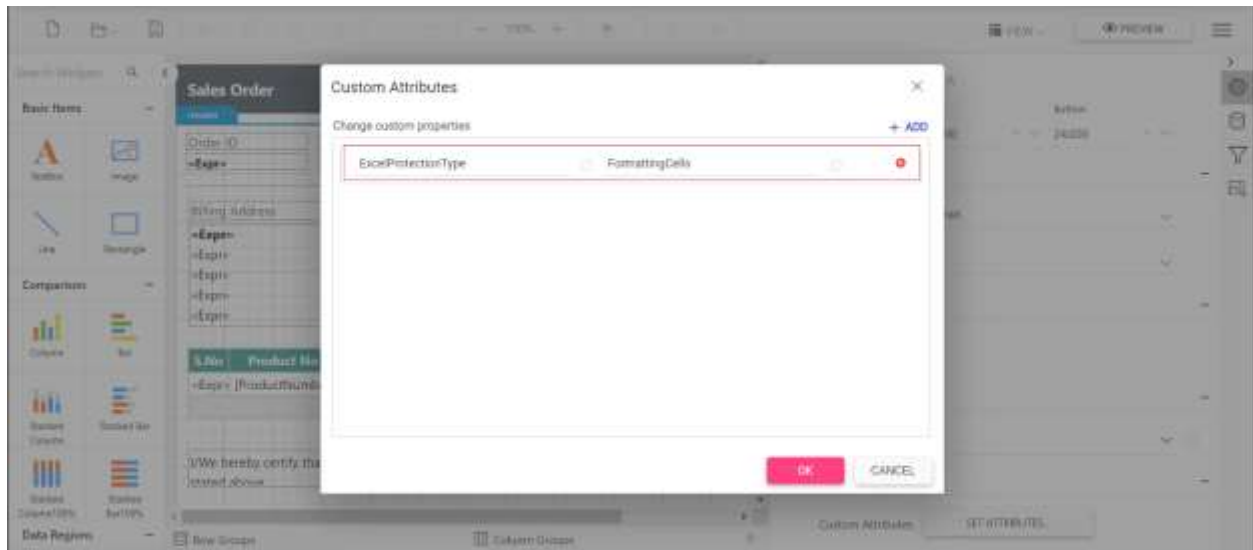


Set excel document protection

The custom property `ExcelProtectionType` allows you to protect the worksheet of excel document either by providing a password or without a password by using following types of protection.

- `None` - Represents no protection in excel sheet
- `Objects` - allows to protect shapes in excel sheet
- `Scenarios` - allows to protect scenarios.
- `FormattingCells` - allows the user to format any cell on a protected worksheet.
- `FormattingColumns` - allows the user to format any column on a protected worksheet.
- `FormattingRows` - allows the user to format any rows on a protected worksheet.
- `InsertingColumns` - allows the user to insert columns on the protected worksheet.
- `InsertingRows` - allows the user to insert rows on the protected worksheet.
- `InsertingHyperlinks` - allows the user to insert hyperlinks on the worksheet.
- `DeletingColumns` - allows the user to delete columns on the protected worksheet, where every cell in the column to be deleted is unlocked.
- `DeletingRows` - allows the user to delete rows on the protected worksheet, where every cell in the row to be deleted is unlocked.
- `LockedCells` - allows to protect locked cells.
- `Sorting` - allows the user to sort on the protected worksheet
- `Filtering` - allows the user to set filters on the protected worksheet. Users can change filter criteria but can not enable or disable an auto filter.
- `UsingPivotTables` - allows the user to use pivot table reports on the protected worksheet.
- `UnLockedCells` - allows to protect the user interface, but not macros.
- `Content` - allows to protect the contents in the excel sheet.
- `All` - allows to protect all type of protection.

You can set the `ExcelProtectionType` custom property as shown below,



Localization of Bold Reports ASP.NET Web Forms Report Viewer

Localization of ASP.NET Web Forms Report Viewer allows you to localize the static text such as tooltip, parameter block, and dialog text based on a specific culture. To render the static text with specific culture, refer to the following corresponding culture script files and set culture name to the **locale** property of the Report Viewer.

- ej.localetexts.fr-FR.min.js
- ej.culture.fr-FR.min.js

Refer this [CDN links for Localization and Culture](#) to get the Localization and Culture scripts for available Culture Code.

1. Install **BoldReports.Global** Nuget package in your ASP.NET application.
2. Refer to this **ej.localetexts.fr-FR.min.js** script file from the **Scripts** folder of your application.

`html

```
<script src="Scripts/bold-reports/i10n/reportdesigner/ej.localetexts.fr-FR.min.js"></script>
```

,

3. Refer to this **ej.culture.fr-FR.min.js** script file from the **Scripts** folder of your application.

`html

```
<script src="Scripts/bold-reports/i18n/ej.culture.fr-FR.min.js"></script>
```

,

4. To render the localization Report Viewer, use the following code snippet in your application.

`js

```

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
<link href="Content/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="Scripts/jquery-1.10.2.min.js"></script>
<script src="Scripts/common/bold.reports.common.min.js"></script>
<script src="Scripts/common/bold.reports.widgets.min.js"></script>
<script src="Scripts/data-visualization/ej.chart.min.js"></script>
<script src="Scripts/bold.report-viewer.min.js"></script>
<script src="Scripts/l10n/ej.localetexts.fr-FR.min.js"></script>
<script src="Scripts/i18n/ej.culture.fr-FR.min.js"></script>
<div style="height: 600px; width: 100%; min-height: 404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
locale="fr-FR">
</Bold:ReportViewer>
</div>
</asp:Content>
`

```

Responsive layout rendering of ASP.NET Report Viewer

Report Viewer will adaptively render itself with optimal user interfaces for phone, tablet, or desktop form factors. This helps your application to scale elegantly on all form factors with ease. You can enable responsive layout rendering in Report Viewer by setting `isResponsive` property to true.

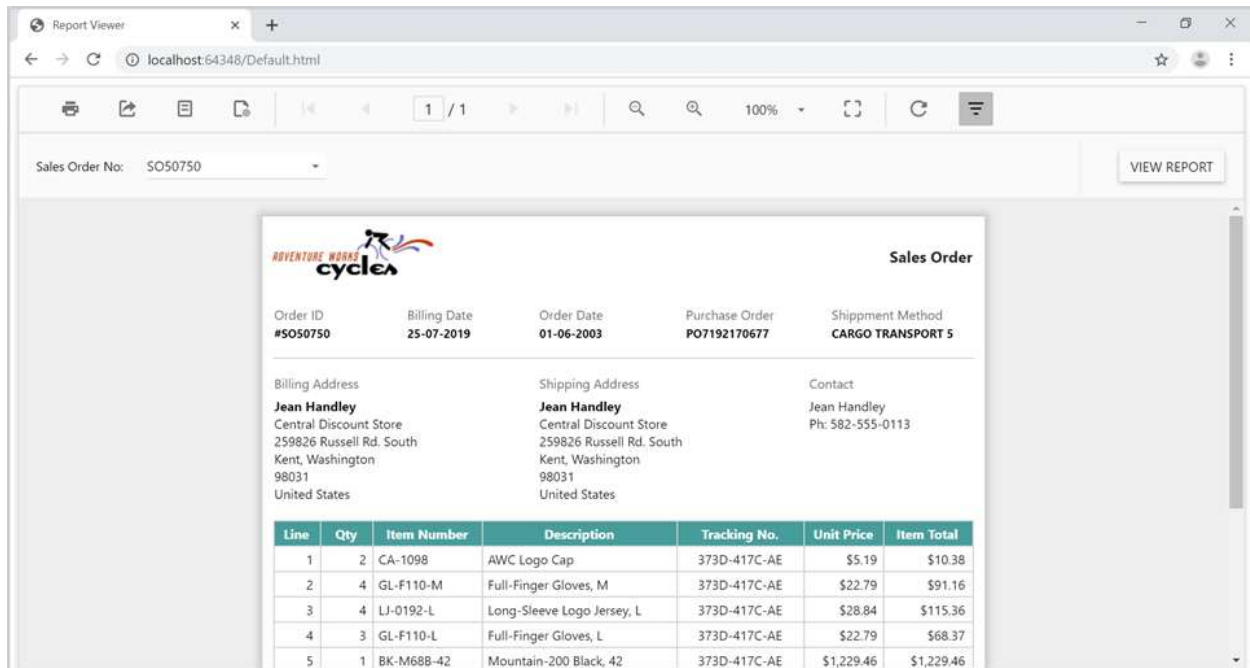
```

`js
<div style="height: 650px; width: 950px; min-height: 404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
IsResponsive="true">
</Bold:ReportViewer>
</div>
`

```

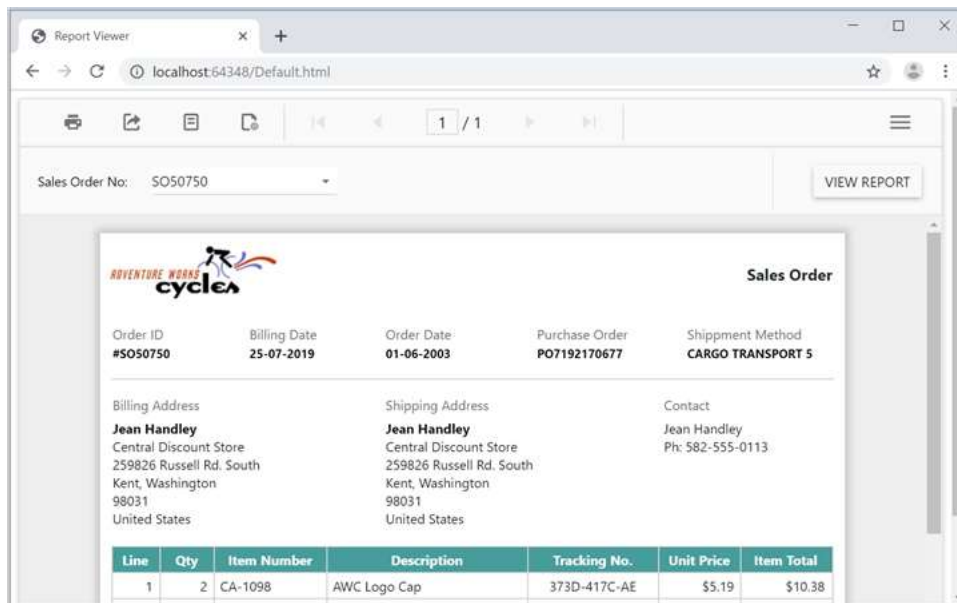
Normal layout

The following output shows the normal layout rendering of Report Viewer tool bar items.



Responsive layout

The following output shows the responsive layout rendering of Report Viewer tool bar items.



Limitations

RDL specification

The Report Viewer control does not support RDL specification for SQL Server 2000 and SQL Server 2005.

Report layout

- Vertical alignment in the text box report item is not supported in web rendering.

- In the Tablix cell split layout process, the entire cell moves to the next page to display the complete cell items, when the table cell width value exceeds the page width.

Expressions

The object function and VB function do not have complete support.

SSRS

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the server. If the report has any data source that uses credentials to connect with the database, then you should specify the data source credentials for each report data source to establish database connection.

HTML Formatted Data with Report

Report viewer supports showing the HTML formatted data with Textbox report items along with the inline CSS. This section provides the information about supported tags and limitations.

Supported HTML Tags

- Hyperlinks:
- Fonts:
- Header, style and block elements: ` ,

, ,

,

,

- , `
- Text format: , , ,

Limitations of Cascading Style Sheet Attributes

The following is a list of attributes that are supported:

- text-align, text-indent
- font-family
- font-size
- Only valid RDL size values are supported in absolute CSS length units. Supported units are: in, cm, mm, pt, pc, px, ex, and em.
- Relative CSS length units are ignored and are not supported. Unsupported units include percentage (%), and rem.
- color
- padding, padding-bottom, padding-top, padding-right, and padding-left
- font-weight

Samples and Demos

Browse and explore the ready-to-use RDL, RDLC reports, samples, online, and offline demos.

Locally installed reports

You can obtain the sample RDL and RDLC files from the Bold Reports installed location

`%localappdata%\Bold Reports\ReportsSDK\Samples\Common\Data\ReportTemplate.`

Offline demos

The offline samples are provided in the Bold Reporting Tools setup. For more details, refer to the [Bold Reporting Tools sample deployment](#).

Online demos

You can view the ASP.NET web forms Report Viewer online demo samples from [here](#).

GitHub demo samples

Click [here](#) to view the GitHub Report Viewer demo samples.

Migrate Report Viewer application

In our Bold Reports new assemblies are introduced for both client and server-side to resolve the compatibility problem between Essential Studio Report Viewer versions. It has changes in both Web API service and client-side scripts.

This section provides step-by-step instructions for migrating Report Viewer from Syncfusion Essential Studio release version to Bold Reports version of ASP.NET Web Forms Report Viewer application:

Client-side migration

1. In the Solution Explorer, right-click the **References** and remove the following assembly references:
 - Syncfusion.EJ
 - Syncfusion.EJ.Web
2. Add the assembly from the Bold Reports NuGet package `BoldReports.WebForms`. To add from NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search for `BoldReports.WebForms` NuGet package, and install in your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Scripts and CSS references

1. Remove the following scripts and CSS references from the Report Viewer `Default.aspx` page.
 - `ej.web.all.min.css`
 - `ej.web.all.min.js`
2. The Report Viewer scripts and style sheets are added to the `Scripts` and `Content` folders in your application when installing the `BoldReports.WebForms` NuGet package. Add scripts as in the following code sample.

```
`html
```

```
<link href="Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
```

```
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<script src="Scripts/bold-reports/bold.report-viewer.min.js"></script>
```

Adding data visualization scripts

To render the report with data visualization components such as chart, gauge, and map items, add scripts of the visualization element. The following table shows the script reference that needs to be added in Report Viewer page for data visualization elements.

Visualization item | Script file

Gauge | ej2-base.min.js, ej2-data.min.js, ej2-pdf-export.min.js, ej2-svg-base.min.js, ej2-lineargauge.min.js and ej2-circulargauge.min.js |

Map | ej2-maps.min.js

Chart | ej.chart.min.js

To render the chart report item, add chart control script `ej.chart.min.js` before the `bold.report-viewer.min.js` reference in the `Default.aspx` page as demonstrated in the following code sample.

```
`html
<link href="Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<!--Used to render the chart item. Add this script only if your report contains the chart report item.-->
<script src="Scripts/bold-reports/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="Scripts/bold-reports/bold.report-viewer.min.js"></script>
```

The following code can be used to render the chart, gauge, and map report items in Report Viewer.

```
`html
<link href="Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<!--Used to render the gauge item. Add this script only if your report contains the gauge report item. -->
<script src="Scripts/bold-reports/common/ej2-base.min.js"></script>
<script src="Scripts/bold-reports/common/ej2-data.min.js"></script>
<script src="Scripts/bold-reports/common/ej2-pdf-export.min.js"></script>
<script src="Scripts/bold-reports/common/ej2-svg-base.min.js"></script>
```

```

<script src="Scripts/bold-reports/data-visualization/ej2-lineargauge.min.js"></script>
<script src="Scripts/bold-reports/data-visualization/ej2-circulargauge.min.js"></script>
<!--Used to render the map item. Add this script only if your report contains the map report item.-->
<script src="Scripts/bold-reports/data-visualization/ej2-maps.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<!--Used to render the chart item. Add this script only if your report contains the chart report item.-->
<script src="Scripts/bold-reports/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="Scripts/bold-reports/bold.report-viewer.min.js"></script>
,

```

Control initialization

1. The component prefix has been changed from **ej** to **Bold**.
2. Open the **Web.config** file and add the **BoldReports.WebForms** assembly reference to the element with **Bold** tag prefix.

```

`html
<configuration>
....
....
<system.web>
....
<pages>
....
<controls>
<add assembly="BoldReports.WebForms" namespace="BoldReports.WebForms" tagPrefix="Bold" />
....
</controls>
</pages>
</system.web>
....
....
</configuration>
,

```


3. Open the Report Viewer component page.
4. Replace the tag `ej:ReportViewer` as `Bold:ReportViewer`.

```
`js
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer">
</Bold:ReportViewer>
</div>
`
```

If your application contains Report Viewer initialization in multiple pages then replace in all the pages.

Server-side migration

1. In the Solution Explorer, right-click the **References** and remove the `Syncfusion.EJ.ReportViewer` assembly reference.
2. Add the assembly from the Bold Reports NuGet package `BoldReports.Web`. To add from NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search for `BoldReports.Web` NuGet package, and then install in your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Web API Controller

1. The `IReportController` interface is moved to `BoldReports.Web.ReportViewer`. Open the Report Viewer Web API Controller file and remove the following using statement.

```
`csharp
using Syncfusion.EJ.ReportViewer;
`
```

2. Add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

Your application is successfully upgraded to the latest version of Report Viewer, and you can run the application with new assemblies.

Report export configuration

Now, the `BoldReports.Web` can export the reports with data visualization components only using web components. It is mandatory to configure the web scripts in Report Viewer Web API controller for

exporting data visualization components such as chart, gauge, and map that are used in report definition. To configure the scripts in Web API, refer to the following steps:

1. Open the Report Viewer Web API controller.
2. Configure the following scripts and styles in `OnInitReportOptions` on Web API controller:
 - o `jquery-1.10.2.min.js`
 - o `ej2-base.min.js`, `ej2-data.min.js`, `ej2-pdf-export.min.js`, `ej2-svg-base.min.js`, `ej2-lineargauge.min.js` and `ej2-circulargauge.min.js` - Exports the gauge item. Add this script only if your report contains the gauge report item.
 - o `ej2-maps.min.js` - Exports the map item. Add this script only if your report contains the map report item.
 - o `bold.reports.common.min.js`
 - o `bold.reports.widgets.min.js`
 - o `ej.chart.min.js` - Exports the chart item. Add this script only if your report contains the chart report item.
 - o `bold.report-viewer.min.js`
3. Replace the following codes in Report Viewer Web API controller.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    var resourcesPath = System.Web.Hosting.HostingEnvironment.MapPath("~/Scripts");
    reportOption.ReportModel.ExportResources.Scripts = new List<string>
    {
        //Gauge component scripts
        resourcesPath + @"\"bold-reports\common\ej2-base.min.js",
        resourcesPath + @"\"bold-reports\common\ej2-data.min.js",
        resourcesPath + @"\"bold-reports\common\ej2-pdf-export.min.js",
        resourcesPath + @"\"bold-reports\common\ej2-svg-base.min.js",
        resourcesPath + @"\"bold-reports\data-visualization\ej2-lineargauge.min.js",
        resourcesPath + @"\"bold-reports\data-visualization\ej2-circulargauge.min.js",
        //Map component script
        resourcesPath + @"\"bold-reports\data-visualization\ej2-maps.min.js",
        resourcesPath + @"\"bold-reports\common\bold.reports.common.min.js",
        resourcesPath + @"\"bold-reports\common\bold.reports.widgets.min.js",
        //Chart component script
        resourcesPath + @"\"bold-reports\data-visualization\ej.chart.min.js",
```

```
//Report Viewer Script
resourcesPath + @"\"bold-reports\"bold.report-viewer.min.js"
};
reportOption.ReportModel.ExportResources.DependentScripts = new List<string>
{
resourcesPath + @"\"jquery-1.7.1.min.js"
};
}
`
```

The data visualization components will not export without the above script configurations.

NuGet Packages for ASP.NET Web Forms

Refer to the following steps to configure Bold Reporting NuGet packages for ASP.NET Web Forms application.

Configure NuGet feed URL

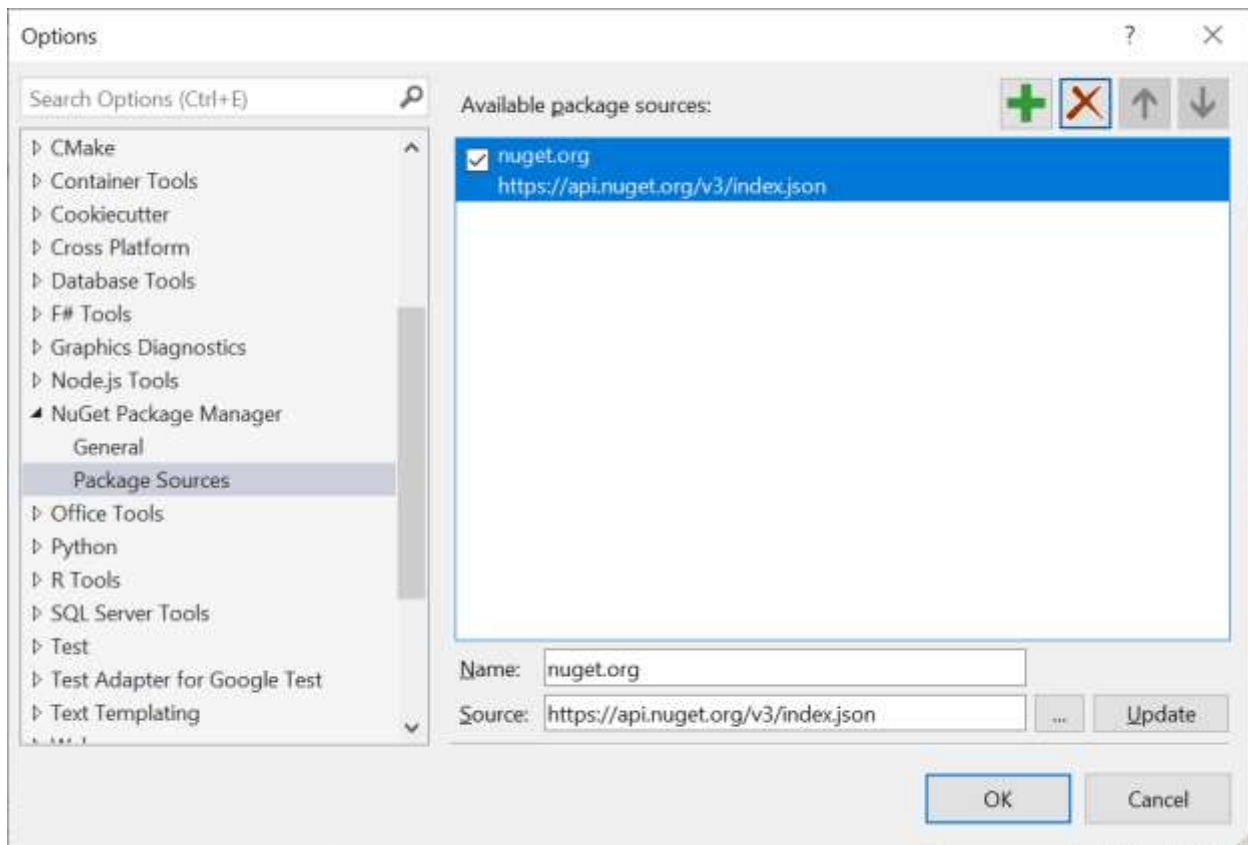
Online NuGet feed URL

The Bold Reporting NuGet packages are published in **Nuget.org**. To configure the online packages, use the following steps:

1. Open Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager** and select **Package Sources**.
4. Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: NuGet.org

Source: https://api.nuget.org/v3/index.json



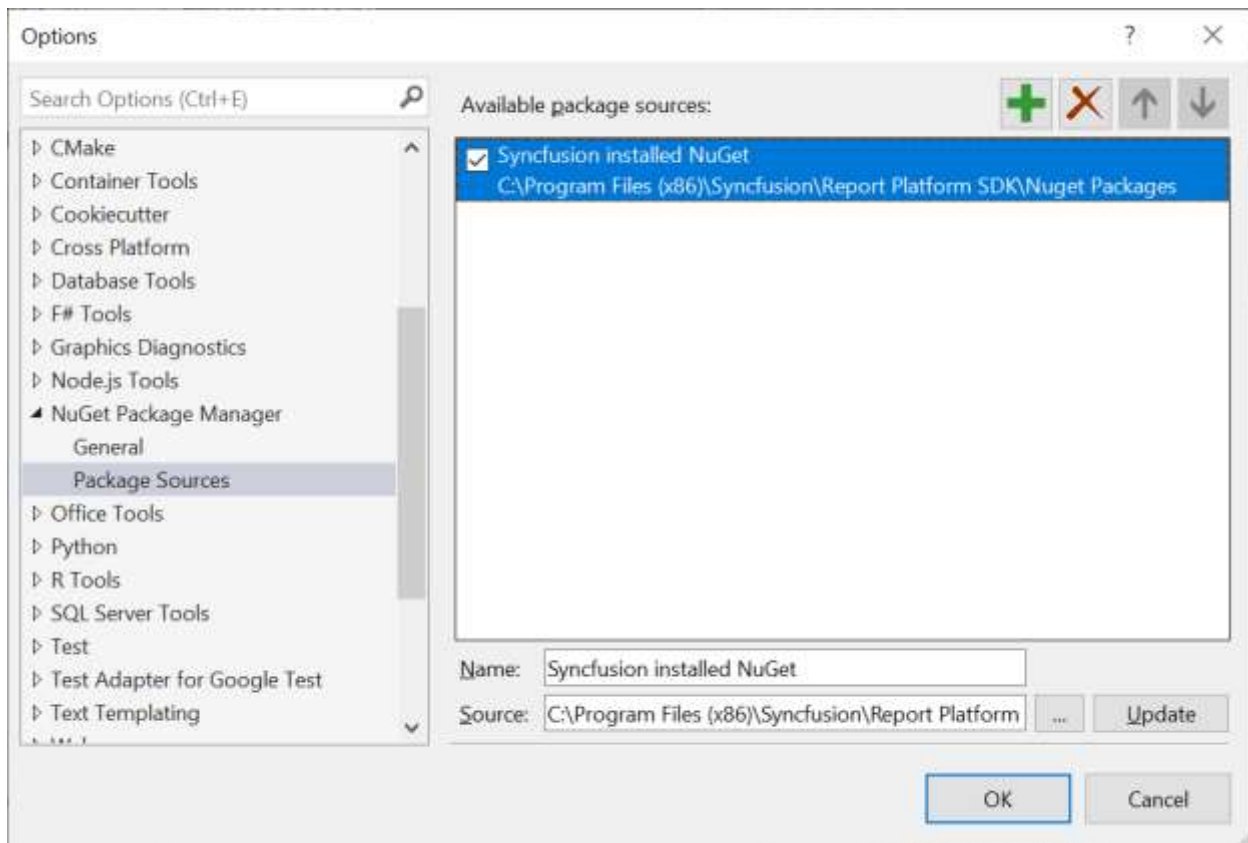
Offline NuGet feed URL

Bold Reporting NuGet packages are shipped into our Bold Reporting Tools build. To configure the packages from Bold Reports installed location, use the following steps:

1. Open your Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager**, and then select **Package Sources**.
4. Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: Bold Reports installed NuGet

Source: {System Drive}:\Program Files (x86)\Bold Reports\Reporting Tools\Nuget Packages.



The system drive varies based on the installed location in your machine.

Installing NuGet packages

Install using NuGet Package Manager

The NuGet Package Manager can be used to search and install NuGet packages in Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution**. Alternatively, right-click the project/solution in Solution Explorer tab, and choose **Manage NuGet Packages**.
2. By default, the **NuGet.org** package is selected in the **Package source** drop-down. Select package source, search for the packages (**BoldReports.WebForms** or **BoldReports.Web**), and then click **Install** button.

Install using Package Manager Console

To install the Bold Reporting component using the Package Manager Console as NuGet packages,

1. On the **Tools** menu, select **NuGet Package Manager**, and then click **Package Manager Console**.
2. Run the following NuGet installation commands:

```
`cmd
```

install specified package in default project

Install-Package <Package Name>

install specified package in default project with specified package source

Install-Package <Package Name> -Source <Source Location>

install specified package in specified project

Install-Package <Package Name> -ProjectName <Project Name>

,

For example:

`cmd

install specified package in default project

Install-Package BoldReports.WebForms

install specified package in default project with specified Package Source

Install-Package BoldReports.Web -Source "https://api.nuget.org/v3/index.json"

install specified package in specified project

Install-Package BoldReports.WebForms -ProjectName BoldReportsApplication

,

Upgrading NuGet packages

Upgrading using NuGet Package Manager

NuGet packages can be updated to their specific version or latest version available in the Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution....** Alternatively, right-click the project/solution in the Solution Explorer tab, and then choose **Manage NuGet Packages**.
2. Select the **Updates** tab to see the packages available for update from the desired package sources, select the required packages and specific version from the drop-down, and then click the **Update** button.

Upgrading using Package Manger Console

To update the installed Bold Reporting NuGet packages using the Package Manager Console:

1. On the **Tools** menu, select **NuGet Package Manager**, and then select **Package Manager Console**.
2. Run the following NuGet installation commands:

`cmd

Update specific NuGet package in default project

Update-Package <Package Name>

Update all the packages in default project

Update-Package

Update specified package in default project with specified package source

Update-Package <Package Name> -Source <Source Location>

Update specified package in specified project

Update-Package <Package Name> - ProjectName <Project Name>

,

For example:

`cmd

Update specified Bold Reporting NuGet package

Update-Package BoldReports.Web

Update specified package in default project with specified Package Source

Update-Package BoldReports.Web -Source "https://api.nuget.org/v3/index.json"

Update specified package in specified project

Update-Package BoldReports.Web -ProjectName BoldReportsApplication

,

Upgrading using NuGet CLI

Using the NuGet CLI, all the NuGet packages in the project can be updated to the available latest version:

1. Download the latest [NuGet CLI](#).

To update the existing `nuget.exe` to latest version use the following command:

`cmd

nuget update -self

,

2. Open the downloaded executable location in the command window. Run the following "update commands" to update the Bold Reporting NuGet packages.

```
`cmd
```

update all NuGet packages from config file

```
nuget update <configPath> [options]
```

update all NuGet packages from specified Packages Source

```
nuget update -Source <Source Location> [optional]
```

,

`configPath` is optional. It identifies the `package.config` or solutions file lists the packages utilized in the project.

For example:

```
`cmd
```

Update all NuGet packages from config file

```
nuget update "C:\Users\BoldReportsApplication\package.config"
```

Update all NuGet packages from specified Packages Source

```
nuget update -Source "https://api.nuget.org/v3/index.json"
```

,

The Update command is not working as expected in Mono (Mac and Linux) and projects using PackageReference format.

Frequently asked questions

This section helps to get the answer for the frequently asked questions in Bold Reports ASP.NET Webforms Report Viewer.

1. [How can improve the performance and handle the large amounts of data with Report Viewer?](#)
2. [Is Report Viewer compatible with latest version of JQuery library?](#)
3. [Is the back action from drillthrough report will load the report again with Report Viewer?](#)
4. [Is possible to change the culture of the date time parameter](#)
5. [Is it possible to hide report parameters?](#)
6. [Is it possible to hide the export options in Report Viewer?](#)
7. [Is it possible to load reports providing parameter values at runtime?](#)

Is possible to change the culture of the date time parameter

Yes, we can change the culture of the date time parameter for Report Viewer by changing the locale. You can make use the following reference to change the locale of Report Viewer.

[ReportViewer Localization](#)

In Report Viewer, we could not change the culture of specific parameter or UI. So, we have to achieve the requirements only by changing the culture.

Is it possible to hide the parameters in Report Viewer

Yes, it is possible to hide the parameters in Report Viewer. On hiding the parameters, the users will not be able to have the parameter block in the Report Viewer. Refer to this [Hide parameter block and toolbar items](#) section.

Is it possible to hide the export options in Report Viewer

Yes, it is possible to hide the export options in Report Viewer. The Report Viewer provides the `exportOptions` property to show or hide the default export types available in the component. Refer to this [Decide or Hide the export options](#) section.

Is it possible to load reports providing parameter values at runtime

Yes, it is possible to load reports with application inputs as parameters in Report Viewer. You need to provide the input values to the Report Viewer from client side at runtime using the `parameters` property, which accepts JSON array values. Refer to this [Set parameter at client](#) section.

How to Create RDL/RDLC Report

The following sections explain about how to create a new RDL/RDLC report using Bold Report Designer, Microsoft Report Builder and Visual Studio Report Server project template.

- [Create a RDL report](#)
- [Create a RDLC report](#)
- [Change the exporting document file name based on the parameter](#)
- [Change the connection string datasource dynamically](#)
- [Disable the vertical scrollbar in parameter panel](#)
- [Pass multiple values using custom data](#)
- [How to clear cache when closing the Report Viewer?](#)

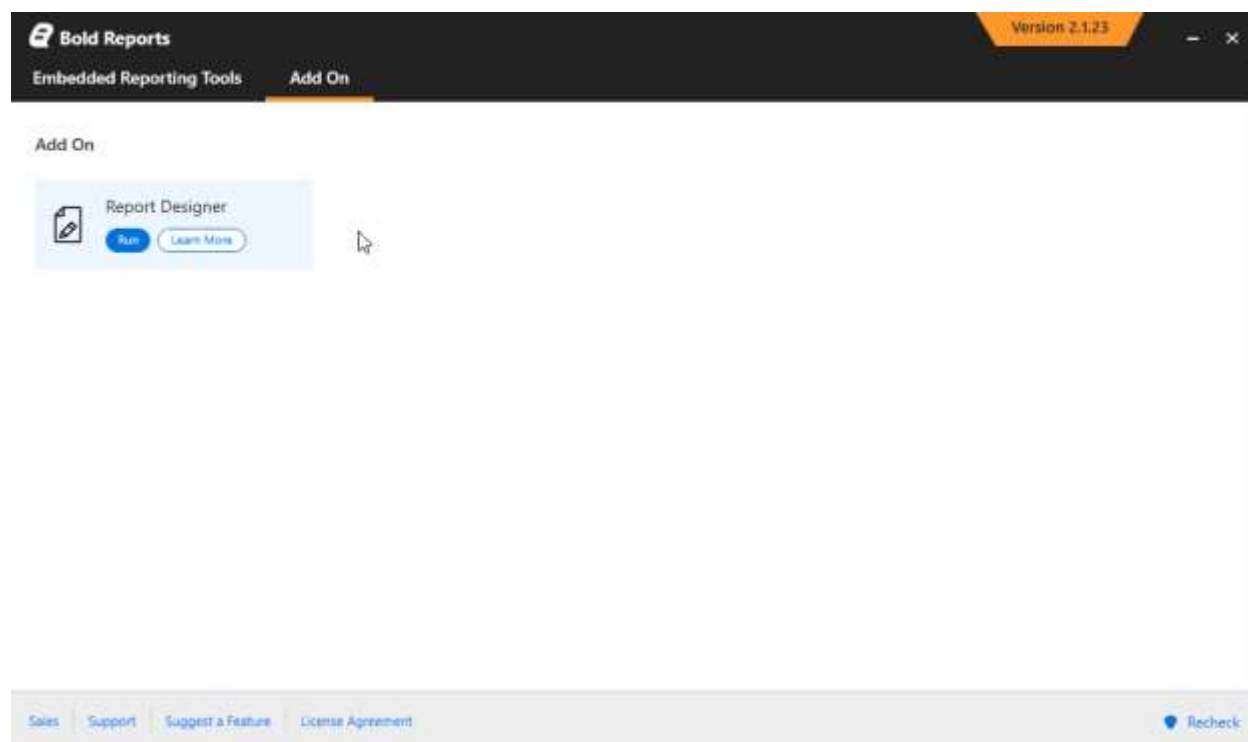
Create a SSRS RDL report

You can create an RDL report using any of the following reporting tools:

- Bold Reports Report Designer.
- Microsoft Report Builder.
- Visual Studio Report Server project template.

Bold Reports Report Designer

Bold Reports Report Designer provides the intuitive user interface to create and edit the RDL reports, which is available in Bold Embedded Reporting Tools Control Panel Add On.



Microsoft SQL Report Builder

You can create an RDL report using the Microsoft stand-alone Report Builder. For more details, refer to this [online documentation](#).

Visual Studio Report Server template

To create an RDL report in Visual Studio, a Report Server project is required where you can save your report definition (.rdl) file. For more details, refer to this [Visual Studio documentation](#).

If you do not have the Business Intelligence or Report Server Project options, you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create a RDLC report using business object data source

This section describes step by step procedure to create an RDLC report using Visual Studio Reporting project type.

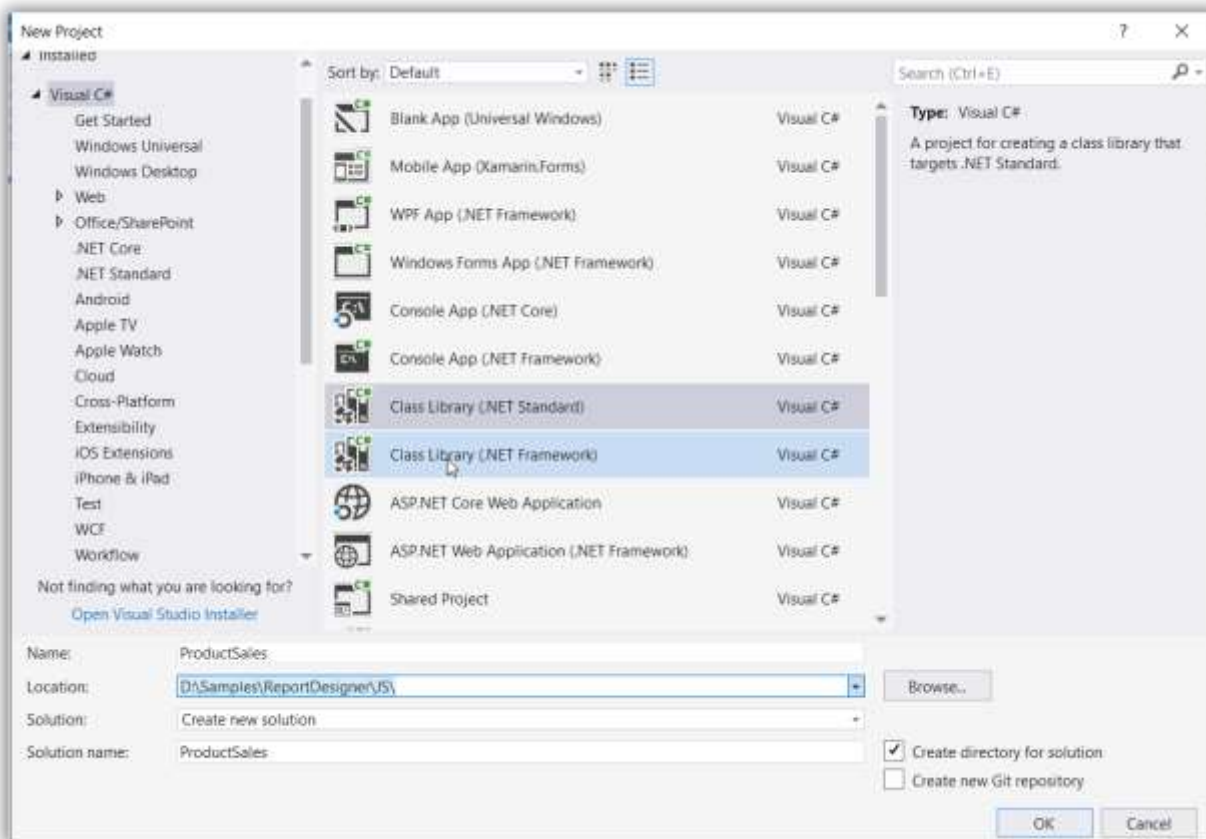
Prerequisites

- Microsoft Visual Studio 2017 or higher
- [Microsoft RDLC Report Designer](#)

If you are using Microsoft Visual Studio lower to 2017 version then you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create business object class

1. Open Visual Studio from the File menu and select **New Project**.
2. Create project with class library type from the project type list.



3. Create the class with necessary properties. You can find the reference below,

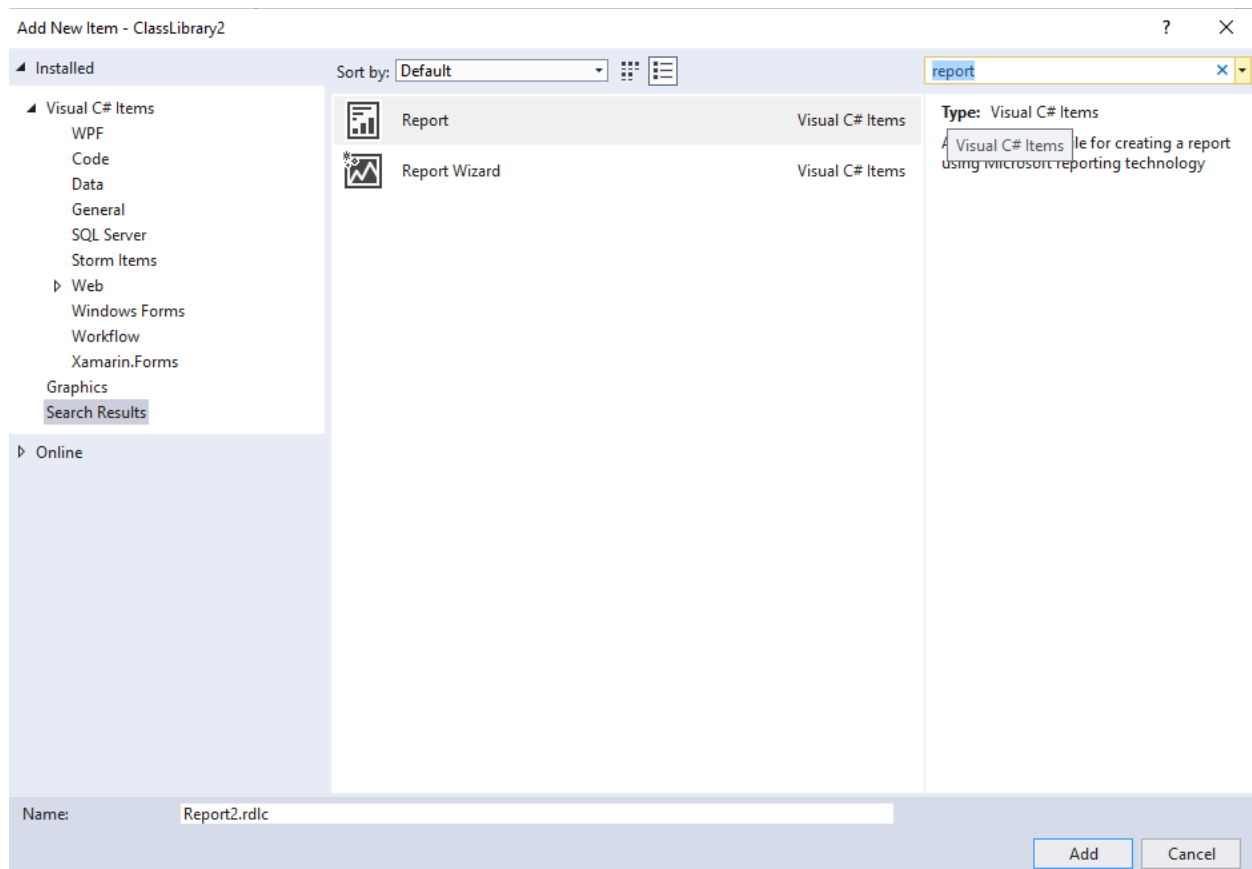
```
`csharp
public class ProductSales
{
    public string ProdCat { get; set; }
    public string SubCat { get; set; }
    public string OrderYear { get; set; }
    public string OrderQtr { get; set; }
    public double Sales { get; set; }
}
`
```

4. Clean and build the application.

Add an RDLC report

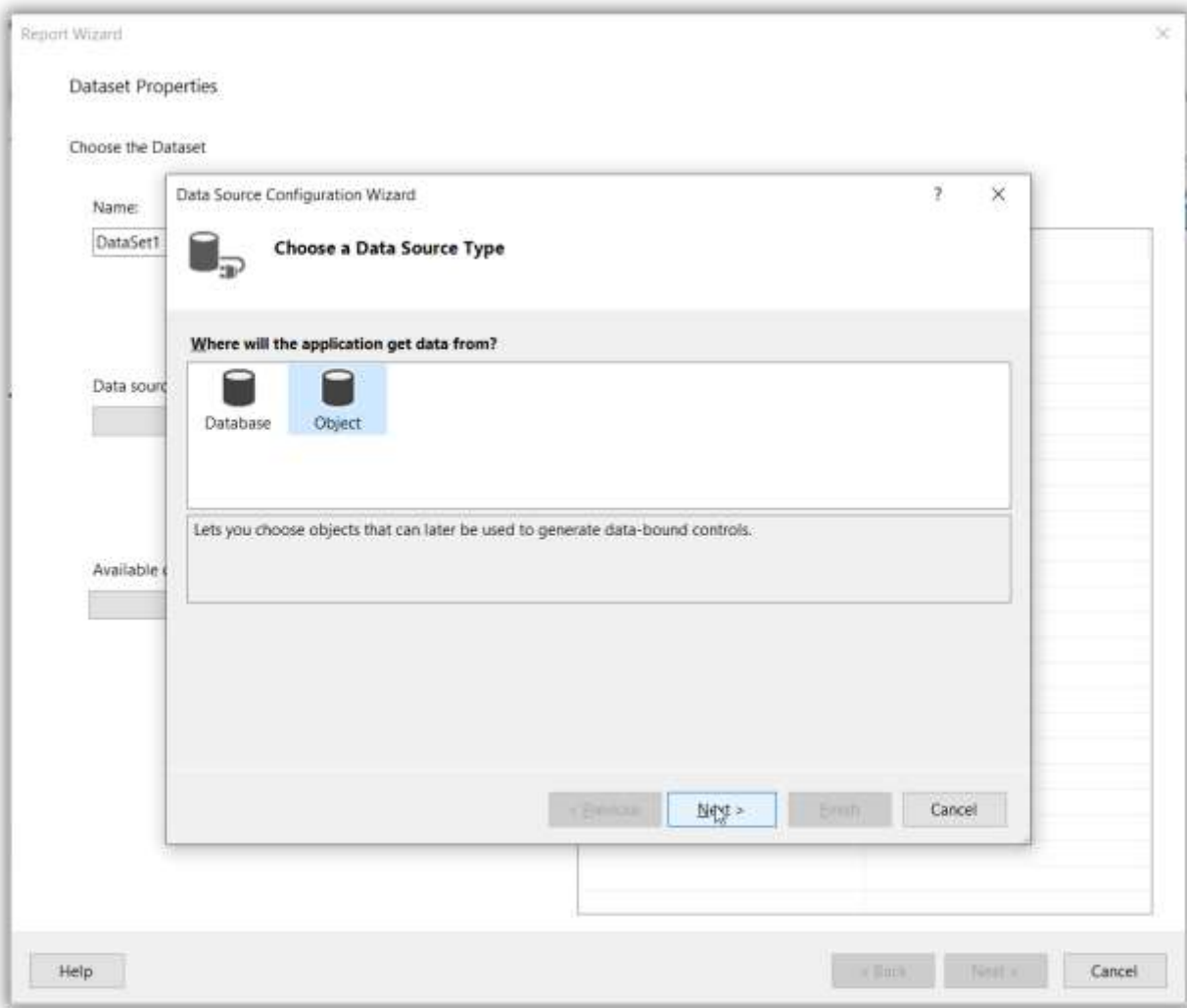
1. Right-click the project and click **Add > New Item**.

2. Search Report with new item and select **Report Wizard** to start the creation with dataset selection.
3. Click **Add**.

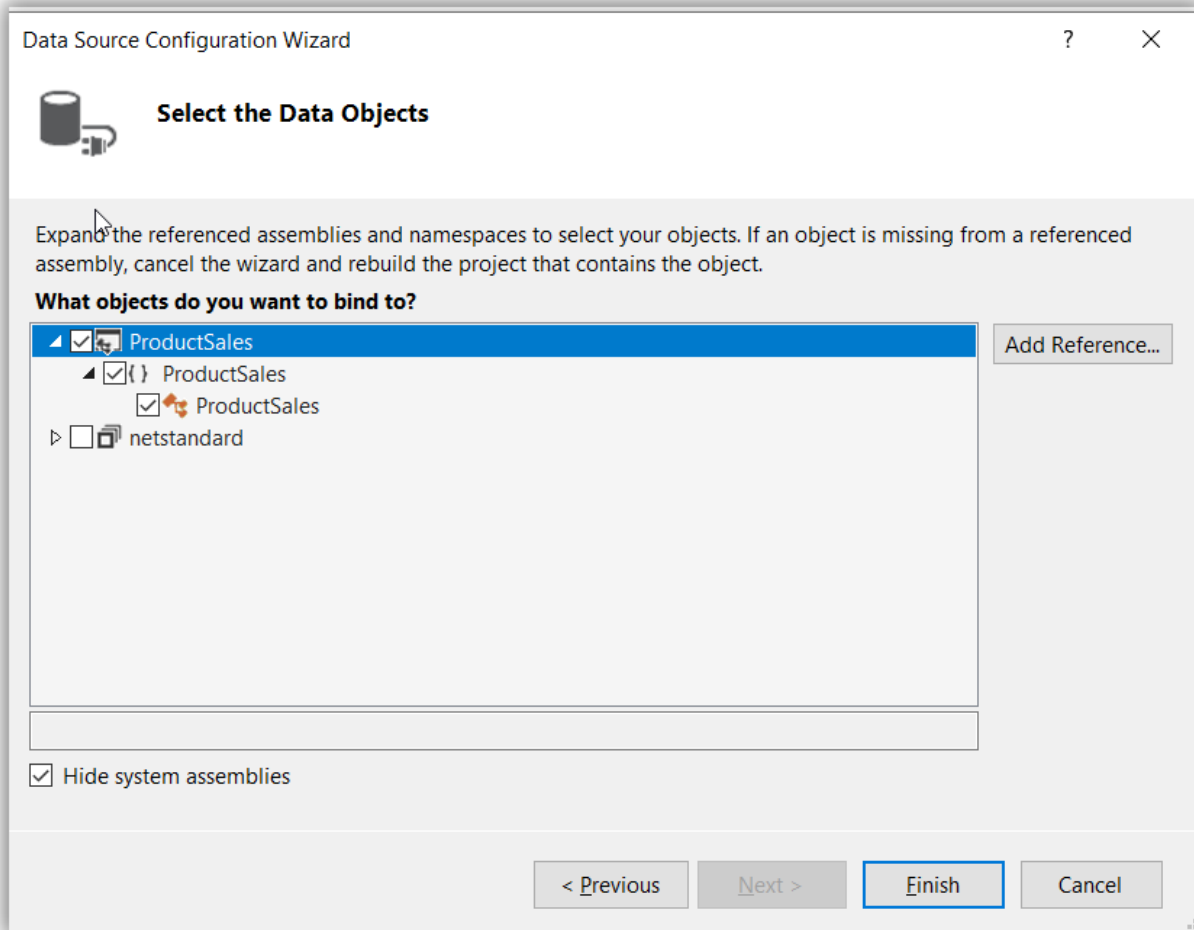


Data source and table configuration wizard

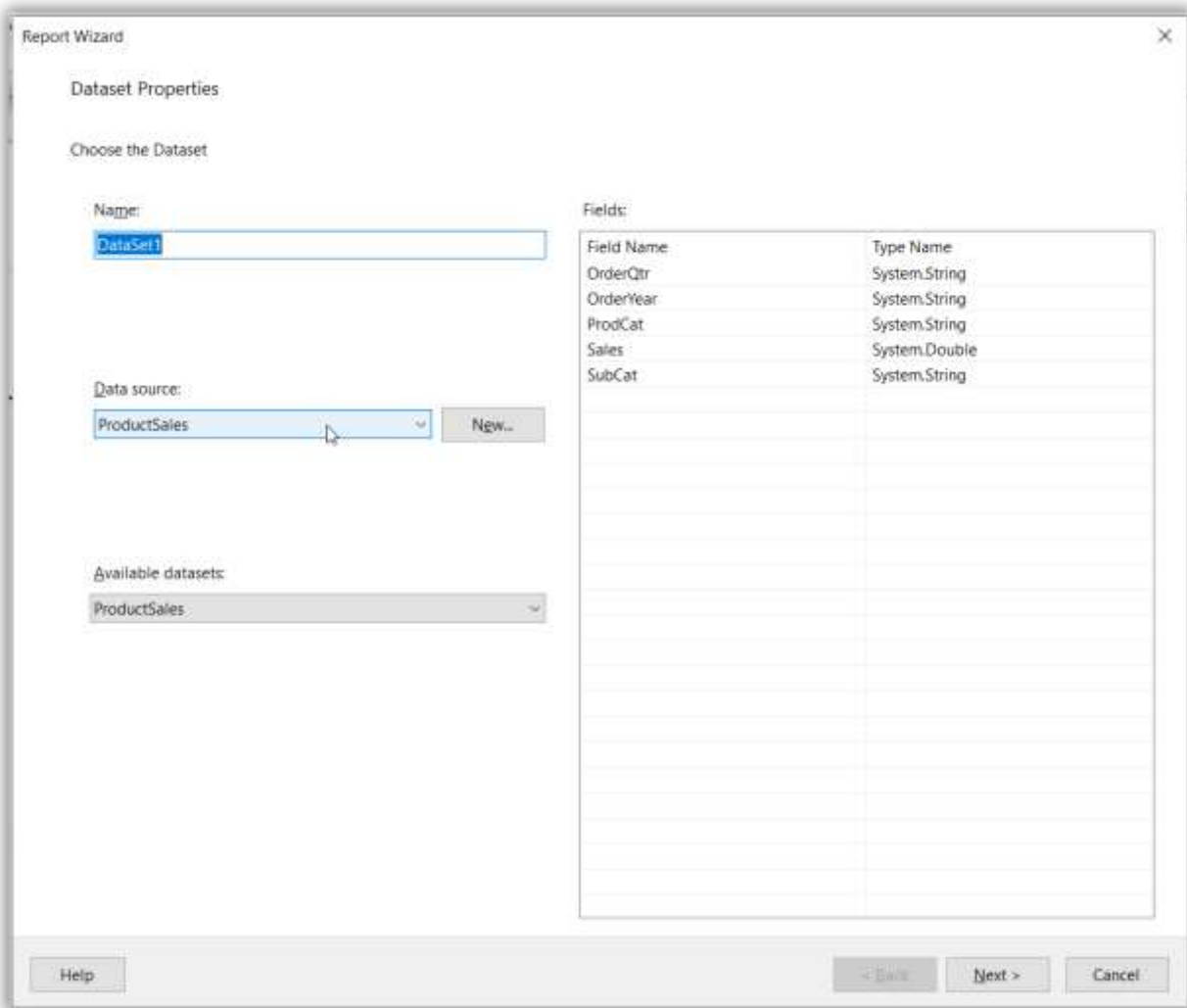
1. Choose object type from the Data Source Configuration wizard and click **Next**.



2. Expand the tree view and select **ProductSales**, and then click **Finish**.



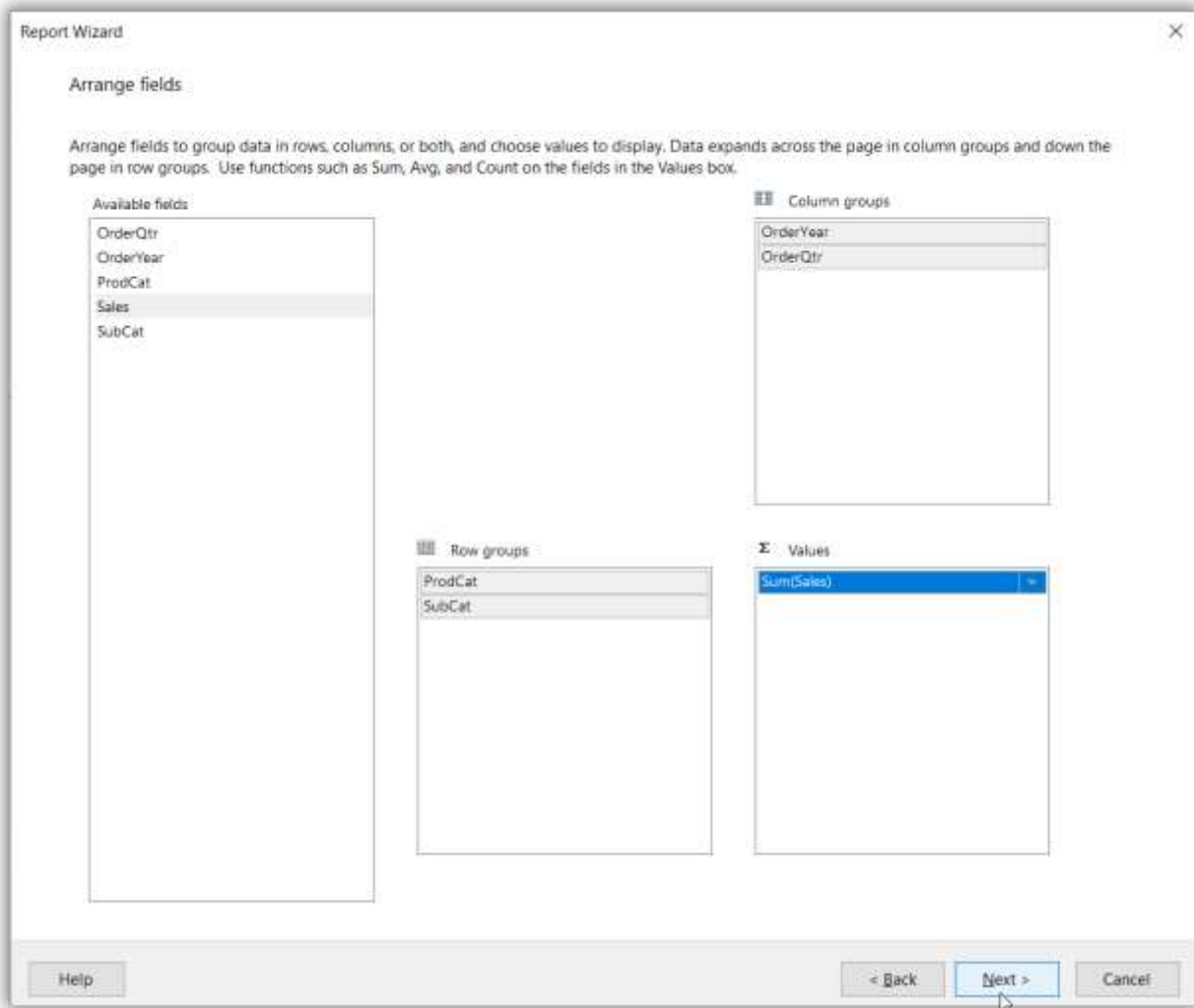
3. In the DataSet Properties wizard, specify the dataset name as `SalesData`.



The image shows the 'Report Wizard' dialog box, specifically the 'Dataset Properties' tab. The 'Choose the Dataset' section contains a 'Name:' label followed by a text box with 'DataSet1'. Below this is a 'Data source:' label followed by a dropdown menu showing 'ProductSales' and a 'New...' button. At the bottom left, there is an 'Available datasets:' label followed by a dropdown menu also showing 'ProductSales'. On the right side, there is a 'Fields:' label above a table. The table has two columns: 'Field Name' and 'Type Name'. It lists five fields: 'OrderQtr' (System.String), 'OrderYear' (System.String), 'ProdCat' (System.String), 'Sales' (System.Double), and 'SubCat' (System.String). At the bottom of the dialog, there are three buttons: 'Help', '< Back', and 'Next >', and a 'Cancel' button on the far right.

Field Name	Type Name
OrderQtr	System.String
OrderYear	System.String
ProdCat	System.String
Sales	System.Double
SubCat	System.String

4. Drag the fields into Values, Row, and Column groups, and then click **Next**.



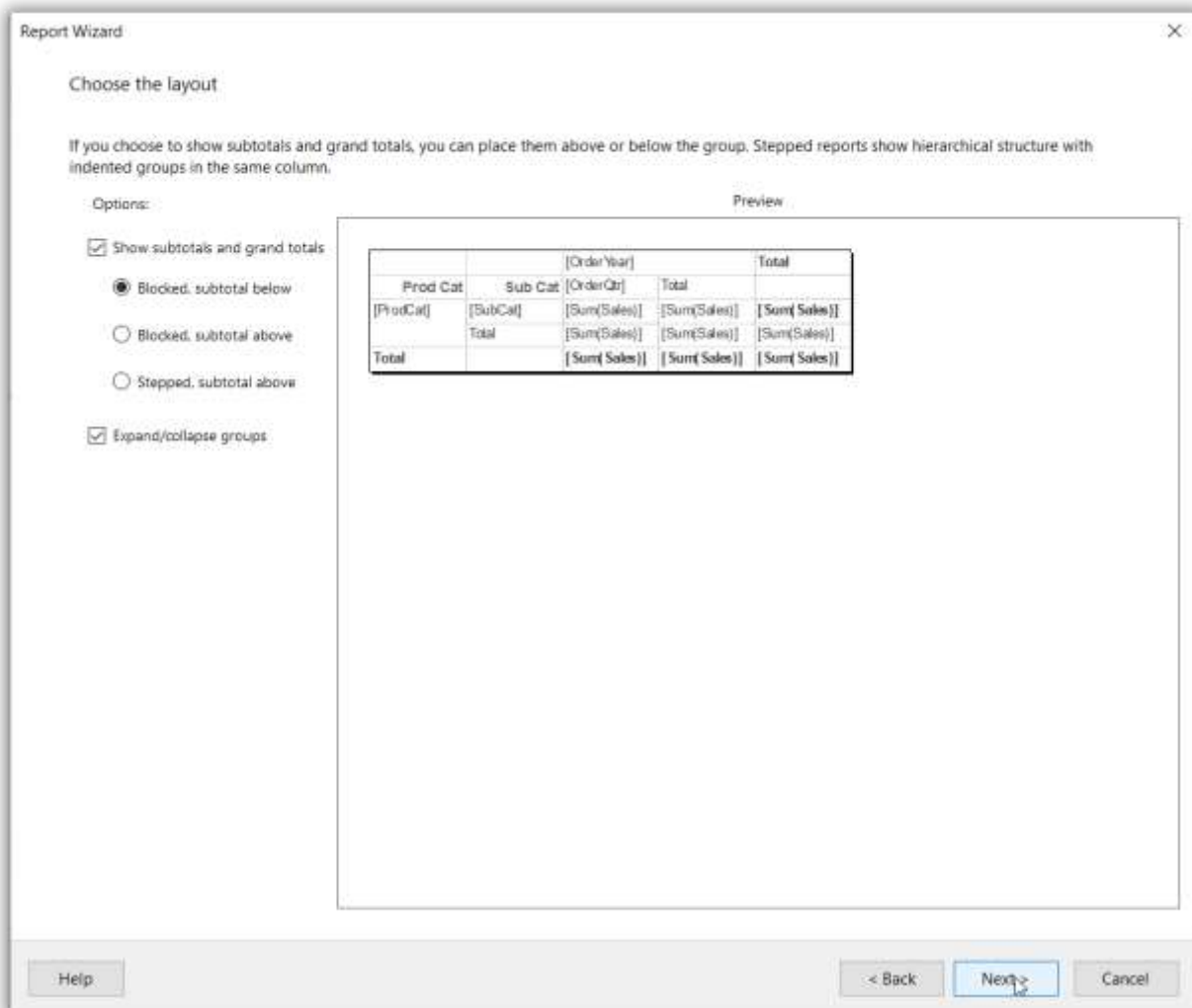
The image shows the 'Report Wizard' dialog box, specifically the 'Arrange fields' step. The dialog has a title bar with 'Report Wizard' and a close button. Below the title bar is the section 'Arrange fields' with a descriptive text: 'Arrange fields to group data in rows, columns, or both, and choose values to display. Data expands across the page in column groups and down the page in row groups. Use functions such as Sum, Avg, and Count on the fields in the Values box.'

The dialog is divided into four main areas:

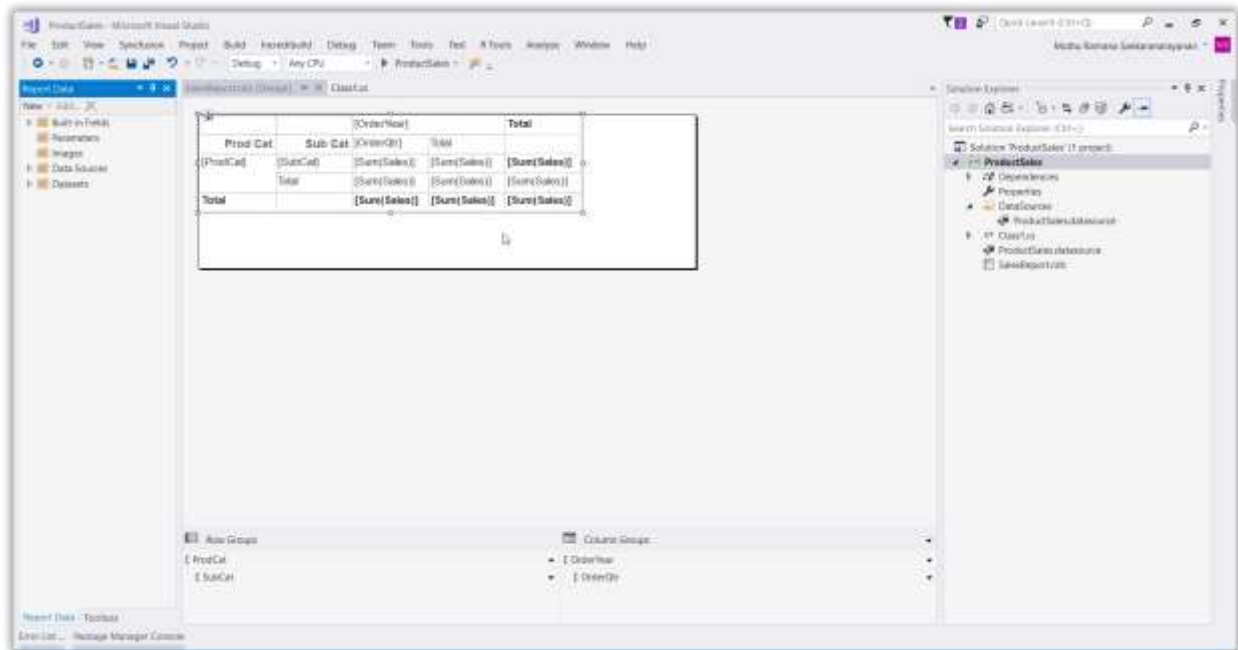
- Available fields:** A list box containing 'OrderQty', 'OrderYear', 'ProdCat', 'Sales', and 'SubCat'. 'Sales' is currently selected.
- Column groups:** A list box containing 'OrderYear' and 'OrderQty'.
- Row groups:** A list box containing 'ProdCat' and 'SubCat'.
- Σ Values:** A list box containing 'Sum(Sales)'.

At the bottom of the dialog, there are three buttons: 'Help', '< Back', and 'Next >', and a 'Cancel' button on the far right. A mouse cursor is pointing at the 'Next >' button.

5. Choose the table layout and click **Next**.
6. Select table style and click **Finish**.



Now, the RDLC report is displayed in the Visual Studio as follows.



Adding data visualization scripts

To render the report with data visualization components such as chart, gauge and map items, must add scripts of the visualization element. The following table shows the script reference that need to be added in Report Viewer page for data visualization elements.

Visualization Item | Script File

Gauge | ej2-base.min.js, ej2-data.min.js, ej2-pdf-export.min.js, ej2-svg-base.min.js, ej2-lineargauge.min.js and ej2-circulargauge.min.js

Map | ej2-maps.min.js

Chart | ej.chart.min.js

To render the chart report item, add chart control script **ej.chart.min.js** before the **bold.report-viewer.min.js** reference in **Default.aspx** page as in following code sample.

```
`html
<link href="Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<!--Used to render the chart item. Add this script, only if your report contains the chart report item.-->
<script src="Scripts/bold-reports/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="Scripts/bold-reports/bold.report-viewer.min.js"></script>
`
```

The following code can be used to render the chart, gauge and map report items in Report Viewer.

```
`html
<link href="Content/bold-reports/material/bold.reports.all.min.css" rel="stylesheet" />
<script src="https://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
<!--Used to render the gauge item. Add this script, only if your report contains the gauge report item. -->
<script src="Scripts/bold-reports/common/ej2-base.min.js"></script>
<script src="Scripts/bold-reports/common/ej2-data.min.js"></script>
<script src="Scripts/bold-reports/common/ej2-pdf-export.min.js"></script>
<script src="Scripts/bold-reports/common/ej2-svg-base.min.js"></script>
<script src="Scripts/bold-reports/data-visualization/ej2-circulargauge.min.js"></script>
<script src="Scripts/bold-reports/data-visualization/ej2-lineargauge.min.js"></script>
<!--Used to render the map item. Add this script, only if your report contains the map report item.-->
<script src="Scripts/bold-reports/data-visualization/ej2-maps.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.common.min.js"></script>
<script src="Scripts/bold-reports/common/bold.reports.widgets.min.js"></script>
<!--Used to render the chart item. Add this script, only if your report contains the chart report item.-->
<script src="Scripts/bold-reports/data-visualization/ej.chart.min.js"></script>
<!-- Report Viewer component script-->
<script src="Scripts/bold-reports/bold.report-viewer.min.js"></script>
`
```

How to change the exporting document file name based on parameter

Find the following steps to change the file based on parameter values in Report.

1. Create the file exporting file name using the parameters in **onRenderingComplete** event and store it in local variable.

```
`html
function onRenderingComplete(event) {
var parameters = event.reportParameters;
if(parameters){
for (var i = 0; i < parameters.length; i++) {
if(parameters[i].Name == "Department"){
this.exportFileName = "Sales for " + parameters[i].Value;
}
}
}
```

```
}
,
```

2. Use the file Name property with export, click event to change the file using the value stored in local variable used for having the file using the parameters.

```
`html
function onExportItemClick(event) {
event.fileName = this.exportFileName ;
}
,
```

How to change the data source dynamically

You have to use the `reportOption.ReportModel.DataSourceCredentials` available with the `OnInitReportOptions` method to dynamically change the data source in the web API controller. The following code sample shows how to change the connection string of the `<database>` data source in the report.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
DataSourceCredentials dataSourceCredentials = new DataSourceCredentials();
string connectionString = "Data Source = <instancename>; Initial Catalog = <database>; User ID =
'<username>'; Password = '<password>'";
//You have to provide the shared data source name used with the report or the data source name
available with the report.
dataSourceCredentials.Name = "<database>";
dataSourceCredentials.ConnectionString = connectionString;
reportOption.ReportModel.DataSourceCredentials = new List<DataSourceCredentials> {
dataSourceCredentials };
}
,
```

How to generate the RDL and RDLC reports programmatically in the report viewer

The Bold Reports reporting library allows you to generate and preview RDL and RDLC reports programmatically in the report viewer. The `ReportDefinition` class is defined as a report object model.

In the report definition instance, you can create, add, and modify the report properties, report sections such as header and footer, and report items.

The following steps illustrates how to create a basic report object model:

[Initialize a report definition](#)

The following code snippet guides you in initializing the report definition.

```
`csharp
ReportDefinition CreateReport()
{
    ReportDefinition report = new ReportDefinition();
    report.ReportSections = new ReportSections();
    var reportSection = new ReportSection();
    report.ReportSections.Add(reportSection);
    reportSection.Width = new BoldReports.RDL.DOM.Size("6in");
    report.ReportUnitType = "Inch";
    report.RDLType = RDLType.RDL2010;
    return report;
}

BoldReports.RDL.DOM.Style AddStyle()
{
    BoldReports.RDL.DOM.Style style = new BoldReports.RDL.DOM.Style();
    style.Border = new BoldReports.RDL.DOM.Border();
    style.Border.Width = new BoldReports.RDL.DOM.Size("1pt");
    style.Border.Style = "Solid";
    style.Border.Color = "Black";
    return style;
}
`
```

[Create a Data source for report](#)

The following code snippet guides you in creating a **Datasource** for the report and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
    ...
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a Dataset for the report

```
...
...
reportSection.Page = new BoldReports.RDL.DOM.Page();
reportSection.Page.Style = AddStyle();
reportSection.Page.PageHeight = "4in";
reportSection.Page.PageWidth = "6in";
var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
<database>");
report.DataSources = new DataSources();
report.DataSources.Add(newDataSource);
...
...
...
}
//If you are using RDLC report then you can pass any string value as connection string
BoldReports.RDL.DOM.DataSource CreateDataSource(string name, string dataProvider, string
connectionString)
{
var dataSource = new BoldReports.RDL.DOM.DataSource();
dataSource.Name = name;
dataSource.SecurityType = BoldReports.RDL.DOM.SecurityType.Integrated;
dataSource.ConnectionProperties = new BoldReports.RDL.DOM.ConnectionProperties();
dataSource.ConnectionProperties.DataProvider = dataProvider;
dataSource.ConnectionProperties.ConnectionString = connectionString;
dataSource.ConnectionProperties.IntegratedSecurity = true;
return dataSource;
}
`
```

Create a Dataset for the report

The following code snippet guides you in creating a **Dataset** for the report and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create** a Dataset for the report

```
...
...
...
reportSection.Page.PageWidth = "6in";
var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
<database>");
report.DataSources = new DataSources();
report.DataSources.Add(newDataSource);
var newDataSet = CreateDataSet("DataSource1", "DataSet1");
report.DataSets = new DataSets();
report.DataSets.Add(newDataSet);
...
...
...
}
BoldReports.RDL.DOM.DataSet CreateDataSet(string dataSourceName, string dataSetName)
{
var dataSet = new BoldReports.RDL.DOM.DataSet();
dataSet.Name = dataSetName;
dataSet.Query = new Query();
dataSet.Query.DataSourceName = dataSourceName;
dataSet.Query.CommandText = "SELECT
HumanResources.Department.DepartmentID,HumanResources.Department.Name,HumanResources.De
partment.GroupName,HumanResources.Department.ModifiedDate FROM
HumanResources.Department";
dataSet.Query.QueryDesignerState = new QueryDesignerState();
var table = CreateTable();
dataSet.Query.QueryDesignerState.Tables = new List<Table>();
dataSet.Query.QueryDesignerState.Tables.Add(table);
dataSet.Fields = new Fields();
dataSet.Fields.Add(CreateField("DepartmentID", "System.Int16"));
dataSet.Fields.Add(CreateField("Name", "System.String"));
dataSet.Fields.Add(CreateField("GroupName", "System.String"));
dataSet.Fields.Add(CreateField("ModifiedDate", "System.DateTime"));
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Create a report page header**

```
return dataSet;
}

BoldReports.RDL.DOM.Table CreateTable()
{
    var table = new Table();
    table.Schema = "HumanResources";
    table.Name = "Department";
    table.Columns = new List<Column>();
    table.Columns.Add(CreateColumn("DepartmentID"));
    table.Columns.Add(CreateColumn("Name"));
    table.Columns.Add(CreateColumn("GroupName"));
    table.Columns.Add(CreateColumn("ModifiedDate"));
    return table;
}

BoldReports.RDL.DOM.Column CreateColumn(string columnName)
{
    var column = new Column();
    column.Name = columnName;
    return column;
}

BoldReports.RDL.DOM.Field CreateField(string fieldName, string type)
{
    var field = new Field();
    field.Name = fieldName;
    field.TypeName = type;
    field.DataField = fieldName;
    return field;
}
`
```

Create a report page header

The following code snippet guides you in creating a **PageHeader** report section and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
```


How to generate the RDL and RDLC reports programmatically in the report viewer **Create a report page footer**

```
{
...
...
...
reportSection.Page = new BoldReports.RDL.DOM.Page();
reportSection.Page.Style = AddStyle();
reportSection.Page.PageHeight = "4in";
reportSection.Page.PageWidth = "6in";
reportSection.Page.PageHeader = CreateHeader();
...
...
...
}
PageHeader CreateHeader()
{
PageHeader pageHeader = new PageHeader();
pageHeader.Height = new BoldReports.RDL.DOM.Size("0.59167in");
pageHeader.Style = AddStyle();
pageHeader.PrintOnFirstPage = true;
pageHeader.PrintOnLastPage = true;
pageHeader.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
return pageHeader;
}
\
```

Create a report page footer

The following code snippet guides you in creating a **PageFooter** report section and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
...
...
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
...
reportSection.Page.PageHeader = CreateFooter();
...
...
}
PageFooter CreateFooter()
{
PageFooter pageFooter = new PageFooter();
pageFooter.Style = AddStyle();
pageFooter.Height = new BoldReports.RDL.DOM.Size("0.59167in");
pageFooter.ReportItems = new ReportItems();
pageFooter.PrintOnFirstPage = true;
pageFooter.PrintOnLastPage = true;
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
return pageFooter;
}
`
```

Initialize a report body section

- Initialize a report body object and set the values to their properties like height, styles, and report items as shown in the following code snippet.

```
`csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Body = CreateBody();
...
...
...
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

Body CreateBody()

```
{
var body = new Body();
body.Height = new BoldReports.RDL.DOM.Size("2.03333in");
body.Style = AddStyle();
body.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
body.ReportItems.Add(textBox);
return body;
}
`
```

- Using the following code snippets, you can create a **Textbox** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

`csharp

PageHeader CreateHeader()

```
{
...
...
...
pageHeader.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
...
...
...
}
```

PageFooter CreateFooter()

```
{
...
...
...
pageFooter.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
pageFooter.ReportItems.Add(textBox);
```

```
...
```

```
...
```

```
...
```

```
}
```

```
Body CreateBody()
```

```
{
```

```
...
```

```
...
```

```
...
```

```
body.ReportItems = new ReportItems();
```

```
var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
```

```
body.ReportItems.Add(textBox);
```

```
...
```

```
...
```

```
...
```

```
}
```

BoldReports.RDL.DOM.TextBox CreateTextBox(string name, string text, string left, string top, string width, string height)

```
{
```

```
var textBox = new BoldReports.RDL.DOM.TextBox();
```

```
textBox.Name = name;
```

```
textBox.Height = new BoldReports.RDL.DOM.Size(height);
```

```
textBox.Width = new BoldReports.RDL.DOM.Size(width);
```

```
textBox.Left = new BoldReports.RDL.DOM.Size(left);
```

```
textBox.Top = new BoldReports.RDL.DOM.Size(top);
```

```
textBox.Style = new BoldReports.RDL.DOM.Style();
```

```
textBox.Style.TextAlign = "Center";
```

```
textBox.Style.VerticalAlign = "Top";
```

```
textBox.Paragraphs = new Paragraphs();
```

```
BoldReports.RDL.DOM.Paragraph paragraph = new BoldReports.RDL.DOM.Paragraph();
```

```
TextRuns runs = new TextRuns();
```

```
TextRun run = new TextRun();
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
run.Style = new BoldReports.RDL.DOM.Style();
run.Style.FontStyle = "Default";
run.Style.TextAlign = "Center";
run.Style.FontFamily = "Arial";
run.Style.FontSize = new BoldReports.RDL.DOM.Size("10pt");
run.Value = text;
runs.Add(run);
paragraph.Style = new BoldReports.RDL.DOM.Style();
paragraph.Style.VerticalAlign = "Top";
paragraph.Style.TextAlign = "Center";
paragraph.TextRuns = runs;
textBox.Paragraphs.Add(paragraph);
return textBox;
}
```

Create a Table for the report

- Using the following code snippets, you can create a **Table** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

`csharp

BoldReports.RDL.DOM.Tablix CreateTablix(string name, string text, string left, string top, string width, string height)

```
{
var tableItem = new BoldReports.RDL.DOM.Tablix();
tableItem.Name = name;
tableItem.Height = new BoldReports.RDL.DOM.Size(height);
tableItem.Width = new BoldReports.RDL.DOM.Size(width);
tableItem.Left = new BoldReports.RDL.DOM.Size(left);
tableItem.Top = new BoldReports.RDL.DOM.Size(top);
tableItem.Style = new BoldReports.RDL.DOM.Style();
tableItem.Style.Border = new BoldReports.RDL.DOM.Border();
tableItem.Style.Border.Style = "Solid";
tableItem.DataSetName = "DataSet1";
tableItem.TablixBody = new TablixBody();
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
tableItem.TablixBody.TablixColumns = new TablixColumns();
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
tableItem.TablixBody.TablixRows = new TablixRows();
var rowOne = new List<TablixRowValues>();
rowOne.Add(new TablixRowValues { TextBoxName = "TextBox1", TextBoxValue = "Department ID" });
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox3", TextBoxValue = "Name" });
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox5", TextBoxValue = "Group Name" });
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowOne));
var rowTwo = new List<TablixRowValues>();
rowTwo.Add(new TablixRowValues { TextBoxName = "DepartmentID", TextBoxValue =
"=Fields!DepartmentID.Value" });
rowTwo.Add(new TablixRowValues { TextBoxName = "Name", TextBoxValue = "=Fields!Name.Value" });
rowTwo.Add(new TablixRowValues { TextBoxName = "GroupName", TextBoxValue =
"=Fields!GroupName.Value" });
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowTwo));
tableItem.TablixColumnHierarchy = new TablixColumnHierarchy();
tableItem.TablixColumnHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixRowHierarchy = new TablixRowHierarchy();
tableItem.TablixRowHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixRowHierarchy.TablixMembers.Add(new TablixMember() { KeepWithGroup =
KeepWithGroup.After });
tableItem.TablixRowHierarchy.TablixMembers.Add(CreateTablixMember("Details"));
return tableItem;
}

BoldReports.RDL.DOM.TablixColumn CreateTablixColumn()
{
var tablixColumn = new TablixColumn();
tablixColumn.Width = "1in";
return tablixColumn;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
}  
  
BoldReports.RDL.DOM.TablixRow CreateTablixRow(List<TablixRowValues> values)  
{  
    var tablixRow = new TablixRow();  
    tablixRow.Height = "0.25in";  
    tablixRow.TablixCells = new TablixCells();  
    for(int i = 0; i < values.Count; i++)  
    {  
        tablixRow.TablixCells.Add(CreateTablixCell(values[i].TextBoxName, values[i].TextBoxValue));  
    }  
    return tablixRow;  
}  
  
BoldReports.RDL.DOM.TablixCell CreateTablixCell(string textBoxName, string textBoxValue)  
{  
    var tablixCell = new TablixCell();  
    tablixCell.CellContents = new CellContents();  
    tablixCell.CellContents.ReportItem = CreateTextBox(textBoxName, textBoxValue);  
    return tablixCell;  
}  
  
BoldReports.RDL.DOM.TablixMember CreateTablixMember(string groupName)  
{  
    var tablixMember = new TablixMember();  
    tablixMember.Group = new Group();  
    tablixMember.Group.Name = groupName;  
    return tablixMember;  
}  
  
internal class TablixRowValues  
{  
    public string TextBoxName { get; set; }  
    public string TextBoxValue { get; set; }  
}  
,
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

Create a Chart for the report

- Using the following code snippets, you can create a **Chart** report item and assign the values for their properties like name, styles, and dimension values.

```
`csharp
BoldReports.RDL.DOM.Chart CreateChart(string name, string left, string top, string width, string height)
{
    var chartItem = new Chart();
    chartItem.Name = name;
    chartItem.Height = new BoldReports.RDL.DOM.Size("10in");
    chartItem.Width = new BoldReports.RDL.DOM.Size("10in");
    chartItem.Left = new BoldReports.RDL.DOM.Size("5in");
    chartItem.Top = new BoldReports.RDL.DOM.Size("5in");
    chartItem.DataSetName = "DataSet1";
    chartItem.Style = new BoldReports.RDL.DOM.Style();
    chartItem.Bookmark= "=First(Fields!Name.Value, \"DataSet1\")";
    chartItem.ChartCategoryHierarchy = new ChartCategoryHierarchy();
    chartItem.ChartCategoryHierarchy.ChartMembers = new ChartMembers();
    chartItem.ChartCategoryHierarchy.ChartMembers.Add(ChartCategoryMember());
    chartItem.ChartSeriesHierarchy = new ChartSeriesHierarchy();
    chartItem.ChartSeriesHierarchy.ChartMembers = new ChartMembers();
    chartItem.ChartSeriesHierarchy.ChartMembers.Add(ChartSeriesMember());
    chartItem.ChartData = new ChartData();
    chartItem.ChartData.ChartDerivedSeriesCollection = new ChartDerivedSeriesCollection();
    chartItem.ChartData.ChartSeriesCollection = CreateChartSeriesCollection();
    chartItem.ChartLegends = new ChartLegends();
    chartItem.ChartAreas = new ChartAreas();
    chartItem.ChartAreas.Add(CreateChartArea());
    chartItem.ChartTitles = new ChartTitles();
    chartItem.Palette = "Pacific";
    chartItem.ChartBorderSkin = new ChartBorderSkin();
    chartItem.ChartNoDataMessage = new ChartNoDataMessage();
    chartItem.ChartNoDataMessage.Caption = "No Data Available";
    chartItem.ChartNoDataMessage.Name = "NoDataMessage";
```


How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
return chartItem;
}

BoldReports.RDL.DOM.ChartMember ChartCategoryMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = new Group();
    ChartMember.Group = CreateChartGroup();
    ChartMember.Label="=Fields!DepartmentID.Value";
    ChartMember.SortExpressions = SortExpressions();
    return ChartMember;
}

BoldReports.RDL.DOM.ChartMember ChartSeriesMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = null;
    ChartMember.Label = "Department ID";
    ChartMember.SortExpressions = new SortExpressions();
    return ChartMember;
}

BoldReports.RDL.DOM.SortExpressions SortExpressions()
{
    var SortExpressions = new SortExpressions();
    var SortExpression = new SortExpression();
    SortExpression.Direction = SortDirection.Ascending;
    SortExpression.Value = "=Fields!DepartmentID.Value";
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
SortExpressions.Add(SortExpression);
return SortExpressions;
}

BoldReports.RDL.DOM.Group CreateChartGroup()
{
    var ChartGroup = new Group();
    ChartGroup.DataElementName = null;
    ChartGroup.DataElementOutput = DataElementOutputs.Auto;
    ChartGroup.DocumentMapLabel = null;
    ChartGroup.DocumentMapLabelLocID = null;
    ChartGroup.DomainScope = null;
    ChartGroup.Filters = new Filters();
    ChartGroup.GroupExpressions = CreateGroupExpressions();
    ChartGroup.Name = "Chart2_CategoryGroup";
    ChartGroup.PageBreak = null;
    ChartGroup.PageName = null;
    ChartGroup.Parent = null;
    ChartGroup.Variables = new Variables();
    return ChartGroup;
}

BoldReports.RDL.DOM.GroupExpressions CreateGroupExpressions()
{
    var GroupExpressions = new GroupExpressions();
    var GroupExpression = new GroupExpression();
    GroupExpression.Value = "=Fields!DepartmentID.Value";
    GroupExpressions.Add(GroupExpression);
    return GroupExpressions;
}

BoldReports.RDL.DOM.ChartSeriesCollection CreateChartSeriesCollection()
{
    var ChartSeriesCollection = new ChartSeriesCollection();
    ChartSeriesCollection.Add(CreateChartSeries());
    return ChartSeriesCollection;
}
```

How to generate the RDL and RDLC reports programmatically in the report viewer **Initialize** a report body section

```
}  
  
BoldReports.RDL.DOM.ChartSeries CreateChartSeries()  
{  
    var ChartSeries = new ChartSeries();  
    ChartSeries.CategoryAxisName = "Primary";  
    ChartSeries.ChartAreaName = null;  
    ChartSeries.ChartDataLabel = null;  
    ChartSeries.ChartDataPoints = new ChartDataPoints();  
    ChartSeries.ChartDataPoints.Add(ChartDataPoint());  
    ChartSeries.ChartEmptyPoints = new ChartEmptyPoints();  
    ChartSeries.ChartEmptyPoints.ChartDataLabel = new ChartDataLabel();  
    ChartSeries.ChartEmptyPoints.ChartDataLabel.Style = new Style();  
    ChartSeries.ChartEmptyPoints.ChartMarker = new ChartMarker();  
    ChartSeries.ChartEmptyPoints.ChartMarker.Style = new Style();  
    ChartSeries.ChartSmartLabel = new ChartSmartLabel();  
    ChartSeries.Name = "DepartmentID";  
    ChartSeries.Type = VisualizationType.Column;  
    ChartSeries.Subtype = VisualizationSubType.Plain;  
    ChartSeries.Style = new Style();  
    return ChartSeries;  
}  
  
BoldReports.RDL.DOM.ChartDataPoint ChartDataPoint()  
{  
    var ChartDataPoint = new ChartDataPoint();  
    ChartDataPoint.ActionInfo = null;  
    ChartDataPoint.ChartDataLabel = null;  
    ChartDataPoint.ChartDataLabel = ChartDataLabel();  
    ChartDataPoint.ChartDataPointValues = new ChartDataPointValues();  
    ChartDataPoint.ChartDataPointValues.Y = "=Sum(Fields!DepartmentID.Value)";  
    ChartDataPoint.ChartItemInLegend = null;  
    ChartDataPoint.ChartMarker = new ChartMarker();  
    ChartDataPoint.ChartMarker.Size = null;  
    ChartDataPoint.ChartMarker.Type = null;
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
ChartDataPoint.ChartMarker.Style= chartStyle("Transparent", "Arial");
ChartDataPoint.CustomProperties = new CustomProperties();
ChartDataPoint.DataElementName = null;
ChartDataPoint.DataElementOutput = DataElementOutputs.Output;
ChartDataPoint.ToolTip = null;
ChartDataPoint.Style= chartStyle("Transparent", "Arial");
return ChartDataPoint;
}
```

BoldReports.RDL.DOM.ChartDataLabel ChartDataLabel()

```
{
var ChartDataLabel = new ChartDataLabel();
ChartDataLabel.ActionInfo = null;
ChartDataLabel.Label = null;
ChartDataLabel.Position = null;
ChartDataLabel.Rotation = "0";
ChartDataLabel.Style = new Style();
ChartDataLabel.Style = chartStyle("Transparent", "Arial");
ChartDataLabel.ToolTip = null;
return ChartDataLabel;
}
```

BoldReports.RDL.DOM.Style chartStyle(string BackgroundColor, string FontFamily)

```
{
var style = new Style();
style.BackgroundColor = BackgroundColor;
style.FontFamily = FontFamily;
return style;
}
```

BoldReports.RDL.DOM.ChartArea CreateChartArea()

```
{
var chartArea = new ChartArea();
chartArea.Style = new Style();
chartArea.Style = chartStyle("Transparent", "Arial");
chartArea.AlignOrientation = AlignOrientation.None;
```

How to generate the RDL and RDLC reports programmatically in the report viewer**Initialize** a report body section

```
chartArea.AlignWithChartArea = null;
chartArea.ChartAlignType = null;
chartArea.ChartElementPosition = null;
chartArea.ChartInnerPlotPosition = null;
chartArea.ChartThreeDProperties = null;
chartArea.Hidden = false;
chartArea.EquallySizedAxesFont = false;
chartArea.Name = "Default";
chartArea.ChartCategoryAxes = new ChartCategoryAxes();
chartArea.ChartCategoryAxes.Add(createChartAxis("Primary"));
chartArea.ChartCategoryAxes.Add(createChartAxis("Secondary"));
chartArea.ChartValueAxes = new ChartValueAxes();
chartArea.ChartValueAxes.Add(createChartAxis("Primary"));
chartArea.ChartValueAxes.Add(createChartAxis("Secondary"));
return chartArea;
}
```

BoldReports.RDL.DOM.ChartAxis createChartAxis(string Name)

```
{
var ChartAxis = new ChartAxis();
ChartAxis.Style = new Style();
ChartAxis.AllowLabelRotation = AllowLabelRotation.None;
ChartAxis.Angle = "0";
ChartAxis.Arrows = Arrows.None;
ChartAxis.ChartAxisScaleBreak = new ChartAxisScaleBreak();
ChartAxis.ChartAxisTitle = new ChartAxisTitle();
ChartAxis.ChartMajorGridLines = new ChartMajorGridLines();
ChartAxis.ChartMajorTickMarks = new ChartMajorTickMarks();
ChartAxis.ChartMinorGridLines = new ChartMinorGridLines();
ChartAxis.ChartMinorTickMarks = new ChartMinorTickMarks();
ChartAxis.ChartStripLines = new ChartStripLines();
ChartAxis.CrossAt = "NaN";
ChartAxis.CustomProperties = new CustomProperties();
ChartAxis.LineStyle = LineStyle.Solid;
```

```

ChartAxis.Name = Name;
return ChartAxis;
}
`

```

Render the generated report in ReportViewer

You can render the report using the ReportViewer, after the report definition is created. The following code snippet illustrates how to render the report using the ReportViewer by report definition.

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    var report = CreateReport();
    reportOption.ReportModel.ReportDefinition = reportDefinition;
}
`

```

How to disable the vertical scrollbar in parameter panel

To disable the vertical scrollbar in parameter panel, set the `enableparameterblockscroller` property to false.

Example

```

`js
<div id="report viewer"></div>
<script>
$("#report viewer").boldReportViewer(
{
    enableParameterBlockScroller: false
});
</script>
`

```

How to pass multiple values using custom data

Pass the multiple data values as JSON in `ajaxBeforeLoad` event using the 'data' property, which needs to be serialized in the server side API using known class type and `JsonConvert.DeserializeObject`.

1. Map the `ajaxBeforeLoad` event with `onAjaxRequest` function in the script to pass custom data to the server.

```
`html
<div style="height: 650px;width: 950px;min-height:404px;">
<Bold:ReportViewer runat="server" ID="viewer"
ReportServiceUrl="/api/ReportViewer"
ReportPath="~/Resources/sales-order-detail.rdl"
OnClientAjaxBeforeLoad="onAjaxRequest">
</Bold:ReportViewer>
</div>
`,`js
<script>
function onAjaxRequest(args) {
//Passing custom data to server
}
</script>
`,`
```

2. You need to pass the multiple custom data values as JSON and use the `data` property to send custom data to the server in the Ajax request.

```
`js
<script>
function onAjaxRequest(args){
//Passing custom data to server
var jsonData = {
customerID: "CI0021",
productID: "PO0022"
}
args.data = jsonData;
}
</script>
`,`
```

3. The custom data values are stored in the `customData` header key, and you can store them in the local property. The following code sample demonstrates deserialization of the JSON string

and change the data source connection strings based on Customer ID and Product ID in the `OnInitReportOptions` method.

```
`csharp
public SampleData customDatas = new SampleData();
public class SampleData
{
    public string customerID { get; set; }
    public string productID { get; set; }
}
[HttpPost]
public object PostReportAction([FromBody]Dictionary<string, object> jsonResult)
{
    if (jsonResult != null)
    {
        if (jsonResult.ContainsKey("customData"))
        {
            //Get client side custom data and store in local variable.
            customDatas = JsonConvert.DeserializeObject<SampleData>(jsonResult["customData"].ToString());
        }
    }
    return ReportHelper.ProcessReport(jsonResult, this, this._cache);
}
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    if (customDatas != null)
    {
        if (customDatas.customerID == "CI0021" && customDatas.productID == "PO0022") {
            //If you are changing the connection string based on customer id then could you please change the
            connection string as below.

            //reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<Datasource name>", "<connection string>"));

            reportOption.ReportModel.DataSourceCredentials.Add(new
            BoldReports.Web.DataSourceCredentials("<database>", "Data Source=<instancename>;Initial
            Catalog=<database>;"));
        }
    }
}
```



```

}
}
}
`

```

How to load the report from the database

You must load the report using the **Stream** option available with **reportOption.ReportModel**. To load the report from the database, please follow these steps:

From byte array

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    byte[] bytes = ""; // provide your byte array report data
    MemoryStream reportStream = new MemoryStream(bytes);
    reportOption.ReportModel.Stream = reportStream;
}
`

```

From base64String

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string base64String = ""; // provide your base64 report data
    byte[] bytes = System.Convert.FromBase64String(base64String);
    MemoryStream reportStream = new MemoryStream(bytes);
    reportOption.ReportModel.Stream = reportStream;
}
`

```

From string

```

`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string reportData = ""; // provide your database report data

```

```
byte[] bytes = System.Text.Encoding.ASCII.GetBytes(reportFile);
MemoryStream reportStream = new MemoryStream(bytes);
reportOption.ReportModel.Stream = reportStream;
}
`
```

How to clear cache when closing the Report Viewer

By using the `clearReportCache` and `destroy` method, you can clear the server side cache. You have to use this method when closing report viewer and switching to another report within your application.

CSHTML

```
`js
<div id="reportviewer"></div>

<script>
var isSubmit = true;
$(document.body).bind('submit', $.proxy(this.formSubmit, this));
function formSubmit(args)
{
isSubmit = false;
}
window.onbeforeunload = function () {
if (isSubmit) {
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.clearReportCache();
reportviewerObj.destroy();
}
isSubmit = true;
};
</script>
`
```

Web API

In the `PostReportAction` method, you have to collect the GC with `ClearCache` as shown in the following code sample.

```
`csharp
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
{
`
```

```

bool isclearcache = false;
if (jsonArray != null && jsonArray.ContainsKey("reportAction") && jsonArray["reportAction"].ToString()
== "ClearCache")
{
    isclearcache = true;
}
var reportresult = ReportHelper.ProcessReport(jsonArray, this, this._cache);
if (isclearcache)
{
    GC.Collect(); isclearcache = false;
}
return reportresult;
}
`

```

Bold Report Writer

Report Writer is a class library that enables the user to render reports defined in Microsoft's RDL format (2008 or 2008 R2) as PDF, Word, Excel, HTML and CSV documents.

The important features of Report Writer are listed as follows:

- RDL Specification - Supports RDL specification for the SQL Server 2008 and RDL specification for the SQL Server 2008 R2 only. List of available report definition formats: [msdn.microsoft.com/library/dd297486\(SQL.100\)](https://msdn.microsoft.com/library/dd297486(SQL.100)).
- Data sources - You can use advanced database servers Data Sources in Report Writer (SQL and Oracle).
- Charts - Show all basic types of charts that are available in Microsoft RDL reports.
- Tablix - Shows the summaries and simple tables.
- Gauge - Shows measurement values by using expression values.
- Textbox - Shows textbox data with expression support.
- Export - Export report as PDF, Word, Excel, HTML and CSV.
- Report Parameter - Views the report based on the report parameter value.

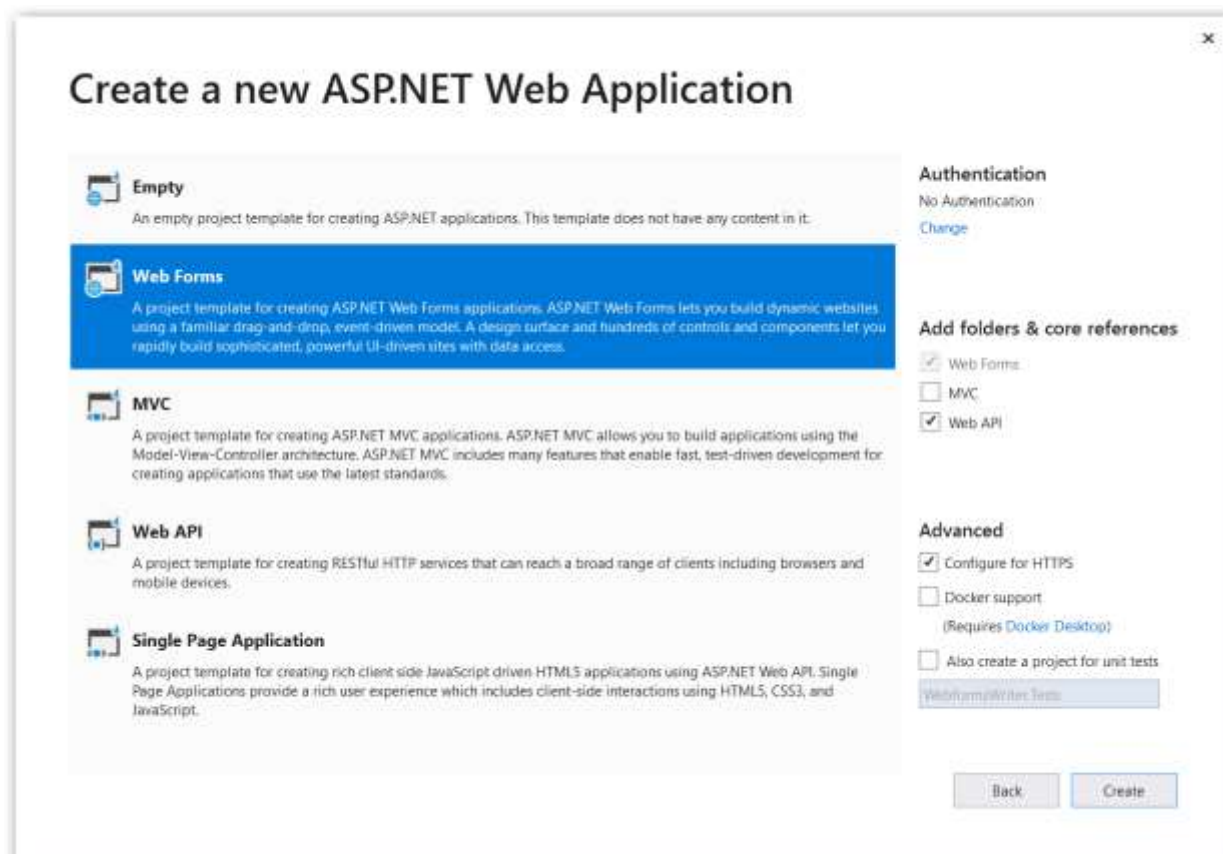
Export SSRS RDL Report in Bold Reports ASP.NET Report Writer

The Report Writer is a class library that is used to export the RDL report with popular file formats like PDF, Microsoft Word, Microsoft CSV, and Microsoft Excel without previewing the report on a webpage. This section describes how to export the RDL report to an ASP.NET application using the **Report Writer**.

Create an ASP.NET application

1. Start Visual Studio 2019 and click **Create new project**.
2. Choose **ASP.NET Web Application (.NET Framework)**, and then click **Next**.

3. Change the project name, and then click **Create**.
4. Choose **Web Forms** and **Web API**, then click **OK**.



List of dependency libraries

1. In the Solution Explorer tab, right-click the project or solution, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.Webpackage**, and install this in your application.

The following table provides details about the dependency packages and their usage.

Package | Purpose

Syncfusion.Pdf.AspNet | Exports the report to a PDF.

Syncfusion.DocIO.AspNet | Exports the report to a Word.

Syncfusion.XlsIO.AspNet | Exports the report to an Excel.

Syncfusion.Presentation.AspNet | Exports the report to a PowerPoint.

Newtonsoft.Json | Serializes and deserializes data for the Report Writer. It is a mandatory package for Report Writer, and the package version should be 10.0.1 or higher.

In this tutorial, the `sales-order-detail.rdl` report is used and it can be downloaded [here](#). You can get the reports from the Bold Reports installation location. For more information, refer to [samples and demos](#) section.

Server side Report Writer changes

1. Create a folder `Resources` in your application. Copy and paste the sample RDL reports into the `Resources` folder.
2. Open `Default.aspx.cs` and add the following using namespace.

```
`csharp
using BoldReports.Writer;
`
```

3. Initialize the Report Writer using the following code example.

```
`csharp
protected void ExportButton_Click(object sender, EventArgs e)
{
    string fileName = null;
    WriterFormat format;
    HttpContext httpContext = System.Web.HttpContext.Current;
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportPath= Server.MapPath("~/Resources/sales-order-detail.rdl");
    if (this.ExportFormat.SelectedValue == "PDF")
    {
        fileName = "sales-order-detail.pdf";
        format = WriterFormat.PDF;
    }
    else if (this.ExportFormat.SelectedValue == "Word")
    {
        fileName = "sales-order-detail.docx";
        format = WriterFormat.Word;
    }
    else if (this.ExportFormat.SelectedValue == "Html")
    {
        fileName = "sales-order-detail.Html";
    }
}
```

```
format = WriterFormat.HTML;
}
else if (this.ExportFormat.SelectedValue == "PPT")
{
    fileName = "sales-order-detail.ppt";
    format = WriterFormat.PPT;
}
else
{
    fileName = "sales-order-detail.xlsx";
    format = WriterFormat.Excel;
}
writer.Save(fileName, format, httpContext.Response);
}
```

Client side changes

1. Open the `Default.aspx` home page. Replace the following code snippet on the `Default.aspx` home page. Ensure that inherits value is same for application name `Inherits=". _Default"`.

```
`html
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="WebformsWriter._Default" %>
<!DOCTYPE html>
<html>
<head runat="server">
<title> BoldReports Writer </title>
</head>
<body>
<form id="form1" runat="server">
<div>
</div>
</form>
</body>
</html>
```

2. Add the following code example in the `Page_Load` tag of the Default.aspx page to view Report Writer export options.

```

`html
<body>
<div class="container">
<div class="content_section">
<form id="form1" runat="server">
<div id="description_Pane" style="text-align: justify;">
<h3>Description</h3>
<span>
Bold ReportWriter is a powerful control for exporting RDL and RDLC files into specified
format files.
</span>
</div>
<div id="export_Pane" style="margin-top: 4%;">
<h3>Export Report</h3>
<span>
Choose a file format to view the selected document generated from Report file by using Essential
ReportWriter.
</span>
<div id="selection_Pane" style="margin-top: 2%;">
<asp:Label Style="font-size: large;" runat="server">
Save As :
</asp:Label>
<asp:RadioButtonList RepeatLayout="Flow" ID="ExportFormat" RepeatDirection="Horizontal"
runat="server">
<asp:ListItem Selected="True">PDF</asp:ListItem>
<asp:ListItem>Word</asp:ListItem>
<asp:ListItem>Excel</asp:ListItem>
<asp:ListItem>Html</asp:ListItem>
<asp:ListItem>PPT</asp:ListItem>
<asp:ListItem>csv</asp:ListItem>

```

```

</asp:RadioButtonList>
<asp:Button style="width: 18%; margin-left: 2%;" ID="ExportButton" runat="server"
OnClick="ExportButton_Click" Text="Generate" />
</div>
</div>
</form>
</div>
</div>
</body>
`

```

3. Now, run and export the report with a specified export format in your Report Writer application.

Congratulations! You have completed your first Web Forms Writer application! Click [here](#) to download the already created WEB Forms Report Writer application.

Note: You can refer to our feature tour page for the [ASP.NET Web Forms Report Writer](#) to see its innovative features. Additionally, you can view our [ASP.NET Web Forms Report Writer examples](#) which demonstrate the rendering of SSRS RDLC and RDL reports.

Export RDLC Report

You can export the RDLC reports that exist on the local file system with custom business object data collection. The following steps demonstrates how to export a RDLC report using Report Writer.

This section requires an ASP.NET Web Forms Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

Bind data source in Web API controller

The following steps help you to configure the Web API to render the RDLC report with business object data collection.

1. Create a class and methods that returns business object data collection. Use the following code in your application Web API Service.

```

`csharp
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
}

```



```
public string ProductImage { get; set; }
public static IList GetData()
{
    List<ProductList> datas = new List<ProductList>();
    ProductList data = new ProductList()
    {
        ProductName = "Baked Chicken and Cheese",
        OrderId = "323B60",
        Price = 55,
        Category = "Non-Veg",
        Ingredients = "grilled chicken, corn and olives.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Delite",
        OrderId = "323B61",
        Price = 100,
        Category = "Non-Veg",
        Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Tikka",
        OrderId = "323B62",
        Price = 64,
        Category = "Non-Veg",
        Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
        ProductImage = ""
    };
};
```

```

datas.Add(data);
return datas;
}
}
`

```

2. In this tutorial, the **Product List.rdlc** report is used, and it can be downloaded [here](#). Copy and paste the sample RDLC reports into the **Resources** folder. For more information, see [Samples and demos](#).
3. Create a folder **Resources** folder in your application. Copy and paste the sample RDLC reports into the **Resource** folder.
4. Open the **Default.aspx.cs** file in your application and add the **Export()**function to load the report as stream. Refer to the following code snippet.

```

`csharp
public partial class _Default : Page
{
protected void ExportButton_Click(object sender, EventArgs e)
{
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.ReportPath = Server.MapPath("~/Resources/Product List.rdl");
}
}
`

```

5. Initialize the Report Writer instance with created **reportStream** and Set the value of the **ReportProcessingMode** property value **ProcessingMode.Local** to provide the dataset collection for that RDLC report.

```

`csharp
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.ReportProcessingMode = ProcessingMode.Local;
`

```

6. Bind the business object data values collection by adding a new item to the **DataSources** as in the following code snippet.

```

`csharp
//Pass the dataset collection for report

```

```
writer.DataSources.Clear();
```

```
writer.DataSources.Add(new BoldReports.Web.ReportDataSource { Name = "list", Value =  
ProductList.GetData() });
```

```
,
```

Data source **Name** is case sensitive and it should be same as in the data source name in the report definition and the **Value** accepts IList, DataSet, and DataTable inputs.

7. You can use the **Save** method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as Stream.

```
`csharp
```

```
protected void ExportButton_Click(object sender, EventArgs e)
```

```
{
```

```
    string fileName = null;
```

```
    WriterFormat format;
```

```
    HttpContext httpContext = System.Web.HttpContext.Current;
```

```
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
```

```
    writer.ReportPath = Server.MapPath("~/Resources/Product List.rdlc");
```

```
    if (this.ExportFormat.SelectedValue == "PDF")
```

```
    {
```

```
        fileName = "Product List.pdf";
```

```
        format = WriterFormat.PDF;
```

```
    }
```

```
    else if (this.ExportFormat.SelectedValue == "Word")
```

```
    {
```

```
        fileName = "Product List.docx";
```

```
        format = WriterFormat.Word;
```

```
    }
```

```
    else if (this.ExportFormat.SelectedValue == "Html")
```

```
    {
```

```
        fileName = "Product List.Html";
```

```
        format = WriterFormat.HTML;
```

```
    }
```

```
    else if (this.ExportFormat.SelectedValue == "PPT")
```

```
    {
```

```

fileName = "Product List.ppt";
format = WriterFormat.PPT;
}
else
{
fileName = "Product List.xlsx";
format = WriterFormat.Excel;
}
writer.Save(fileName, format, httpContext.Response);
}
`

```

8. Now, the RDLC report exported successfully in your application.

Export SSRS Report Server Report

Report Writer has support to Export RDL reports into popular file formats such as PDF, Microsoft Word, Microsoft Excel, and CSV from SSRS Report Server.

To export the SSRS Reports, set the `ReportProcessingMode`, `ReportServerURL`, and `ReportPath` properties in Web API as shown in the following steps.

1. To create your first ASP.NET Web Forms Report Writer application, refer to the [Getting-started](#) section.
2. Set the `reportProcessignMode` API for Bold Report Writer as shown in the following code snippet.

```

`csharp
protected void ExportButton_Click(object sender, EventArgs e)
{
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.ReportProcessingMode = ProcessingMode.Remote;
}
`

```

3. Set the `reportServerUrl` API for Bold Report Writer with `Web Service URL` in the WebAPI as shown in the following code snippet.

```

`csharp
protected void ExportButton_Click(object sender, EventArgs e)

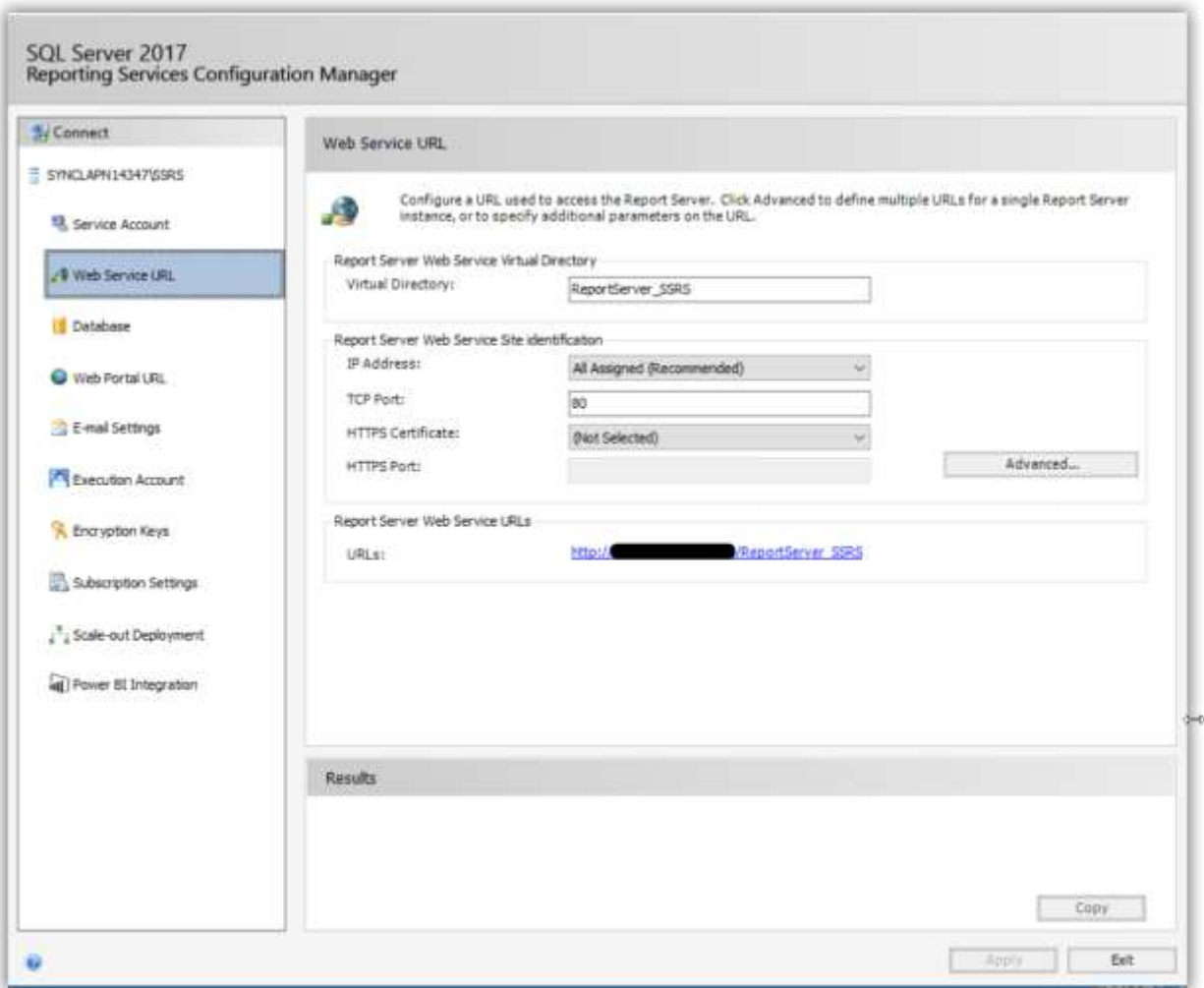
```

```

{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
    writer.ReportServerUrl = "http://<servername>/Reports_SSRS";
}

```

The Web Service URL should be set as `reportServerUrl` when using ASP.NET Web Forms service. The Web Service URL can be found from the Reporting Services Configuration manager under the **Web Service URL** section, as shown in the following image.



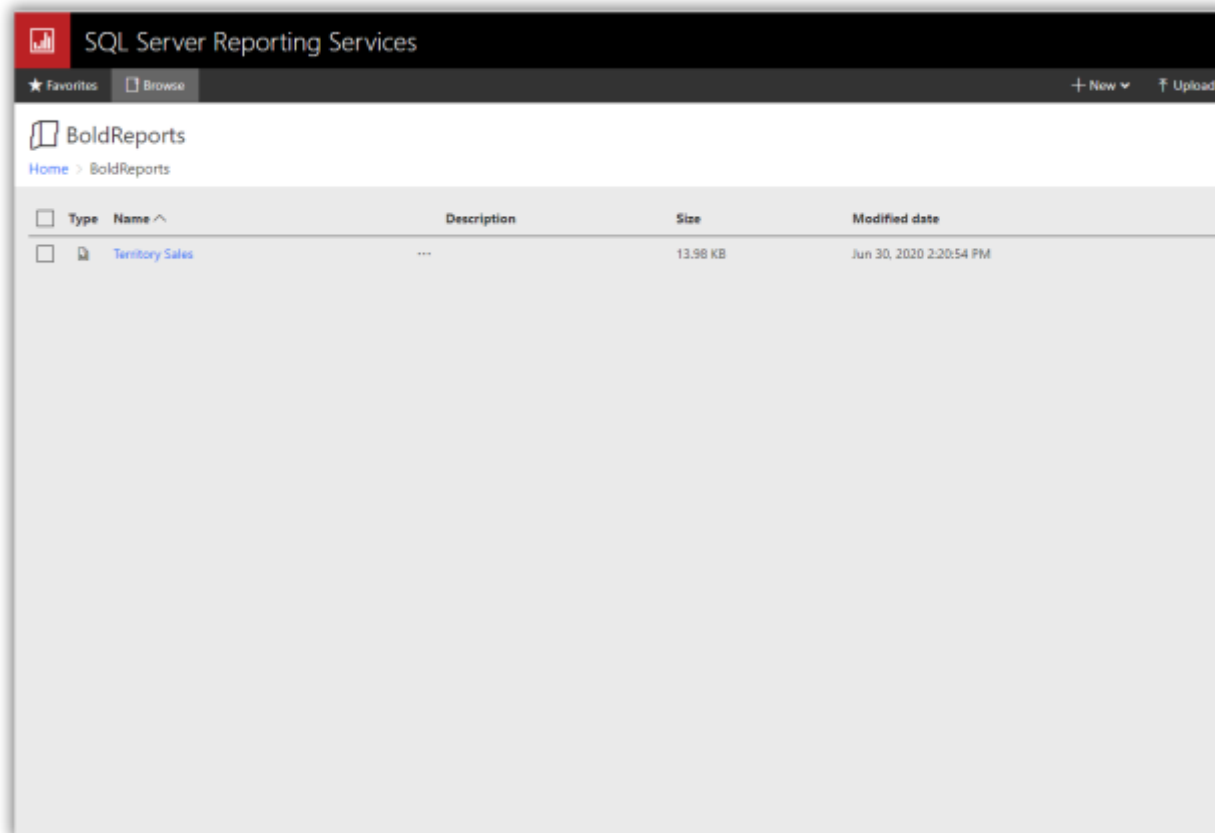
- Set the report path for loading the reports from the SSRS Report Server. The report path should be in the format of `/folder name/report name`.

```
`csharp
```

```
protected void ExportButton_Click(object sender, EventArgs e)
```

```
{
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.ReportProcessingMode = ProcessingMode.Remote;
writer.ReportServerUrl = "http://<servername>/Reports_SSRS";
writer.ReportPath= Server.MapPath("/SSRSSamples2/Territory Sales");
}
`
```

The report path can be found from the SSRS Report Server by navigating to the path of the report to be loaded, as shown in the following image.



Network credentials for SSRS

The network credentials are required to load specified SSRS report from the specified SSRS Report Server using the Report Writer. Specify the `ReportServerCredential` property in writer instance.

```
`csharp
writer.ReportServerCredential = new System.Net.NetworkCredential("username", "password");
`
```

If you are facing problem to access the SSRS Report server reports, you can refer [How to provide the permission for user to access the SSRS Report Server reports](#).

Set data source credential to shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS Server. If the report has any data source that uses credentials to connect with the database, then you should specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp
```

```
List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new  
List<BoldReports.Web.DataSourceCredentials>();  
  
dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",  
"username", "password"));  
  
writer.SetDataSourceCredentials(dataSourceCredentialsList);  
`
```

Data source credentials should be added to the shared data sources that do not have credentials in the connection strings.

Refer to the following final code snippet example to export the SSRS report server reports.

```
`csharp
```

```
protected void ExportButton_Click(object sender, EventArgs e)  
{  
    string fileName = null;  
    WriterFormat format;  
    HttpContext httpContext = System.Web.HttpContext.Current;  
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();  
    writer.ReportProcessingMode = ProcessingMode.Remote;  
    writer.ReportServerUrl = "http://<servername>/Reports_SSRS";  
    writer.ReportPath = Server.MapPath("/SSRSSamples2/Territory Sales");  
    writer.ReportServerCredential = new System.Net.NetworkCredential("username", "password");  
    List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new  
    List<BoldReports.Web.DataSourceCredentials>();  
    dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",  
    "username", "password"));  
    writer.SetDataSourceCredentials(dataSourceCredentialsList);  
    if (this.ExportFormat.SelectedValue == "PDF")  
    {  
        fileName = "sales-order-detail.pdf";  
        format = WriterFormat.PDF;  
    }  
}
```

```
else if (this.ExportFormat.SelectedValue == "Word")
{
    fileName = "sales-order-detail.docx";
    format = WriterFormat.Word;
}
else if (this.ExportFormat.SelectedValue == "Html")
{
    fileName = "sales-order-detail.Html";
    format = WriterFormat.HTML;
}
else if (this.ExportFormat.SelectedValue == "PPT")
{
    fileName = "sales-order-detail.ppt";
    format = WriterFormat.PPT;
}
else
{
    fileName = "sales-order-detail.xlsx";
    format = WriterFormat.Excel;
}
writer.Save(fileName, format, httpContext.Response);
}
```

Export SharePoint Server Report

Report Writer has support to Export RDL reports into popular file formats like PDF, Microsoft Word, Microsoft Excel, and HTML from SharePoint server reports.

This section requires an ASP.NET Web Forms Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

To export the SharePoint server reports, Initialize the Report Writer and set the `ReportProcessingMode`, `ReportServerURL`, and `ReportPath` properties in Web API as shown in the following code snippet.

```
`csharp
[HttpPost]
public ActionResult Export(string writerFormat)
```



```
{
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
    writer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
    writer.ReportPath = "http://<servername>/reportserver$instanceName/Report_Path";
}
`csharp
```

In SharePoint server, the **ReportServerUrl** will be same as your site URL. The **ReportPath** is relative to the Report Server URL with the file extension.

Forms credential for SharePoint Server

The forms credentials are required to load the SharePoint integrated SSRS report from the specified SharePoint integrated SSRS Report Server using the Report Viewer. Specify the **ReportServerFormsCredential** property to access the share point reports.

```
`csharp
writer.ReportServerFormsCredential = new
    BoldReports.Web.ReportServerFormsCredential("username", "password");
`csharp
```

Set data source credential to shared data sources

The shared data source credentials can be added to the **DataSourceCredentials** property to connect with the database.

```
`csharp
List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new
    List<BoldReports.Web.DataSourceCredentials>();
dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",
    "username", "password"));
writer.SetDataSourceCredentials(dataSourceCredentialsList);
`csharp
```

Data source credentials should be added to shared data sources that do not have credentials in the connection strings.

Refer to the following final code snippet example to export the SharePoint server reports.

```
`csharp
protected void ExportButton_Click(object sender, EventArgs e)
{
    HttpContext httpContext = System.Web.HttpContext.Current;
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportProcessingMode = ProcessingMode.Remote;
}
```

```
writer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
writer.ReportPath = "http://<servername>/reportserver$instanceName/Report_Path";
writer.ReportServerFormsCredential = new
BoldReports.Web.ReportServerFormsCredential("username", "password");
List<BoldReports.Web.DataSourceCredentials> dataSourceCredentialsList = new
List<BoldReports.Web.DataSourceCredentials>();
dataSourceCredentialsList.Add(new BoldReports.Web.DataSourceCredentials("datasource name",
"username", "password"));
writer.SetDataSourceCredentials(dataSourceCredentialsList);
string fileName = null;
WriterFormat format;
if (this.ExportFormat.SelectedValue == "PDF")
{
fileName = "Testingreport.pdf";
format = WriterFormat.PDF;
}
else if (this.ExportFormat.SelectedValue == "Word")
{
fileName = "Product List.docx";
format = WriterFormat.Word;
}
else if (this.ExportFormat.SelectedValue == "Html")
{
fileName = "Testingreport.Html";
format = WriterFormat.HTML;
}
else if (this.ExportFormat.SelectedValue == "PPT")
{
fileName = "Testingreport.ppt";
format = WriterFormat.PPT;
}
else
{
fileName = "Testingreport.xlsx";
```

```
format = WriterFormat.Excel;
}
writer.Save(fileName, format, httpContext.Response);
}
`
```

Export Subreport

You can export a Main report with subreport with popular file formats such as PDF, Microsoft Word, and Microsoft Excel without previewing the report in webpage using ASP.NET Report Writer application.

This section requires an ASP.NET Web Forms Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

Export RDL Subreport

In this tutorial, the `SideBySideMainReport.rdl`, `SideBySideSubReport.rdl` reports is used, and it can be downloaded [here](#). You can get the report from Bold Reports installation location. The reports used from installed location requires `NorthwindIO_Reports.sdf` database to run, so add the database to your application. For more information, refer to [Samples and demos](#).

Refer to the following steps to export the RDL sub report with specified format.

1. Create a folder `Resources` in your application. Copy and paste the sample RDL reports into the `Resources` folder.
2. Open the `Default.aspx.cs` file in your application and add the `Export()` function to load the report as stream. Refer to the following code snippet.

```
`csharp
public partial class _Default : Page
{
protected void ExportButton_Click(object sender, EventArgs e)
{
// Here, we have loaded the sample reports from application the Resources folder.
FileStream mainReportStream = new
FileStream(Server.MapPath("~/Resources/SideBySideMainReport.rdl"), FileMode.Open,
FileAccess.Read);

FileStream subReportStream = new
FileStream(Server.MapPath("~/Resources/SideBySideSubReport.rdl"), FileMode.Open, FileAccess.Read);
}
}
`
```

3. Initialize the Report Writer instance and pass the subreport name and stream to the `LoadSubreport()` method in Report Writer instance.

```
`csharp
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadSubreport("SideBySideSubReport", subReportStream);
`
```

4. After loading the main report stream using Report Writer instance. Refer to the following code snippet.

```
`csharp
writer.LoadReport(mainReportStream);
`
```

5. You can use the `Save` method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as Stream.

```
`csharp
protected void ExportButton_Click(object sender, EventArgs e)
{
    FileStream mainReportStream = new
    FileStream(Server.MapPath("~/Resources/SideBySideMainReport.rdl"), FileMode.Open,
    FileAccess.Read);

    FileStream subReportStream = new
    FileStream(Server.MapPath("~/Resources/SideBySideSubReport.rdl"), FileMode.Open, FileAccess.Read);

    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    HttpContext httpContext = System.Web.HttpContext.Current;
    writer.LoadSubreport("SideBySideSubReport", subReportStream);
    writer.LoadReport(mainReportStream);
    string fileName = null;
    WriterFormat format;
    string type = null;
    if (this.ExportFormat.SelectedValue == "PDF")
    {
        fileName = "SideBySideMainReport.pdf";
        type = "pdf";
        format = WriterFormat.PDF;
    }
}
```

```

}
else if (this.ExportFormat.SelectedValue == "Word")
{
fileName = "SideBySideMainReport.docx";
type = "docx";
format = WriterFormat.Word;
}
else if (this.ExportFormat.SelectedValue == "CSV")
{
fileName = "SideBySideMainReport.csv";
type = "csv";
format = WriterFormat.CSV;
}
else
{
fileName = "SideBySideMainReport.xlsx";
type = "xlsx";
format = WriterFormat.Excel;
}
writer.Save(fileName, format, HttpContext.Response);
}
`

```

6. Now, the RDL report exported successfully in your application.

Export RDLC subreport

To export the RDLC report along with subreport, we need to load the subreport streams before loading a main report to the Report Writer as in the following code example, then only subreport processing event will work.

```

`csharp
protected void ExportButton_Click(object sender, EventArgs e)
{
// Here, we have loaded the sample report from application the folder App_Data.
FileStream mainReportStream = new FileStream(Server.MapPath(@"Resources\MainReport.rdlc"),
FileStream.Open, FileAccess.Read);

```

```
FileStream subreport1Stream = new FileStream(Server.MapPath(@"\Resources\SubReport1.rdlc"),
FileStream.Open, FileAccess.Read);

FileStream subreport2Stream = new FileStream(Server.MapPath(@"\Resources\SubReport2.rdlc"),
FileStream.Open, FileAccess.Read);

ReportWriter writer = new ReportWriter();

writer.ReportProcessingMode = ProcessingMode.Local;

writer.SubreportProcessing += ReportWriter_SubreportProcessing;

string fileName = null;

WriterFormat format;

//Adding sub report stream going to be used with exporting report.
writer.LoadSubreport("SubReport1", subreport1Stream);
writer.LoadSubreport("SubReport2", subreport2Stream);

//Loading the report going to export as PDF.
writer.LoadReport(mainReportStream);

writer.DataSources.Clear();

writer.DataSources.Add(new ReportDataSource { Name = "DataSet", Value = MainReport.GetData() });

HttpContext httpContext = System.Web.HttpContext.Current;

writer.Save(fileName, format, httpContext.Response);
}

private void ReportWriter_SubreportProcessing(object sender, SubreportProcessingEventArgs e)
{
    if (e.ReportPath == "SubReport1")
    {
        //Pass the dataset collection for subreport
        e.DataSources.Clear();
        e.DataSources.Add(new ReportDataSource { Name = "DataSet1", Value = SubReport1.GetData() });
    }
    else if (e.ReportPath == "SubReport2")
    {
        //Pass the dataset collection for subreport
        e.DataSources.Clear();
        e.DataSources.Add(new ReportDataSource { Name = "DataSet2", Value = SubReport2.GetData() });
    }
}
```

Export Parameter Report

You can set report parameter default values or modify the values using the `SetParameters()` method of Report Writer instance. This section describes how to modify the exported document report parameter values.

This section requires an ASP.NET Web Forms Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

In this tutorial, the `sales-order-detail.rdl` report is used and it can be downloaded [here](#). You can get the reports from Bold Reports installation location. For more information, refer to [samples and demos](#) section.

Set report parameters to export file

1. Create a folder `Resources` folder in your application. Copy and paste the sample RDL reports into the `Resources` folder.
2. To load the report path on a `Export()` function to load the report as stream.

```
`csharp
public partial class _Default : Page
{
    protected void ExportButton_Click(object sender, EventArgs e)
    {
        // Here, we have loaded the sample reports from application the Resources folder.
        BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
        writer.ReportPath = Server.MapPath("~/Resources/sales-order-detail.rdl");
        writer.ReportProcessingMode = ProcessingMode.Remote;
        .....
    }
}
```

3. You can add collection of report parameters programmatically using the `ReportParameter` property that is the list type of Report Parameter class. Refer to the following code snippet to set the report parameter name and value.

```
`csharp
List<BoldReports.Web.ReportParameter> userParameters = new
List<BoldReports.Web.ReportParameter>();
userParameters.Add(new BoldReports.Web.ReportParameter()
```

```
{
Name = "SalesOrderNumber",
Values = new List<string>() { "SO50756" }
});
writer.SetParameters(userParameters);
,
```

4. You can use the **Save** method in Report Writer to generate the export document along with information of the report stream, it will return the generated file as Stream.

```
`csharp
protected void ExportButton_Click(object sender, EventArgs e)
{
// Here, we have loaded the sample reports from application the Resources folder.
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.ReportPath = Server.MapPath("~/Resources/sales-order-detail.rdl");
writer.ReportProcessingMode = ProcessingMode.Remote;
HttpContext httpContext = System.Web.HttpContext.Current;
List<BoldReports.Web.ReportParameter> userParameters = new
List<BoldReports.Web.ReportParameter>();
userParameters.Add(new BoldReports.Web.ReportParameter()
{
Name = "SalesOrderNumber",
Values = new List<string>() { "SO50756" }
});
writer.SetParameters(userParameters);
string fileName = null;
WriterFormat format;
if (this.ExportFormat.SelectedValue == "PDF")
{
fileName = "sales-order-detail.pdf";
format = WriterFormat.PDF;
}
else if (this.ExportFormat.SelectedValue == "Word")
{
```



```
fileName = "sales-order-detail.docx";
format = WriterFormat.Word;
}
else if (this.ExportFormat.SelectedValue == "Html")
{
    fileName = "sales-order-detail.Html";
    format = WriterFormat.HTML;
}
else if (this.ExportFormat.SelectedValue == "PPT")
{
    fileName = "sales-order-detail.ppt";
    format = WriterFormat.PPT;
}
else
{
    fileName = "sales-order-detail.xlsx";
    format = WriterFormat.Excel;
}
writer.Save(fileName, format, httpContext.Response);
}
```

5. Now, the parameter report exported successfully in your application.

Report Writer Events

You can handle the Report Writer events with reports using the following events.

- SubreportProcessing
- ReportErrorOccurred

Subreport Processing

The **SubreportProcessing** event occurs when subreport is loaded and it is ready to start the processing. You can handle the event and specify the data source, parameters, and subreport loading. The following sample code loads a subreport by assigning the report data source credentials and parameter values in the **SubreportProcessing** event.

1. Report Writer instance used to register the **SubreportProcessing** event in your Report Writer application.

```
`csharp
writer.SubreportProcessing += (sen, arg) =>
{
    // Assign the data source and parameter values to the sub report.
};
`
```

2. Set the subreport data source details to the subreport processing event argument.

```
`csharp
writer.SubreportProcessing += (sen, arg) =>
{
    arg.DataSources.Add(new BoldReports.Web.ReportDataSource {Name= "Datasource name", Value =
    value });
}
`
```

Data source **Name** is case sensitive and it should be same as in the data source name in the report definition. The **Value** accepts IList, DataSet, and DataTable inputs.

3. Set the subreport parameters value to the subreport processing event argument.

```
`csharp
writer.SubreportProcessing += (sen, arg) =>
{
    arg.Parameters.Add(new BoldReports.Web.ReportParameterInfo { Name = "Parameter name", Values =
    values});
}
`
```

Report Error

The **ReportErrorOccurred** event raises when an error occurs in the report processing. You can handle the event and get the report error details from the event arguments. Refer to the following sample code to handle the report errors in the **ReportErrorOccurred** event.

```
`csharp
writer.ReportErrorOccurred += (sen, arg) =>
{
```

```
// You can handle the report errors using event arguments.
```

```
};  
,
```

Error logging in ASP.NET Web Forms Report Writer

If an error occurred in report processing, ASP.NET Web Forms Report writer **ReportErrorOccurred** event raised. You can register the event and get the report error details from the event arguments to save all logs, stack trace, and error information into a physical file location.

This section explains how to log the detailed error information to your ASP.NET Web Forms application.

This section requires an ASP.NET Web Forms Report Writer application, if you don't have, then create by referring to the [Getting-Started](#) documentation.

1. In Solution Explorer, open the **Default.aspx.cs** file.
2. Add the following sample code to register the report errors in the **ReportErrorOccurred** event.

```
`csharp  
protected void ExportButton_Click(object sender, EventArgs e)  
{  
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();  
    writer.ReportErrorOccurred += Writer_ReportErrorOccurred;  
}  
private void Writer_ReportErrorOccurred(object sender, ReportErrorOccurredEventArgs e)  
{  
    // You can register the report errors using event arguments.  
}  
,
```

3. Create a method in **Default.aspx.cs** to write the error text into application folder.

```
`csharp  
private void WriteLogs(string errorMessage)  
{  
    string filePath = Path.Combine(writer.ReportPath, "ErrorDetails.txt");  
    using (StreamWriter writer = new StreamWriter(filePath, true))  
    {  
        writer.AutoFlush = true;  
        writer.WriteLine(errorMessage);  
    }  
}
```

```

}
}
`

```

4. Invoke the newly created function in `ReportErrorOccurred` as follows.

```

`csharp
private void Writer_ReportErrorOccurred(object sender, ReportErrorOccurredEventArgs e)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2}", e.ClassName,
        e.MethodName, e.Message));
}
`

```

In cases of any issues faced in the report rendering, share the log file to our technical support team to get assistance on that.

5. The final event is given as follows, you can replace it in your application.

```

`csharp
protected void ExportButton_Click(object sender, EventArgs e)
{
    // Here, we have loaded the sales-order-detail sample report from application the folder Resources.
    FileStream reportStream = new FileStream(Server.MapPath("~/Resources/sales-order-detail.rdl"),
        FileMode.Open, FileAccess.Read);
    BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
    writer.ReportErrorOccurred += Writer_ReportErrorOccurred;
    string fileName = null;
    WriterFormat format;
    string type = null;
    if (this.ExportFormat.SelectedValue == "PDF")
    {
        fileName = "sales-order-detail.pdf";
        format = WriterFormat.PDF;
    }
    else if (this.ExportFormat.SelectedValue == "Word")
    {

```

```
fileName = "sales-order-detail.docx";
format = WriterFormat.Word;
}
else if (this.ExportFormat.SelectedValue == "Html")
{
    fileName = "sales-order-detail.Html";
    format = WriterFormat.HTML;
}
else if (this.ExportFormat.SelectedValue == "PPT")
{
    fileName = "sales-order-detail.ppt";
    format = WriterFormat.PPT;
}
else
{
    fileName = "sales-order-detail.xlsx";
    format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
HttpContext httpContext = System.Web.HttpContext.Current;
writer.Save(fileName, format, httpContext.Response);
}
}

private void Writer_ReportErrorOccurred(object sender, ReportErrorOccurredEventArgs e)
{
    WriteLogs(string.Format("Class Name: {0} \n Method Name: {1} \n Error Message: {2}", e.ClassName,
e.MethodName, e.Message));
}

private void WriteLogs(string errorMessage)
{
    string filePath = Path.Combine(Server.MapPath("~/Resources/ErrorDetails.txt"));
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
```

```
writer.AutoFlush = true;  
writer.WriteLine(errorMessage);  
}  
}  
`
```

Encrypt and Secure Documents

Encrypt and Secure Documents option allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password. The following code snippet explains how to encrypt the exported document with the user defined password.

Password Protected PDF document

You can protect the exported PDF document using the following code snippet.

```
`csharp  
writer.PDFOptions = new BoldReports.Writer.PDFOptions();  
writer.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity  
{  
    UserPassword = "Password"  
};  
`
```

Password Protected Word document

You can protect the exported Word document using the following code snippet.

```
`csharp  
writer.WordOptions = new BoldReports.Writer.WordOptions()  
{  
    EncryptionPassword = "password"  
};  
`
```

Password Protected Excel document

You can protect the exported Excel document using the following code snippet.

```
`csharp  
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()  
{  
    PasswordToModify = "password",  
    PasswordToOpen = "password"  
};  
`
```

Advance Layouts For Merged Cells

Report Writer supports additional export layouts in Word and Excel export formats to eliminate the tiny columns, rows, and merged cells. The benefits of the layout are:

- Overcomes the merging problem with tiny cells, rows, and columns in SSRS Excel and Word exporting.
- Provides clear readability by eliminating the tiny columns, rows, and merge cells.
- Allows you to perform data manipulations such as applying sorting, filters, and grouping in Excel.

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` to `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
```

```
writer.WordOptions.LayoutOption = WordLayoutOptions.TopLevel;
writer.WordOptions.ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
    Bottom = 0.5f,
    Top = 0.5f
};
```

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior. The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, add an empty paragraph between two tables.

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` as `IgnoreCellMerge`.

```
`csharp
```

```
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
};
```

Change the Default File Format

Users can change the default file format to any other file format that is supported by the selected file type. It includes APIs to set the formats before exporting the document. Refer to the following section to change the default file formats.

Change the Word document type

You can save the report to the required Word document version by setting the `FormatType` property.

```
`csharp
writer.WordOptions = new BoldReports.Writer.WordOptions()
{
    FormatType = WordFormatType.Docx
};
`
```

Change the Excel document type

You can save the report to the required Excel document version by setting the `ExcelSaveType` property.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013
};
`
```

PDF Settings

The PDF export options provides properties to manage PDF export behaviors. You can set the customization properties in the `PDFOptions`. Refer to the following code snippet to initialize the `PDFOptions` property.

```
`csharp
writer.PDFOptions = new PDFOptions();
`
```

Export with complex scripts

To export reports with the complex script language texts, set the `ComplexScript` property of `PDFOptions` instance to `true`.

```
`csharp
writer.PDFOptions.EnableComplexScript = true;
`
```


PDF conformance

You can export the report as a PDF/A-1b document by specifying the PdfConformanceLevel.Pdf_A1B conformance level in the PdfConformanceLevel property.

```
`csharp
writer.PDFOptions.PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B;
`
```

Embedding Custom PDF fonts

You can add custom fonts to the PDF exported document by adding the font streams to Fonts collection in PDFOptions instance.

To add custom fonts to the PDF exported document, follow these steps:

1. Add the font .ttf files to your application Resources folder.
2. In the Solution Explorer, open the properties of the font file and set the Copy property to Output Directory as Copy always.
3. Initialize the Font collection and add the font stream to it.

The key value provided in the font collection should be same as in the report item font property.

```
`csharp
//Load Missing font stream to the pdf document
writer.PDFOptions.Fonts = new Dictionary<string, System.IO.Stream>
{
    { "Segoe UI", new FileStream(server.MapPath(@"Resources\Font\filename.ttf"), FileMode.Open,
    FileAccess.Read) }
};
`
```

If any fonts used in the report definition is not installed or available in the local system, then you should load the font stream like above code snippet.

Password Protected PDF document

Allows you to protect the exported PDF document from unauthorized users by encrypting the document using encryption password. The following code snippet explains how to encrypt the exported document with user-defined password.

```
`csharp
writer.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity
{
    UserPassword = "Password"
};
`
```

Excel Settings

The Excel export options provides properties to manage Excel document export behaviors. You can set the customization properties in the `ExcelOptions`. Refer to the following code snippet to initialize the `ExcelOptions` property.

```
`csharp
writer.ExcelOptions = new ExcelOptions();
`
```

Excel document type

You can save the report to the required Excel version by setting the `ExcelSaveType` property.

```
`csharp
writer.ExcelOptions.ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013;
`
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the `LayoutOption` as `IgnoreCellMerge`.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    LayoutOption = BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge
};
`
```

Protecting Excel document from editing

You can restrict the Excel document from editing by providing the `ExcelSheetProtection` or enabling the `ReadOnlyRecommended` properties.

```
`csharp
writer.ExcelOptions.ReadOnlyRecommended = true;
writer.ExcelOptions.ExcelSheetProtection = Syncfusion.XlsIO.ExcelSheetProtection.DeletingColumns;
`
```

Change Excel export format

Allows you to change the default file format to any other file format using the `ExcelSaveType` properties.

```
`csharp
writer.ExcelOptions.ExcelSaveType = ExcelVersion.Excel2013;
`
```

Password Protected Excel document

Allows you to protect the exported Excel document from unauthorized users by encrypting the document using the encryption password. The following code snippet explains how to encrypt the exported document with user-defined password.

```
`csharp
writer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    PasswordToModify = "password",
    PasswordToOpen = "password"
};
`
```

Word Settings

The Word export options provides properties to manage Word document export behaviors. You can set the customization properties in the **WordOptions**. Refer to the following code snippet to initialize the **WordOptions** property.

```
`csharp
writer.WordOptions = new WordOptions();
`
```

Word document type

You can save the report to the required document version by setting the **FormatType** property.

```
`csharp
writer.WordOptions.FormatType = WordFormatType.Docx;
`
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the **LayoutOption** to **TopLevel**. The **ParagraphSpacing** is the distance value added between two elements in the document.

```
`csharp
writer.WordOptions.LayoutOption = WordLayoutOptions.TopLevel;
writer.WordOptions.ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
    Bottom = 0.5f,
    Top = 0.5f
};
`
```

A paragraph element is inserted between two tables in the exported document to overcome the Word document auto merging behavior. The table in the word document is not a stand-alone object. If you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, add an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without password. The following are the types of protection:

- **AllowOnlyComments**: Adds or modifies only the comments in the Word document.
- **AllowOnlyFormFields**: Modifies the form field values in the Word document.
- **AllowOnlyRevisions**: Accepts or rejects the revisions in the Word document.
- **AllowOnlyReading**: Views the content only in the Word document.
- **NoProtection**: Accesses or edits the Word document contents as normally.

```
`csharp
```

```
writer.WordOptions.ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading;
```

```
`
```

Password Protected Word document

Allows you protect the exported Word document from unauthorized users by encrypting the document using encryption password. The following code snippet explains how to encrypt the exported document with user-defined password.

```
`csharp
```

```
writer.WordOptions = new BoldReports.Writer.WordOptions()
```

```
{
```

```
EncryptionPassword = "password"
```

```
};
```

```
`
```

Export Data Visualization in Web Forms Environment

we have provided the below option to export the data visualization report items

1. [External browser - Puppeteer](#).
2. [PhantomJS - Classic Tool](#)

External Browser - Puppeteer

1. It supports **ASP.NET Core 2.1** or above version only.
2. Open the Report Writer application and install the **PuppeteerSharp** package from [NuGet Gallery](#).
3. Create a folder like **wwwroot/ResourcesTemp** in your **Report Writer** application.
4. Open the Report Writer application Web API file.

5. Set the `BrowserTypes` Enum property to `External`.
6. Create and initialize `CustomBrowserType` by inheriting the `ExportSettings` class.
7. Set the `ResourcePath`, `Scripts` and `DependentScripts` property in `Report Writer` instance.
The following code example demonstrates how to configure `Puppeteer` in your ASP.NET Core application.

In this tutorial, the `product-line-sales.rdl` report is used, and it can be downloaded in this [link](#).

```
`csharp
public class HomeController : Controller
{
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;
    // IWebHostEnvironment initialized with controller to get the data from application data folder.
    public HomeController(Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _hostingEnvironment = hostingEnvironment;
    }
    public IActionResult Index()
    {
        return View();
    }
    [HttpPost]
    public IActionResult Export(string writerFormat)
    {
        string fileName = null;
        WriterFormat format;
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sample report from application the folder wwwroot.
        FileStream inputStream = new FileStream(basePath + @"\Resources\product-line-sales.rdl",
        FileMode.Open, FileAccess.Read);
        MemoryStream reportStream = new MemoryStream();
        inputStream.CopyTo(reportStream);
        reportStream.Position = 0;
        inputStream.Close();
        BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
```

```
// Initialize the class for puppeteer
writer.ExportSettings = new CustomBrowserType();
// Puppeteer Option to export the data visualization report items.
writer.ExportResources.BrowserType = ExportResources.BrowserTypes.External;
writer.ExportResources.ResourcePath = basePath + @"/ResourcesTemp/";
writer.ExportResources.Scripts = new List<string>
{
//Gauge component scripts
"https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js",
"https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js",
"https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js",
"https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js",
"https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-lineargauge.min.js",
"https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-circulargauge.min.js",
//Map component script
"https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js",
"https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js",
"https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js",
//Chart component script
"https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js",
//Report Viewer Script
"https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js",
};
writer.ExportResources.DependentScripts = new List<string>
{
"https://code.jquery.com/jquery-1.10.2.min.js"
};
if (writerFormat == "PDF")
{
fileName = "ProductLineSales.pdf";
format = WriterFormat.PDF;
}
else if (writerFormat == "Word")
```

```
{
fileName = "ProductLineSales.docx";
format = WriterFormat.Word;
}
else if (writerFormat == "CSV")
{
fileName = "ProductLineSales.CSV";
format = WriterFormat.CSV;
}
else
{
fileName = "ProductLineSales.xlsx";
format = WriterFormat.Excel;
}
writer.LoadReport(reportStream);
MemoryStream memoryStream = new MemoryStream();
writer.Save(memoryStream, format);
// Download the generated document from client.
memoryStream.Position = 0;
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/pdf");
fileStreamResult.FileNameDownload = fileName;
return fileStreamResult;
}
// Add the class to override the method for puppeteer
public class CustomBrowserType : ExportSettings
{
// Can be used to convert the image from external browser
public override string GetImageFromHTML(string url)
{
return LaunchPuppeteer(url).Result;
}
// Can be used to convert the image from Puppeteer
public async Task<string> LaunchPuppeteer(string url)
```

```

{
var browserFetcher = new BrowserFetcher();
await browserFetcher.DownloadAsync();
await using var browser = await Puppeteer.LaunchAsync(new LaunchOptions { Headless = true });
await using var page = await browser.NewPageAsync();
await page.GoToAsync(url);

return await
page.WaitForSelectorAsync("#imagejsonData").Result.GetPropertyAsync("innerText").Result.JsonValueA
sync<string>();
}
}
}
`

```

The **Scripts** and **Dependent** scripts must be added to export the data visualization report items.

`PhantomJS - Classic Tool`

To download **PhantomJS** application and deploy it on your machine, you should accept its license terms on [LICENSE](#) and [Third-Party](#) document.

1. Download **PhantomJS** for windows [here](#) and for linux [here](#) and extract the download file.
2. Copy the **PhantomJS** file from the extracted bin folder and paste into **Resources/PhantomJS** folder in your application.
3. Open the Report Writer application Web API file.
4. Set the **UsePhantomJS** property to true.
5. Set the **PhantomJSPath**, **Scripts** and **DependentScripts** property in **Report Writer** instance. The following code example demonstrates how to configure **PhantomJS** in your ASP.NET Web Forms application.

In this tutorial, the **product-line-sales.rdl** report is used, and it can be downloaded in this [link](#).

```

`csharp
protected void ExportButton_Click(object sender, EventArgs e)
{
string fileName = null;
WriterFormat format;
HttpContext httpContext = System.Web.HttpContext.Current;
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.ReportPath = Server.MapPath("~/Resources/sales-order-detail.rdl");
writer.ExportResources.UsePhantomJS = true;

```


// Set the IncludeText property to true, when text not displayed in the data visualization images.

```
writer.ExportResources.IncludeText = true;
```

```
writer.ExportResources.PhantomJSPath = Server.MapPath("/Resources/PhantomJS");
```

```
writer.ExportResources.Scripts = new List<string>
```

```
{
```

```
//Gauge component scripts
```

```
"https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js",
```

```
"https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js",
```

```
"https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js",
```

```
"https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js",
```

```
"https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-lineargauge.min.js",
```

```
"https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-circulargauge.min.js",
```

```
//Map component script
```

```
"https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js",
```

```
"https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js",
```

```
"https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js",
```

```
//Chart component script
```

```
"https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js",
```

```
//Report Viewer Script
```

```
"https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js",
```

```
};
```

```
writer.ExportResources.DependentScripts = new List<string>
```

```
{
```

```
"https://code.jquery.com/jquery-1.10.2.min.js"
```

```
};
```

```
if (this.ExportFormat.SelectedValue == "PDF")
```

```
{
```

```
fileName = "sales-order-detail.pdf";
```

```
format = WriterFormat.PDF;
```

```
}
```

```
else if (this.ExportFormat.SelectedValue == "Word")
```

```
{
```

```
fileName = "sales-order-detail.docx";
```

```
format = WriterFormat.Word;
}
else if (this.ExportFormat.SelectedValue == "Html")
{
fileName = "sales-order-detail.Html";
format = WriterFormat.HTML;
}
else if (this.ExportFormat.SelectedValue == "PPT")
{
fileName = "sales-order-detail.ppt";
format = WriterFormat.PPT;
}
else
{
fileName = "sales-order-detail.xlsx";
format = WriterFormat.Excel;
}
writer.Save(fileName, format, httpContext.Response);
}
}
`
```

The **Scripts** and **Dependent** scripts must be added to export the data visualization report items.

Limitations

RDL Specification support

ReportWriter control does not support RDL Specification for SQL Server 2000 and RDL Specification for SQL Server 2005.

Layout Process

Syncfusion ReportWriter control has some limitations in Tablix Cell split Layout process in comparison with MS ReportWriter. When table cell width value exceeds the page width, the entire cell moves to the next page in order to display the complete cell items.

Unsupported expression

- Object function and VB function do not have complete support in ReportWriter.
- VB Code functions are not supported in **Silverlight** and **WinRT** ReportWriter.

HTML Formatted Data with Report

Report viewer supports showing the HTML formatted data with Textbox report items along with the inline CSS. This section provides the information about supported tags and limitations.

Supported HTML Tags

- Hyperlinks:
- Fonts:
- Header, style and block elements: `

, ,

,

,

- , `
- Text format: , , ,

Limitations of Cascading Style Sheet Attributes

The following is a list of attributes that are supported:

- text-align, text-indent
- font-family
- font-size
- Only valid RDL size values are supported in absolute CSS length units. Supported units are: in, cm, mm, pt, pc, px, ex, and em.
- Relative CSS length units are ignored and are not supported. Unsupported units include percentage (%), and rem.
- color
- padding, padding-bottom, padding-top, padding-right, and padding-left
- font-weight

How to section for Report Writer component

1. [Generate and export RDL and RDLC reports programmatically](#)

How to generate and export RDL and RDLC reports programmatically

The Bold Reports reporting library allows you to generate and export RDL and RDLC reports programmatically. The `ReportDefinition` class is defined as an object model of a report. You can create, add, and modify the report properties, report sections like header and footer and report items in the report definition instance.

The following steps illustrates how to create a basic report object model:

Initialize a report definition

The following code snippet guides you in initializing the report definition.

```
`csharp
ReportDefinition CreateReport()
{
    ReportDefinition report = new ReportDefinition();
    report.ReportSections = new ReportSections();
    var reportSection = new ReportSection();
    report.ReportSections.Add(reportSection);
    reportSection.Width = new BoldReports.RDL.DOM.Size("6in");
    report.ReportUnitType = "Inch";
    report.RDLType = RDLType.RDL2010;
    return report;
}

BoldReports.RDL.DOM.Style AddStyle()
{
    BoldReports.RDL.DOM.Style style = new BoldReports.RDL.DOM.Style();
    style.Border = new BoldReports.RDL.DOM.Border();
    style.Border.Width = new BoldReports.RDL.DOM.Size("1pt");
    style.Border.Style = "Solid";
    style.Border.Color = "Black";
    return style;
}
`
```

Create a Data source for report

The following code snippet guides you in creating a **Datasource** for the report and setting values to their properties.

```
`csharp
ReportDefinition CreateReport()
{
    ...
    ...
    ...
    reportSection.Page = new BoldReports.RDL.DOM.Page();
    reportSection.Page.Style = AddStyle();
    reportSection.Page.PageHeight = "4in";
}
```

```

reportSection.Page.PageWidth = "6in";

var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
<database>");

report.DataSources = new DataSources();
report.DataSources.Add(newDataSource);

...

...

...
}

//If you are using RDLC report then you can pass any string value as connection string
BoldReports.RDL.DOM.DataSource CreateDataSource(string name, string dataProvider, string
connectionString)
{
var dataSource = new BoldReports.RDL.DOM.DataSource();
dataSource.Name = name;
dataSource.SecurityType = BoldReports.RDL.DOM.SecurityType.Integrated;
dataSource.ConnectionProperties = new BoldReports.RDL.DOM.ConnectionProperties();
dataSource.ConnectionProperties.DataProvider = dataProvider;
dataSource.ConnectionProperties.ConnectionString = connectionString;
dataSource.ConnectionProperties.IntegratedSecurity = true;
return dataSource;
}
`

```

Create a Dataset for the report

The following code snippet guides you in creating a **Dataset** for the report and setting values to their properties.

```

`csharp
ReportDefinition CreateReport()
{
...
...
...

reportSection.Page.PageWidth = "6in";

var newDataSource = CreateDataSource("DataSource1", "SQL", "Data Source =.; Initial Catalog =
<database>");

```

```
report.DataSources = new DataSources();
report.DataSources.Add(newDataSource);
var newDataSet = CreateDataSet("DataSource1", "DataSet1");
report.DataSets = new DataSets();
report.DataSets.Add(newDataSet);
...
...
...
}

BoldReports.RDL.DOM.DataSet CreateDataSet(string dataSourceName, string dataSetName)
{
    var dataSet = new BoldReports.RDL.DOM.DataSet();
    dataSet.Name = dataSetName;
    dataSet.Query = new Query();
    dataSet.Query.DataSourceName = dataSourceName;
    dataSet.Query.CommandText = "SELECT
HumanResources.Department.DepartmentID,HumanResources.Department.Name,HumanResources.De
partment.GroupName,HumanResources.Department.ModifiedDate FROM
HumanResources.Department";
    dataSet.Query.QueryDesignerState = new QueryDesignerState();
    var table = CreateTable();
    dataSet.Query.QueryDesignerState.Tables = new List<Table>();
    dataSet.Query.QueryDesignerState.Tables.Add(table);
    dataSet.Fields = new Fields();
    dataSet.Fields.Add(CreateField("DepartmentID", "System.Int16"));
    dataSet.Fields.Add(CreateField("Name", "System.String"));
    dataSet.Fields.Add(CreateField("GroupName", "System.String"));
    dataSet.Fields.Add(CreateField("ModifiedDate", "System.DateTime"));
    return dataSet;
}

BoldReports.RDL.DOM.Table CreateTable()
{
    var table = new Table();
    table.Schema = "HumanResources";
```

```

table.Name = "Department";
table.Columns = new List<Column>();
table.Columns.Add(CreateColumn("DepartmentID"));
table.Columns.Add(CreateColumn("Name"));
table.Columns.Add(CreateColumn("GroupName"));
table.Columns.Add(CreateColumn("ModifiedDate"));
return table;
}

BoldReports.RDL.DOM.Column CreateColumn(string columnName)
{
    var column = new Column();
    column.Name = columnName;
    return column;
}

BoldReports.RDL.DOM.Field CreateField(string fieldName, string type)
{
    var field = new Field();
    field.Name = fieldName;
    field.TypeName = type;
    field.DataField = fieldName;
    return field;
}

```

Create a report page header

The following code snippet guides you in creating a **PageHeader** report section and setting values to their properties.

```

`csharp
ReportDefinition CreateReport()
{
    ...
    ...
    ...
    reportSection.Page = new BoldReports.RDL.DOM.Page();
    reportSection.Page.Style = AddStyle();
}

```

```

reportSection.Page.PageHeight = "4in";
reportSection.Page.PageWidth = "6in";
reportSection.Page.PageHeader = CreateHeader();
...
...
...
}
PageHeader CreateHeader()
{
PageHeader pageHeader = new PageHeader();
pageHeader.Height = new BoldReports.RDL.DOM.Size("0.59167in");
pageHeader.Style = AddStyle();
pageHeader.PrintOnFirstPage = true;
pageHeader.PrintOnLastPage = true;
pageHeader.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
return pageHeader;
}
`

```

Create a report page footer

The following code snippet guides you in creating a **PageFooter** report section and setting values to their properties.

```

`csharp
ReportDefinition CreateReport()
{
...
...
...
reportSection.Page.PageHeader = CreateFooter();
...
...
...
}

```



```
PageFooter CreateFooter()
{
    PageFooter pageFooter = new PageFooter();
    pageFooter.Style = AddStyle();
    pageFooter.Height = new BoldReports.RDL.DOM.Size("0.59167in");
    pageFooter.ReportItems = new ReportItems();
    pageFooter.PrintOnFirstPage = true;
    pageFooter.PrintOnLastPage = true;
    var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
    pageFooter.ReportItems.Add(textBox);
    return pageFooter;
}
```

Initialize a report body section

- Initialize a report body object and set the values to their properties like height, styles, and report items as shown in the following code snippet.

```
`csharp
ReportDefinition CreateReport()
{
    ...
    ...
    ...
    reportSection.Body = CreateBody();
    ...
    ...
    ...
}

Body CreateBody()
{
    var body = new Body();
    body.Height = new BoldReports.RDL.DOM.Size("2.03333in");
    body.Style = AddStyle();
    body.ReportItems = new ReportItems();
}
```

```
var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
body.ReportItems.Add(textBox);
return body;
}
```

- Using the following code snippets, you can create a **Textbox** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

`csharp

PageHeader CreateHeader()

```
{
...
...
...
pageHeader.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox2", "Header", "1.61333in", "0.13833in", "2.2in", "0.34167in");
pageHeader.ReportItems.Add(textBox);
...
...
...
}
```

PageFooter CreateFooter()

```
{
...
...
...
pageFooter.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox3", "Footer", "1.61333in", "0.05416in", "2.2in", "0.34167in");
pageFooter.ReportItems.Add(textBox);
...
...
...
}
```

Body CreateBody()

```
{
...
...
...
body.ReportItems = new ReportItems();
var textBox = CreateTextBox("TextBox1", "Body", "1.58833in", "0.66333in", "2.225in", "0.59167in");
body.ReportItems.Add(textBox);
...
...
...
}

BoldReports.RDL.DOM.TextBox CreateTextBox(string name, string text, string left, string top, string
width, string height)
{
var textBox = new BoldReports.RDL.DOM.TextBox();
textBox.Name = name;
textBox.Height = new BoldReports.RDL.DOM.Size(height);
textBox.Width = new BoldReports.RDL.DOM.Size(width);
textBox.Left = new BoldReports.RDL.DOM.Size(left);
textBox.Top = new BoldReports.RDL.DOM.Size(top);
textBox.Style = new BoldReports.RDL.DOM.Style();
textBox.Style.TextAlign = "Center";
textBox.Style.VerticalAlign = "Top";
textBox.Paragraphs = new Paragraphs();
BoldReports.RDL.DOM.Paragraph paragraph = new BoldReports.RDL.DOM.Paragraph();
TextRuns runs = new TextRuns();
TextRun run = new TextRun();
run.Style = new BoldReports.RDL.DOM.Style();
run.Style.FontStyle = "Default";
run.Style.TextAlign = "Center";
run.Style.FontFamily = "Arial";
run.Style.FontSize = new BoldReports.RDL.DOM.Size("10pt");
run.Value = text;
```

```

runs.Add(run);
paragraph.Style = new BoldReports.RDL.DOM.Style();
paragraph.Style.VerticalAlign = "Top";
paragraph.Style.TextAlign = "Center";
paragraph.TextRuns = runs;
textBox.Paragraphs.Add(paragraph);
return textBox;
}
`

```

Create a Table for the report

- Using the following code snippets, you can create a **Table** report item and assign the values for their properties like name, styles, paragraphs, and dimension values.

```
`csharp
```

```
BoldReports.RDL.DOM.Tablix CreateTablix(string name, string text, string left, string top, string width,
string height)
```

```

{
var tableItem = new BoldReports.RDL.DOM.Tablix();
tableItem.Name = name;
tableItem.Height = new BoldReports.RDL.DOM.Size(height);
tableItem.Width = new BoldReports.RDL.DOM.Size(width);
tableItem.Left = new BoldReports.RDL.DOM.Size(left);
tableItem.Top = new BoldReports.RDL.DOM.Size(top);
tableItem.Style = new BoldReports.RDL.DOM.Style();
tableItem.Style.Border = new BoldReports.RDL.DOM.Border();
tableItem.Style.Border.Style = "Solid";
tableItem.DataSetName = "DataSet1";
tableItem.TablixBody = new TablixBody();
tableItem.TablixBody.TablixColumns = new TablixColumns();
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
tableItem.TablixBody.TablixColumns.Add(CreateTablixColumn());
tableItem.TablixBody.TablixRows = new TablixRows();
var rowOne = new List<TablixRowValues>();

```

```

rowOne.Add(new TablixRowValues { TextBoxName = "TextBox1", TextBoxValue = "Department ID" });
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox3", TextBoxValue = "Name" });
rowOne.Add(new TablixRowValues { TextBoxName = "Textbox5", TextBoxValue = "Group Name" });
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowOne));
var rowTwo = new List<TablixRowValues>();
rowTwo.Add(new TablixRowValues { TextBoxName = "DepartmentID", TextBoxValue =
    "=Fields!DepartmentID.Value" });
rowTwo.Add(new TablixRowValues { TextBoxName = "Name", TextBoxValue = "=Fields!Name.Value" });
rowTwo.Add(new TablixRowValues { TextBoxName = "GroupName", TextBoxValue =
    "=Fields!GroupName.Value" });
tableItem.TablixBody.TablixRows.Add(CreateTablixRow(rowTwo));
tableItem.TablixColumnHierarchy = new TablixColumnHierarchy();
tableItem.TablixColumnHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixColumnHierarchy.TablixMembers.Add(new TablixMember());
tableItem.TablixRowHierarchy = new TablixRowHierarchy();
tableItem.TablixRowHierarchy.TablixMembers = new TablixMembers();
tableItem.TablixRowHierarchy.TablixMembers.Add(new TablixMember() { KeepWithGroup =
    KeepWithGroup.After });
tableItem.TablixRowHierarchy.TablixMembers.Add(CreateTablixMember("Details"));
return tableItem;
}

BoldReports.RDL.DOM.TablixColumn CreateTablixColumn()
{
    var tablixColumn = new TablixColumn();
    tablixColumn.Width = "1in";
    return tablixColumn;
}

BoldReports.RDL.DOM.TablixRow CreateTablixRow(List<TablixRowValues> values)
{
    var tablixRow = new TablixRow();
    tablixRow.Height = "0.25in";
    tablixRow.TablixCells = new TablixCells();

```

```

for(int i = 0; i < values.Count; i++)
{
    tablixRow.TablixCells.Add(CreateTablixCell(values[i].TextBoxName, values[i].TextBoxValue));
}
return tablixRow;
}

BoldReports.RDL.DOM.TablixCell CreateTablixCell(string textBoxName, string textBoxValue)
{
    var tablixCell = new TablixCell();
    tablixCell.CellContents = new CellContents();
    tablixCell.CellContents.ReportItem = CreateTextBox(textBoxName, textBoxValue);
    return tablixCell;
}

BoldReports.RDL.DOM.TablixMember CreateTablixMember(string groupName)
{
    var tablixMember = new TablixMember();
    tablixMember.Group = new Group();
    tablixMember.Group.Name = groupName;
    return tablixMember;
}

internal class TablixRowValues
{
    public string TextBoxName { get; set; }
    public string TextBoxValue { get; set; }
}

```

Create a Chart for the report

- Using the following code snippets, you can create a **Chart** report item and assign the values for their properties like name, styles, and dimension values.

```

`csharp
BoldReports.RDL.DOM.Chart CreateChart(string name, string left, string top, string width, string height)
{
    var chartItem = new Chart();

```

```
chartItem.Name = name;
chartItem.Height = new BoldReports.RDL.DOM.Size("10in");
chartItem.Width = new BoldReports.RDL.DOM.Size("10in");
chartItem.Left = new BoldReports.RDL.DOM.Size("5in");
chartItem.Top = new BoldReports.RDL.DOM.Size("5in");
chartItem.DataSetName = "DataSet1";
chartItem.Style = new BoldReports.RDL.DOM.Style();
chartItem.Bookmark= "=First(Fields!Name.Value, \"DataSet1\")";
chartItem.ChartCategoryHierarchy = new ChartCategoryHierarchy();
chartItem.ChartCategoryHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartCategoryHierarchy.ChartMembers.Add(ChartCategoryMember());
chartItem.ChartSeriesHierarchy = new ChartSeriesHierarchy();
chartItem.ChartSeriesHierarchy.ChartMembers = new ChartMembers();
chartItem.ChartSeriesHierarchy.ChartMembers.Add(ChartSeriesMember());
chartItem.ChartData = new ChartData();
chartItem.ChartData.ChartDerivedSeriesCollection = new ChartDerivedSeriesCollection();
chartItem.ChartData.ChartSeriesCollection = CreateChartSeriesCollection();
chartItem.ChartLegends = new ChartLegends();
chartItem.ChartAreas = new ChartAreas();
chartItem.ChartAreas.Add(CreateChartArea());
chartItem.ChartTitles = new ChartTitles();
chartItem.Palette = "Pacific";
chartItem.ChartBorderSkin = new ChartBorderSkin();
chartItem.ChartNoDataMessage = new ChartNoDataMessage();
chartItem.ChartNoDataMessage.Caption = "No Data Available";
chartItem.ChartNoDataMessage.Name = "NoDataMessage";
return chartItem;
}

BoldReports.RDL.DOM.ChartMember ChartCategoryMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
```

```
ChartMember.DataElementName = null;
ChartMember.DataElementOutput = DataElementOutputs.Auto;
ChartMember.Group = new Group();
ChartMember.Group = CreateChartGroup();
ChartMember.Label="=Fields!DepartmentID.Value";
ChartMember.SortExpressions = SortExpressions();
return ChartMember;
}

BoldReports.RDL.DOM.ChartMember ChartSeriesMember()
{
    var ChartMember = new ChartMember();
    ChartMember.ChartMembers = new ChartMembers();
    ChartMember.CustomProperties = new CustomProperties();
    ChartMember.DataElementName = null;
    ChartMember.DataElementOutput = DataElementOutputs.Auto;
    ChartMember.Group = null;
    ChartMember.Label = "Department ID";
    ChartMember.SortExpressions = new SortExpressions();
    return ChartMember;
}

BoldReports.RDL.DOM.SortExpressions SortExpressions()
{
    var SortExpressions = new SortExpressions();
    var SortExpression = new SortExpression();
    SortExpression.Direction = SortDirection.Ascending;
    SortExpression.Value = "=Fields!DepartmentID.Value";
    SortExpressions.Add(SortExpression);
    return SortExpressions;
}

BoldReports.RDL.DOM.Group CreateChartGroup()
{
    var ChartGroup = new Group();
    ChartGroup.DataElementName = null;
```



```
ChartGroup.DataElementOutput = DataElementOutputs.Auto;
ChartGroup.DocumentMapLabel = null;
ChartGroup.DocumentMapLabelLocID = null;
ChartGroup.DomainScope = null;
ChartGroup.Filters = new Filters();
ChartGroup.GroupExpressions = CreateGroupExpressions();
ChartGroup.Name = "Chart2_CategoryGroup";
ChartGroup.PageBreak = null;
ChartGroup.PageName = null;
ChartGroup.Parent = null;
ChartGroup.Variables = new Variables();
return ChartGroup;
}

BoldReports.RDL.DOM.GroupExpressions CreateGroupExpressions()
{
    var GroupExpressions = new GroupExpressions();
    var GroupExpression = new GroupExpression();
    GroupExpression.Value = "=Fields!DepartmentID.Value";
    GroupExpressions.Add(GroupExpression);
    return GroupExpressions;
}

BoldReports.RDL.DOM.ChartSeriesCollection CreateChartSeriesCollection()
{
    var ChartSeriesCollection = new ChartSeriesCollection();
    ChartSeriesCollection.Add(CreateChartSeries());
    return ChartSeriesCollection;
}

BoldReports.RDL.DOM.ChartSeries CreateChartSeries()
{
    var ChartSeries = new ChartSeries();
    ChartSeries.CategoryAxisName = "Primary";
    ChartSeries.ChartAreaName = null;
    ChartSeries.ChartDataLabel = null;
```

```
ChartSeries.ChartDataPoints = new ChartDataPoints();
ChartSeries.ChartDataPoints.Add(ChartDataPoint());
ChartSeries.ChartEmptyPoints = new ChartEmptyPoints();
ChartSeries.ChartEmptyPoints.ChartDataLabel = new ChartDataLabel();
ChartSeries.ChartEmptyPoints.ChartDataLabel.Style = new Style();
ChartSeries.ChartEmptyPoints.ChartMarker = new ChartMarker();
ChartSeries.ChartEmptyPoints.ChartMarker.Style = new Style();
ChartSeries.ChartSmartLabel = new ChartSmartLabel();
ChartSeries.Name = "DepartmentID";
ChartSeries.Type = VisualizationType.Column;
ChartSeries.Subtype = VisualizationSubType.Plain;
ChartSeries.Style = new Style();
return ChartSeries;
}

BoldReports.RDL.DOM.ChartDataPoint ChartDataPoint()
{
    var ChartDataPoint = new ChartDataPoint();
    ChartDataPoint.ActionInfo = null;
    ChartDataPoint.ChartDataLabel = null;
    ChartDataPoint.ChartDataLabel = ChartDataLabel();
    ChartDataPoint.ChartDataPointValues = new ChartDataPointValues();
    ChartDataPoint.ChartDataPointValues.Y = "=Sum(Fields!DepartmentID.Value)";
    ChartDataPoint.ChartItemInLegend = null;
    ChartDataPoint.ChartMarker = new ChartMarker();
    ChartDataPoint.ChartMarker.Size = null;
    ChartDataPoint.ChartMarker.Type = null;
    ChartDataPoint.ChartMarker.Style= chartStyle("Transparent", "Arial");
    ChartDataPoint.CustomProperties = new CustomProperties();
    ChartDataPoint.DataElementName = null;
    ChartDataPoint.DataElementOutput = DataElementOutputs.Output;
    ChartDataPoint.ToolTip = null;
    ChartDataPoint.Style= chartStyle("Transparent", "Arial");
    return ChartDataPoint;
}
```

```
}  
BoldReports.RDL.DOM.ChartDataLabel ChartDataLabel()  
{  
    var ChartDataLabel = new ChartDataLabel();  
    ChartDataLabel.ActionInfo = null;  
    ChartDataLabel.Label = null;  
    ChartDataLabel.Position = null;  
    ChartDataLabel.Rotation = "0";  
    ChartDataLabel.Style = new Style();  
    ChartDataLabel.Style = chartStyle("Transparent", "Arial");  
    ChartDataLabel.ToolTip = null;  
    return ChartDataLabel;  
}  
BoldReports.RDL.DOM.Style chartStyle(string BackgroundColor, string FontFamily)  
{  
    var style = new Style();  
    style.BackgroundColor = BackgroundColor;  
    style.FontFamily = FontFamily;  
    return style;  
}  
BoldReports.RDL.DOM.ChartArea CreateChartArea()  
{  
    var chartArea = new ChartArea();  
    chartArea.Style = new Style();  
    chartArea.Style = chartStyle("Transparent", "Arial");  
    chartArea.AlignOrientation = AlignOrientation.None;  
    chartArea.AlignWithChartArea = null;  
    chartArea.ChartAlignType = null;  
    chartArea.ChartElementPosition = null;  
    chartArea.ChartInnerPlotPosition = null;  
    chartArea.ChartThreeDProperties = null;  
    chartArea.Hidden = false;  
    chartArea.EquallySizedAxesFont = false;
```

```

chartArea.Name = "Default";
chartArea.ChartCategoryAxes = new ChartCategoryAxes();
chartArea.ChartCategoryAxes.Add(createChartAxis("Primary"));
chartArea.ChartCategoryAxes.Add(createChartAxis("Secondary"));
chartArea.ChartValueAxes = new ChartValueAxes();
chartArea.ChartValueAxes.Add(createChartAxis("Primary"));
chartArea.ChartValueAxes.Add(createChartAxis("Secondary"));
return chartArea;
}

BoldReports.RDL.DOM.ChartAxis createChartAxis(string Name)
{
var ChartAxis = new ChartAxis();
ChartAxis.Style = new Style();
ChartAxis.AllowLabelRotation = AllowLabelRotation.None;
ChartAxis.Angle = "0";
ChartAxis.Arrows = Arrows.None;
ChartAxis.ChartAxisScaleBreak = new ChartAxisScaleBreak();
ChartAxis.ChartAxisTitle = new ChartAxisTitle();
ChartAxis.ChartMajorGridLines = new ChartMajorGridLines();
ChartAxis.ChartMajorTickMarks = new ChartMajorTickMarks();
ChartAxis.ChartMinorGridLines = new ChartMinorGridLines();
ChartAxis.ChartMinorTickMarks = new ChartMinorTickMarks();
ChartAxis.ChartStripLines = new ChartStripLines();
ChartAxis.CrossAt = "NaN";
ChartAxis.CustomProperties = new CustomProperties();
ChartAxis.LineStyle = LineStyle.Solid;
ChartAxis.Name = Name;
return ChartAxis;
}

```

[Export the generated report in ReportWriter](#)

You can export the report using the ReportWriter, after the report definition or stream is created. The following code snippet illustrates how to export the report into PDF format using the Report Writer by report definition.

```
`csharp
ReportWriter reportWriter = new ReportWriter();
reportWriter.LoadReport(reportDefinition);
reportWriter.Save(fileName, WriterFormat.PDF);
`
```

How to load the report from the database

You must load the report using the **Stream** option available with **writer.LoadReport()** method. To load the report from the database, please follow these steps:

From byte array

```
`csharp
byte[] bytes = ""; // provide your byte array report data
MemoryStream reportStream = new MemoryStream(bytes);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadReport(reportStream);
`
```

From base64String

```
`csharp
string base64String = ""; // provide your base64 report data
byte[] bytes = System.Convert.FromBase64String(base64String);
MemoryStream reportStream = new MemoryStream(bytes);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadReport(reportStream);
`
```

From string

```
`csharp
string reportData = ""; // provide your Report data
byte[] bytes = System.Text.Encoding.ASCII.GetBytes(reportData);
MemoryStream reportStream = new MemoryStream(bytes);
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
writer.LoadReport(reportStream);
`
```

Reporting tools for WPF

Enterprise-class reporting tools for WPF development to embed reporting functionalities such as viewing, exporting, and printing reports in your WPF applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

Reporting tools for WPF

Enterprise-class reporting tools for WPF development to embed reporting functionalities such as viewing, exporting, and printing reports in your WPF applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

System Requirements

This topic describes the software and hardware requirements for setting up the development environment of Bold Reports WPF.

Supported Operating Systems

- Windows 7+, 8+
- Windows Server 2008 R2+

Hardware Environments

The following hardware environments are necessary for setting up the Bold Reports WPF.

- 1 GHZ or faster, 32 bit or 64 bit processor.
- 1 GB RAM for 32 bit or 2 GB RAM for 64 bit.

Development Environments

The following development environment are necessary to run the Bold Reports WPF.

- [Microsoft Visual Studio 2010](#) or [later](#)

Framework

The below tool is required for Bold Reports WPF.

.NET Framework 4.0 or higher

See Also

- [Licensing procedure for deployment](#)

Overview

The Report Viewer is a visualization control used to display SSRS, RDL, RDLC, and Bold Report Server reports within web applications. It allows you to view RDL/RDLC reports with or without using SSRS or Bold Report Server. You can bind data sources, parameters, and render reports with all major capabilities of RDL reporting and export the report to PDF, Microsoft Excel, CSV, Microsoft Word, and HTML formats. Some of the key features are,

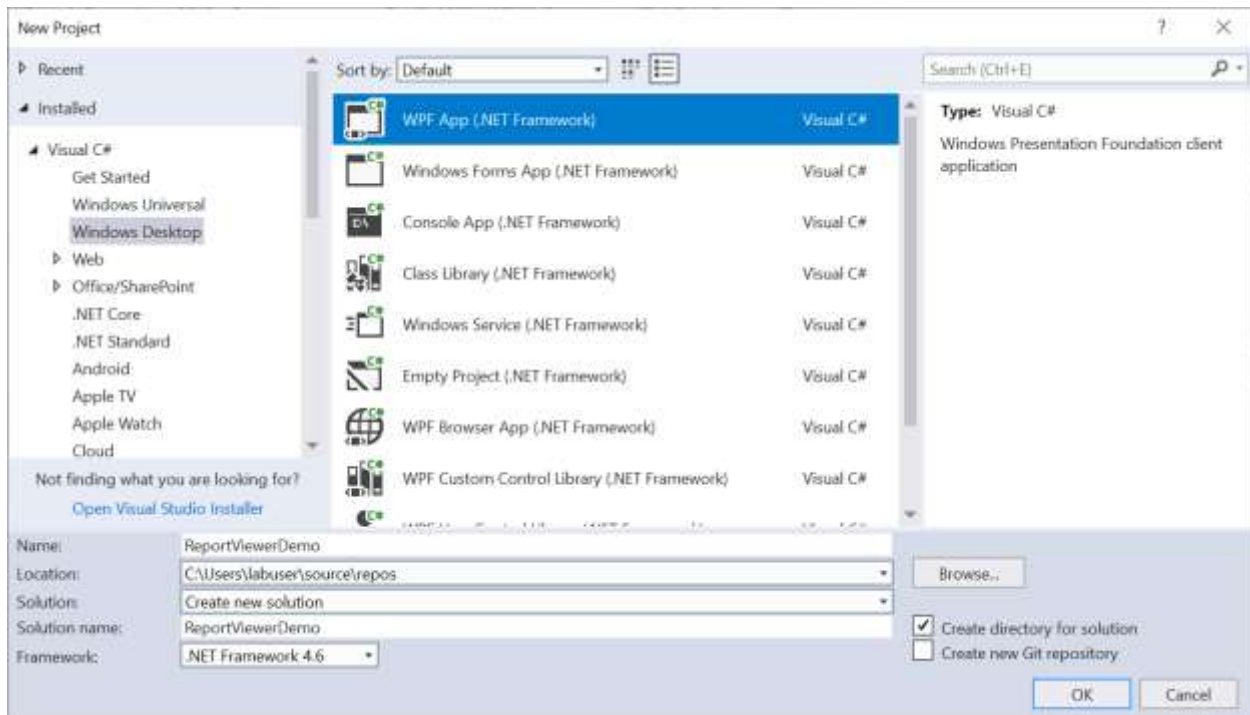
- Renders interactive reports with drill down, drill through, hyperlinks, and interactive sorting.
- Easily customize each element of Report Viewer and provide events for report processing customization.
- Supports jQuery, Angular, React, Blazor, ASP.NET Core, ASP.NET MVC, ASP.NET WebForms, WPF, and UWP.

Display SSRS RDL report in Bold Reports WPF Report Viewer

This section explains you the steps required to create your first WPF reporting application to display an already created SSRS RDL report in the Bold Reports WPF Report Viewer without using a Report Server.

Create the application project

1. Open Visual Studio 2017 and select **File > New > Project**.
2. Go to **Installed > Visual C# > Windows Desktop**, and then select the required **.NET Framework** in the dropdown.
3. Select the **WPF App (.NET Framework)**, change the application name, and then click **OK**.



Configure Report Viewer in an application

1. Right-click the project or solution on the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select **Tools > NuGet Package Manager > Manage NuGet Packages for Solution**.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring Report Viewer NuGet packages.

2. Search for **BoldReports.Wpf** NuGet package, and install them in your WPF application.

Package | Purpose

BoldReports.Wpf | Contains WPF Reporting controls (Report Viewer and Report Writer) to preview and export the reports.

Initialize Report Viewer

1. Import the Report Viewer namespace as shown below in the **MainWindow.xaml** file,

```
`csharp
xmlns:syncfusion="clr-namespace:BoldReports.UI.Xaml;assembly=BoldReports.Wpf"
`
```

2. Initialize the Report Viewer component inside the **Page** tag as shown below in the **MainWindow.xaml** file,


```

`csharp
<Window
.....
.....
.....
.....
xmlns:syncfusion="clr-namespace:BoldReports.UI.Xaml;assembly=BoldReports.Wpf"
.....
.....>
<Grid>
<syncfusion:ReportViewer Name="reportViewer" />
</Grid>
</Window>
`

```

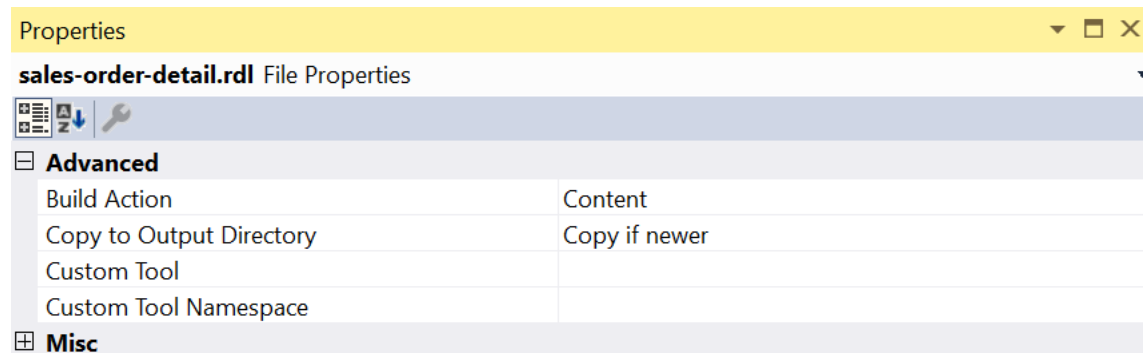
Add already created reports

The Report Viewer is only for rendering reports. You must use a report generation tool to create a report. To learn more about this, refer to the [create RDL report](#) section.

1. Create a folder **Resources** in your application to store the RDL reports.
2. Add already created reports to the newly created folder.

In this tutorial, the **sales-order-detail.rdl** report is used, and it can be downloaded at this [link](#). You can add the reports from the Syncfusion installation location. For more information, refer to the [samples and demos](#) section.

3. Set the **Build Action** to **content** and **Copy to Output Directory** to either **Copy always** or **Copy if newer**.



Set report path

1. Open the `MainWindow.xaml.cs` file.
2. Initialize the window loaded event inside the `MainWindow()` constructor.

```
`csharp
public MainWindow()
{
    InitializeComponent();
    this.Loaded += new RoutedEventHandler(MainWindow_Loaded);
}
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
}
`
```

3. Set the `ReportPath` property in the `MainWindow_Loaded` event and invoke the `RefreshReport()` method to render the report.
4. You can replace the following code in your `MainWindow_Loaded` event method.

```
`csharp
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\sales-order-detail.rdl");
    this.reportViewer.RefreshReport();
}
`
```

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

Sales Order No. 5050750 View Report

Adventure Works **Sales Order**

Order ID: #5050750 Billing Date: 21-05-2019 Order Date: 01-06-2008 Purchase Order: PO7192170677 Shipment Method: CARGO TRANSPORT 5

Billing Address:
Jean Handley
 Central Discount Store
 259826 Russell Rd. South
 Kent, Washington
 98031
 United States

Shipping Address:
Jean Handley
 Central Discount Store
 259826 Russell Rd. South
 Kent, Washington
 98031
 United States

Contact:
Jean Handley
 Ph: 582-555-0113

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	WAC Logo Cap	373D-417C-AE	\$3.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LI-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.84	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	BK-M688-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	BK-MM85-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

Note: You can refer to our feature tour page for the [WPF Report Viewer](#) to see its innovative features.

See Also

[Create RDLC report](#)

[Migrate Report Viewer](#)

[List of SSRS server versions are supported in Bold Reports](#)

Display SSRS RDL report in WPF .NET Core app using Report Viewer

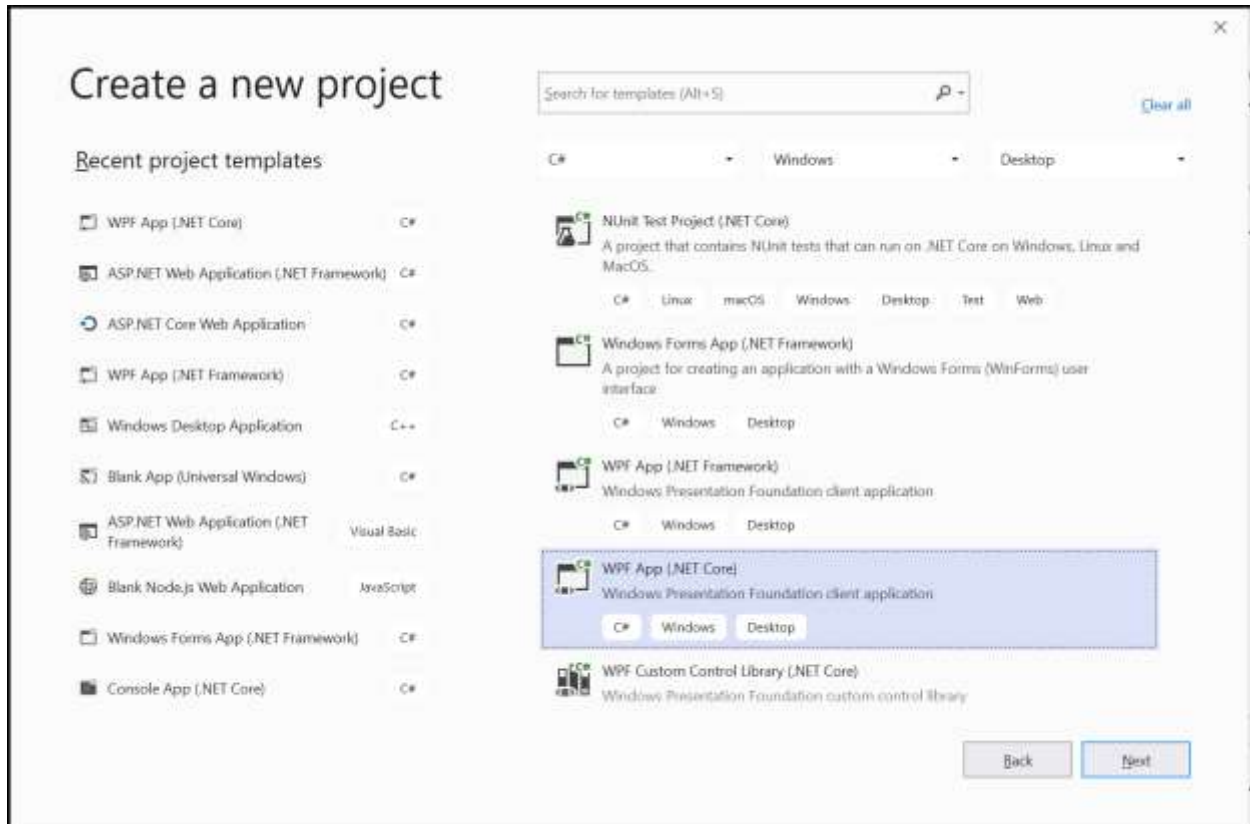
Create your first WPF .NET Core reporting application to display an already created SSRS RDL report in the WPF Report Viewer without using a Report Server, using this step-by-step instruction.

To get start quickly with Report Viewer, you can check on this video:

youtube: <https://youtu.be/KNuiMyIL7VQ>

[Create the application project](#)

1. Start Visual Studio 2019 and click **Create new project**.
2. Select **WPF App (.NET Core)**, and then choose **Next**.



3. Change the project name, and then click **Create**.

Configure Report Viewer in an application

1. Right-click the project or solution on the Solution Explorer tab and choose **Manage NuGet Packages**. Alternatively, select **Tools > NuGet Package Manager > Manage NuGet Packages for Solution**.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring Report Viewer NuGet packages.

2. Search for **BoldReports.Wpf** NuGet package and install them in your WPF application.

Package | Purpose

BoldReports.Wpf | Contains WPF Reporting controls (Report Viewer and Report Writer) to preview and export the reports.

Initialize Report Viewer

1. Import the Report Viewer namespace as shown below in the **MainWindow.xaml** file,

```
`csharp
```

```
xmlns:syncfusion="clr-namespace:BoldReports.UI.Xaml;assembly=BoldReports.Wpf"
`
```

2. Initialize the Report Viewer component inside the `tag` as shown below in the `MainWindow.xaml` file,

```
`csharp
<Window
.....
.....
.....
.....
xmlns:syncfusion="clr-namespace:BoldReports.UI.Xaml;assembly=BoldReports.Wpf"
.....
.....>
<Grid>
<syncfusion:ReportViewer Name="reportViewer" />
</Grid>
</Window>
`
```

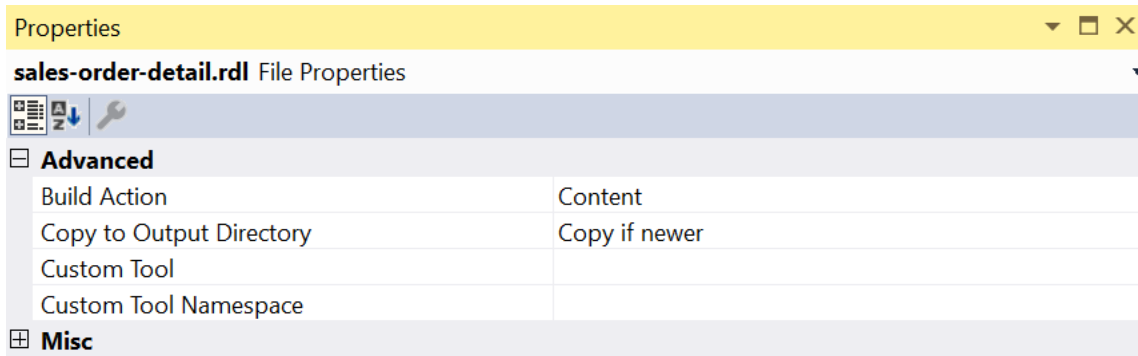
Add already created reports

The Report Viewer is only for rendering reports. You must use a report generation tool to create a report. To learn more about this, refer to the [create RDL report](#) section.

1. Create a folder `Resources` in your application to store the RDL reports.
2. Add already created reports to the newly created folder.

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded at this [link](#). You can add the reports from the Syncfusion installation location. For more information, refer to the [samples and demos](#) section.

3. Set the `Build Action` to **content** and `Copy to Output Directory` to either **Copy always** or **Copy if newer**.



Set report path

1. Open the `MainWindow.xaml.cs` file.
2. Initialize the window loaded event inside the `MainWindow()` constructor.

```
`csharp
public MainWindow()
{
    InitializeComponent();
    this.Loaded += new RoutedEventHandler(MainWindow_Loaded);
}

private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
}
`
```

3. Set the `ReportPath` property in the `MainWindow_Loaded` event and invoke the `RefreshReport()` method to render the report.
4. You can replace the following code in your `MainWindow_Loaded` event method.

```
`csharp
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
        @"Resources\sales-order-detail.rdl");
    this.reportViewer.RefreshReport();
}
`
```

[Preview the report](#)

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

The screenshot shows a web application window titled 'Main Window' with a 'View Report' button. The report is a sales order for 'Adventure Sports Cycles' with Sales Order No. 5050750. It includes fields for Order ID, Billing Date, Order Date, Purchase Order, and Shipment Method. It also lists Billing and Shipping addresses for Jean Handley at Central Discount Store. A table of items is shown with columns for Line, Qty, Item Number, Description, Tracking No., Unit Price, and Item Total.

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	WAC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LI-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.84	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	BK-M688-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	BK-M685-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

[See Also](#)

[Create RDLC report](#)

[Migrate Report Viewer](#)

[List of SSRS server versions are supported in Bold Reports](#)

Load SSRS rdl reports

Report Viewer has support to load RDL reports from SSRS Report Server. To render SSRS Reports set the `ReportServerUrl` and `ReportServerCredential` properties as in the following code snippet.

```
`csharp
this.reportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
this.reportViewer.ReportServerCredential = new System.Net.NetworkCredential("ssrs", "RDLReport1");
this.reportViewer.ReportPath = "/SSRSSamples/Territory Sales"; //The report path should be in the
format of "/folder name/report name"
this.reportViewer.RefreshReport();
`
```

Set data source credential for shared data sources

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS server. If the report has any data source that uses credentials to connect with the database, then you must specify the `DataSourceCredentials` for each report data source to establish database connection.

```

`csharp
this.reportViewer.ReportLoaded += (sen, arg) =>
{
    List<BoldReports.Windows.DataSourceCredentials> dataSourceCrdentials = new
    List<BoldReports.Windows.DataSourceCredentials>();
    foreach (var dataSource in this.reportViewer.GetDataSources())
    {
        BoldReports.Windows.DataSourceCredentials crdentials = new
        BoldReports.Windows.DataSourceCredentials();
        crdentials.Name = dataSource.Name;
        crdentials.UserId = "ssrs1";
        crdentials.Password = "RDLReport1";
        dataSourceCrdentials.Add(crdentials);
    }
    this.reportViewer.SetDataSourceCredentials(dataSourceCrdentials);
};
this.reportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
this.reportViewer.ReportServerCredential = new System.Net.NetworkCredential("ssrs", "RDLReport1");
this.reportViewer.ReportPath = "/SSRSSamples/Territory Sales"; //The report path should be in the
format of "/folder name/report name"
this.reportViewer.RefreshReport();
`

```

Data source credentials must be added for shared data sources that do not have credentials in the connection strings.

Render linked reports

You can render a linked report that point to an existing report, which is published in the SSRS Report Server. Also, it is possible to set the parameter, data source, credential, and other properties as like normal SSRS reports using the Report Viewer.

```

`csharp
this.reportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
this.reportViewer.ReportServerCredential = new System.Net.NetworkCredential("ssrs", "RDLReport1");
this.reportViewer.ReportPath = "/SSRSSamples/Territory Sales_Link"; //The report path should be in the
format of "/folder name/report name"
this.reportViewer.RefreshReport();
`

```


The **Territory Sales_Link** is a linked report created for **Territory Sales** report that is already available on the SSRS Report Server.

Load SharePoint integrated reports

To render SharePoint integrated SSRS reports set the **ReportPath**, **ReportServerUrl** and **ReportServerFormsCredential** properties as in the following code snippet.

```
`csharp
this.reportViewer.ReportPath =
"http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl";
this.reportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
this.reportViewer.ReportServerFormsCredential = new
BoldReports.Windows.ReportServerFormsCredential("ssrs", "RDLReport1");
this.reportViewer.RefreshReport();
`
```

Set data source credential for shared data sources

The shared data source credentials can be added to the **DataSourceCredentials** property to connect with the database.

```
`csharp
this.reportViewer.ReportLoaded += (sen, arg) =>
{
    List<BoldReports.Windows.DataSourceCredentials> dataSourceCrdentials = new
    List<BoldReports.Windows.DataSourceCredentials>();
    foreach (var dataSource in this.reportViewer.GetDataSources())
    {
        BoldReports.Windows.DataSourceCredentials crdentials = new
        BoldReports.Windows.DataSourceCredentials();
        crdentials.Name = dataSource.Name;
        crdentials.UserId = "ssrs1";
        crdentials.Password = "RDLReport1";
        dataSourceCrdentials.Add(crdentials);
    }
    this.reportViewer.SetDataSourceCredentials(dataSourceCrdentials);
};
this.reportViewer.ReportPath =
"http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl";
this.reportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
`
```

```

this.reportViewer.ReportServerFormsCredential = new
BoldReports.Windows.ReportServerFormsCredential("ssrs", "RDLReport1");
this.reportViewer.RefreshReport();
`

```

Data source credentials must be added for shared data sources that do not have credentials in the connection strings.

Render RDLC report

The data binding support, allows you to view RDLC reports that exist on the custom business object data collection. The following steps demonstrates how to render a RDLC report with custom business object data collection.

Add the RDLC report `product-list.rdlc` from Bold Reports installation location to your application `Resources` folder. For more information, see [Samples and demos](#).

- Set the `ReportPath` and `ProcessingMode` to `ProcessingMode.Local` to the RDLC report location.
- Bind the business object data values collection by adding new item to the `DataSources` as in the following code snippet.

```

`csharp
this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\product-list.rdlc");
this.reportViewer.ProcessingMode = BoldReports.UI.Xaml.ProcessingMode.Local;
this.reportViewer.DataSources.Clear();
this.reportViewer.DataSources.Add(new BoldReports.Windows.ReportDataSource { Name = "list", Value
= ProductList.GetData() });
this.reportViewer.RefreshReport();
.....
public class ProductList
{
public string ProductName { get; set; }
public string OrderId { get; set; }
public double Price { get; set; }
public string Category { get; set; }
public string Ingredients { get; set; }
public string ProductImage { get; set; }
public static IList GetData()
{

```

```
List<ProductList> datas = new List<ProductList>();
ProductList data = null;
data = new ProductList()
{
    ProductName = "Baked Chicken and Cheese",
    OrderId = "323B60",
    Price = 55,
    Category = "Non-Veg",
    Ingredients = "grilled chicken, corn and olives.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Delite",
    OrderId = "323B61",
    Price = 100,
    Category = "Non-Veg",
    Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
```

```

}
}
`

```

Load report as stream

To load report as a stream, create a report stream using the `FileStream` class and assign the report stream to the `Stream` property.

```

`csharp
FileStream reportStream = new FileStream(System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\product-list.rdlc"), FileMode.Open, FileAccess.Read);
this.reportViewer.ProcessingMode = BoldReports.UI.Xaml.ProcessingMode.Local;
this.reportViewer.LoadReport(reportStream);
this.reportViewer.DataSources.Clear();
this.reportViewer.DataSources.Add(new BoldReports.Windows.ReportDataSource { Name = "list", Value
= ProductList.GetData() });
this.reportViewer.RefreshReport();
.....
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = null;
        data = new ProductList()
        {
            ProductName = "Baked Chicken and Cheese",
            OrderId = "323B60",
            Price = 55,
            Category = "Non-Veg",

```

```

Ingredients = "grilled chicken, corn and olives.",
ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Delite",
    OrderId = "323B61",
    Price = 100,
    Category = "Non-Veg",
    Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
    ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
,

```

Render subreport

You can display another report inside the body of a main report using the Report Viewer. The following steps help you to customize the subreport properties such as data source, report path, and parameters.

- Add the sub report and main reports to your application **Resources** folder. In this tutorial, using the already created reports. Refer to the [Create RDL Report section](#) or [Create RDLC Report section](#) for creating new reports.

Download the **side-by-side-main-report.rdl**, **side-by-side-sub-report.rdl** reports from [here](#). Also, you can add the report from Syncfusion installation location. For more information, see [Samples and demos](#). The reports used from installed location, requires **NorthwindIO_Reports.sdf** database to run, so add it to your application.

- Set the main report path **ReportPath** properties of the Report Viewer as in following code snippet.

```
`csharp
this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\side-by-side-main-report.rdl");
this.reportViewer.RefreshReport();
`
```

Load subreport stream

To load subreport as stream, set the sub report stream in **LoadSubreport** method of the Report Viewer as in following code snippet.

```
`csharp
FileStream subReportStream = new FileStream(System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\product-list.rdlc"), FileMode.Open, FileAccess.Read);

FileStream mainReportStream = new
FileStream(System.IO.Path.Combine(Environment.CurrentDirectory, @"Resources\product-list-
main.rdlc"), FileMode.Open, FileAccess.Read);

this.reportViewer.SubreportProcessing += (sen, arg) =>
{
    arg.DataSources.Clear();
    arg.DataSources.Add(new BoldReports.Windows.ReportDataSource { Name = "list", Value =
    ProductList.GetData() });
};

this.reportViewer.LoadSubreport("product-list", subReportStream);
this.reportViewer.LoadReport(mainReportStream);
this.reportViewer.RefreshReport();

.....

public class ProductList
{
```

```
public string ProductName { get; set; }
public string OrderId { get; set; }
public double Price { get; set; }
public string Category { get; set; }
public string Ingredients { get; set; }
public string ProductImage { get; set; }
public static IList GetData()
{
    List<ProductList> datas = new List<ProductList>();
    ProductList data = null;
    data = new ProductList()
    {
        ProductName = "Baked Chicken and Cheese",
        OrderId = "323B60",
        Price = 55,
        Category = "Non-Veg",
        Ingredients = "grilled chicken, corn and olives.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Delite",
        OrderId = "323B61",
        Price = 100,
        Category = "Non-Veg",
        Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Tikka",
```

```

OrderId = "323B62",
Price = 64,
Category = "Non-Veg",
Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
ProductImage = ""
};
datas.Add(data);
return datas;
}
}
`

```

Report parameters

Provides property options to pass or set report parameters default values at run-time using the parameters property. You can set the report parameters while creating the Report Viewer control in a application loaded.

In this tutorial, the `sales-order-dtail.rdl` report is used, and it can be downloaded from [here](#).

Set report parameter

1. Set the default value data to the `Values` property and name of the report parameter to the `Name` property.

The parameter name is case sensitive, it should be same as available in the report definition.

2. The following code example illustrates how to set report parameter in the script.

```

`csharp
this.reportViewer.ReportLoaded += (sen, arg) =>
{
    List<BoldReports.Windows.ReportParameter> parameters = new
    List<BoldReports.Windows.ReportParameter>();
    BoldReports.Windows.ReportParameter parameter = new BoldReports.Windows.ReportParameter();
    parameter.Name = "SalesOrderNumber";
    parameter.Values = new List<string>() { "SO50756" };
    parameters.Add(parameter);
    this.reportViewer.SetParameters(parameters);
};

```



```

this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\sales-order-detail.rdl");
this.reportViewer.RefreshReport();
`

```

Get report parameter

Methods | Description

GetParameters | Returns the parameters that are used in the current report without the processed values.

You can use the following code sample to get parameter names and set parameter default values.

```

`csharp
this.reportViewer.ReportLoaded += (sen, arg) =>
{
    var reportParameters = this.reportViewer.GetParameters();
    List<BoldReports.Windows.ReportParameter> parameters = new
    List<BoldReports.Windows.ReportParameter>();
    if (reportParameters != null)
    {
        foreach (var parameter in reportParameters)
        {
            parameters.Add(new BoldReports.Windows.ReportParameter()
            {
                Name = parameter.Name,
                Values = new List<string>() { "SO50756" }
            });
        }
    }
    this.reportViewer.SetParameters(parameters);
};

this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\sales-order-detail.rdl");
this.reportViewer.RefreshReport();
`

```

Report interaction events

You can handle the report processing actions and interact with reports using the following events.

- ReportLoaded
- ReportError
- Drill through
- Hyperlink

Report loaded

The **ReportLoaded** event fires once the report loading is completed and ready to start the processing. You can handle the event and specify data source, parameters at client-side. The following sample code loads a report by assigning report data source input in the **ReportLoaded** event.

In this tutorial, **product-list.rdlc** report is used, you can add the report from Bold Reports installation location. For more information, see [Samples and demos](#).

```
`csharp
this.reportViewer.ReportLoaded += (sen, arg) =>
{
    this.reportViewer.DataSources.Clear();
    this.reportViewer.DataSources.Add(new BoldReports.Windows.ReportDataSource { Name = "list", Value
    = ProductList.GetData() });
};
this.reportViewer.ProcessingMode = BoldReports.UI.Xaml.ProcessingMode.Local;
this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\product-list.rdlc");
this.reportViewer.RefreshReport();
`
```

Report error

When an error occurs in the report processing, it raises the **ReportError** event. You can handle the event and show the details in your custom dialog instead of component default error detail interface.

```
`csharp
this.reportViewer.ReportError += (sen, arg) =>
{
    MessageBox.Show(arg.Message, "ReportError", MessageBoxButton.OK);
};
this.reportViewer.ProcessingMode = BoldReports.UI.Xaml.ProcessingMode.Local;
//this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\product-list.rdlc");
this.reportViewer.RefreshReport();
`
```

Drill through

When a drill through item is selected in a report, it invokes the **DrillThroughReport** event. You can change the drill through arguments such as report parameter and data sources. The following sample code can be used to change the drill through report name and set the parameter value before the drill through report is rendered.

```
`csharp
this.reportViewer.DrillThroughReport += (sen, arg) =>
{
    List<BoldReports.Windows.ReportParameter> parameters = new
    List<BoldReports.Windows.ReportParameter>();

    BoldReports.Windows.ReportParameter parameter = new BoldReports.Windows.ReportParameter();
    parameter.Name = "EmployeeID";
    parameter.Values = new List<string>() { "4" };
    parameters.Add(parameter);

    //Assign the drill through report path and parameters
    arg.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory, @"Resources\personal-
    details.rdl");
    arg.Parameters = parameters;
};

this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\sales-person-details.rdl");
this.reportViewer.RefreshReport();
`
```

Hyperlink

The **Hyperlink** event occurs when the user clicks a hyperlink in a report, before the hyperlink is followed. The following sample code redirects to a new custom URL and cancels the component default action.

```
`csharp
this.reportViewer.Hyperlink += (sen, arg) =>
{
    arg.Cancel = true;
    System.Diagnostics.Process.Start(arg.Hyperlink);
};

this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\customer-support-analysis.rdl");
this.reportViewer.RefreshReport();
`
```

Error logging in WPF Report Viewer

If an error occurred in report processing, WPF Report Viewer displays short messages about the error in view. In report viewer `ReportError` event raised. You can register the event and get the report error details from the event arguments to save all logs error information into a physical file location.

This section explains how to log the detailed error information to your WPF application.

This section requires a WPF Report Viewer application, if you don't have then create using the [Getting-Started](#).

1. In Solution Explorer, Open report viewer initialization cs file.
2. Register the `ReportError` event.

```
`csharp
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    this.reportViewer.ReportError += ReportViewer_ReportError;
}
private void ReportViewer_ReportError(object sender, ReportErrorEventArgs e)
{
    // You can register the report errors using event arguments.
}
`
```

3. Create a method in `MainWindow.xaml.cs` to write the error text into application folder.

```
`csharp
private void WriteLogs(string errorMessage)
{
    string filePath = System.IO.Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)+
@"\Errordetails.txt";
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
        writer.WriteLine(errorMessage);
    }
}
```

,

4. Invoke the newly created function in `ReportViewer_ReportError` as follows.

```
`csharp
private void ReportViewer_ReportError(object sender, ReportErrorEventArgs e)
{
    string errorLog;
    if (e.Exception != null)
    {
        errorLog = (string.Format("Error Message: {0} \n Stack Trace: {1}", e.Message, e.Exception.StackTrace));
    }
    else
    {
        errorLog = e.Message;
    }
    WriteLogs(errorLog);
}
,
```

In cases of any issues faced in the report rendering, share the log file to our technical support team to get assistance on that.

5. The final `MainWindow.xaml.cs` file is given as follows, you can replace it in your application.

```
`csharp
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        this.Loaded += new RoutedEventHandler(MainWindow_Loaded);
    }
    private void MainWindow_Loaded(object sender, RoutedEventArgs e)
    {
        this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\sales-order-detail.rdl");
    }
}
```

```

this.ReportViewer.ReportError += ReportViewer_ReportError;
this.reportViewer.RefreshReport();
}
private void ReportViewer_ReportError(object sender, ReportErrorEventArgs e)
{
    string errorLog;
    if (e.Exception != null)
    {
        errorLog = (string.Format("Error Message: {0} \n Stack Trace: {1}", e.Message, e.Exception.StackTrace));
    }
    else
    {
        errorLog = e.Message;
    }
    WriteLogs(errorLog);
}
private void WriteLogs(string errorMessage)
{
    string filePath = System.IO.Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)+
@"\Errordetails.txt";
    using (StreamWriter writer = new StreamWriter(filePath, true))
    {
        writer.AutoFlush = true;
        writer.WriteLine(errorMessage);
    }
}
,

```

Print report

The Report Viewer provides print option in the toolbar to print a copy of the report. The page setup dialog allows you to set the paper size or other page setup properties. To see print margins, click Print Layout on the toolbar.

The values allow you to set in the Page Setup dialog box for current session only. When you close the report and reopen it, it will have the default values again. The default values for the page setup dialog come from the report properties, which are set in the design view.

View report in print mode

Print margins are displayed in the Print Layout only, to view report in print mode by default, set the `ViewMode` property to `ViewMode.Print`.

```
`csharp
```

```
this.reportViewer.ViewMode = BoldReports.UI.Xaml.ViewMode.Print;
```

```
`
```

By default, the Report Viewer renders report in normal layout in which the print margins are not displayed.

Remove empty spaces in printing

The extra blank page is created when the Body of your report is too wide for your page. If you want the report to appear on a single page, all the content within the report body must fit on the physical page and the body width should be lesser or equal to the following formula:

Body Width <= Page Width - (Left Margin + Right Margin)

For more details on designing a report to remove the empty pages in the report, refer to the knowledge base article of [report page sizing](#).

Export report

The Report Viewer provides events and properties to control and customize the exporting functionality.

PDF export options

The `PDFOptions` provides properties to manage PDF export behaviors. You have to set the properties in the application window loaded method.

Export with complex scripts

To export reports with the complex scripts, set the `EnableComplexScript` property of `PDFOptions` instance to true.

```
`csharp
```

```
this.reportViewer.PDFOptions = new BoldReports.Writer.PDFOptions()
```

```
{
```

```
EnableComplexScript = true
```

```
};
```

```
`
```

PDF Conformance

You can export the report as `PDF/A-1b` document by specifying the conformance level `PdfConformanceLevel.Pdf_A1B` in the `PdfConformanceLevel` property.

```
`csharp
```

```
this.reportViewer.PDFOptions = new BoldReports.Writer.PDFOptions()
```

```
{
PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B
};
`
```

Word export options

The **WordOptions** provides properties to manage Word document export behaviors.

Word document type

You can save the report to the required document version by setting the **FormatType** property.

```
`csharp
this.reportViewer.WordOptions = new BoldReports.Writer.WordOptions();
this.reportViewer.WordOptions.FormatType = BoldReports.Writer.WordFormatType.Docx;
`
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the **LayoutOption** as **TopLevel**. The **ParagraphSpacing** is the distance value added between two elements in the document.

```
`csharp
this.reportViewer.WordOptions = new BoldReports.Writer.WordOptions();
this.reportViewer.WordOptions.LayoutOption = BoldReports.Writer.WordLayoutOptions.TopLevel;
this.reportViewer.WordOptions.ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
    Bottom = 0.5f,
    Top = 0.5f
};
`
```

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior.

The table in word document is not a stand-alone object, if you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, added an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without a password. The following are the types of protection.

1. **AllowOnlyComments**: You can add or modify only the comments in the Word document.
2. **AllowOnlyFormFields**: You can modify the form field values in the Word document.
3. **AllowOnlyRevisions**: You can accept or reject the revisions in the Word document.

4. **AllowOnlyReading**: You can only view the content in the Word document.
5. **NoProtection**: You can access or edit the Word document contents as normally.

```
`csharp
```

```
this.reportViewer.WordOptions = new BoldReports.Writer.WordOptions();  
this.reportViewer.WordOptions.ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading;  
`
```

Excel export options

The **ExcelOptions** provides properties to manage Excel document export behaviors.

Excel document type

You can save the report to the required excel version by setting the **ExcelSaveType** property.

```
`csharp
```

```
this.reportViewer.ExcelOptions = new BoldReports.Writer.ExcelOptions();  
this.reportViewer.ExcelOptions.ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013;  
`
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the **LayoutOption** as **IgnoreCellMerge**.

```
`csharp
```

```
this.reportViewer.ExcelOptions = new BoldReports.Writer.ExcelOptions();  
this.reportViewer.ExcelOptions.LayoutOption =  
BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge;  
`
```

Protecting Excel document from editing

You can restrict the Excel document from editing either by providing the **ExcelSheetProtection** or enabling the **ReadOnlyRecommended** properties.

```
`csharp
```

```
this.reportViewer.ExcelOptions = new BoldReports.Writer.ExcelOptions()  
{  
    ReadOnlyRecommended = true,  
    ExcelSheetProtection = Syncfusion.XlsIO.ExcelSheetProtection.DeletingColumns  
};  
`
```

PowerPoint export options

You can save the report to the required PowerPoint version by setting the **FormatType** property.

```
`csharp
```

```
this.reportViewer.PPTOptions = new BoldReports.Writer.PPTOptions();  
this.reportViewer.PPTOptions.FormatType = BoldReports.Writer.PPTSaveType.PowerPoint2013;  
`
```

CSV export options

The **CsvOptions** allows you to change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document.

```
`csharp  
this.reportViewer.CsvOptions = new BoldReports.Writer.CsvOptions()  
{  
    Encoding = System.Text.Encoding.Default,  
    FieldDelimiter = ",",  
    UseFormattedValues = false,  
    Qualifier = "#",  
    RecordDelimiter = "@",  
    SuppressLineBreaks = true,  
    FileExtension = ".txt"  
};  
`
```

HTML export options

You can hide the separator added at the end of each page by setting the **HidePageSeparator** property to true.

```
`csharp  
this.reportViewer.HTMLOptions = new BoldReports.Writer.HTMLOptions();  
this.reportViewer.HTMLOptions.HidePageSeparator = true;  
`
```

Password protect exported document

This allows you to protect the exported document such as PDF, Word, Excel, and PowerPoint from unauthorized users by encrypting the document using encryption password. The following code snippet illustrates how to encrypt the exported document with the user defined password.

```
`csharp  
//PDF encryption  
this.reportViewer.PDFOptions = new BoldReports.Writer.PDFOptions();  
this.reportViewer.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity()  
{  
    UserPassword = "Password"  
}
```

```
};  
//Word encryption  
this.reportViewer.WordOptions = new BoldReports.Writer.WordOptions()  
{  
    EncryptionPassword = "password"  
};  
//Excel encryption  
this.reportViewer.ExcelOptions = new BoldReports.Writer.ExcelOptions()  
{  
    PasswordToModify = "password",  
    PasswordToOpen = "password"  
};  
//PPT encryption  
this.reportViewer.PPTOptions = new BoldReports.Writer.PPTOptions()  
{  
    EncryptionPassword = "password"  
};  
`
```

Password protection is not supported for HTML export format.

Change file name in export

You can change the file name of report in export using the `FileName` property.

```
`csharp  
this.reportViewer.ExportSettings = new BoldReports.Writer.ExportSettings();  
this.reportViewer.ExportSettings.FileName = "Invoice";  
`
```

Change image quality in export

You can change image quality of data visualization items in report export using the `ImageQuality` property.

```
`csharp  
this.reportViewer.ExportSettings = new BoldReports.Writer.ExportSettings();  
this.reportViewer.ExportSettings.ImageQuality = 4;  
`
```

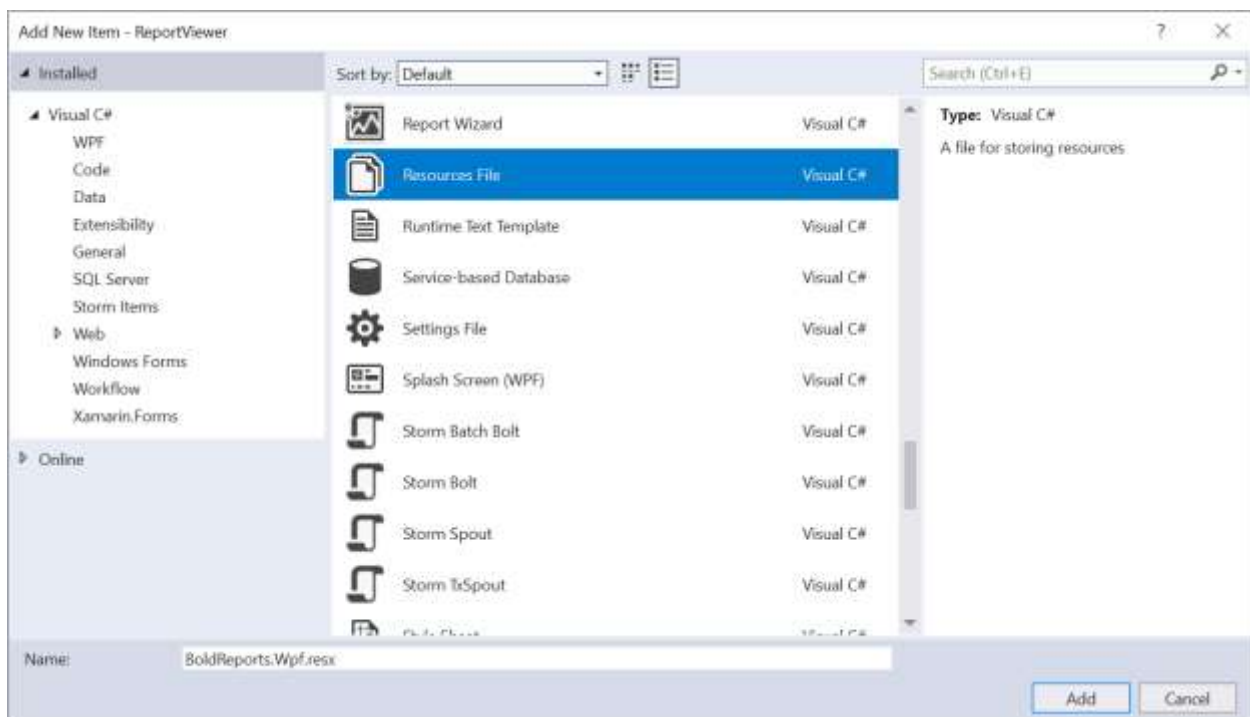
Localization of Bold Report Viewer

Localization of Bold Report Viewer allows you to localize the static text such as tooltip, parameter block, and dialog text based on a specific culture. Refer the following steps to localize the Report Viewer based on the culture.

1. Add Resources file for the different cultures.
2. Assign the value to each culture using key.
3. Assign a Current UI Culture to the application.
4. Set Report Viewer properties.

Add the Resource file for the different cultures

1. Right-click the project and add the **Resources** folder in your application.
2. Right-click on the **Resources** folder and press **Ctrl+Shift+A** keys or select **Add > New Item** from the context menu.
3. In the Add New Item dialog, select **Resources** file and name it as **BoldReports.Wpf.resx**.



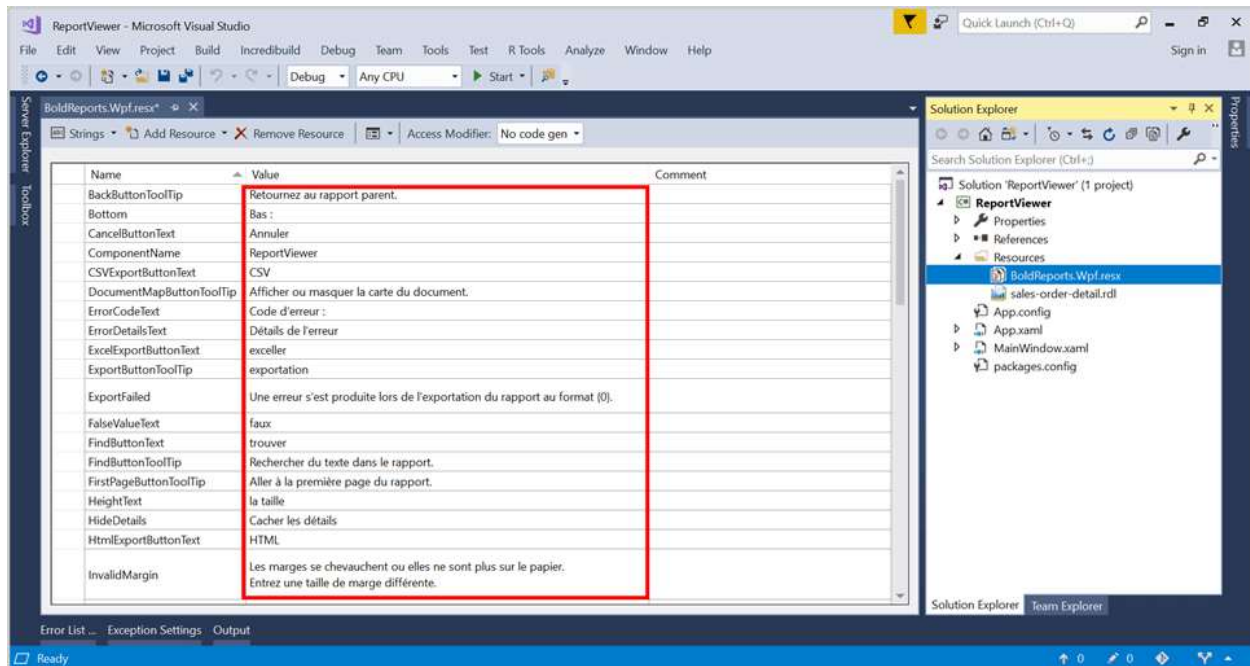
4. Click **Add**.

In case, the another culture is used in the application, then create another resource file in name **[AssemblyName].[CultureInfo Code].resx** and the naming convention needs to be followed mandatorily.

Assign the values to each culture using key

To assign **Values** in Resource, the resource file needs to be updated according to the following steps.

1. Open the **BoldReports.Wpf.resx** file by double clicking it from Solutions Explorer.
2. Add the resource key name, and its corresponding localized value by editing its field. Modify the resource values based on the **fr-FR** culture as shown in the following image.



You can download the modified resource file from [here](#) and replace it in your application.

Assign a Current UI Culture to the application

Mention the culture to be referred while initializing the application, so that application refer to the appropriate value provided in resource file.

Set the **CultureInfo** in the application before the **InitializeComponent()** method call. In this application, **MainWindow.xaml.cs** is the startup page and culture is assigned as like below.

```
`csharp
```

```
public MainWindow()
```

```
{
```

```
System.Globalization.CultureInfo.CurrentUICulture = new System.Globalization.CultureInfo("fr-FR");
```

```
InitializeComponent();
```

```
}
```

```
,
```

Set Report Viewer properties

1. Add the Report Viewer initialization code.

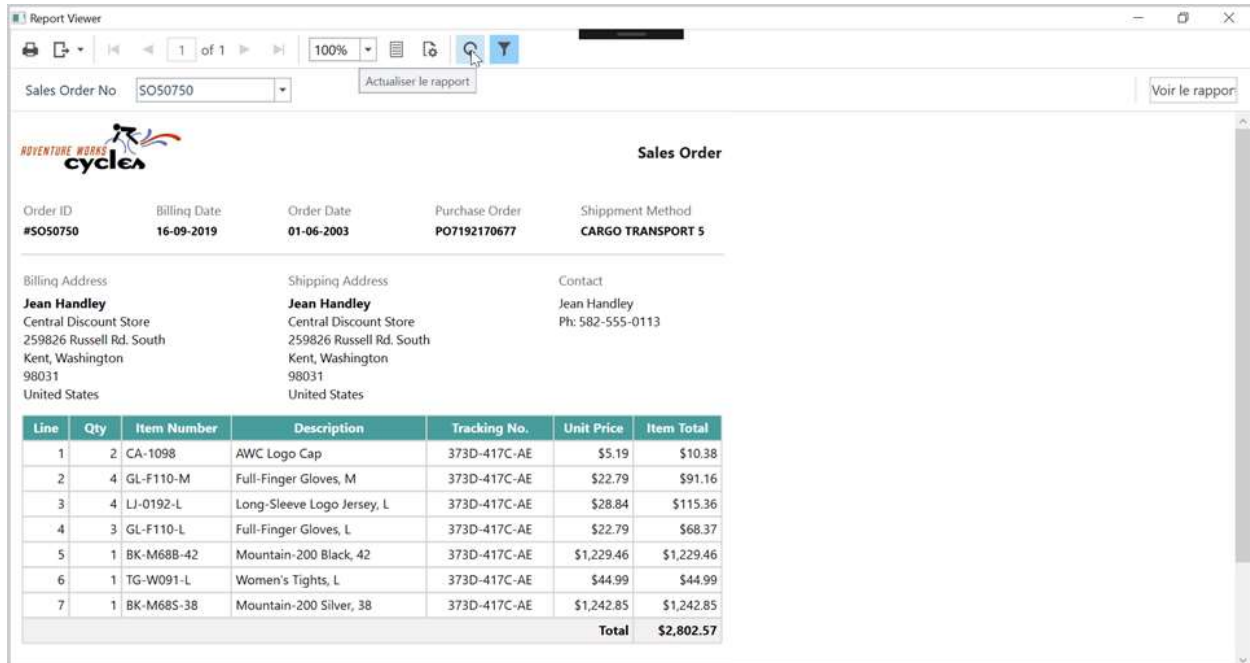
```
`csharp
```

```
this.reportViewer.ReportPath = System.IO.Path.Combine(Environment.CurrentDirectory,
@"Resources\sales-order-detail.rdl");
```

```
this.reportViewer.RefreshReport();
```

In this tutorial, `sales-order-detail` report is used.

- Now, run the application and the below output shows the toolbar items localized `fr-FR` culture.



Limitations

RDL specification

The Report Viewer control does not support RDL specification for SQL Server 2000 and SQL Server 2005.

Report layout

- In the Tablix cell split layout process, the entire cell moves to the next page to display the complete cell items, when the table cell width value exceeds the page width.

Expressions

The object function and VB function do not have complete support.

SSRS

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the server. If the report has any data source that uses credentials to connect with the database, then you should specify the data source credentials for each report data source to establish database connection.

Samples and Demos

Browse and explore the ready-to-use RDL, RDLC reports, samples, and offline demos.

Locally installed reports

You can obtain the sample RDL and RDLC files from the Bold Reports installed location

%localappdata%\Bold Reports\Embedded Reporting\Samples\Common\Data\ReportTemplate.

Offline demos

The offline samples are provided in the Bold Reporting Tools setup. For more details, refer to the [Bold Reporting Tools sample deployment](#).

Migrate Report Viewer application

In our Bold Reports new assemblies are introduced for both client and server-side to resolve the compatibility problem between Essential Studio Report Viewer versions.

This section provides step-by-step instructions for migrating Report Viewer from Syncfusion Essential Studio release version to Bold Reports version of WPF Report Viewer application:

Client-side migration

1. In the Solution Explorer, right-click the **References** and remove the following assembly references:
 - Syncfusion.ReportControls.Wpf
 - Syncfusion.ReportViewer.Wpf
 - Syncfusion.ReportWriter.Base
2. Add the assembly from the Bold Reporting NuGet package **BoldReports.Wpf**. To add from NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search for **BoldReports.Wpf** NuGet package, and install in your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Control initialization

1. Open the **MainWindow.xaml** file and remove the following namespace in your XAML page.

```
`csharp
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
`
```

2. Add the following namespace in your XAML page.

```
`csharp
xmlns:syncfusion="clr-namespace:BoldReports.UI.Xaml;assembly=BoldReports.Wpf"
`
```

NuGet Packages for WPF

Refer to the following steps to configure Bold Reporting NuGet packages for WPF application.

Configure NuGet feed URL

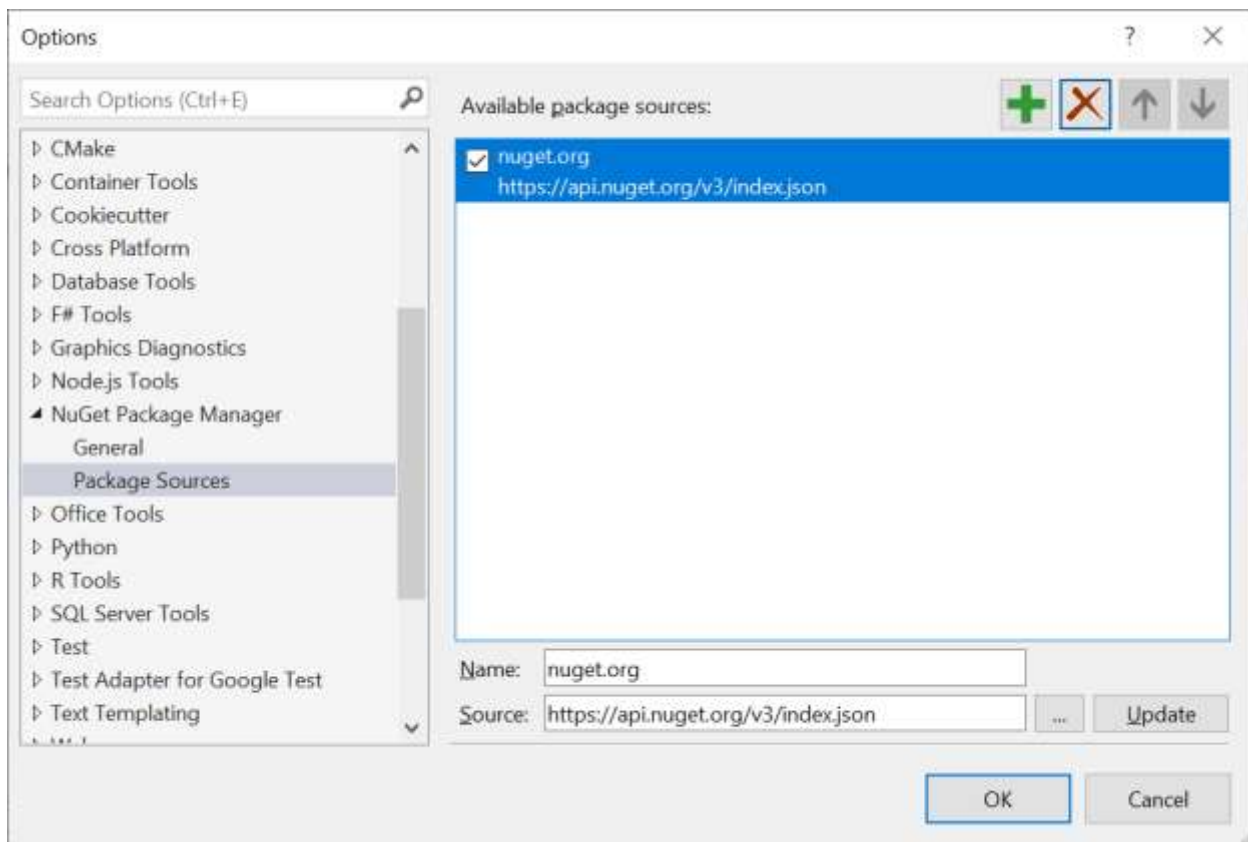
Online NuGet feed URL

The Bold Reporting NuGet packages are published in **Nuget.org**. To configure the online packages, use the following steps:

1. Open Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager** and select **Package Sources**.
4. Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: NuGet.org

Source: <https://api.nuget.org/v3/index.json>



Installing NuGet packages

Install using NuGet Package Manager

The NuGet Package Manager can be used to search and install NuGet packages in Visual Studio solution or project:

1. On the **Tools** menu, click NuGet Package Manager | Manage NuGet Packages for Solution. Alternatively, right-click the project/solution in Solution Explorer tab, and choose **Manage NuGet Packages**.
2. By default, the NuGet.org package is selected in the **Package source** drop-down. Select package source, search for the BoldReports.WPF package, and then click **Install** button.

Install using Package Manager Console

To install the Bold Reporting component using the Package Manager Console as NuGet packages,

1. On the **Tools** menu, select **NuGet Package Manager**, and then click **Package Manager Console**.
2. Run the following NuGet installation commands:

```
`cmd
```

install specified package in default project

```
Install-Package <Package Name>
```

install specified package in default project with specified package source

```
Install-Package <Package Name> -Source <Source Location>
```

install specified package in specified project

```
Install-Package <Package Name> -ProjectName <Project Name>
```

```
`
```

For example:

```
`cmd
```

install specified package in default project

```
Install-Package BoldReports.Wpf
```

install specified package in default project with specified Package Source

```
Install-Package BoldReports.Wpf -Source "https://api.nuget.org/v3/index.json"
```

install specified package in specified project

```
Install-Package BoldReports.Wpf -ProjectName BoldReportsApplication
```

```
`
```

Upgrading NuGet packages

Upgrading using NuGet Package Manager

NuGet packages can be updated to their specific version or latest version available in the Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution....** Alternatively, right-click the project/solution in the Solution Explorer tab, and then choose **Manage NuGet Packages**.
2. Select the **Updates** tab to see the packages available for update from the desired package sources, select the required packages and specific version from the drop-down, and then click the **Update** button.

Upgrading using Package Manger Console

To update the installed Bold Reporting NuGet packages using the Package Manager Console:

1. On the **Tools** menu, select **NuGet Package Manager**, and then select **Package Manager Console**.
2. Run the following NuGet installation commands:

```
`cmd
```

Update specific NuGet package in default project

```
Update-Package <Package Name>
```

Update all the packages in default project

```
Update-Package
```

Update specified package in default project with specified package source

```
Update-Package <Package Name> -Source <Source Location>
```

Update specified package in specified project

```
Update-Package <Package Name> - ProjectName <Project Name>
```

```
,
```

For example:

```
`cmd
```

Update specified Bold Reporting NuGet package

```
Update-Package BoldReports.Wpf
```

Update specified package in default project with specified Package Source

```
Update-Package BoldReports.Wpf -Source "https://api.nuget.org/v3/index.json"
```

Update specified package in specified project

```
Update-Package BoldReports.Wpf -ProjectName BoldReportsApplication
```

```
,
```

Upgrading using NuGet CLI

Using the NuGet CLI, all the NuGet packages in the project can be updated to the available latest version:

1. Download the latest [NuGet CLI](#).

To update the existing `nuget.exe` to latest version use the following command:

```
`cmd
```

```
nuget update -self
```

```
,
```

2. Open the downloaded executable location in the command window. Run the following “update commands” to update the Bold Reporting NuGet packages.

```
`cmd
```

update all NuGet packages from config file

```
nuget update <configPath> [options]
```

update all NuGet packages from specified Packages Source

```
nuget update -Source <Source Location> [optional]
```

```
,
```

`configPath` is optional. It identifies the `package.config` or solutions file lists the packages utilized in the project.

For example:

```
`cmd
```

Update all NuGet packages from config file

```
nuget update "C:\Users\BoldReportsApplication\package.config"
```

Update all NuGet packages from specified Packages Source

```
nuget update -Source "https://api.nuget.org/v3/index.json"
```

```
,
```

The Update command is not working as expected in Mono (Mac and Linux) and projects using PackageReference format.

Custom properties

The custom properties helps you to include additional features that are not natively supported in the RDL reporting. This topic explains the list of custom properties supported in WPF Report Viewer.

See Also

- [Table Custom Properties](#)
- [Report Custom Properties](#)
- [Parameter Custom Properties](#)
- [Export Custom Properties](#)

Table custom properties

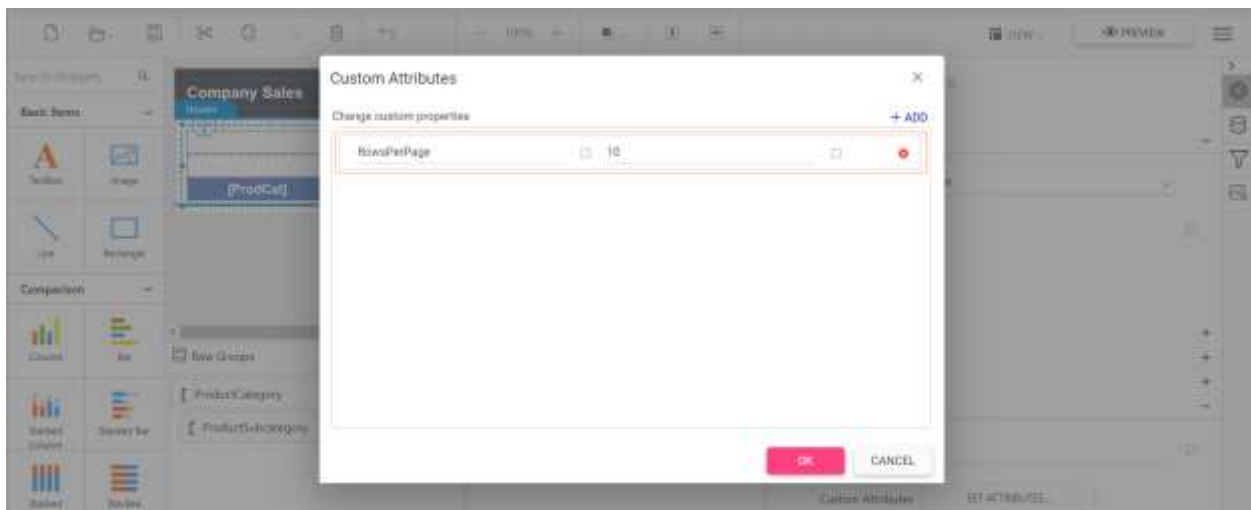
This topic explains about the list of table report item custom properties that are supported to render in WPF Report Viewer.

Limit number of table records on each page

The `RowsPerPage` custom property is used to specify the number of table records to display on each page. It supports integer data value greater than zero.

This property is ignored when table rows heights higher than current page size. Increase the report page height or reduce `RowsPerPage` count that fits within the page.

You can set the `RowsPerPage` property value, as shown in the below.



Report custom properties

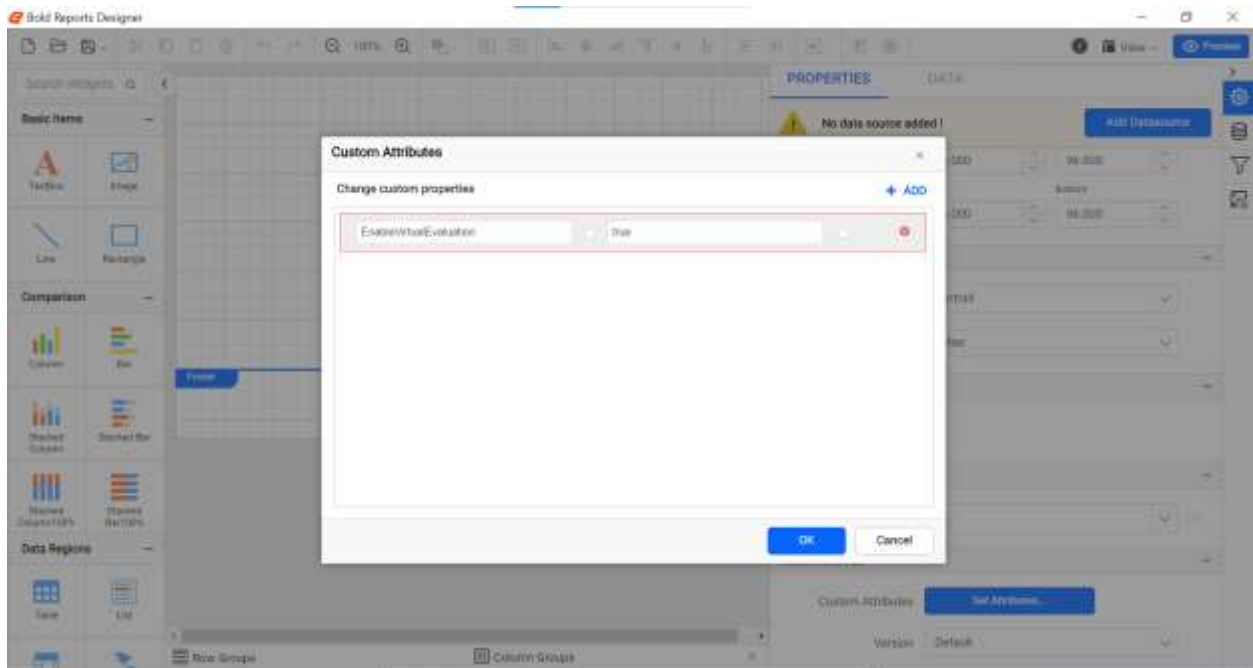
This topic explains about the list of report custom properties that are supported to render in WPF Report Viewer.

Improve performance and handle large amount of data

Set the `EnableVirtualEvaluation` and `DisablePageSplitting` custom properties in a report to improve performance and handle the larger amount of data with less memory footprint.

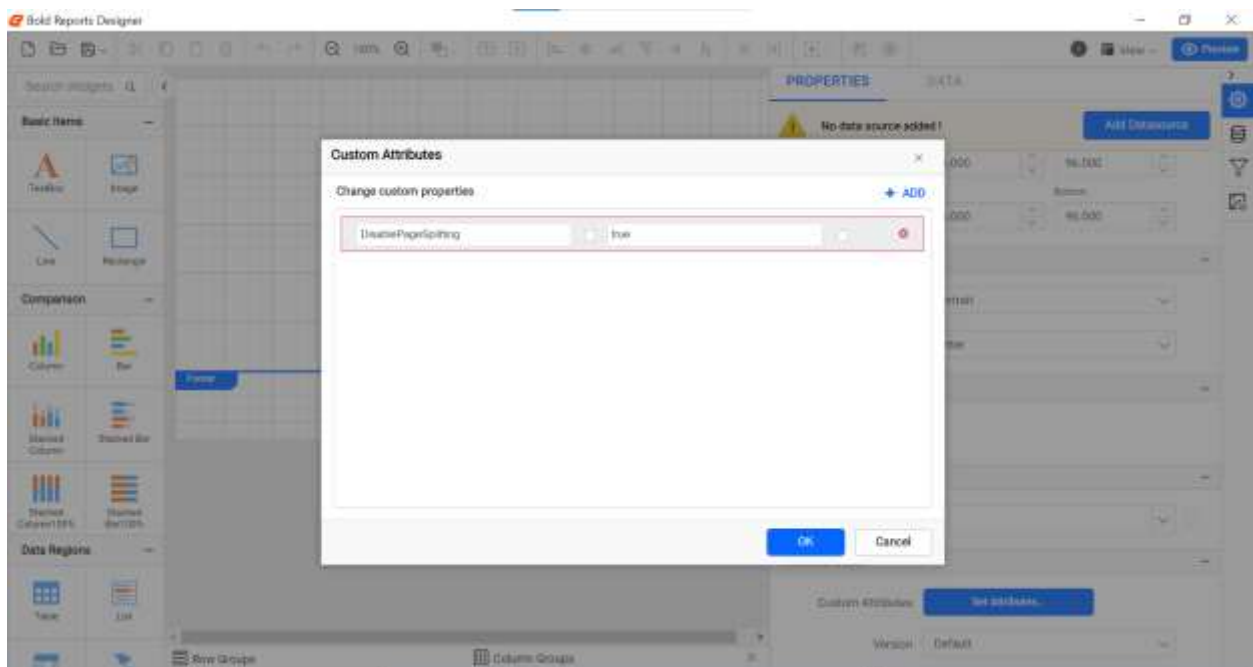
Render large data report faster

The `EnableVirtualEvaluation` custom property is used to render the large data report faster. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Reduce memory footprint for large data report

The `DisablePageSplitting` custom property is used to reduce the memory footprint for large data report. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Improve report items layout and avoid extra blank pages

When the `RoundLayoutMeasures` property is false, all non-integral values that are calculated during the report processing are rounded to whole pixel values. It provides following improvements,

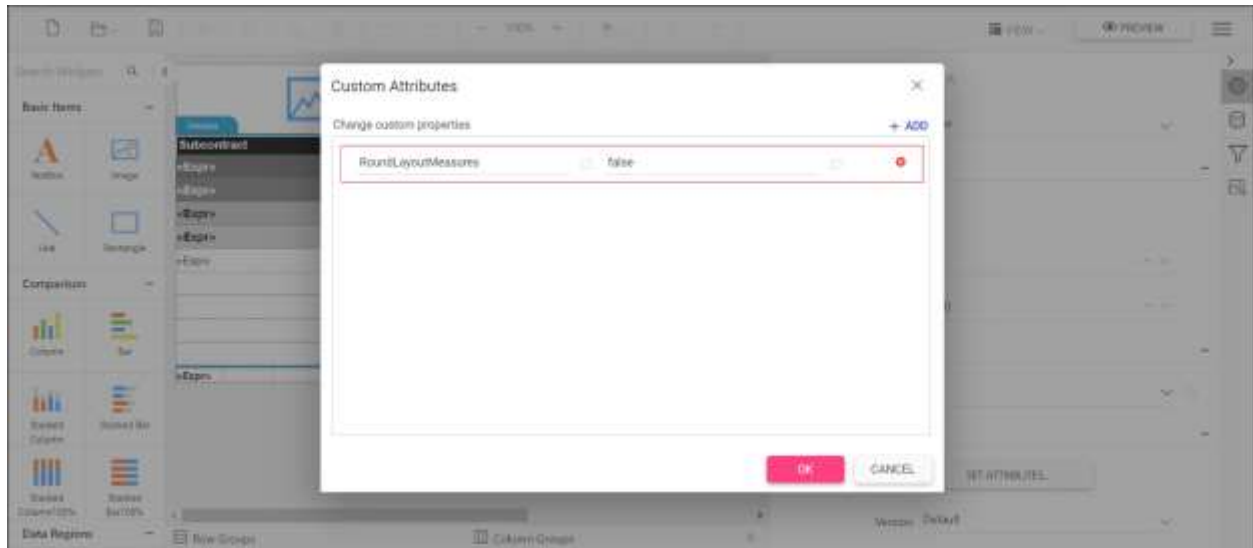
- Report text rendered without cuts.

Report custom properties
timeouts

Handling failure in long-running HTTP requests, report processes, and

- Eliminates extra blank pages.
- Removes item and text overlaps.
- Eliminates the blur semi-transparent edges that are produced by anti-aliasing.
- Produces identical look in report view and export output.

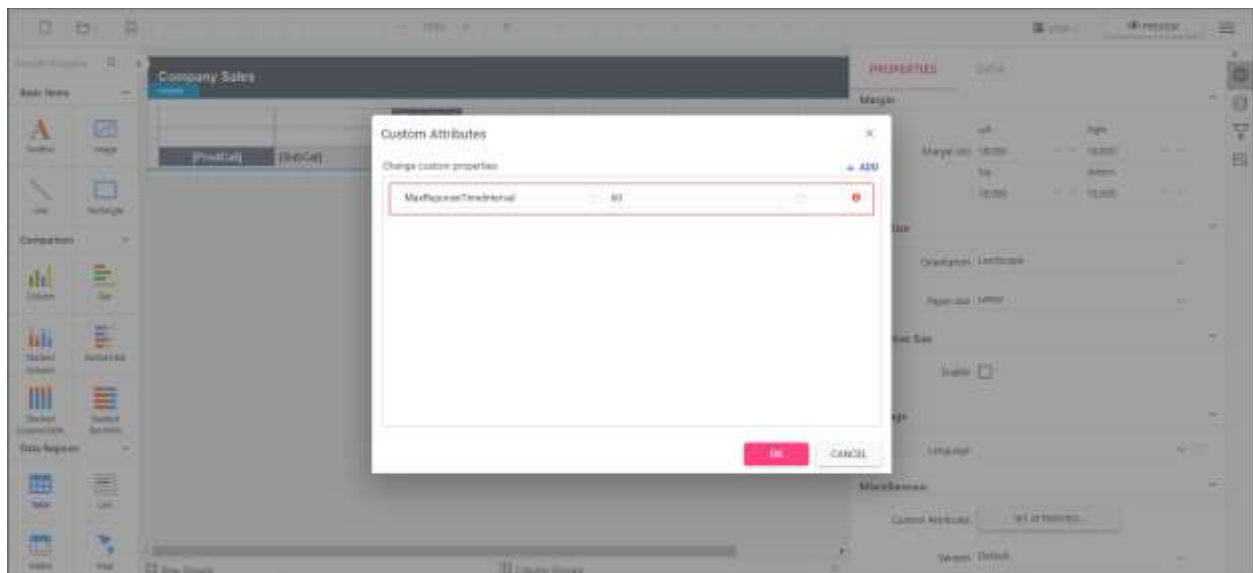
You can set the property value, as shown in the below.



Handling failure in long-running HTTP requests, report processes, and timeouts

When a report process for a long time due to huge records or a HTTP request takes too long to respond then it results in Gateway Timeout or report rendering errors. The `MaxResponseTimeInterval` allows to specify the seconds to process an HTTP request and respond back. It helps to keep the client and server connection live by avoiding the timeout.

You can set the property value as shown in the below.



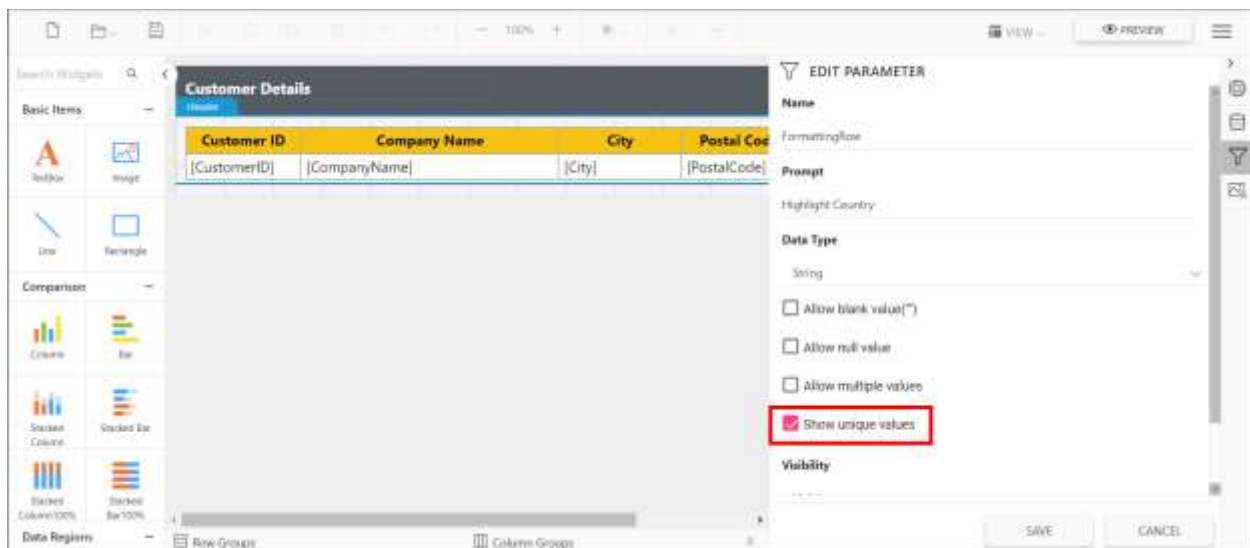
Parameter custom properties

This topic explains about the list of parameter custom properties that are supported to customize the reports preview in Report Viewer.

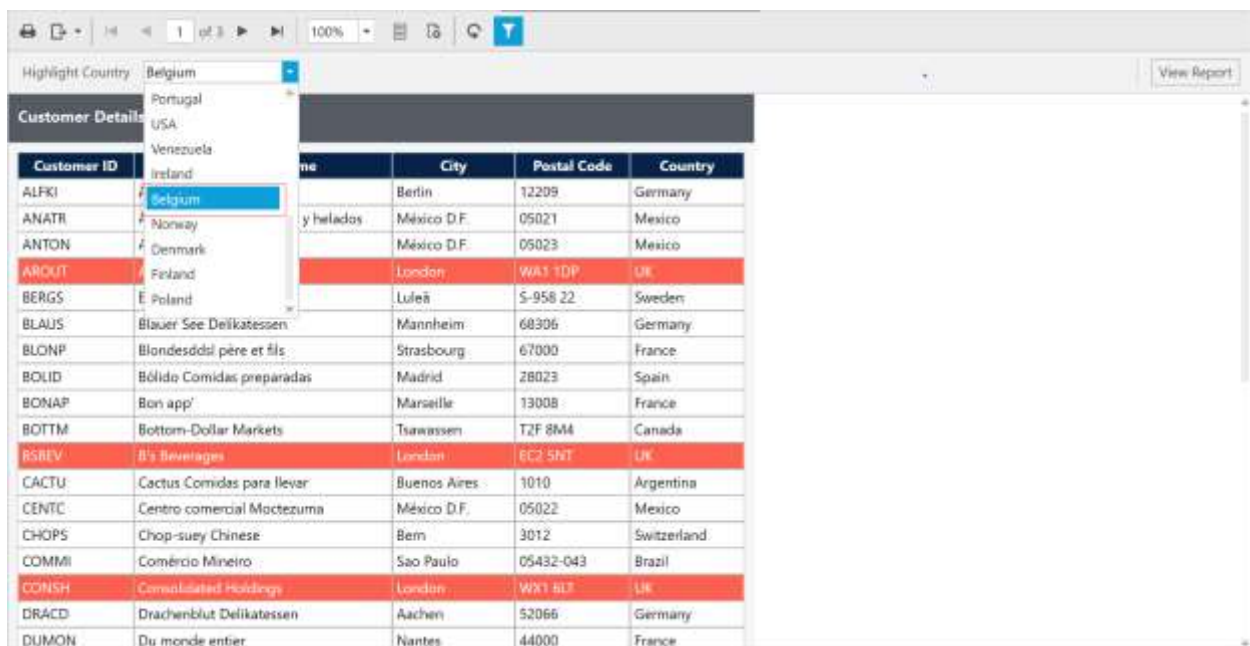
Show only distinct values in parameter

A parameter created with data set query values may contain duplicate values. The Report Viewer supports UniqueValueParameters custom property that helps show only unique values in the parameter drop-down while viewing the report, without creating new a data set. Refer to the below image for property setting option.

You can also provide the parameter names with comma (,) separator to set for multiple parameters.



Preview the report and the unique values showed in the parameter drop down.



Export custom properties

This topic explains the list of custom properties that are supported at the report level to control the export behaviour in WPF Report Viewer.

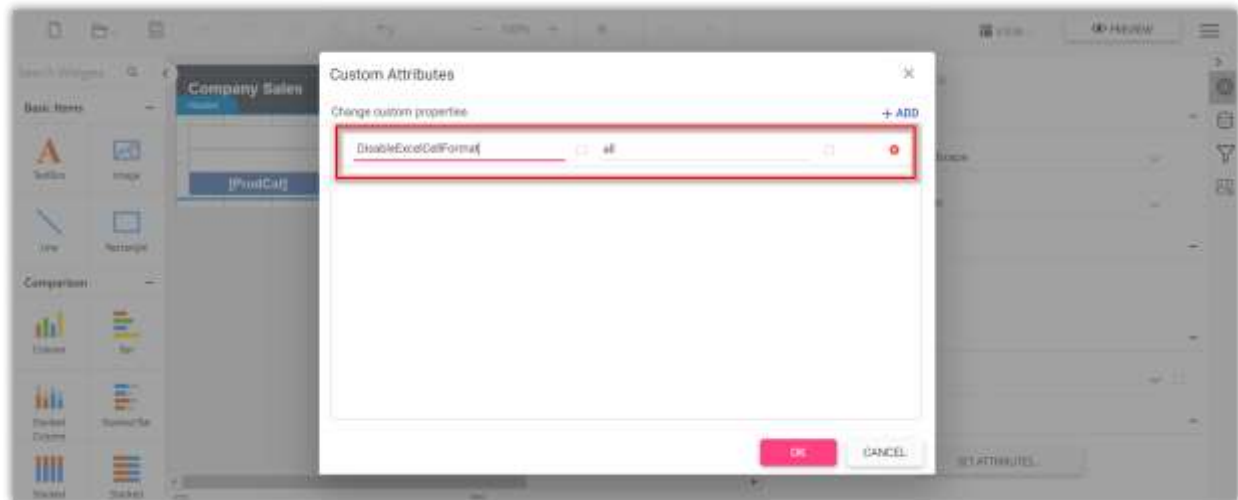
Improve excel export performance

The custom property `DisableExcelCellFormat` helps to improve the excel export performance by ignoring the rendering of report item styles. It allows property value from anyone of the following values.

- **Style** - Disables rendering of the cell styles like padding, background, color, and text style
- **Border** - Disables rendering of the cell border
- **All** - Disables rendering of the cell border, padding, background, color, and text style

Set the property value as **All** to improve maximum excel export performance.

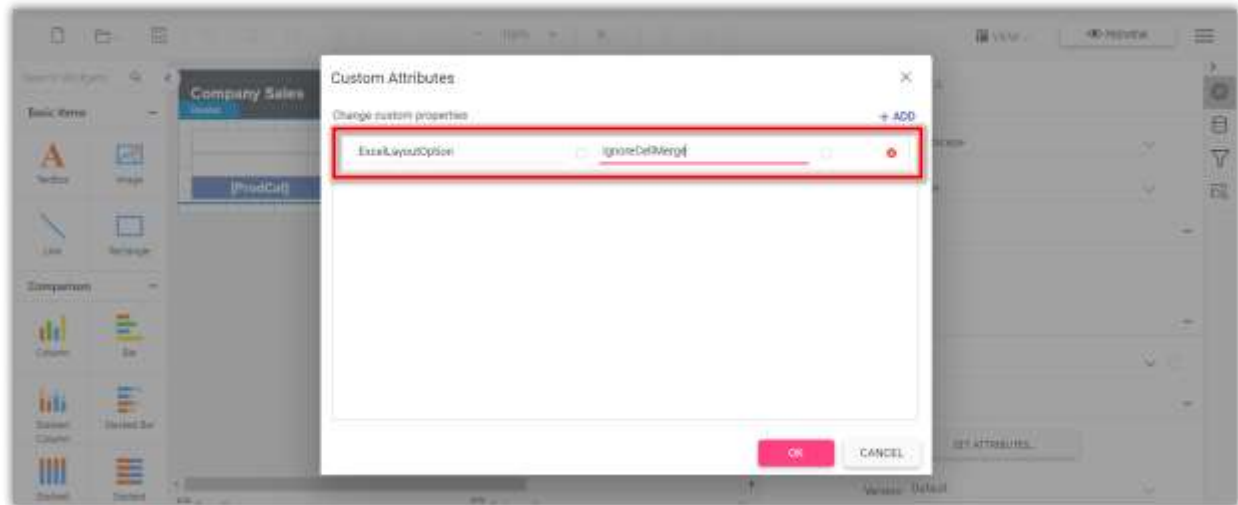
You can set the `DisableExcelCellFormat` custom property as shown below,



The `DisableExcelCellFormat` property must be added to report properties.

Improve excel export readability

Set `ExcelLayoutOption` custom property value as `IgnoreCellMerge` to improve the readability of the excel document by eliminating the tiny columns, rows, and merged cells. You can set the property value, as shown below,



The `ExcelLayoutOption` property must be added to report properties.

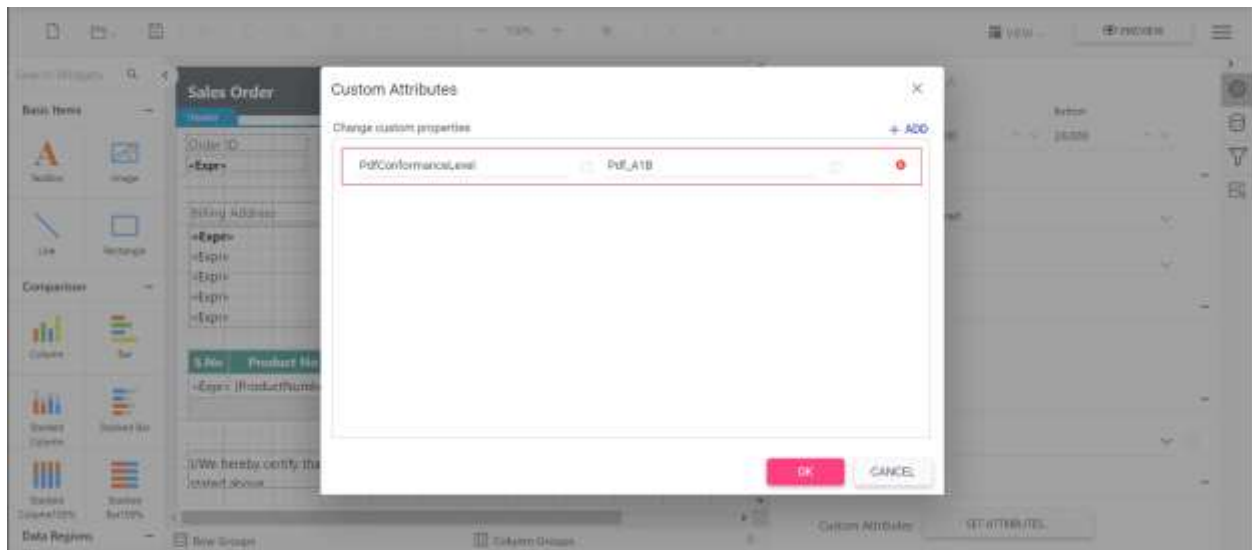
Set pdf conformance level

The `PdfConformanceLevel` allows you set PDF document versions.

You can set anyone of the following PDF conformance option.

- **PdfA1B** - You can create a PdfA1B document by specifying the conformance level as Pdf_A1B through PdfConformanceLevel Enum when creating the new PDF document.
- **PdfA2B** - You can create a PdfA2B document by specifying the conformance level as Pdf_A2B through PdfConformanceLevel Enum when creating the new PDF document
- **PdfA3B** - The PDF/A3B conformance supports the external files as attachment to the PDF document, so you can attach any document format such as Excel, Word, HTML, CAD, or XML files.
- **PdfA1A** - This conformance includes all PdfA1B requirements in addition to the features intended to improve a document's accessibility. PDF/A-1a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2A** - This conformance includes all PDF/A2B requirements in addition to the features intended to improve a document's accessibility. PDF/A-2a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2U** - This conformance includes all PdfA2U requirements, and additionally Unicode mapping for all text in the document.
- **PdfX1A2001** - You can create a PDF/X-1a document by specifying the conformance level as PdfX1A2001 through PdfConformanceLevel Enum when creating the new PDF document.

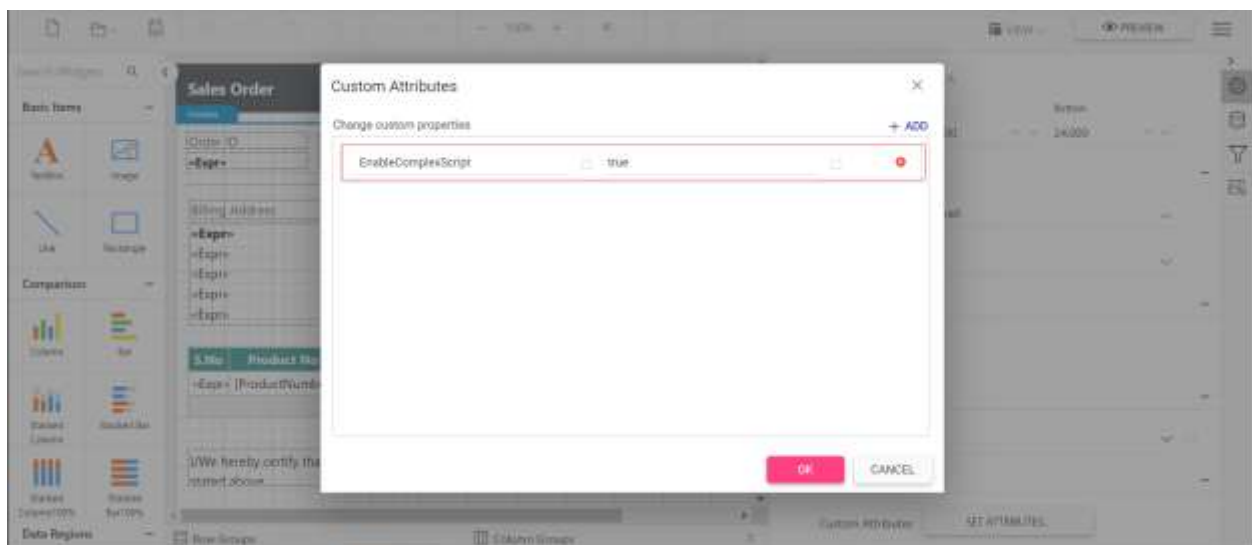
You can set the `PdfConformanceLevel` custom property as shown below,



Export pdf document with complex script

The `EnableComplexScript` custom property allows you to export pdf documents with complex script language texts.

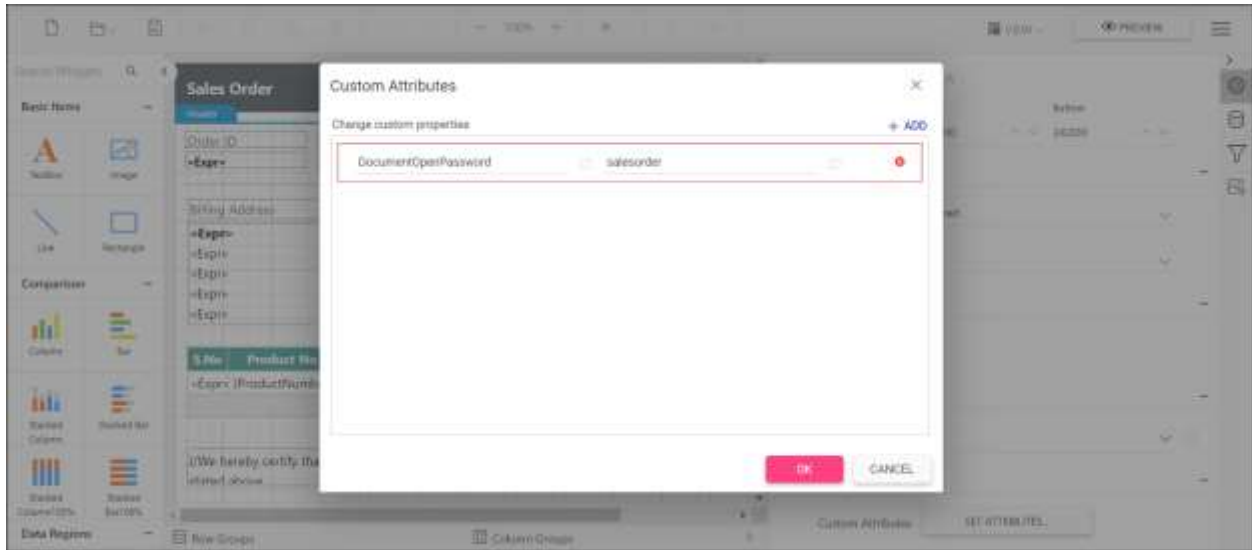
You can set the property value, as shown below,



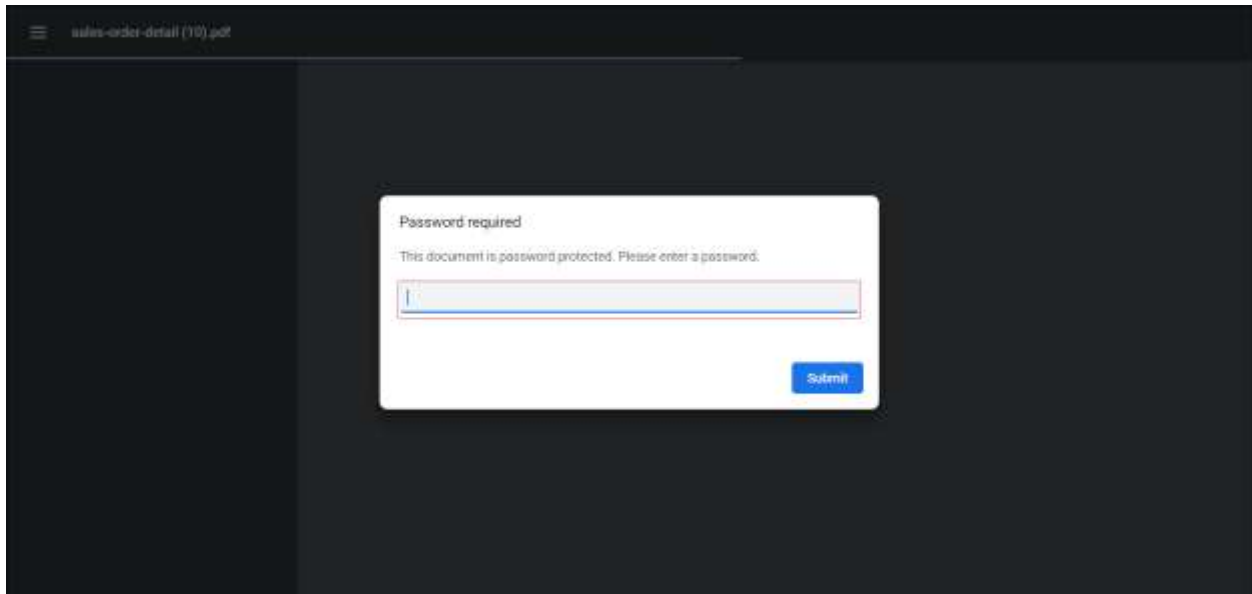
Encrypt and secure export documents

The custom property `DocumentOpenPassword` allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password.

You can set the property value, as shown below,



open the pdf document and see the below output.

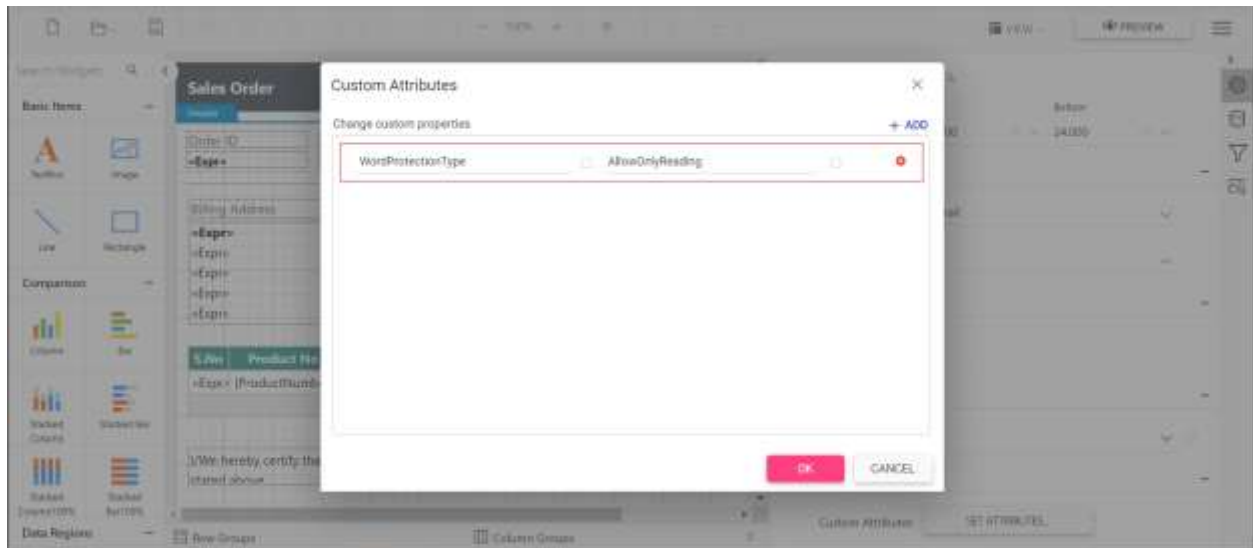


Protecting Word document from editing

The custom property `WordProtectionType` allows you to restrict a Word document from editing either by providing a password or without a password by using following types of protection.

- `AllowOnlyComments`- You can add or modify only the comments in the Word document.
- `AllowOnlyFormFields`- You can modify the form field values in the Word document.
- `AllowOnlyRevisions`- You can accept or reject the revisions in the Word document.
- `AllowOnlyReading`- You can only view the content in the Word document.
- `NoProtection`- You can access or edit the Word document contents as normally.

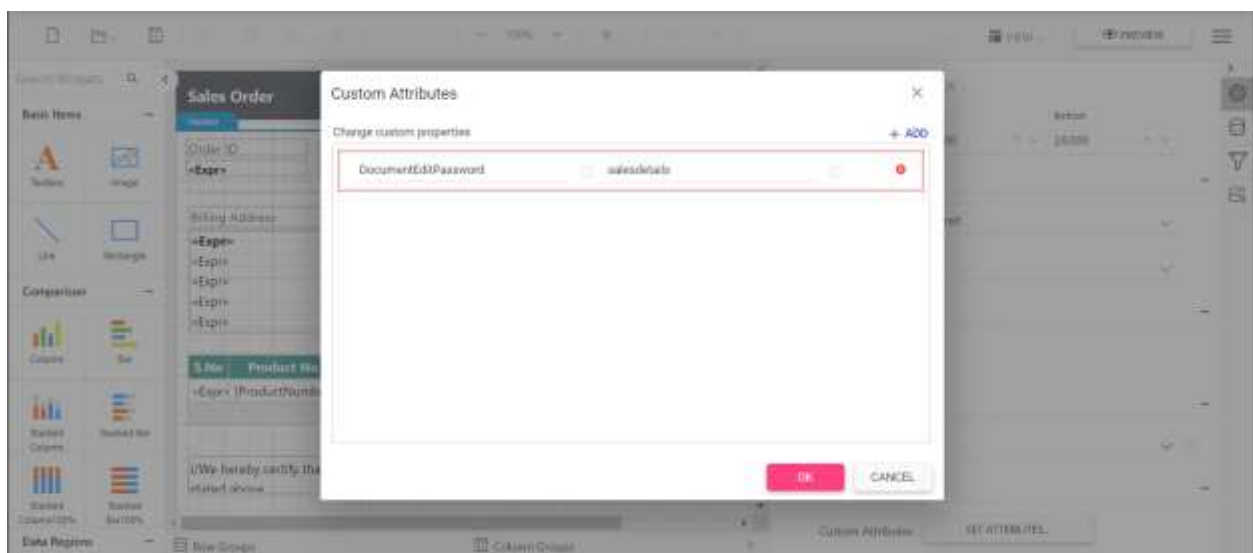
You can set the `WordProtectionType` custom property as shown below,



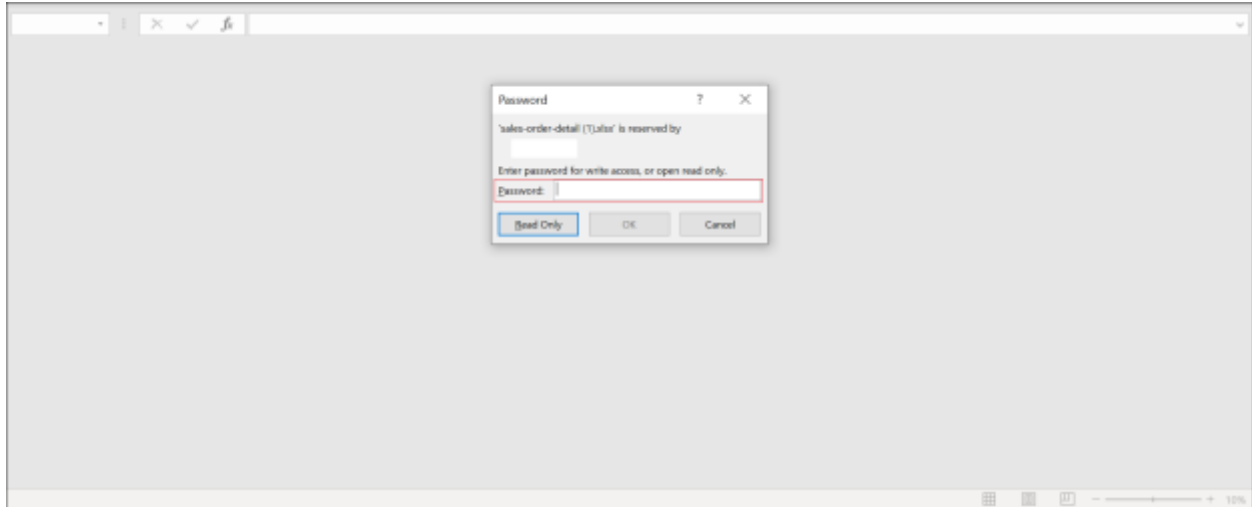
Set excel document edit password

The custom property `DocumentEditPassword` helps to allow specific users permission to modify the workbook data and save changes to the file in the excel document.

You can set the property value, as shown below,



open the excel document and see the below output.

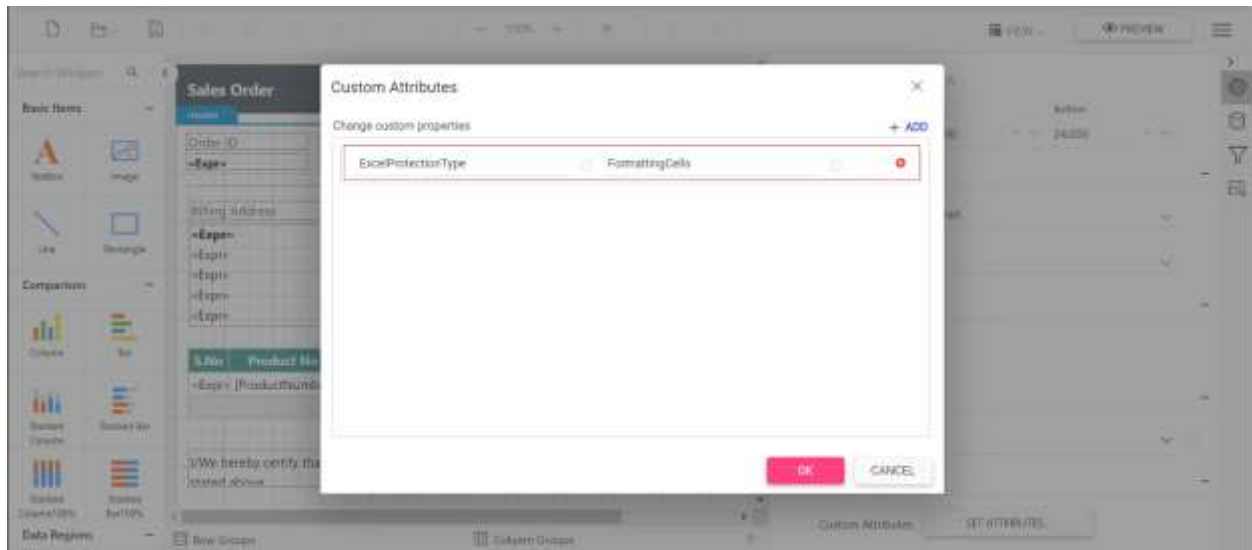


Set excel document protection

The custom property `ExcelProtectionType` allows you to protect the worksheet of excel document either by providing a password or without a password by using following types of protection.

- `None` - Represents no protection in excel sheet
- `Objects` - allows to protect shapes in excel sheet
- `Scenarios` - allows to protect scenarios.
- `FormattingCells` - allows the user to format any cell on a protected worksheet.
- `FormattingColumns` - allows the user to format any column on a protected worksheet.
- `FormattingRows` - allows the user to format any rows on a protected worksheet.
- `InsertingColumns` - allows the user to insert columns on the protected worksheet.
- `InsertingRows` - allows the user to insert rows on the protected worksheet.
- `InsertingHyperlinks` - allows the user to insert hyperlinks on the worksheet.
- `DeletingColumns` - allows the user to delete columns on the protected worksheet, where every cell in the column to be deleted is unlocked.
- `DeletingRows` - allows the user to delete rows on the protected worksheet, where every cell in the row to be deleted is unlocked.
- `LockedCells` - allows to protect locked cells.
- `Sorting` - allows the user to sort on the protected worksheet
- `Filtering` - allows the user to set filters on the protected worksheet. Users can change filter criteria but can not enable or disable an auto filter.
- `UsingPivotTables` - allows the user to use pivot table reports on the protected worksheet.
- `UnLockedCells` - allows to protect the user interface, but not macros.
- `Content` - allows to protect the contents in the excel sheet.
- `All` - allows to protect all type of protection.

You can set the `ExcelProtectionType` custom property as shown below,



How to Create RDL/RDLC Report

The following sections explain about how to create a new RDL/RDLC report using Bold Report Designer, Microsoft Report Builder and Visual Studio Report Server project template.

- [Create a RDL report](#)
- [Create a RDLC report](#)

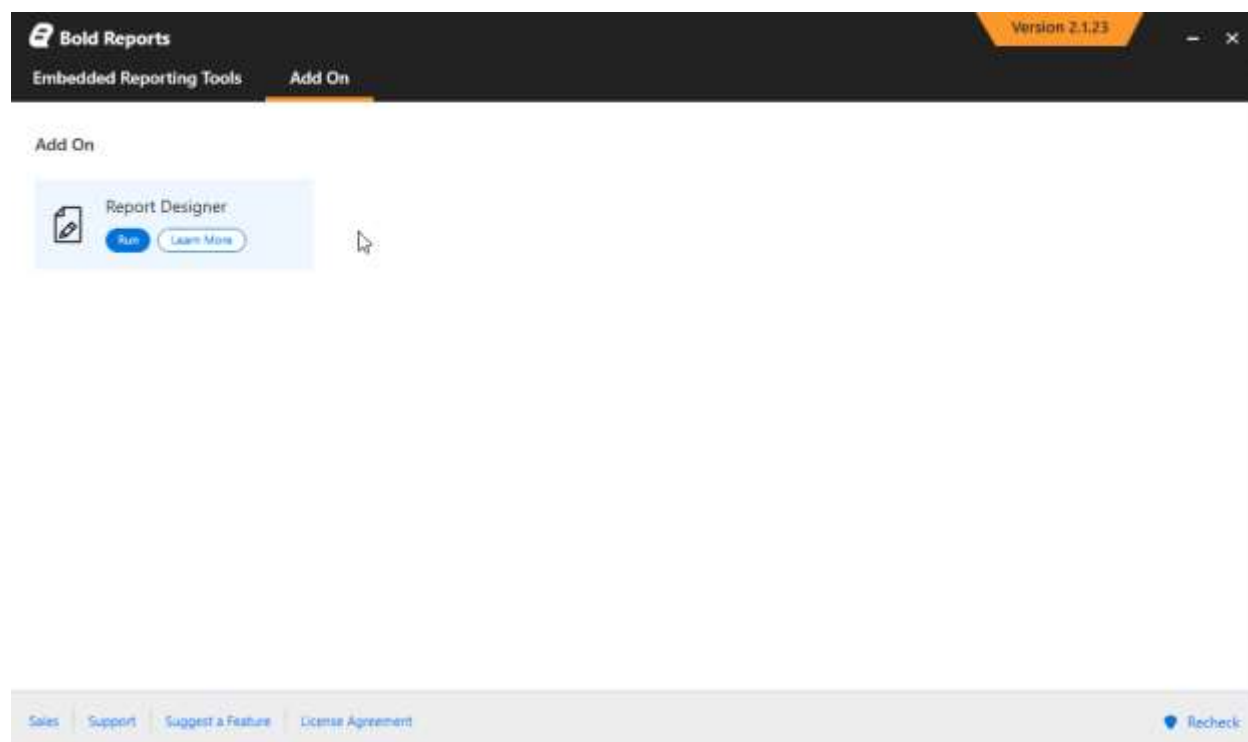
Create a SSRS RDL report

You can create an RDL report using any of the following reporting tools:

- Bold Reports Report Designer.
- Microsoft Report Builder.
- Visual Studio Report Server project template.

Bold Reports Report Designer

Bold Reports Report Designer provides the intuitive user interface to create and edit the RDL reports, which is available in Bold Embedded Reporting Tools Control Panel Add On.



Microsoft SQL Report Builder

You can create an RDL report using the Microsoft stand-alone Report Builder. For more details, refer to this [online documentation](#).

Visual Studio Report Server template

To create an RDL report in Visual Studio, a Report Server project is required where you can save your report definition (.rdl) file. For more details, refer to this [Visual Studio documentation](#).

If you do not have the Business Intelligence or Report Server Project options, you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create a RDLC report using business object data source

This section describes step by step procedure to create an RDLC report using Visual Studio Reporting project type.

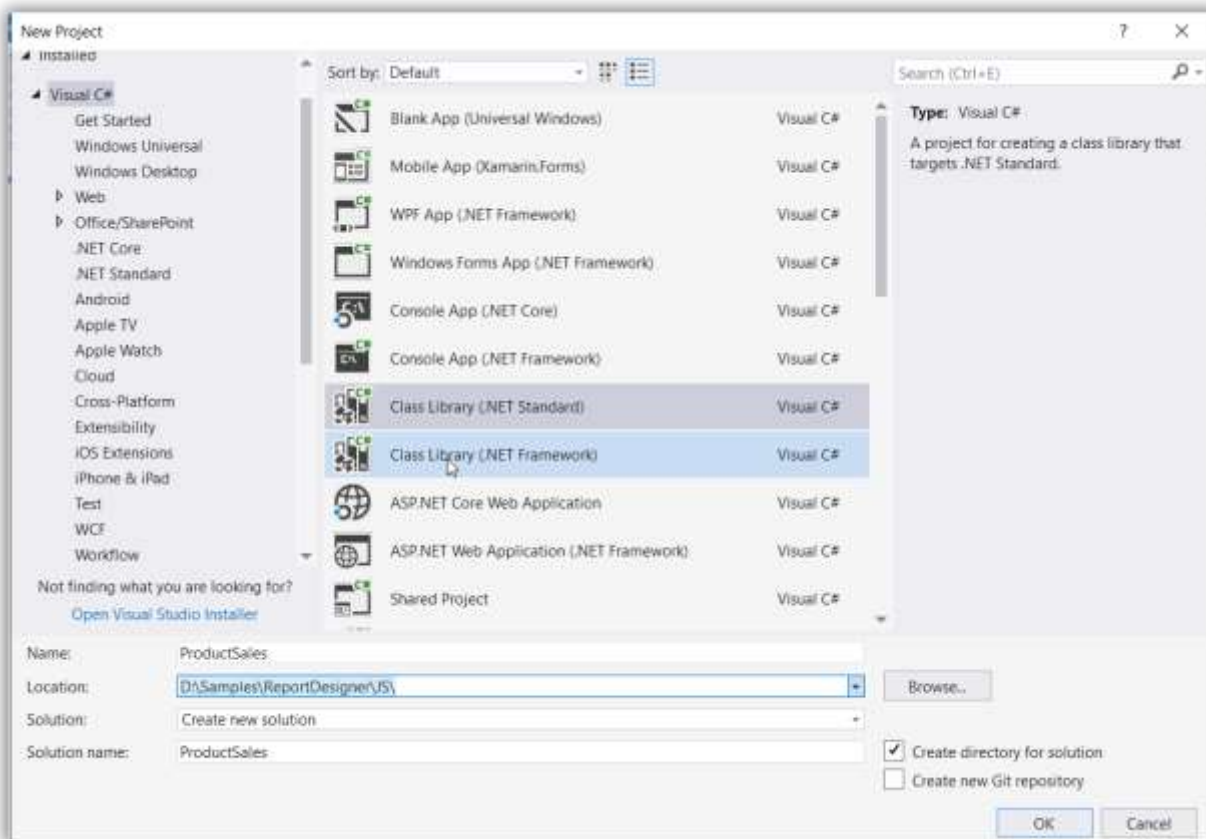
Prerequisites

- Microsoft Visual Studio 2017 or higher
- [Microsoft RDLC Report Designer](#)

If you are using Microsoft Visual Studio lower to 2017 version then you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create business object class

1. Open Visual Studio from the File menu and select **New Project**.
2. Create project with class library type from the project type list.



3. Create the class with necessary properties. You can find the reference below,

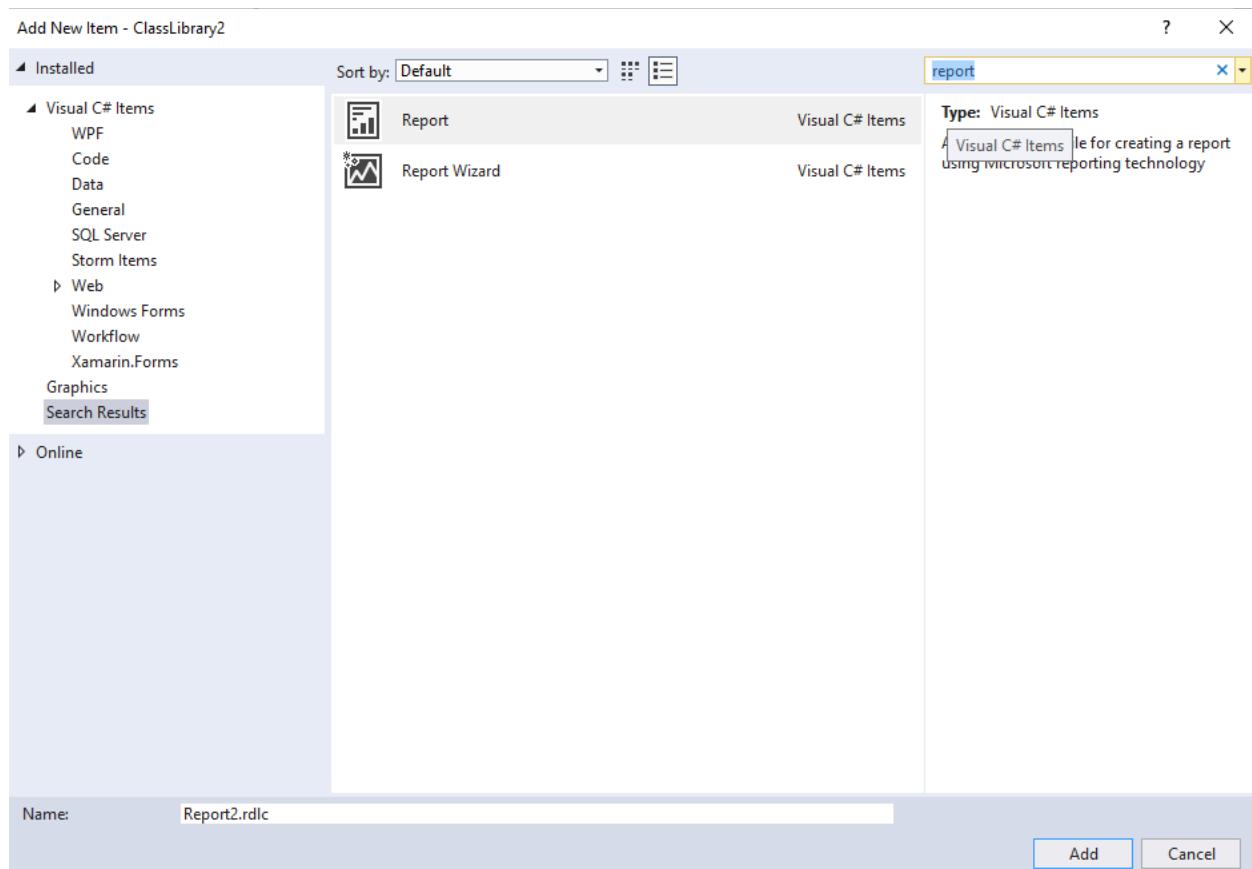
```
`csharp
public class ProductSales
{
    public string ProdCat { get; set; }
    public string SubCat { get; set; }
    public string OrderYear { get; set; }
    public string OrderQtr { get; set; }
    public double Sales { get; set; }
}
`
```

4. Clean and build the application.

Add an RDLC report

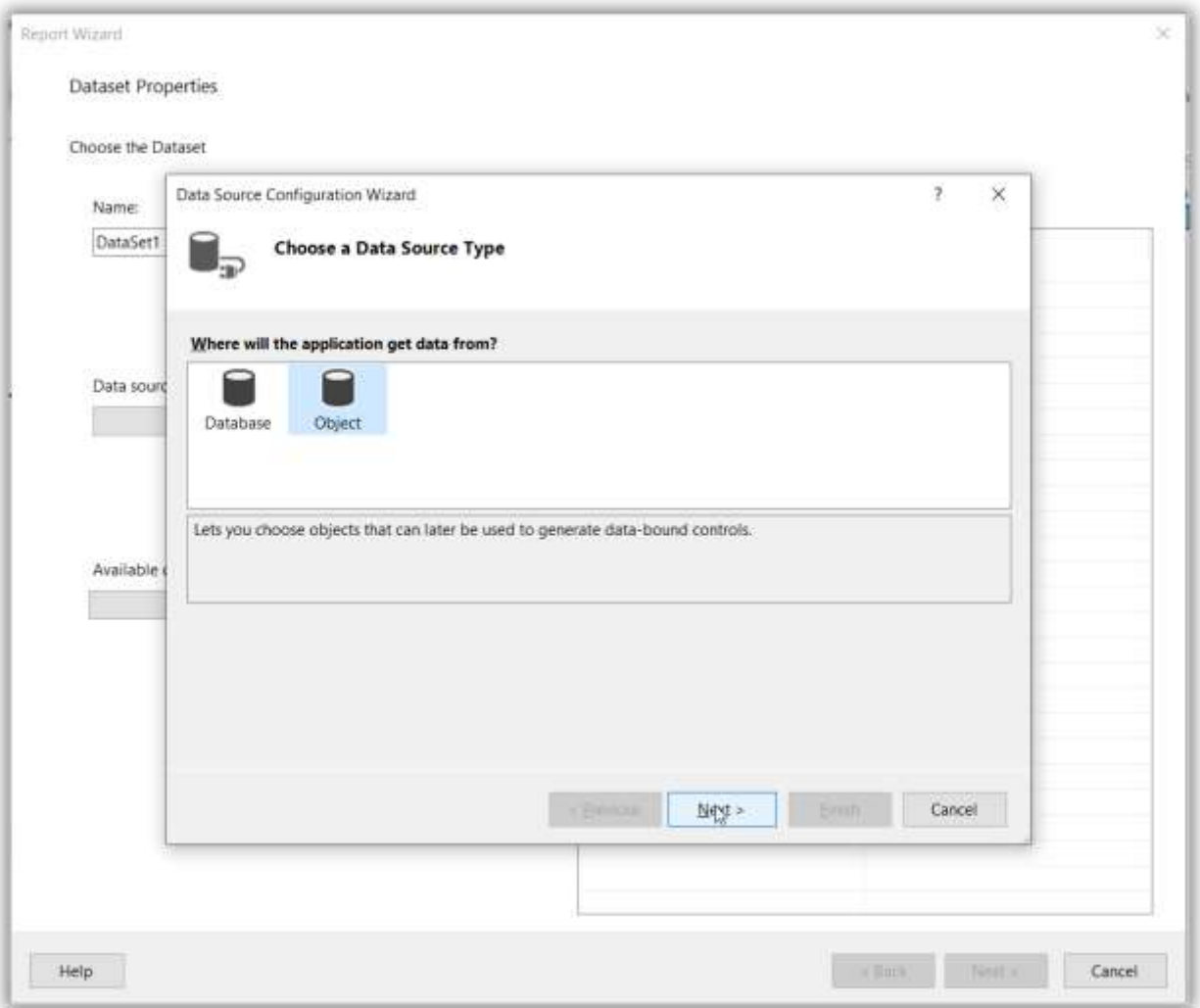
1. Right-click the project and click **Add > New Item**.

2. Search Report with new item and select **Report Wizard** to start the report creation with dataset selection.
3. Click **Add**.

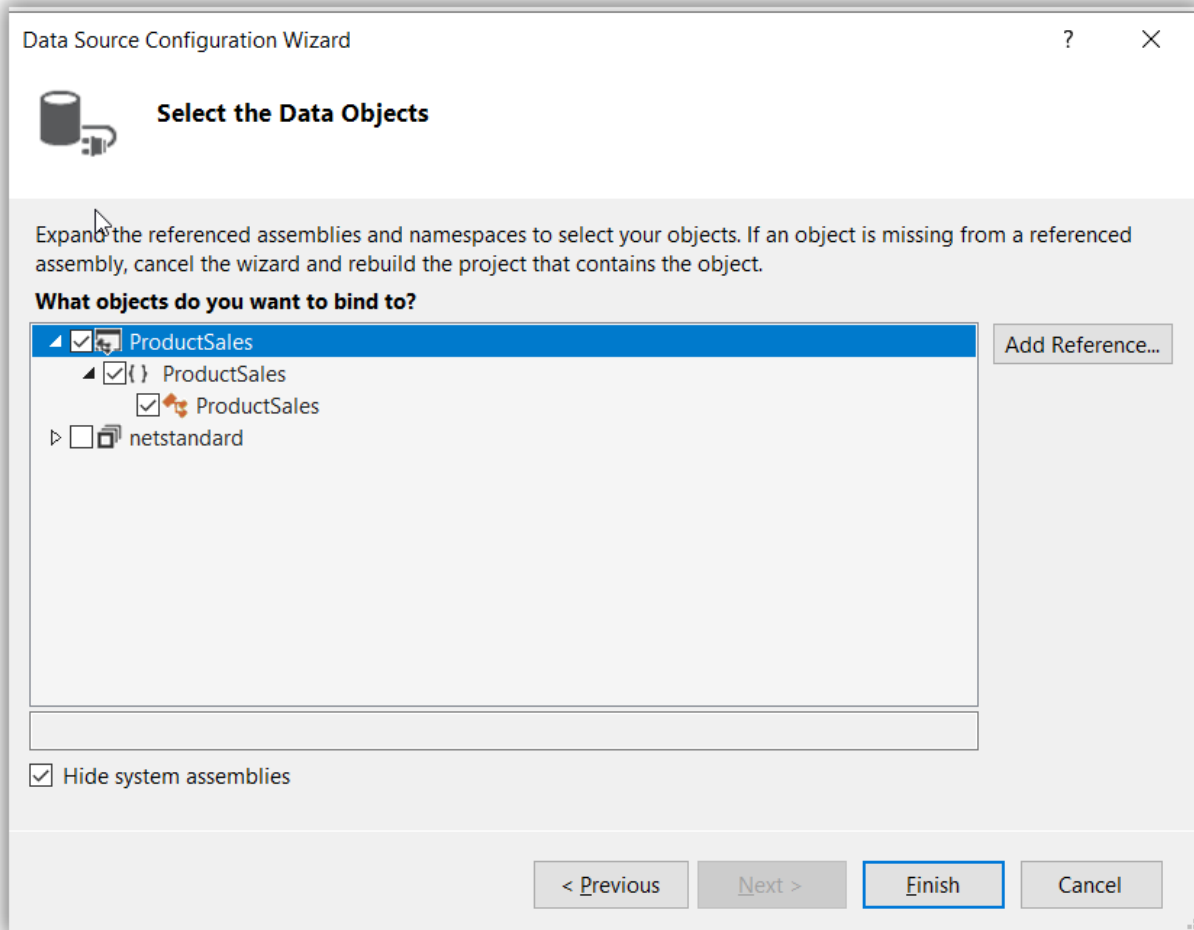


Data source and table configuration wizard

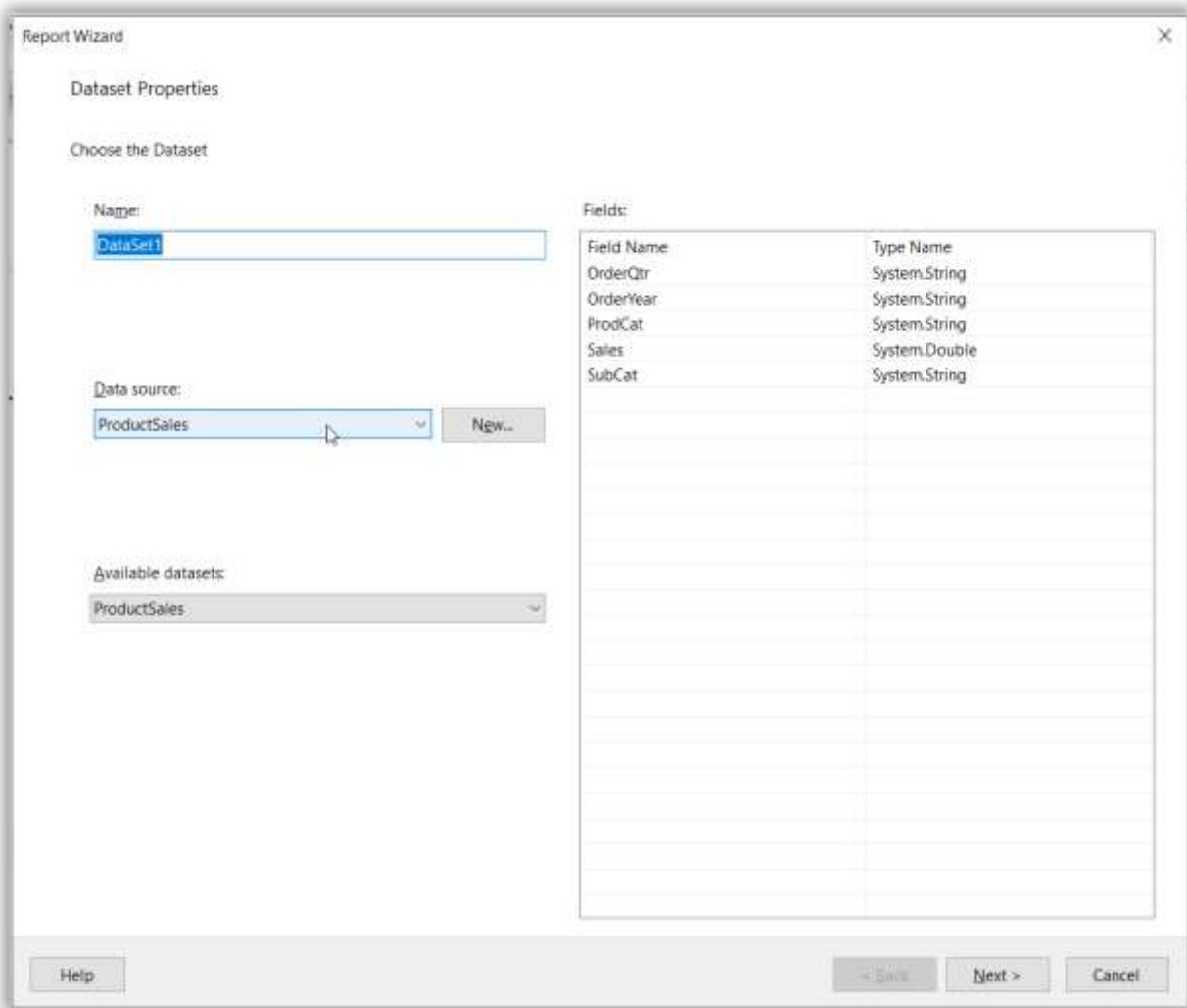
1. Choose object type from the Data Source Configuration wizard and click **Next**.



2. Expand the tree view and select **ProductSales**, and then click **Finish**.



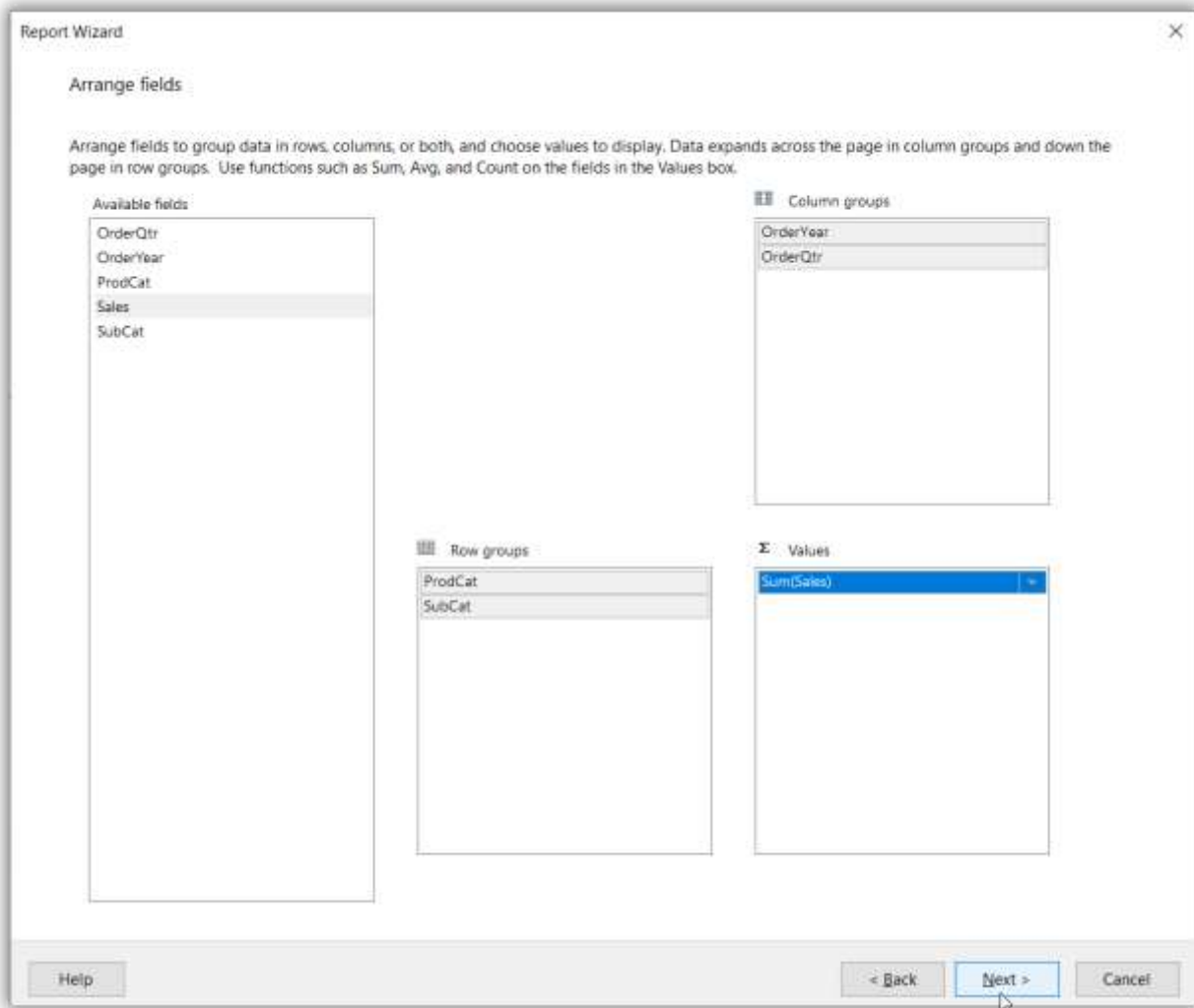
3. In the DataSet Properties wizard, specify the dataset name as **SalesData**.



The image shows the 'Report Wizard' dialog box, specifically the 'Dataset Properties' tab. The 'Choose the Dataset' section contains a 'Name:' label followed by a text box with 'DataSet1'. Below this is a 'Data source:' label followed by a dropdown menu showing 'ProductSales' and a 'New...' button. At the bottom left, there is an 'Available datasets:' label followed by a dropdown menu also showing 'ProductSales'. On the right side, there is a 'Fields:' label above a table. The table has two columns: 'Field Name' and 'Type Name'. It lists five fields: 'OrderQtr' (System.String), 'OrderYear' (System.String), 'ProdCat' (System.String), 'Sales' (System.Double), and 'SubCat' (System.String). At the bottom of the dialog, there are three buttons: 'Help', '< Back', and 'Next >' (disabled), and a 'Cancel' button.

Field Name	Type Name
OrderQtr	System.String
OrderYear	System.String
ProdCat	System.String
Sales	System.Double
SubCat	System.String

4. Drag the fields into Values, Row, and Column groups, and then click **Next**.



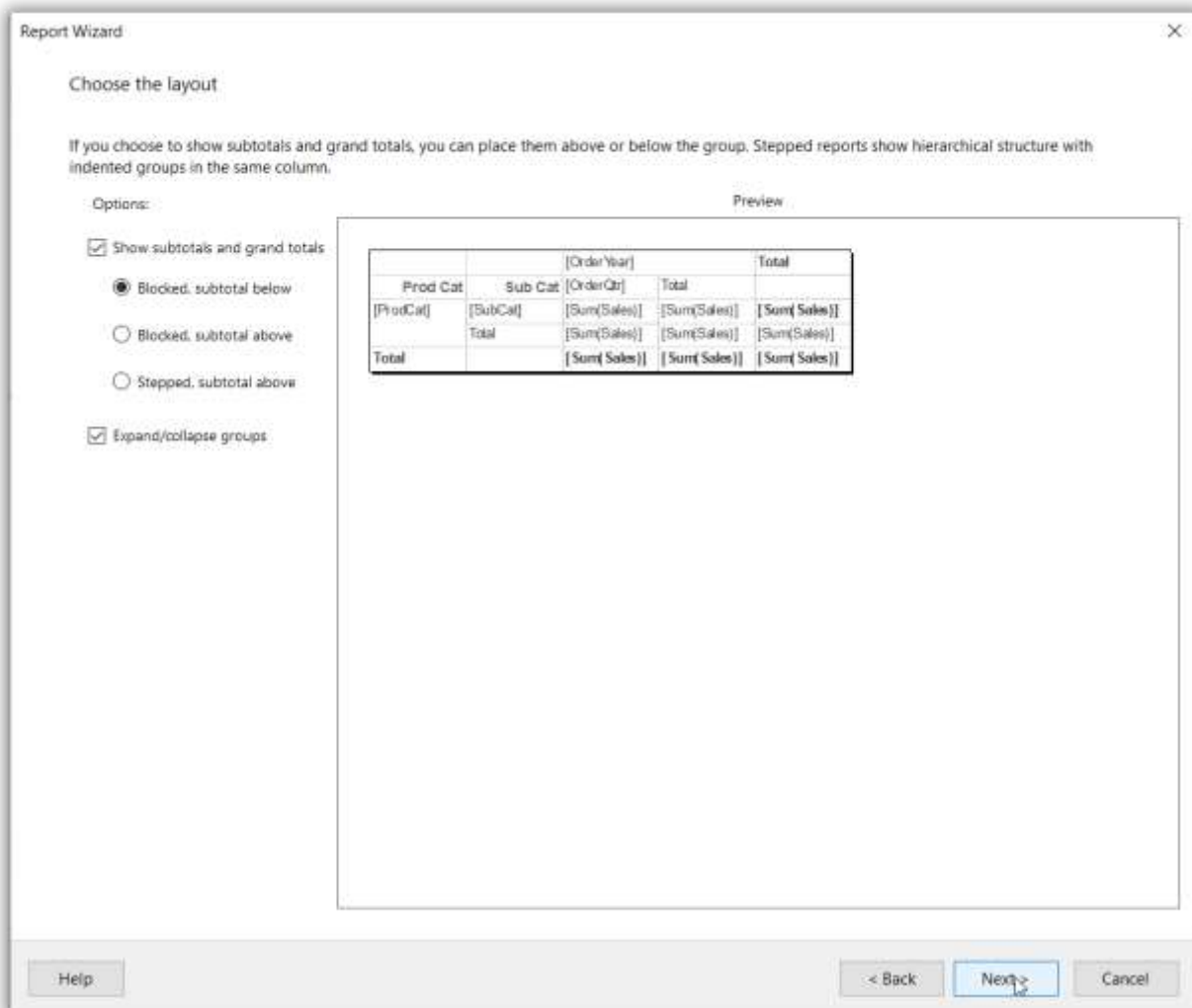
The image shows the 'Report Wizard' dialog box, specifically the 'Arrange fields' step. The dialog has a title bar with 'Report Wizard' and a close button. Below the title bar is the section 'Arrange fields' with a descriptive text: 'Arrange fields to group data in rows, columns, or both, and choose values to display. Data expands across the page in column groups and down the page in row groups. Use functions such as Sum, Avg, and Count on the fields in the Values box.'

The dialog is divided into four main areas:

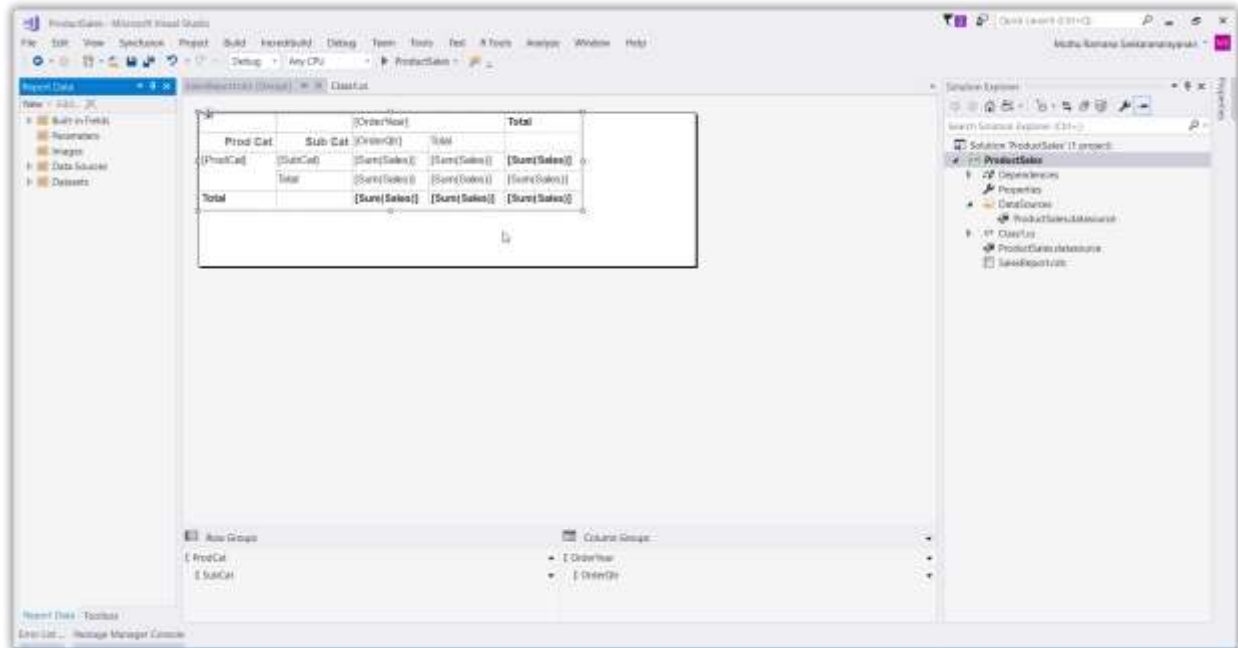
- Available fields:** A list box containing 'OrderQtr', 'OrderYear', 'ProdCat', 'Sales', and 'SubCat'. 'Sales' is currently selected.
- Column groups:** A list box containing 'OrderYear' and 'OrderQtr'.
- Row groups:** A list box containing 'ProdCat' and 'SubCat'.
- Values:** A list box containing 'Sum(Sales)'.

At the bottom of the dialog, there are three buttons: 'Help', '< Back', and 'Next >', and a 'Cancel' button on the far right. A mouse cursor is pointing at the 'Next >' button.

5. Choose the table layout and click **Next**.
6. Select table style and click **Finish**.



Now, the RDLC report is displayed in the Visual Studio as follows.



Frequently asked questions

This section helps to get the answer for the frequently asked questions in Bold Reports JavaScript Report Viewer.

1. [Is Report Viewer compatible with latest version of JQuery library?](#)
2. [Is the back action from drillthrough report will load the report again with Report Viewer?](#)

Bold Report Writer

Report Writer is a class library that enables the user to render reports defined in Microsoft's RDL format (2008 or 2008 R2) as PDF, Word, Excel or CSV documents.

The important features of WPF Report Writer are listed as follows:

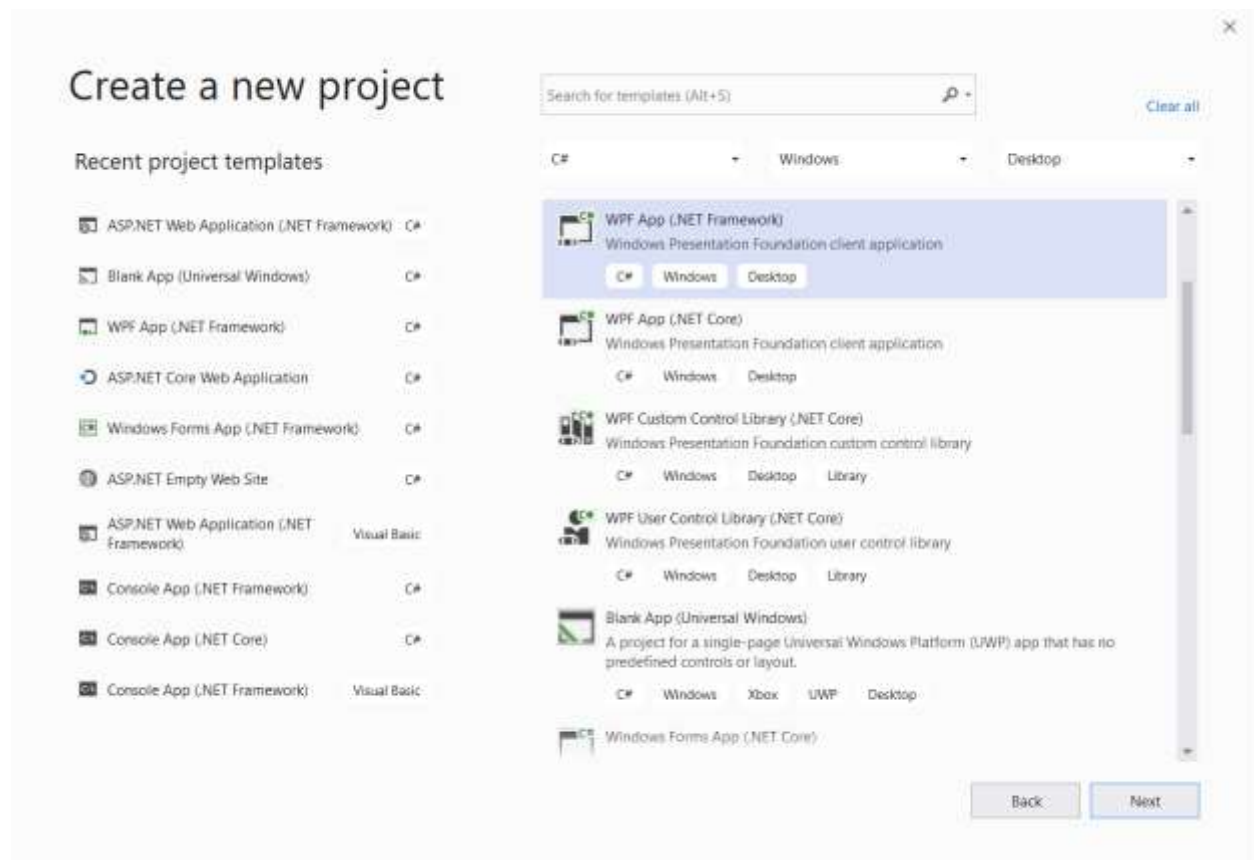
- RDL Specification - Supports RDL specification for the SQL Server 2008 and RDL specification for the SQL Server 2008 R2 only. List of available report definition formats: [msdn.microsoft.com/library/dd297486\(SQL.100\)](https://msdn.microsoft.com/library/dd297486(SQL.100)).
- Data sources - You can use advanced database servers Data Sources in Report Writer (SQL and Oracle).
- Charts - Show all basic types of charts that are available in Microsoft RDL reports.
- Tablix - Shows the summaries and simple tables.
- Gauge - Shows measurement values by using expression values.
- Textbox - Shows textbox data with expression support.
- Export - Export report as PDF, Word, Excel and CSV.
- Report Parameter - Views the report based on the report parameter value.

Export SSRS RDL Report in Bold Reports WPF Report Writer

The Report Writer is a class library that is used to export the RDL report with popular file formats like PDF, Microsoft Word, Microsoft CSV, and Microsoft Excel without previewing the report on the webpage. This section describes how to export the RDL report in WPF application using the **Report Writer**.

Create WPF application

1. Start Visual Studio 2019 and click **Create new project**.
2. Choose **WPF App (.NET Framework)**, and then click **Next**.



3. Change the project name, and then click **Create**.

List of dependency libraries

1. In the Solution Explorer tab, right-click the project or solution, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for **BoldReports.Wpf** package and install this in your WPF application. The following table provides details about the packages and their usage.

Package | Purpose

BoldReports.Wpf | Contains WPF Reporting controls (Report Viewer and Report Writer) to preview and export the reports.

Add already created reports

In this tutorial, the **sales-order-detail.rdl** report is used, and it can be downloaded at this [link](#). You can add the reports from the Syncfusion installation location. For more information, refer to the [samples and demos](#) section.

1. Create a folder **Resources** in your application to store the RDL reports.
2. Add already created reports to the newly created folder.

Initialize Report Writer

1. Open the **MainWindow.xaml** file in your application. Add the following code example in the tag.

```
`csharp
<Window
.....
.....
.....
Title="WPF Writer" WindowStyle="SingleBorderWindow" ResizeMode="NoResize"
WindowStartupLocation="CenterScreen" Width="365" Height="235">
<Grid>
</Grid>
</Window>
`
```

2. Initialize the Report writer export type inside the tag as shown below in the **MainWindow.xaml** file,

```
`csharp
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<TextBlock Grid.Row="0" TextAlignment="Justify" FontFamily="Verdana" FontSize="11"
TextWrapping="Wrap" Padding="5" >
<TextBlock.Background>
```

```
<LinearGradientBrush EndPoint="0.5,-0.04" StartPoint="0.5,1.04">
```

```
<GradientStop Color="#FFD9E9F7" Offset="0"/>
```

```
<GradientStop Color="#FFEFF8FF" Offset="1"/>
```

```
</LinearGradientBrush>
```

```
</TextBlock.Background>
```

```
<TextBlock.Text>
```

Choose a file format to view the selected document generated from Report file by using Essential Report Writer.

```
</TextBlock.Text>
```

```
</TextBlock>
```

```
<StackPanel Grid.Row="1" Orientation="Horizontal">
```

```
<RadioButton Content="PDF" Height="16" HorizontalAlignment="Left" Name="pdf" IsChecked="True" Margin="5"/>
```

```
<RadioButton Content="Excel" Height="16" HorizontalAlignment="Left" Name="excel" Margin="5" />
```

```
<RadioButton Content="Word" Height="16" HorizontalAlignment="Left" Name="word" Margin="5"/>
```

```
<RadioButton Content="HTML" Height="16" HorizontalAlignment="Left" Name="html" Margin="5"/>
```

```
<Button Click="Button_Click" Margin="10,0" Width="100" Height="25" BorderBrush="LightBlue">
```

```
<Button.Background>
```

```
<LinearGradientBrush EndPoint="0.5,-0.04" StartPoint="0.5,1.04">
```

```
<GradientStop Color="#FFD9E9F7" Offset="0"/>
```

```
<GradientStop Color="#FFEFF8FF" Offset="1"/>
```

```
</LinearGradientBrush>
```

```
</Button.Background>
```

```
<StackPanel Orientation="Horizontal">
```

```
<Image Name="image2" Margin="2" />
```

```
<TextBlock Text="Generate" Margin="3" HorizontalAlignment="Right" />
```

```
</StackPanel>
```

```
</Button>
```

```
</StackPanel>
```

```
</Grid>
```

```
,
```

Set report path

1. Open the `MainWindow.xaml.cs` file and add the following using statement.

```
`csharp
using BoldReports.Writer;
`
```

2. Initialize the ReportWriter by using the following code example in the `MainWindow.xaml.cs` file export button click event.

```
`csharp
private void Button_Click(object sender, RoutedEventArgs e)
{
    try
    {
        // Refer the RDL report file location
        string reportPath = @"<root folder>\Resources\sales-order-detail.rdl";
        string fileName = null;
        WriterFormat format;
        //Step 1 : Instantiate the report writer with the parameter "ReportPath".
        ReportWriter reportWriter = new ReportWriter(reportPath);
        //Step 2 : Save the report as Pdf or Word or Excel
        if (pdf.IsChecked == true)
        {
            fileName = "sales-order-detail.pdf";
            format = WriterFormat.PDF;
        }
        else if (word.IsChecked == true)
        {
            fileName = "sales-order-detail.docx";
            format = WriterFormat.Word;
        }
        else if (excel.IsChecked == true)
        {
            fileName = "sales-order-detail.xlsx";
            format = WriterFormat.Excel;
        }
        else
    }
}
```

```
{
fileName = "sales-order-detail.html";
format = WriterFormat.HTML;
}
reportWriter.Save(fileName, format);
//Message box confirmation to view the created report document.
if (MessageBox.Show("Do you want to view the " + format + " file?", "" + format + " report Created",
MessageBoxButton.YesNo, MessageBoxImage.Information) == MessageBoxResult.Yes)
{
//Launching the PDF file using the default Application.[Acrobat Reader]
System.Diagnostics.Process.Start(fileName);
}
}
catch (Exception Ex)
{
MessageBox.Show(Ex.Message);
}
}
```

3. Now, run and export the report with specified export format in your Report Writer application.

Congratulations! You have completed your first WPF Writer application! Click [here](#) to download the already created WPF Report Writer application.

- Exported files are saved in the application bin location.

Note: You can refer to our feature tour page for the [WPF Report Writer](#) to see its innovative features.

Frequently asked questions

This section helps to get the answer for the frequently asked questions. Discussed the following topic in this section.

- [How to print the report silently without preview the report in Report Viewer?](#)

How to print the report silently without preview the report in Report Viewer

You have to use the **Report Writer** and **PdfDocumentView** to print the reports silently without viewing in Report Viewer. Find the following steps to print the report silently:

Initialize the [Report Writer](#) with reports and data sources, use ~Save~ API to get the printable PDF in stream.

```
`csharp
string reportPath = @"../ReportTemplate/Product Details.rdlc";
ReportWriter reportWriter = new ReportWriter(reportPath);
reportWriter.ReportProcessingMode = ProcessingMode.Local;
reportWriter.DataSources.Clear();
reportWriter.DataSources.Add(new ReportDataSource { Name = "DataSet1", Value =
ProductCatalog.GetData() });
MemoryStream stream = new MemoryStream();
reportWriter.Save(stream, WriterFormat.PDF);
`
```

Load the exported stream into the PdfDocumentView using the **PdfDocumentView.Load** API and add the available printers to a list for printing the report using the Windows PrintDialog as shown in the following code example.

```
`csharp
PdfDocumentView pdfdoc = new PdfDocumentView();
pdfdoc.Load(stream);
var doc = pdfdoc.PrintDocument as IDocumentPaginatorSource;
PrintDialog printDialog = new PrintDialog();
List<string> printersList = new List<string>();
List<string> serversList = new List<string>();
var server = new PrintServer();
var queues = server.GetPrintQueues(new[] { EnumeratedPrintQueueTypes.Local });
foreach (var queue in queues)
{
    if (!serversList.Contains(queue.HostingPrintServer.Name))
    {
        serversList.Add(queue.HostingPrintServer.Name);
    }
    printersList.Add(queue.FullName);
}
```

```
}  
server = new PrintServer(serversList[0].ToString());  
PrintQueue printer1 = server.GetPrintQueue(printersList[2].ToString());  
printDialog.PrintQueue = printer1;  
printDialog.PrintDocument(doc.DocumentPaginator, "PDF PRINTER");  
`
```

Reporting tools for UWP

Enterprise-class reporting tools for UWP development to embed reporting functionalities such as viewing, exporting, and printing reports in your UWP applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

Reporting tools for UWP

Enterprise-class reporting tools for UWP development to embed reporting functionalities such as viewing, exporting, and printing reports in your UWP applications.

How to best read this user guide

The best way to get started would be to read the **Getting Started** section of the documentation for the control that you would like to start using first. The **Getting Started** guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

Now that you are familiar with the basics of using the control, the next step would be to start integrating the control into your application.

Getting help

If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please [contact us](#) by creating a support ticket.

System Requirements

This topic describes the software and hardware requirements for setting up the development environment of Bold Reports UWP.

Supported Operating Systems

- Windows 7+, 8+
- Windows Server 2008 R2+

Hardware Environments

The following hardware environments are necessary for setting up the Bold Reports UWP.

- 1 GHZ or faster, 32 bit or 64 bit processor.
- 1 GB RAM for 32 bit or 2 GB RAM for 64 bit.

Development EnvironmentS

The following development environment are necessary to run the Bold Reports UWP.

- [Microsoft Visual Studio 2017](#) or [later](#)

Framework

The below tool is required for Bold Reports UWP.

.NET Framework 4.0 or higher

See Also

- [Licensing procedure for deployment](#)

Overview

The Report Viewer is a visualization control used to display SSRS, RDL, RDLC, and Bold Report Server reports within web applications. It allows you to view RDL/RDLC reports with or without using SSRS or Bold Report Server. You can bind data sources, parameters, and render reports with all major capabilities of RDL reporting and export the report to PDF, Microsoft Excel, Microsoft Word, and HTML formats. Some of the key features are,

- Renders interactive reports with drill down, drill through, hyperlinks, and interactive sorting.
- Easily customize each element of Report Viewer and provide events for report processing customization.
- Supports jQuery, Angular, React, Blazor, ASP.NET Core, ASP.NET MVC, ASP.NET WebForms, WPF, and UWP.

Display SSRS RDL report in Bold Reports UWP Report Viewer

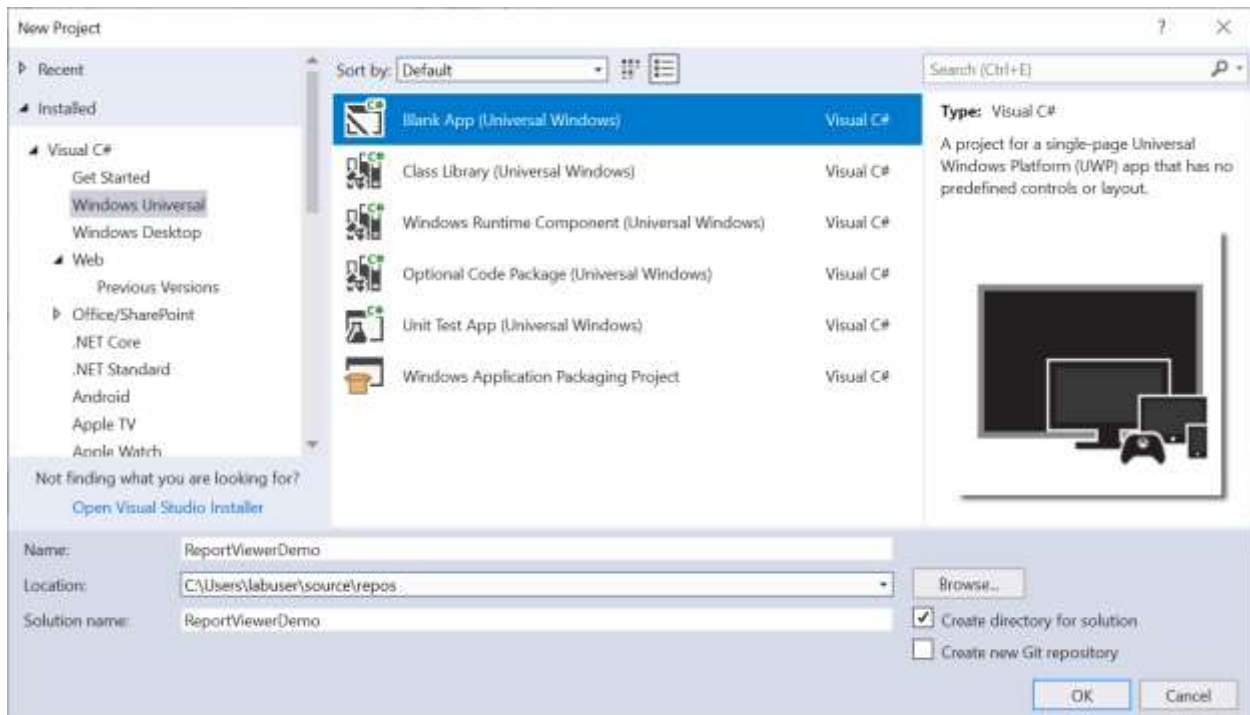
This section explains you the steps required to create your first UWP reporting application to display an already created SSRS RDL report in the Bold Reports UWP Report Viewer without using a Report Server.

To get start quickly with Report Viewer, you can check on this video:

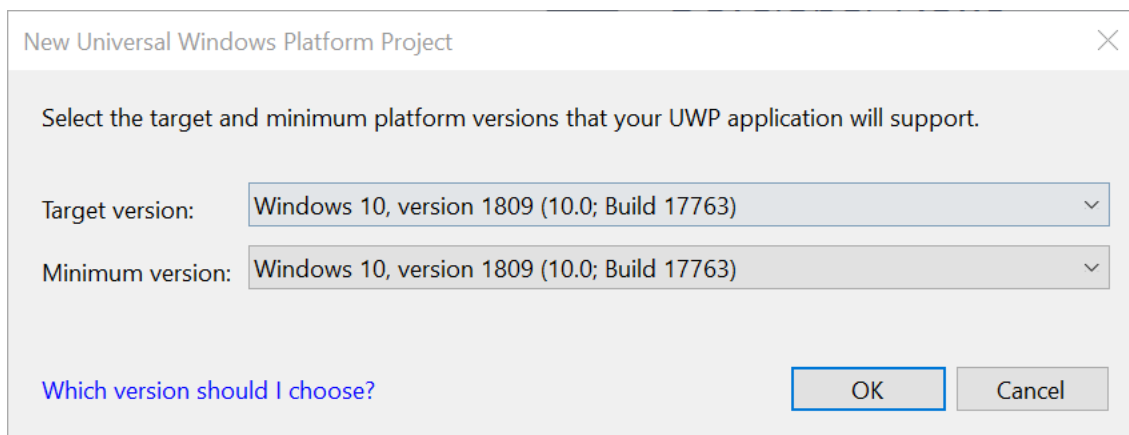
youtube: <https://youtu.be/wNIrV8LDt8E>

Create the application project

1. Open Visual Studio 2017 and select **File > New > Project**.
2. Go to **Installed > Visual C# > Windows Universal**.
3. Select **Blank App (Universal Windows)**, change the application name, and then click **OK**.



4. Select the target and minimum platform version of **Windows 10, version 1809(10.0; Build 17763)** from the dropdown, and then click **OK**.



Set the same version for the target and the minimum platform version.

Configure Report Viewer in an application

1. Right-click the project or solution on the Solution Explorer tab and choose **Manage NuGet Packages**. Alternatively, select **Tools > NuGet Package Manager > Manage NuGet Packages for Solution**.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring Report Viewer NuGet packages.

2. Search for **BoldReports.UWP** NuGet package and install them in your UWP application.

Package | Purpose

BoldReports.UWP | Contains UWP Reporting controls (Report Viewer and Report Writer) to preview and export the reports.

Initialize Report Viewer

1. Open the **MainWindow.xaml** file and import the Report Viewer namespace as shown below,

```
`csharp
xmlns:BoldReports="using:BoldReports.UI.Xaml"
`
```

2. Initialize the Report Viewer component inside the `Page` tag as shown below in the **MainWindow.xaml** file,

```
`csharp
<Page
....
....
....
xmlns:BoldReports="using:BoldReports.UI.Xaml"
....>
<Grid>
<BoldReports:ReportViewer Name="ReportViewer"/>
</Grid>
</Page>
`
```

Create Web API service

The Report Viewer requires a Web API service to process the RDL report files. You can skip this step and use the online [Web API services](#) to preview the already available reports or you should create any one of the following Web API services:

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

Adding already created report

If you have created a new service, you can add the reports from the Syncfusion installation location. For more information, refer to the [samples and demos](#) section.

1. Create a folder `Resources` in your Web API application to store RDL reports and add the already created reports to it.
2. Add already created reports to the newly created folder.

In this tutorial, the `sales-order-detail.rdl` report is used, and it can be downloaded at this [link](#).

Refer to the [create RDL report](#) section for creating new reports.

Set report path and service URL

1. Open the `MainWindow.xaml.cs` file.
2. Initialize the window loaded event inside the `MainWindow()` constructor.

```
`csharp
public MainPage()
{
    this.InitializeComponent();
    this.Loaded += MainPage_Loaded;
}

private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
}
`
```

3. Set the `ReportPath` and `ReportServiceUrl` properties in the `MainWindow_Loaded` event method and invoke the `RefreshReport()` method to render the report.
4. You can replace the following code in your `MainWindow_Loaded` event method.

```
`csharp
private void MainPage_Loaded(object sender, RoutedEventArgs e)
```

```

{
this.ReportViewer.ReportPath = "~/Resources/docs/sales-order-dtail.rdl";
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.RefreshReport();
}

```

In the above code, the `sales-order-detail.rdl` report and `reportServiceUrl` used from online URL.

Preview the report

Build and run the application to view the report output in the Report Viewer as displayed in the following screenshot.

The screenshot displays the ReportViewerDemo application window. The report is titled "Sales Order" and features the Adventure Works Cycles logo. It contains the following information:

Order Details:

Order ID	Billing Date	Order Date	Purchase Order	Shipment Method
#SO50750	21-05-2019	01-06-2003	PO7192170677	CARGO TRANSPORT 5

Billing Address:
Jean Handley
 Central Discount Store
 259826 Russell Rd. South
 Kent, Washington
 98031
 United States

Shipping Address:
Jean Handley
 Central Discount Store
 259826 Russell Rd. South
 Kent, Washington
 98031
 United States

Contact:
 Jean Handley
 Ph: 582-555-0113

Item Details Table:

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1098	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LJ-0192-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.84	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	BK-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	BK-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,802.57

Note: You can refer to our feature tour page for the [UWP Report Viewer](#) to see its innovative features.

See Also
[Create RDLC report](#)

[Migrate Report Viewer](#)

[List of SSRS server versions are supported in Bold Reports](#)

Load SSRS Report Server reports

Report Viewer has support to load RDL reports from SSRS Report Server. To render SSRS Reports set the `ReportServerUrl` and `ReportServerCredential` properties as in the following code snippet.

```
`csharp
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
this.ReportViewer.ReportServerCredential = new System.Net.NetworkCredential("ssrs", "RDLReport1");
this.ReportViewer.ReportPath = "/SSRSSamples/Territory Sales"; //The report path should be in the
format of "/folder name/report name"
this.ReportViewer.RefreshReport();
`
```

Set data source credential for shared data sources

1. The SSRS Report Server does not provide options to get credential information of the report data source deployed on the SSRS server. If the report has any data source that uses credentials to connect with the database, then you must specify the `DataSourceCredentials` for each report data source to establish database connection.

```
`csharp
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
this.ReportViewer.ReportPath = "/SSRSSamples/Territory Sales"; //The report path should be in the
format of "/folder name/report name"
this.ReportViewer.RefreshReport();
`
```

2. To set shared datasource credentials in Web API Controller, use the following code in the `OnReportLoaded` method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Add SSRS Report Server and data source credentials
}
```

```
reportOption.ReportModel.ReportServerCredential = new System.Net.NetworkCredential("ssrs",
"RDLReport1");
```

```
reportOption.ReportModel.DataSourceCredentials.Add(new
BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
```

```
}
```

```
,
```

Render linked reports

You can render a linked report that point to an existing report, which is published in the SSRS Report Server. Also, it is possible to set the parameter, data source, credential, and other properties as like normal SSRS reports using the Report Viewer.

```
`csharp
```

```
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
```

```
this.ReportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
```

```
this.ReportViewer.ReportServerCredential = new System.Net.NetworkCredential("ssrs", "RDLReport1");
```

```
this.ReportViewer.ReportPath = "/SSRSSamples/Territory Sales_Link"; //The report path should be in
the format of "/folder name/report name"
```

```
this.ReportViewer.RefreshReport();
```

```
,
```

The `Territory Sales_Link` is a linked report created for `Territory Sales` report that is already available on the SSRS Report Server.

Load SharePoint Server reports

To render SharePoint integrated SSRS reports set the `ReportPath`, `ReportServiceURL`, `ReportServerUrl` and `ReportServerFormsCredential` properties as in the following code snippet.

```
`csharp
```

```
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
```

```
this.ReportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
```

```
this.ReportViewer.ReportServerFormsCredential = new
BoldReports.UI.Xaml.ReportServerFormsCredential("ssrs", "RDLReport1");
```

```
this.ReportViewer.ReportPath =
"http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl";
```

```
this.ReportViewer.RefreshReport();
```

```
,
```

Set data source credential for shared data sources

1. The shared data source credentials can be added to the `DataSourceCredentials` property to connect with the database.

```
`csharp
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.ReportServerUrl = "http://<servername>/reportserver$instanceName";
this.ReportViewer.ReportPath =
"http://<servername>/reportserver$instanceName/SSRSSamples/Territory Sales.rdl";
this.ReportViewer.RefreshReport();
`
```

2. To set shared datasource credentials in Web API Controller, use the following code in the `OnReportLoaded` method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    //Add ReportServerFormsCredential and data source credentials
    reportOption.ReportModel.ReportServerFormsCredential = new
    BoldReports.Web.ReportServerFormsCredential("ssrs", "RDLReport1");
    reportOption.ReportModel.DataSourceCredentials.Add(new
    BoldReports.Web.DataSourceCredentials("<database>", "ssrs1", "RDLReport1"));
}
`
```

Data source credentials must be added for shared data sources that do not have credentials in the connection strings.

Render RDLC report

The data binding support, allows you to view RDLC reports that exist on the custom business object data collection. The following steps demonstrates how to render a RDLC report with custom business object data collection.

Add the RDLC report `product-list.rdlc` from Bold Reports installation location to your application `Resources` folder. For more information, see [Samples and demos](#).

- Assign the `processingMode` property to `ProcessingMode.Local`.
- To load report as a stream, create a report stream using the `Stream` class and assign the report stream to the `LoadReport()` method.
- Bind the business object data values collection by adding new item to the `DataSources` as in the following code snippet.

```
`csharp
```

```
Assembly assembly = typeof(MainPage).GetTypeInfo().Assembly;
Stream reportStream = assembly.GetManifestResourceStream("ReportViewer.Resources.product-
list.rdlc");
this.ReportViewer.ProcessingMode = BoldReports.UI.Xaml.ProcessingMode.Local;
this.ReportViewer.LoadReport(reportStream);
this.ReportViewer.DataSources.Clear();
this.ReportViewer.DataSources.Add(new BoldReports.UI.Xaml.ReportDataSource { Name = "list", Value
= ProductList.GetData() });
this.ReportViewer.RefreshReport();
```

.....

```
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
    public string ProductImage { get; set; }
    public static IList GetData()
    {
        List<ProductList> datas = new List<ProductList>();
        ProductList data = null;
        data = new ProductList()
        {
            ProductName = "Baked Chicken and Cheese",
            OrderId = "323B60",
            Price = 55,
            Category = "Non-Veg",
            Ingredients = "grilled chicken, corn and olives.",
            ProductImage = ""
        };
        datas.Add(data);
        data = new ProductList()
        {
```

```

ProductName = "Chicken Delite",
OrderId = "323B61",
Price = 100,
Category = "Non-Veg",
Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
ProductImage = ""
};
datas.Add(data);
data = new ProductList()
{
    ProductName = "Chicken Tikka",
    OrderId = "323B62",
    Price = 64,
    Category = "Non-Veg",
    Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
    ProductImage = ""
};
datas.Add(data);
return datas;
}
}
`

```

Render subreport

You can display another report inside the body of a main report using the Report Viewer. The following steps help you to customize the subreport properties such as data source, report path, and parameters.

- Add the sub report and main reports to your application **Resources** folder. In this tutorial, using the already created reports. Refer to the [Create RDL Report section](#) or [Create RDLC Report section](#) for creating new reports.

Download the **side-by-side-main-report.rdl**, **side-by-side-sub-report.rdl** reports from [here](#). Also, you can add the report from Syncfusion installation location. For more information, see [Samples and demos](#). The reports used from installed location, requires **NorthwindIO_Reports.sdf** database to run, so add it to your application.

- Set the main report path **ReportPath** and **ReportServiceURL** properties of the Report Viewer as in following code snippet.

```
`csharp
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.ReportPath = "~/Resources/docs/side-by-side-main-report.rdl";
this.ReportViewer.RefreshReport();
`
```

Load subreport stream

To load subreport as stream, set the sub report stream in **LoadSubreport()** method of the Report Viewer as in following code snippet.

```
`csharp
Assembly assembly = typeof(MainPage).GetTypeInfo().Assembly;
Stream mainReportStream =
assembly.GetManifestResourceStream("ReportViewer_UWP.Resources.product-list-main.rdlc");
Stream subReportStream =
assembly.GetManifestResourceStream("ReportViewer_UWP.Resources.product-list.rdlc");
this.ReportViewer.ProcessingMode = BoldReports.UI.Xaml.ProcessingMode.Local;
this.ReportViewer.SubreportProcessing += (sen, arg) =>
{
    arg.DataSources.Clear();
    arg.DataSources.Add(new BoldReports.UI.Xaml.ReportDataSource { Name = "list", Value =
    ProductList.GetData() });
};
this.ReportViewer.LoadSubreport("product-list", subReportStream);
this.ReportViewer.LoadReport(mainReportStream);
this.ReportViewer.RefreshReport();
.....
public class ProductList
{
    public string ProductName { get; set; }
    public string OrderId { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }
    public string Ingredients { get; set; }
}
```

```
public string ProductImage { get; set; }
public static IList GetData()
{
    List<ProductList> datas = new List<ProductList>();
    ProductList data = null;
    data = new ProductList()
    {
        ProductName = "Baked Chicken and Cheese",
        OrderId = "323B60",
        Price = 55,
        Category = "Non-Veg",
        Ingredients = "grilled chicken, corn and olives.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Delite",
        OrderId = "323B61",
        Price = 100,
        Category = "Non-Veg",
        Ingredients = "cheese, chicken chunks, onions & pineapple chunks.",
        ProductImage = ""
    };
    datas.Add(data);
    data = new ProductList()
    {
        ProductName = "Chicken Tikka",
        OrderId = "323B62",
        Price = 64,
        Category = "Non-Veg",
        Ingredients = "onions, grilled chicken, chicken salami & tomatoes.",
        ProductImage = ""
    };
    datas.Add(data);
}
```

```
};
datas.Add(data);
return datas;
}
}
`
```

Report parameters

Provides property options to pass or set report parameters default values at run-time using the parameters property. You can set the report parameters while creating the Report Viewer control in a application loaded.

In this tutorial, the `sales-order-dtail.rdl` report is used, and it can be downloaded from [here](#).

Set report parameter

1. Set the default value data to the `Values` property and name of the report parameter to the `Name` property.

The parameter name is case sensitive, it should be same as available in the report definition.

2. The following code example illustrates how to set report parameter in the `ReportLoaded` event.

```
`csharp
this.ReportViewer.ReportLoaded += (sen, arg) =>
{
    List<BoldReports.UI.Xaml.ReportParameter> parameters = new
    List<BoldReports.UI.Xaml.ReportParameter>();

    BoldReports.UI.Xaml.ReportParameter parameter = new BoldReports.UI.Xaml.ReportParameter();
    parameter.Name = "SalesOrderNumber";
    parameter.Values = new List<string>() { "SO50756" };
    parameters.Add(parameter);
    this.ReportViewer.SetParameters(parameters);
};

this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.ReportPath = "~/Resources/docs/sales-order-detail.rdl";
this.ReportViewer.RefreshReport();
`
```

Set report parameter in Web API

1. Set the report path **ReportPath** and **ReportServiceURL** properties of the Report Viewer as in following code snippet.

```
`csharp
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.ReportPath = "~/Resources/docs/sales-order-dtail.rdl";
this.ReportViewer.RefreshReport();
`
```

2. To set parameter default value in Web API Controller, use the following code in the **OnReportLoaded** method.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    List<BoldReports.Web.ReportParameter> userParameters = new
    List<BoldReports.Web.ReportParameter>();
    userParameters.Add(new BoldReports.Web.ReportParameter()
    {
        Name = "SalesOrderNumber",
        Values = new List<string>() { "SO50756" }
    });
    reportOption.ReportModel.Parameters = userParameters;
}
`
```

Get report parameter

Methods | Description

GetParameters | Returns the parameters that are used in the current report without the processed values.

GetParametersWithValues | Returns the report parameters with processed data values that are used in the current report.

You can use the following code sample to get parameter names and set parameter default values.

1. Set the report path **ReportPath** and **ReportServiceURL** properties of the Report Viewer as in following code snippet.

```
`csharp
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.ReportPath = "~/Resources/docs/sales-order-dtail.rdl";
this.ReportViewer.RefreshReport();
`
```

2. To set parameter default value based on the parameter name in Web API Controller, use the following code in the **OnReportLoaded** method.

```
`csharp
public class ReportViewerController : ApiController, IReportController
{
    Dictionary<string, object> jsonArray = null;
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        jsonArray = jsonResult;
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    ....
    [NonAction]
    public void OnReportLoaded(ReportViewerOptions reportOption)
    {
        var reportParameters = ReportHelper.GetParameters(jsonArray, this);
        List<BoldReports.Web.ReportParameter> setParameters = new
        List<BoldReports.Web.ReportParameter>();
        if (reportParameters != null)
        {
            foreach (var rptParameter in reportParameters)
            {
                setParameters.Add(new BoldReports.Web.ReportParameter()
                {
                    Name = rptParameter.Name,
                    Values = new List<string>() { "SO50756" }
                });
            }
        }
    }
}
```

```
reportOption.ReportModel.Parameters = setParameters;
}
}
}
`
```

Report interaction events

You can handle the report processing actions and interact with reports using the following events.

- ReportLoaded
- ReportError
- Drill through

Report loaded

The **ReportLoaded** event fires once the report loading is completed and ready to start the processing. You can handle the event and specify data source, parameters at client-side. The following sample code loads a report by assigning report data source input in the **ReportLoaded** event.

In this tutorial, **product-list.rdlc** report is used, you can add the report from Bold Reports installation location. For more information, see [Samples and demos](#).

```
`csharp
Assembly assembly = typeof(MainPage).GetTypeInfo().Assembly;
Stream reportStream = assembly.GetManifestResourceStream("ReportViewer_UWP.Resources.product-list.rdlc");
this.ReportViewer.ReportLoaded += (sen, arg) =>
{
    this.ReportViewer.DataSources.Clear();
    this.ReportViewer.DataSources.Add(new BoldReports.UI.Xaml.ReportDataSource { Name = "list", Value = ProductList.GetData() });
};
this.ReportViewer.ProcessingMode = BoldReports.UI.Xaml.ProcessingMode.Local;
this.ReportViewer.LoadReport(reportStream);
this.ReportViewer.RefreshReport();
`
```

Report error

When an error occurs in the report processing, it raises the **ReportError** event. You can handle the event and show the details in your custom dialog instead of component default error detail interface.

```
`csharp
```

```

this.ReportViewer.ProcessingMode = BoldReports.UI.Xaml.ProcessingMode.Local;
Assembly assembly = typeof(MainPage).GetTypeInfo().Assembly;
Stream reportStream = assembly.GetManifestResourceStream("ReportViewer_UWP.Resources.product-
list.rdlc");
this.ReportViewer.ReportError += (sen, arg) =>
{
    //Handle your report error options here.
}
//this.ReportViewer.LoadReport(reportStream);
this.ReportViewer.RefreshReport();
`

```

Drill through

When a drill through item is selected in a report, it invokes the **DrillThroughReport** event. You can change the drill through arguments such as report parameter and data sources. The following sample code can be used to change the drill through report name and set the parameter value before the drill through report is rendered.

```

`csharp
this.ReportViewer.DrillThroughReport += (sen, arg) =>
{
    List<BoldReports.UI.Xaml.ReportParameter> parameters = new
    List<BoldReports.UI.Xaml.ReportParameter>();
    BoldReports.UI.Xaml.ReportParameter parameter = new BoldReports.UI.Xaml.ReportParameter();
    parameter.Name = "EmployeeID";
    parameter.Values = new List<string>() { "4" };
    parameters.Add(parameter);
    //Assign the drill through report path and parameter
    arg.ReportPath = "~/Resources/docs/personal-details.rdl";
    arg.Parameters = parameters;
};
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.ReportPath = "~/Resources/docs/sales-person-details.rdl";
this.ReportViewer.RefreshReport();
`

```

Error logging in UWP Report Viewer

If an error occurred in report processing, UWP Report Viewer displays short messages about the error in view. In report viewer `ReportError` event raised. You can register the event and get the report error details from the event arguments to save all logs and error information into a physical file location.

This section explains how to log the detailed error information to your UWP application.

This section requires a UWP Report Viewer application, if you don't have then create using the [Getting-Started](#).

1. In Solution Explorer, Open report viewer initialization cs file.
2. Register the `ReportError` event.

```
`csharp
private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    this.ReportViewer.ReportError += ReportViewer_ReportError;
}
private void ReportViewer_ReportError(object sender, ReportErrorEventArgs e)
{
    // You can register the report errors using event arguments.
}
`
```

3. Create a method in `MainPage.xaml.cs` to write the error text into application folder.

```
`csharp
private async void WriteLogs(string errorMessage)
{
    var appFolder = Windows.Storage.ApplicationData.Current.LocalFolder;
    //you can refer to your preferred location path for errordetails text file.
    var filePath = await appFolder.CreateFileAsync("Errordetails.txt",
        Windows.Storage.CreationCollisionOption.OpenIfExists);
    await Windows.Storage.FileIO.AppendTextAsync(filePath, errorMessage + Environment.NewLine);
}
`
```

4. Invoke the newly created function in `ReportViewer_ReportError` as follows.

```
`csharp
```



```

private void ReportViewer_ReportError(object sender, ReportErrorEventArgs e)
{
    string errorLog;
    if (e.Exception != null)
    {
        errorLog = (string.Format("Error Message: {0} \n Stack Trace: {1}", e.Message, e.Exception.StackTrace));
    }
    else
    {
        errorLog = e.Message;
    }
    WriteLogs(errorLog);
}
`

```

In cases of any issues faced in the report rendering, share the log file to our technical support team to get assistance on that.

5. The final `MainPage.xaml.cs` file is given as follows, you can replace it in your application.

```

`csharp
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
        this.Loaded += MainPage_Loaded;
    }
    private void MainPage_Loaded(object sender, RoutedEventArgs e)
    {
        this.ReportViewer.ReportPath = "~/Resources/docs/sales-order-dtail.rdl";
        this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
        this.ReportViewer.ReportError += ReportViewer_ReportError;
        this.ReportViewer.RefreshReport();
    }
    private void ReportViewer_ReportError(object sender, ReportErrorEventArgs e)

```

```

{
string errorLog;
if (e.Exception != null)
{
errorLog = (string.Format("Error Message: {0} \n Stack Trace: {1}", e.Message, e.Exception.StackTrace));
}
else
{
errorLog = e.Message;
}
WriteLogs(errorLog);
}

private async void WriteLogs(string errorMessage)
{
var appFolder = Windows.Storage.ApplicationData.Current.LocalFolder;
//you can refer to your preferred location path for errordetails text file.
var filePath = await appFolder.CreateFileAsync("Errordetails.txt",
Windows.Storage.CreationCollisionOption.OpenIfExists);
await Windows.Storage.FileIO.AppendTextAsync(filePath, errorMessage + Environment.NewLine);
}
}
`

```

Print report

The Report Viewer provides print report option in the toolbar to print a copy of the report. The page setup dialog allows you to set the paper size or other page setup properties. To see print margins, click Print Layout on the toolbar.

The values allow you to set in the Page Setup dialog box for current session only. When you close the report and reopen it, it will have the default values again. The default values for the page setup dialog come from the report properties, which are set in the design view.

Remove empty spaces in printing

The extra blank page is created when the Body of your report is too wide for your page. If you want the report to appear on a single page, all the content within the report body must fit on the physical page and the body width should be lesser or equal to the following formula:

Body Width <= Page Width - (Left Margin + Right Margin)

For more details on designing a report to remove the empty pages in the report, refer to the knowledge base article of [report page sizing](#).

Export report

The Report Viewer provides events and properties to control and customize the report exporting functionality.

PDF export options

The **PDFOptions** provides properties to manage PDF export behaviors. You have to set the properties in the application window loaded method.

Export with complex scripts

To export reports with the complex scripts, set the **EnableComplexScript** property of **PDFOptions** instance to true.

```
`csharp
this.ReportViewer.PDFOptions = new BoldReports.Writer.PDFOptions();
this.ReportViewer.PDFOptions.EnableComplexScript = true;
`
```

PDF Conformance

You can export the report as **PDF/A-1b** document by specifying the conformance level **PdfConformanceLevel.Pdf_A1B** in the **PdfConformanceLevel** property.

```
`csharp
this.ReportViewer.PDFOptions = new BoldReports.Writer.PDFOptions();
this.ReportViewer.PDFOptions.PdfConformanceLevel = Syncfusion.Pdf.PdfConformanceLevel.Pdf_A1B;
`
```

Add custom PDF fonts

This allows you to have custom fonts in the PDF exported document by adding the font streams to **Fonts** collection in **PDFOptions** instance.

1. Add the font **.ttf** files into your application **Resources** folder.
2. In the Solution Explorer, open the properties of the font file and set the property Copy to Output Directory as Copy always.
3. Initialize the **Font** collection and add the font stream to it.

The key value provided in the font collection should be same as in the report item font property.

```
`csharp
this.ReportViewer.PDFOptions = new BoldReports.Writer.PDFOptions()
{
    Fonts = new Dictionary<string, System.IO.Stream>
    {
```

```
{ "Segoe UI", assembly.GetManifestResourceStream("ReportViewerUWP.Resources.fontsymbols.ttf") }
}
};
`
```

Any fonts used in the report definition that is not installed or available in the local system, then you must load the font stream. In the above code, loaded `font_symbols` font stream to export `sales-order-detail.rdl` report as symbols.

Word export options

The `WordOptions` provides properties to manage Word document export behaviors.

Word document type

You can save the report to the required document version by setting the `FormatType` property.

```
`csharp
this.ReportViewer.WordOptions = new BoldReports.Writer.WordOptions();
this.ReportViewer.WordOptions.FormatType = BoldReports.Writer.WordFormatType.Docx;
`
```

Word document advance layout for merged cells

Eliminate the tiny columns, rows, merged cells, and render the word document elements without nested grid layout by setting the `LayoutOption` as `TopLevel`. The `ParagraphSpacing` is the distance value added between two elements in the document.

```
`csharp
this.ReportViewer.WordOptions = new BoldReports.Writer.WordOptions();
this.ReportViewer.WordOptions.LayoutOption = BoldReports.Writer.WordLayoutOptions.TopLevel;
this.ReportViewer.WordOptions.ParagraphSpacing = new BoldReports.Writer.ParagraphSpacing()
{
    Bottom = 0.5f,
    Top = 0.5f
};
`
```

A paragraph element is inserted between two tables in the exported document to overcome word document auto merging behavior.

The table in word document is not a stand-alone object, if you draw two tables one after another, it will automatically get merged into a single table. To prevent this merging, added an empty paragraph between two tables.

Protecting Word document from editing

You can restrict a Word document from editing either by providing a password or without a password. The following are the types of protection.

1. **AllowOnlyComments**: You can add or modify only the comments in the Word document.
2. **AllowOnlyFormFields**: You can modify the form field values in the Word document.
3. **AllowOnlyRevisions**: You can accept or reject the revisions in the Word document.
4. **AllowOnlyReading**: You can only view the content in the Word document.
5. **NoProtection**: You can access or edit the Word document contents as normally.

```
`csharp
```

```
this.ReportViewer.WordOptions = new BoldReports.Writer.WordOptions();
this.ReportViewer.WordOptions.ProtectionType = Syncfusion.DocIO.ProtectionType.AllowOnlyReading;
`
```

Excel export options

The **ExcelOptions** provides properties to manage Excel document export behaviors.

Excel document type

You can save the report to the required excel version by setting the **ExcelSaveType** property.

```
`csharp
```

```
this.ReportViewer.ExcelOptions = new BoldReports.Writer.ExcelOptions();
this.ReportViewer.ExcelOptions.ExcelSaveType = BoldReports.Writer.ExcelVersion.Excel2013;
`
```

Excel document advance layout for merged cells

Eliminate the tiny columns, rows, and merged cells to provide clear readability and perform data manipulations by setting the **LayoutOption** as **IgnoreCellMerge**.

```
`csharp
```

```
this.ReportViewer.ExcelOptions = new BoldReports.Writer.ExcelOptions();
this.ReportViewer.ExcelOptions.LayoutOption =
BoldReports.Writer.ExcelLayoutOptions.IgnoreCellMerge;
`
```

Protecting Excel document from editing

You can restrict the Excel document from editing either by providing the **ExcelSheetProtection** or enabling the **ReadOnlyRecommended** properties.

```
`csharp
```

```
this.ReportViewer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    ReadOnlyRecommended = true,
    ExcelSheetProtection = Syncfusion.XlsIO.ExcelSheetProtection.DeletingColumns
};
`
```

PowerPoint export options

You can save the report to the required PowerPoint version by setting the `FormatType` property.

```
`csharp
this.ReportViewer.PPTOptions = new BoldReports.Writer.PPTOptions();
this.ReportViewer.PPTOptions.FormatType = BoldReports.Writer.PPTSaveType.PowerPoint2013;
`
```

CSV export options

The `CsvOptions` allows you to change encoding, delimiters, qualifiers, extension, and line break of a CSV exported document.

```
`csharp
this.ReportViewer.CsvOptions = new BoldReports.Writer.CsvOptions()
{
    Encoding = System.Text.Encoding.Default,
    FieldDelimiter = ",",
    UseFormattedValues = false,
    Qualifier = "#",
    RecordDelimiter = "@",
    SuppressLineBreaks = true,
    FileExtension = ".txt"
};
`
```

HTML export options

You can hide the separator added at the end of each page by setting the `HidePageSeparator` property to true.

```
`csharp
this.ReportViewer.HTMLOptions = new BoldReports.Writer.HTMLOptions();
this.ReportViewer.HTMLOptions.HidePageSeparator = true;
`
```

Password protect exported document

This allows you to protect the exported document such as PDF, Word, Excel, and PowerPoint from unauthorized users by encrypting the document using encryption password. The following code snippet illustrates how to encrypt the exported document with the user defined password.

```
`csharp
//PDF encryption
this.ReportViewer.PDFOptions = new BoldReports.Writer.PDFOptions();
```

```
this.ReportViewer.PDFOptions.Security = new Syncfusion.Pdf.Security.PdfSecurity()
{
    UserPassword = "Password"
};
//Word encryption
this.ReportViewer.WordOptions = new BoldReports.Writer.WordOptions()
{
    EncryptionPassword = "password"
};
//Excel encryption
this.ReportViewer.ExcelOptions = new BoldReports.Writer.ExcelOptions()
{
    PasswordToModify = "password",
    PasswordToOpen = "password"
};
//PPT encryption
this.ReportViewer.PPTOptions = new BoldReports.Writer.PPTOptions()
{
    EncryptionPassword = "password"
};
`
```

Password protection is not supported for HTML export format.

Change file name in export

You can change the file name of report in export using the `FileName` property.

```
`csharp
this.ReportViewer.ExportSettings = new BoldReports.Writer.ExportSettings();
this.ReportViewer.ExportSettings.FileName = "Invoice";
`
```

Change image quality in export

You can change image quality of data visualization items in report export using the `ImageQuality` property.

```
`csharp
this.ReportViewer.ExportSettings = new BoldReports.Writer.ExportSettings();
this.ReportViewer.ExportSettings.ImageQuality = 4;
```

Toolbar customization

Decide or hide the export option

The Report Viewer provides the `ExportOptions` property to show or hide the default export types available in the component. The following code hides the HTML export type from the default export options.

```
`csharp
this.ReportViewer.ExportOptions = BoldReports.UI.Xaml.ExportOptions.All &
~BoldReports.UI.Xaml.ExportOptions.Html;
```

Handle post actions

You can handle the report processing post actions in following Report Viewer events to customize the rendered report.

- `RenderingBegin`
- `RenderingCompleted`
- `ReportServiceRequestBegin`
- `EncryptCredentials`

RenderingBegin

This event occurs when the report begins rendering. Information about this event is passed in a `CancelEventArgs` object to a `CancelEventHandler` delegate, which handles the event. You can use the following code in your application.

```
`csharp
this.ReportViewer.RenderingBegin += async (sen, arg) =>
{
    var dialog = new MessageDialog("This event handler will be called before the report rendering.");
    await dialog.ShowAsync();
};
```

RenderingCompleted

This event occurs when the report finishes rendering. Information about this event is passed in a `RenderingCompletedEventArgs` object to the `RenderingCompletedEventHandler` delegate, which handles the event. You can use the following code in your application.

```
`csharp
this.ReportViewer.RenderingCompleted += async (sen, arg) =>
{
```



```
var dialog = new MessageDialog("This event handler will be called after complete the report rendering.");
await dialog.ShowAsync();
};
`
```

ReportServiceRequestBegin

This event occurs when service request started for RDL processing mode to interact with web api for ReportViewer. You can use the following code in your application.

```
`csharp
this.ReportViewer.ReportServiceRequestBegin += async (sen, arg) =>
{
var dialog = new MessageDialog("This event handler will be called before interact the report viewer web api services");
await dialog.ShowAsync();
};
`
```

EncryptCredentials

Encrypt the report data source credentials and report server credential to pass the report viewer controller to render the report securely. You can use the following code in your application.

```
`csharp
this.ReportViewer.EncryptCredentials += (sen, arg) =>
{
DataSourceCredentials credential = new DataSourceCredentials();
credential.Name = "<database>";
credential.UserId = "ssrs1";
credential.Password = "RDLReport1";
// Set the Report server credential and Data source credential for the report server reports.
arg.ReportServerCredential = new System.Net.NetworkCredential("ssrs", "RDLReport1");
arg.DataSourceCredentials.Add(credential);
};
`
```

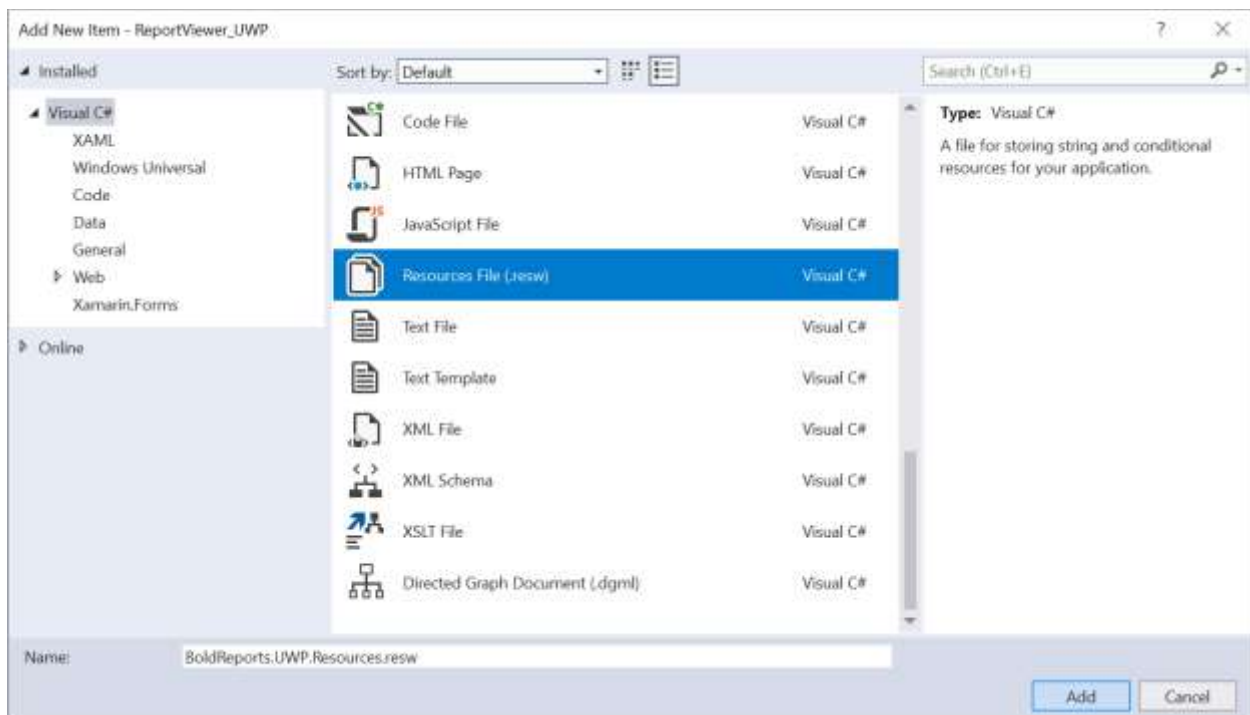
Localization of Bold Report Viewer

Localization of Bold Report Viewer allows you to localize the static text such as tooltip, parameter block, and dialog text based on a specific culture. Refer the following steps to localize the Report Viewer based on the culture.

1. Add Resources file for the different cultures.
2. Assign the value to each culture using key.
3. Assign a Current UI Culture to the application.
4. Set Report Viewer properties.

Add Resources file for the different cultures

1. Right-click the project and add the **Resources** folder in your application.
2. Right-click on the **Resources** folder and add the new folder then name it as **fr-FR**.
3. Right-click on the **fr-FR** folder and press **Ctrl+Shift+A** keys or select **Add > New Item** from the context menu.
4. In the Add New Item dialog, select **Resources** file and name it as **BoldReports.UWP.Resources.resw**.



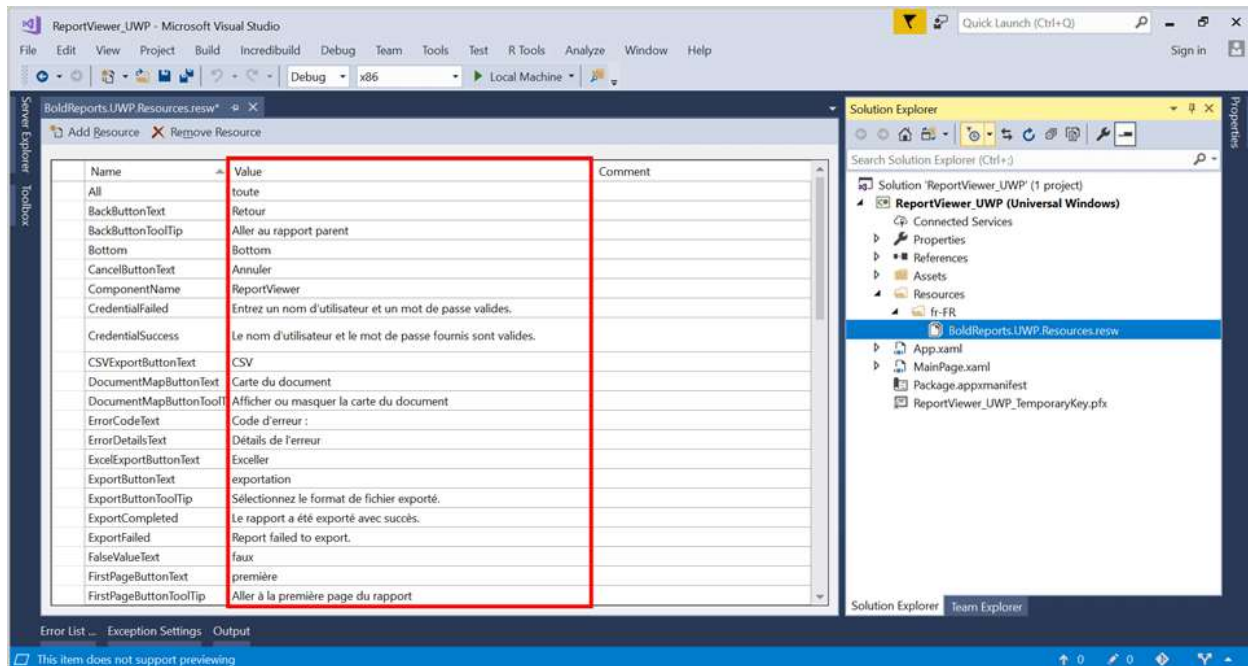
5. Click **Add**.

In case, the another culture is used in the application, then create another folder in name **[CultureInfo Code]** under the **Resources** folder. For example, **fr-FR** and the naming convention needs to be followed mandatorily.

Assign the value to each culture using key

To assign **Values** in Resource, the resource file needs to be updated according to the following steps.

1. Open the **BoldReports.UWP.Resources.resw** file by double clicking it from Solutions Explorer.
2. Add the key, Name, and its corresponding localized value by editing its field. Modify the resource values based on the **fr-FR** culture as shown in the following image.



you can download the modified resource file from [here](#) and replace it in your application.

Assign a Current UI Culture to the application

Mention the culture to be referred while initializing the application, so that application refer to the appropriate value provided in resource file.

Report Viewer application culture can be changed by setting the `PrimaryLanguageOverride` in the `MainPage()` constructor. In this application, `MainPage.xaml.cs` is the startup page and culture is assigned as like below.

```
`csharp
public MainPage()
{
    this.InitializeComponent();
    ApplicationLanguages.PrimaryLanguageOverride = "fr-FR";
}
```

Set Report Viewer properties

1. Add the Report Viewer initialization code.

```
`csharp
this.ReportViewer.ReportServiceURL = @"https://demos.boldreports.com/services/api/ReportViewer";
this.ReportViewer.ReportPath = "~/Resources/docs/sales-order-dtail.rdl";
this.ReportViewer.RefreshReport();
```

In this tutorial, `sales-order-detail` report is used.

- Now, run the application and the below output shows the toolbar items localized `fr-FR` culture.

Report Viewer - UNIP

Actualisation le rapport

Sales Order

Order ID: #5050750 Billing Date: 17-09-2019 Order Date: 01-06-2003 Purchase Order: PO7132170677 Shipment Method: CARGO TRANSPORT 5

Billing Address:
 Jean Handley
 Central Discount Store
 259826 Russell Rd. South
 Kent, Washington
 98031
 United States

Shipping Address:
 Jean Handley
 Central Discount Store
 259826 Russell Rd. South
 Kent, Washington
 98031
 United States

Contact:
 Jean Handley
 Ph: 562-555-0113

Line	Qty	Item Number	Description	Tracking No.	Unit Price	Item Total
1	2	CA-1088	AWC Logo Cap	373D-417C-AE	\$5.19	\$10.38
2	4	GL-F110-M	Full-Finger Gloves, M	373D-417C-AE	\$22.79	\$91.16
3	4	LI-0752-L	Long-Sleeve Logo Jersey, L	373D-417C-AE	\$28.64	\$115.36
4	3	GL-F110-L	Full-Finger Gloves, L	373D-417C-AE	\$22.79	\$68.37
5	1	BK-M68B-42	Mountain-200 Black, 42	373D-417C-AE	\$1,229.46	\$1,229.46
6	1	TG-W091-L	Women's Tights, L	373D-417C-AE	\$44.99	\$44.99
7	1	BK-M68S-38	Mountain-200 Silver, 38	373D-417C-AE	\$1,242.85	\$1,242.85
Total						\$2,892.57

Limitations

RDL specification

The Report Viewer control does not support RDL specification for SQL Server 2000 and SQL Server 2005.

Report layout

- In the Tablix cell split layout process, the entire cell moves to the next page to display the complete cell items, when the table cell width value exceeds the page width.

Expressions

The object function and VB function do not have complete support.

SSRS

The SSRS Report Server does not provide options to get credential information of the report data source deployed on the server. If the report has any data source that uses credentials to connect with the database, then you should specify the data source credentials for each report data source to establish database connection.

Samples and Demos

Browse and explore the ready-to-use RDL, RDLC reports, samples, and offline demos.

Locally installed reports

You can obtain the sample RDL and RDLC files from the Bold Reports installed location

`%localappdata%\Bold Reports\Embedded Reporting\Samples\Common\Data\ReportTemplate.`

Offline demos

The offline samples are provided in the Bold Reporting Tools setup. For more details, refer to the [Bold Reporting Tools sample deployment](#).

Migrate Report Viewer application

In our Bold Reports new assemblies are introduced for both client and server-side to resolve the compatibility problem between Essential Studio Report Viewer versions. It has changes in both Web API service and client-side scripts.

This section provides step-by-step instructions for migrating Report Viewer from Syncfusion Essential Studio release version to Bold Reports version of UWP Report Viewer application:

Client-side migration

1. In the Solution Explorer, right-click the **References** and remove the following assembly references:
 - Syncfusion.SfReportViewer.UWP
2. Add the assembly from the Syncfusion NuGet package **BoldReports.UWP**. To add from NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search for **BoldReports.UWP** NuGet package, and install in your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Control initialization

1. Open the **MainWindow.xaml** file and remove the following namespace in your XAML page.

```
`csharp
xmlns:syncfusion="using:BoldReports.UI.Xaml"
`
```

2. Add the following namespace in your XAML page.

```
`csharp
xmlns:syncfusion="using:BoldReports.UI.Xaml"
`
```

Server-side migration

1. In the Solution Explorer, right-click the **References** and remove the **Syncfusion.EJ.ReportViewer** assembly reference.
2. Add the assembly from the Bold Reporting NuGet package **BoldReports.Web**. To add from NuGet, right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages**. Search for **BoldReports.Web** NuGet package, and then install in your application.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

Web API Controller

1. The `IReportController` interface is moved to `BoldReports.Web.ReportViewer`. Open the Report Viewer Web API Controller file and remove the following using statement.

```
`csharp
using Syncfusion.EJ.ReportViewer;
`
```

2. Add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

Your application is successfully upgraded to the latest version of Report Viewer, and you can run the application with new assemblies.

Report export configuration

Now, the `BoldReports.Web` can export the reports with data visualization components only using web components. It is mandatory to configure the web scripts in Report Viewer Web API controller for exporting data visualization components such as chart, gauge, and map that are used in report definition. To configure the scripts in Web API, refer to the following steps:

1. Open the Report Viewer Web API controller.
2. Configure the following scripts and styles in `OnInitReportOptions` on Web API controller:
 - o `jquery-1.10.2.min.js`
 - o `bold.reports.common.min.js`
 - o `bold.reports.widgets.min.js`
 - o `ej.chart.min.js` - Exports the chart item. Add this script only if your report contains the chart report item.
 - o `ej2-base.min.js`, `ej2-data.min.js`, `ej2-pdf-export.min.js`, `ej2-svg-base.min.js`, `ej2-lineargauge.min.js` and `ej2-circulargauge.min.js` - Exports the gauge item. Add this script only if your report contains the gauge report item.
 - o `ej.map.min.js` - Exports the map item. Add this script only if your report contains the map report item.
 - o `bold.report-viewer.min.js`
3. Replace the following codes in Report Viewer Web API controller.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    var resourcesPath = System.Web.Hosting.HostingEnvironment.MapPath("~/Scripts");
```

```

reportOption.ReportModel.ExportResourceOption.Scripts = new List<string>
{
    "https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js",
    //Chart component script
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js",
    //Gauge component scripts
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-base.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-data.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-pdf-export.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/common/ej2-svg-base.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-lineargauge.min.js",
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-circulargauge.min.js",
    //Map component script
    "https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej2-maps.min.js",
    //Report Viewer Script
    "https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js",
};
reportOption.ReportModel.ExportResourceOption.DependentScripts = new List<string>
{
    "https://code.jquery.com/jquery-1.10.2.min.js"
};
}
,

```

The data visulization components will not export without the above script configurations.

NuGet Packages for UWP

Refer to the following steps to configure Bold Reporting NuGet packages for UWP application.

Configure NuGet feed URL

Online NuGet feed URL

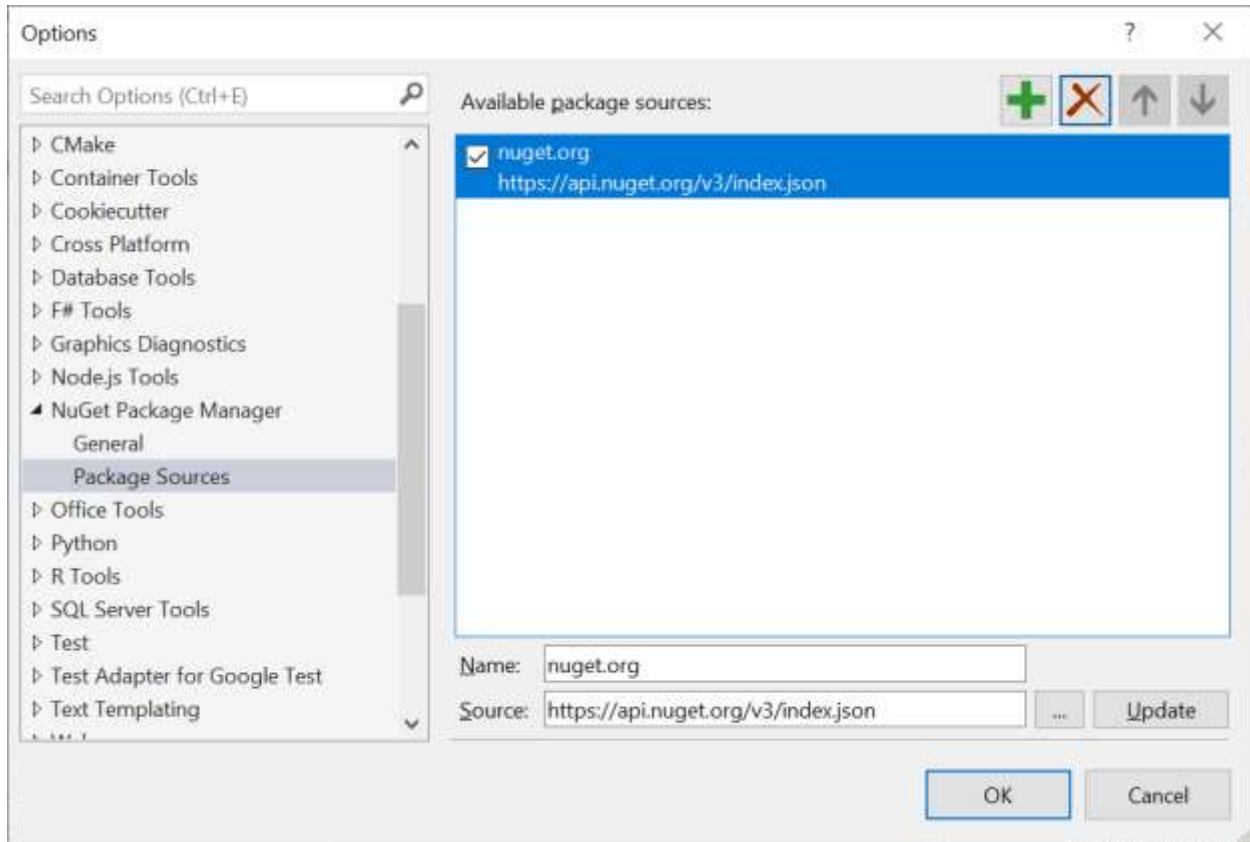
The Bold Reporting NuGet packages are published in [Nuget.org](https://www.nuget.org). To configure the online packages, use the following steps:

1. Open Visual Studio application.
2. On the **Tools** menu, select **Options**.
3. Expand the **NuGet Package Manager** and select **Package Sources**.

- Click the **Add** button, enter the following **Package Name** and **Package Source URL**, and then click **Update**.

Name: NuGet.org

Source: <https://api.nuget.org/v3/index.json>



Installing NuGet packages

Install using NuGet Package Manager

The NuGet Package Manager can be used to search and install NuGet packages in Visual Studio solution or project:

- On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution**. Alternatively, right-click the project/solution in Solution Explorer tab, and choose **Manage NuGet Packages**.
- By default, the **NuGet.org** package is selected in the **Package source** drop-down. Select package source, search for the **BoldReports.UWP** package, and then click **Install** button.

Install using Package Manager Console

To install the Reporting component using the Package Manager Console as NuGet packages,

- On the **Tools** menu, select **NuGet Package Manager**, and then click **Package Manager Console**.
- Run the following NuGet installation commands:


```
`cmd
```

install specified package in default project

```
Install-Package <Package Name>
```

install specified package in default project with specified package source

```
Install-Package <Package Name> -Source <Source Location>
```

install specified package in specified project

```
Install-Package <Package Name> -ProjectName <Project Name>
```

```
,
```

For example:

```
`cmd
```

install specified package in default project

```
Install-Package BoldReports.UWP
```

install specified package in default project with specified Package Source

```
Install-Package BoldReports.UWP -Source "https://api.nuget.org/v3/index.json"
```

install specified package in specified project

```
Install-Package BoldReports.UWP -ProjectName BoldReportsApplication
```

```
,
```

Upgrading NuGet packages

Upgrading using NuGet Package Manager

NuGet packages can be updated to their specific version or latest version available in the Visual Studio solution or project:

1. On the **Tools** menu, click **NuGet Package Manager | Manage NuGet Packages for Solution....** Alternatively, right-click the project/solution in the Solution Explorer tab, and then choose **Manage NuGet Packages**.
2. Select the **Updates** tab to see the packages available for update from the desired package sources, select the required packages and specific version from the drop-down, and then click the **Update** button.

Upgrading using Package Manager Console

To update the installed Bold Reporting NuGet packages using the Package Manager Console:

1. On the **Tools** menu, select **NuGet Package Manager**, and then select **Package Manager Console**.
2. Run the following NuGet installation commands:

```
`cmd
```

Update specific NuGet package in default project

Update-Package <Package Name>

Update all the packages in default project

Update-Package

Update specified package in default project with specified package source

Update-Package <Package Name> -Source <Source Location>

Update specified package in specified project

Update-Package <Package Name> - ProjectName <Project Name>

,

For example:

`cmd

Update specified Bold Reporting NuGet package

Update-Package BoldReports.UWP

Update specified package in default project with specified Package Source

Update-Package BoldReports.UWP -Source "https://api.nuget.org/v3/index.json"

Update specified package in specified project

Update-Package BoldReports.UWP -ProjectName BoldReportsApplication

,

Upgrading using NuGet CLI

Using the NuGet CLI, all the NuGet packages in the project can be updated to the available latest version:

1. Download the latest [NuGet CLI](#).

To update the existing `nuget.exe` to latest version use the following command:

`cmd

nuget update -self

,

2. Open the downloaded executable location in the command window. Run the following "update commands" to update the Bold Reporting NuGet packages.

```
`cmd
```

update all NuGet packages from config file

```
nuget update <configPath> [options]
```

update all NuGet packages from specified Packages Source

```
nuget update -Source <Source Location> [optional]
```

,

`configPath` is optional. It identifies the `package.config` or solutions file lists the packages utilized in the project.

For example:

```
`cmd
```

Update all NuGet packages from config file

```
nuget update "C:\Users\BoldReportsApplication\package.config"
```

Update all NuGet packages from specified Packages Source

```
nuget update -Source "https://api.nuget.org/v3/index.json"
```

,

The Update command is not working as expected in Mono (Mac and Linux) and projects using PackageReference format.

Frequently asked questions

This section helps to get the answer for the frequently asked questions in Bold Reports UWP Report Viewer.

1. [How can improve the performance and handle the large amounts of data with Report Viewer?](#)
2. [Is Report Viewer compatible with latest version of JQuery library?](#)
3. [Is the back action from drillthrough report will load the report again with Report Viewer?](#)

Custom properties

The custom properties helps you to include additional features that are not natively supported in the RDL reporting. This topic explains the list of custom properties supported in UWP Report Viewer.

See Also

- [Table Custom Properties](#)
- [Report Custom Properties](#)
- [Parameter Custom Properties](#)
- [Export Custom Properties](#)

Table custom properties

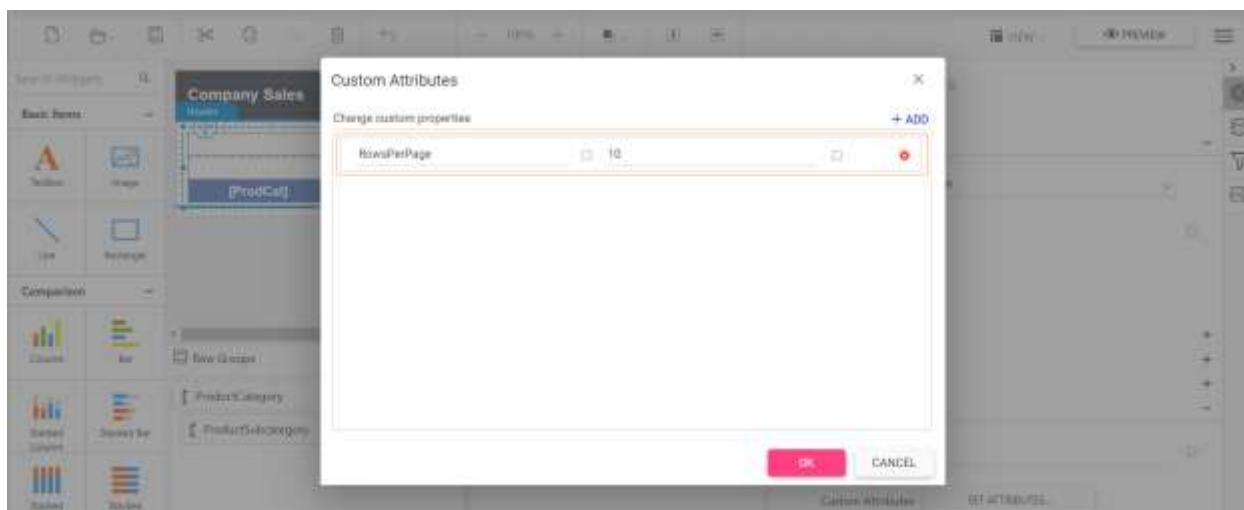
This topic explains about the list of table report item custom properties that are supported to render in UWP Report Viewer.

Limit number of table records on each page

The `RowsPerPage` custom property is used to specify the number of table records to display on each page. It supports integer data value greater than zero.

This property is ignored when table rows heights higher than current page size. Increase the report page height or reduce `RowsPerPage` count that fits within the page.

You can set the `RowsPerPage` property value, as shown in the below.



Report custom properties

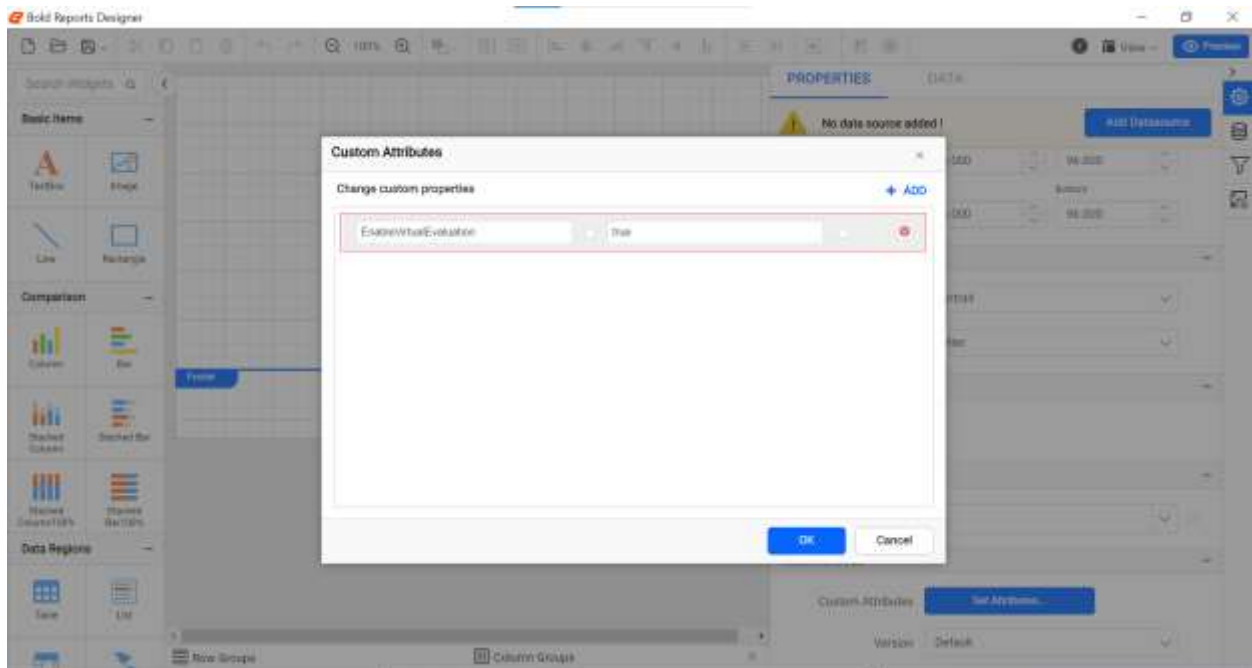
This topic explains about the list of report custom properties that are supported to render in UWP Report Viewer.

Improve performance and handle large amount of data

Set the `EnableVirtualEvaluation` and `DisablePageSplitting` custom properties in a report to improve performance and handle the larger amount of data with less memory footprint.

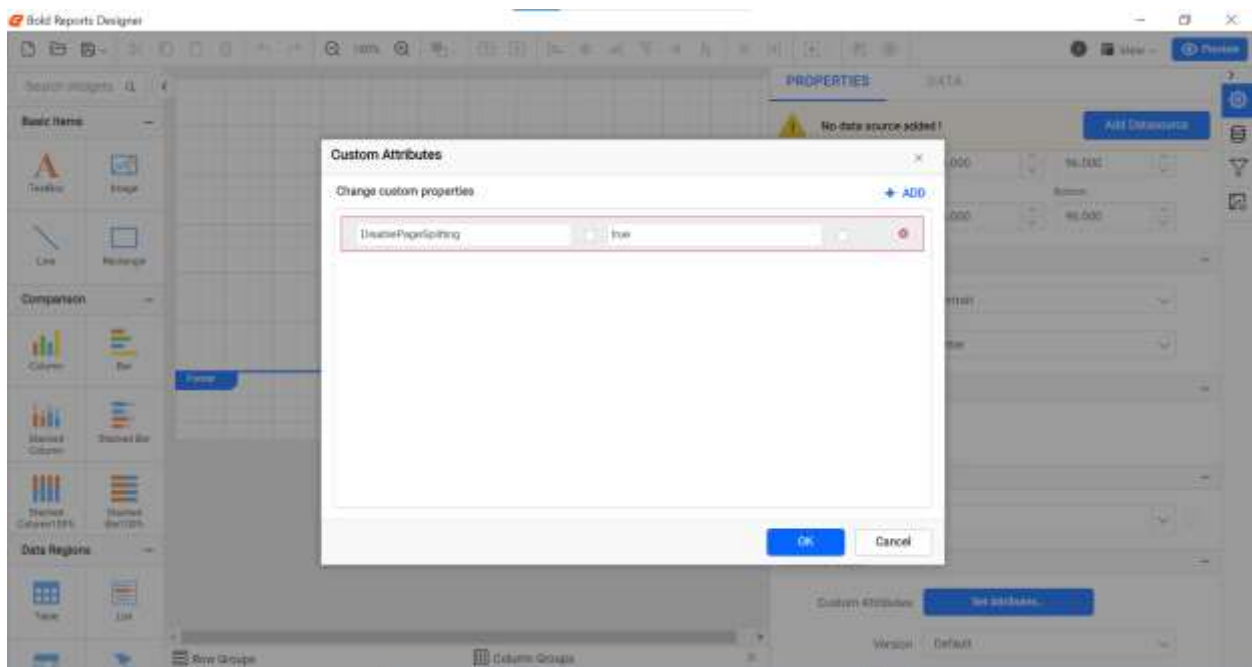
Render large data report faster

The `EnableVirtualEvaluation` custom property is used to render the large data report faster. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Reduce memory footprint for large data report

The `DisablePageSplitting` custom property is used to reduce the memory footprint for large data report. The property value should be static or an expression that returns a boolean value. You can set the property value, as shown in the below.



Improve report items layout and avoid extra blank pages

When the `RoundLayoutMeasures` property is false, all non-integral values that are calculated during the report processing are rounded to whole pixel values. It provides following improvements,

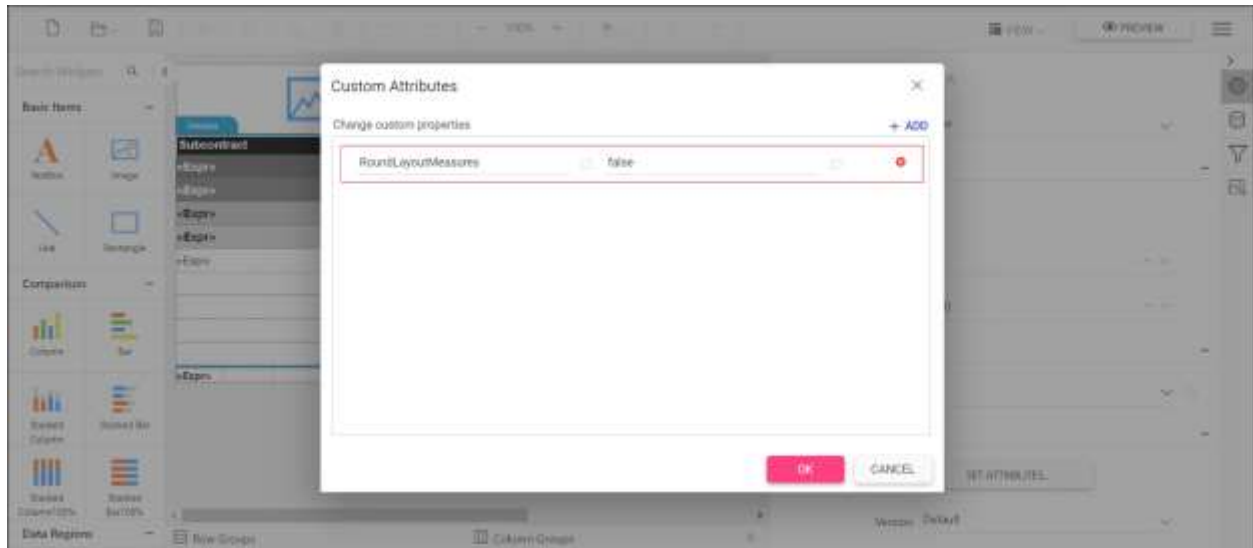
- Report text rendered without cuts.

Report custom properties
timeouts

Handling failure in long-running HTTP requests, report processes, and

- Eliminates extra blank pages.
- Removes item and text overlaps.
- Eliminates the blur semi-transparent edges that are produced by anti-aliasing.
- Produces identical look in report view and export output.

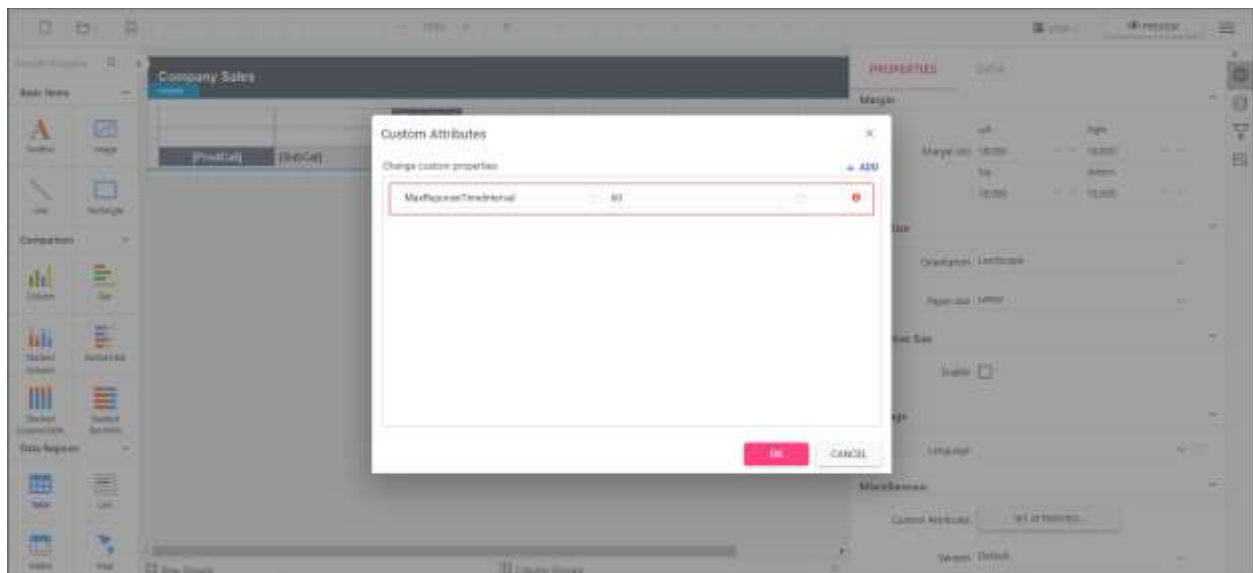
You can set the property value, as shown in the below.



Handling failure in long-running HTTP requests, report processes, and timeouts

When a report process for a long time due to huge records or a HTTP request takes too long to respond then it results in Gateway Timeout or report rendering errors. The `MaxResponseTimeInterval` allows to specify the seconds to process an HTTP request and respond back. It helps to keep the client and server connection live by avoiding the timeout.

You can set the property value as shown in the below.



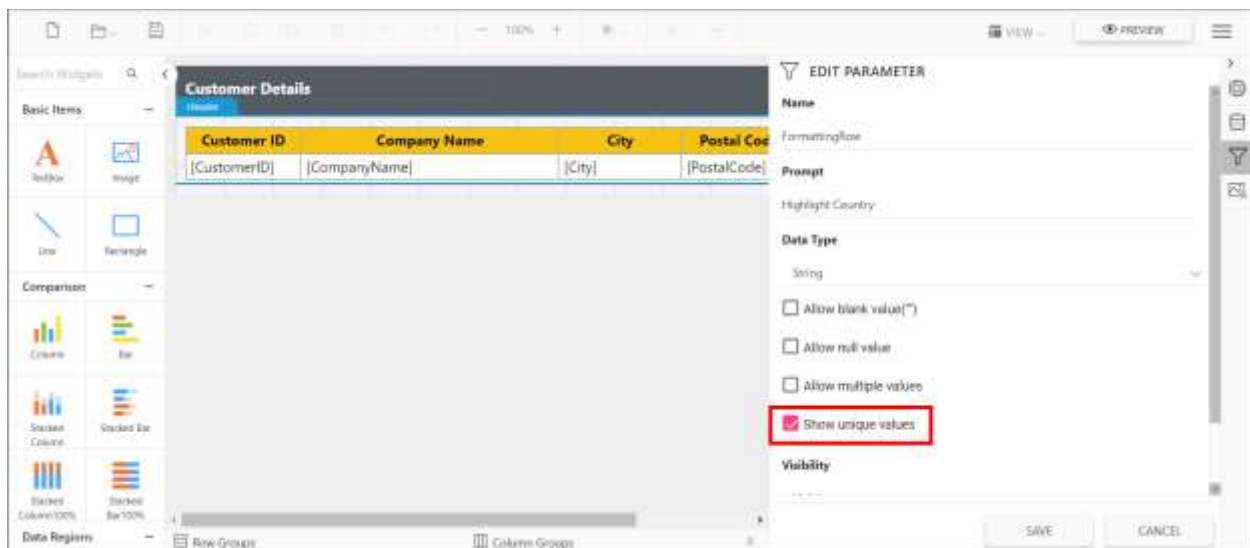
Parameter custom properties

This topic explains about the list of parameter custom properties that are supported to customize the reports preview in Report Viewer.

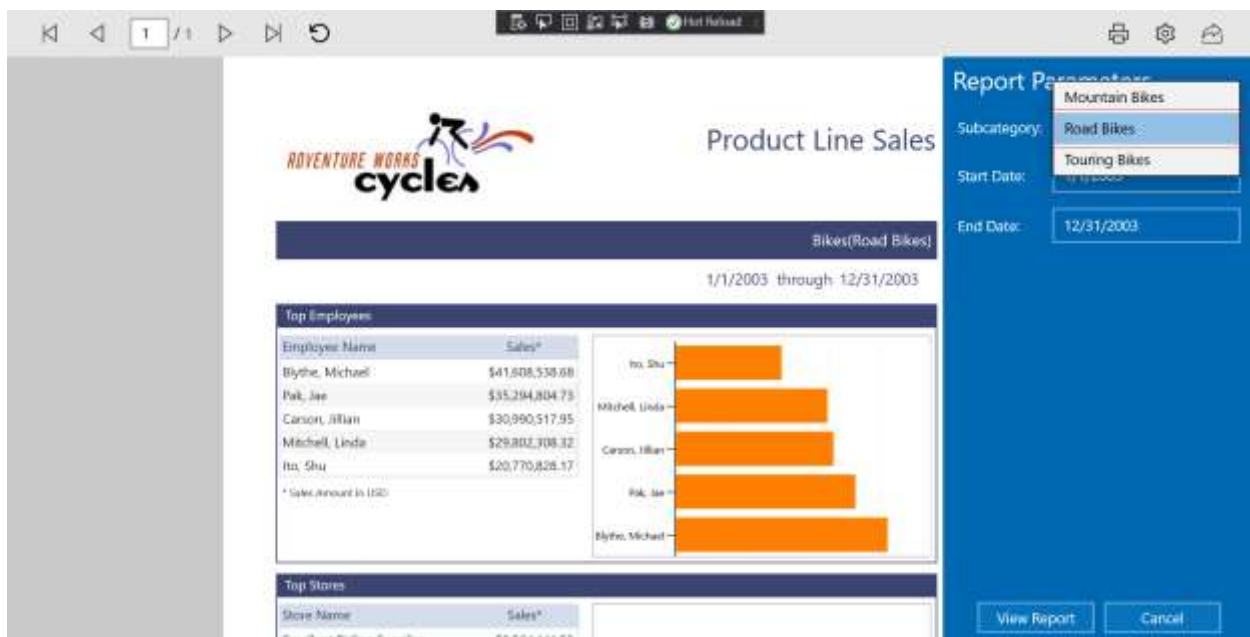
Show only distinct values in parameter

A parameter created with data set query values may contain duplicate values. The Report Viewer supports UniqueValueParameters custom property that helps show only unique values in the parameter drop-down while viewing the report, without creating new a data set. Refer to the below image for property setting option.

You can also provide the parameter names with comma (,) separator to set for multiple parameters.



Preview the report and the unique values showed in the parameter drop down.



Export custom properties

This topic explains the list of custom properties that are supported at the report level to control the export behaviour in UWP Report Viewer.

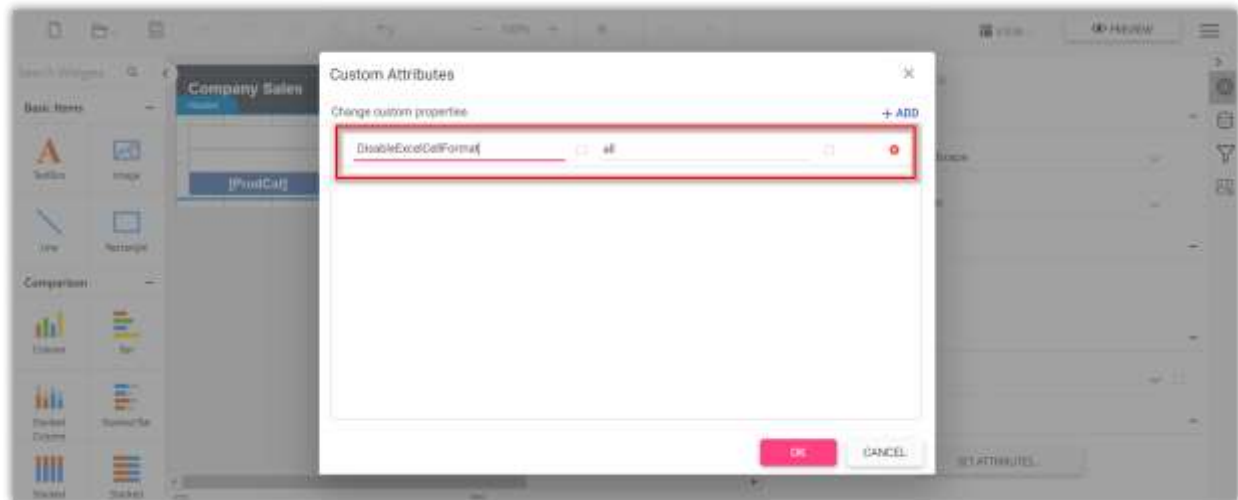
Improve excel export performance

The custom property `DisableExcelCellFormat` helps to improve the excel export performance by ignoring the rendering of report item styles. It allows property value from anyone of the following values.

- **Style** - Disables rendering of the cell styles like padding, background, color, and text style
- **Border** - Disables rendering of the cell border
- **All** - Disables rendering of the cell border, padding, background, color, and text style

Set the property value as **All** to improve maximum excel export performance.

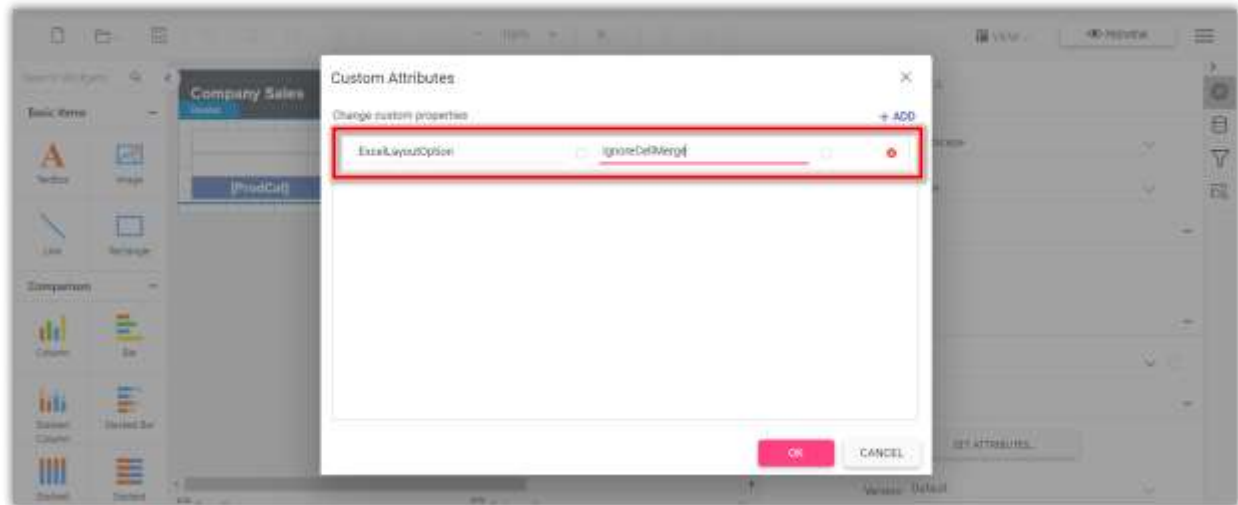
You can set the `DisableExcelCellFormat` custom property as shown below,



The `DisableExcelCellFormat` property must be added to report properties.

Improve excel export readability

Set `ExcelLayoutOption` custom property value as `IgnoreCellMerge` to improve the readability of the excel document by eliminating the tiny columns, rows, and merged cells. You can set the property value, as shown below,



The `ExcelLayoutOption` property must be added to report properties.

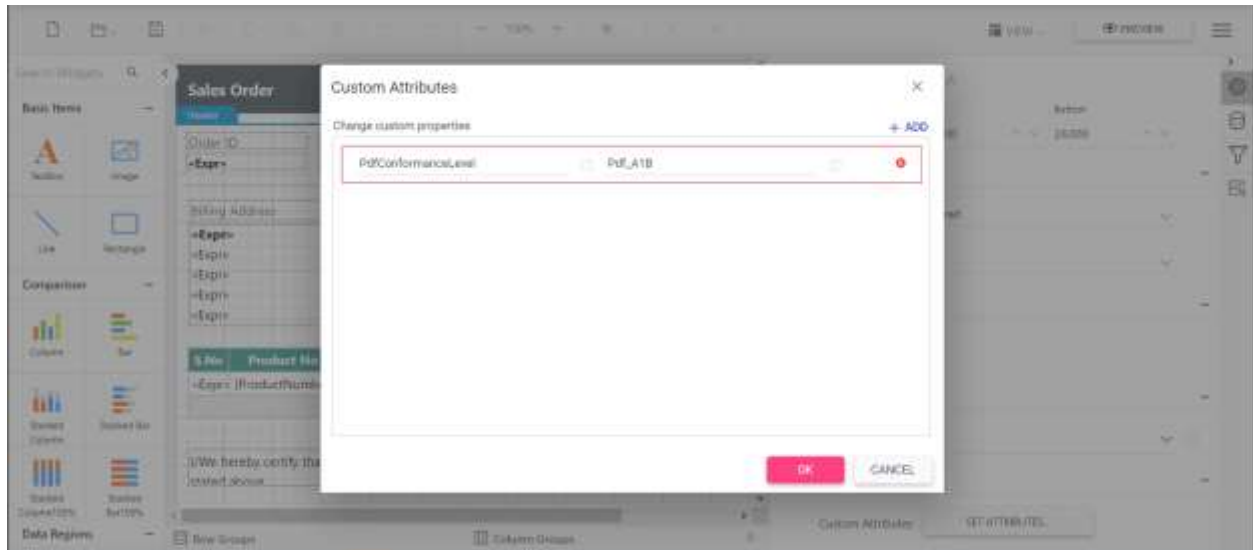
Set pdf conformance level

The `PdfConformanceLevel` allows you set PDF document versions.

You can set anyone of the following PDF conformance option.

- **PdfA1B** - You can create a PdfA1B document by specifying the conformance level as Pdf_A1B through PdfConformanceLevel Enum when creating the new PDF document.
- **PdfA2B** - You can create a PdfA2B document by specifying the conformance level as Pdf_A2B through PdfConformanceLevel Enum when creating the new PDF document
- **PdfA3B** - The PDF/A3B conformance supports the external files as attachment to the PDF document, so you can attach any document format such as Excel, Word, HTML, CAD, or XML files.
- **PdfA1A** - This conformance includes all PdfA1B requirements in addition to the features intended to improve a document's accessibility. PDF/A-1a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2A** - This conformance includes all PDF/A2B requirements in addition to the features intended to improve a document's accessibility. PDF/A-2a conformance additionally have crucial properties of Tagged PDF.
- **PdfA2U** - This conformance includes all PdfA2U requirements, and additionally Unicode mapping for all text in the document.
- **PdfX1A2001** - You can create a PDF/X-1a document by specifying the conformance level as PdfX1A2001 through PdfConformanceLevel Enum when creating the new PDF document.

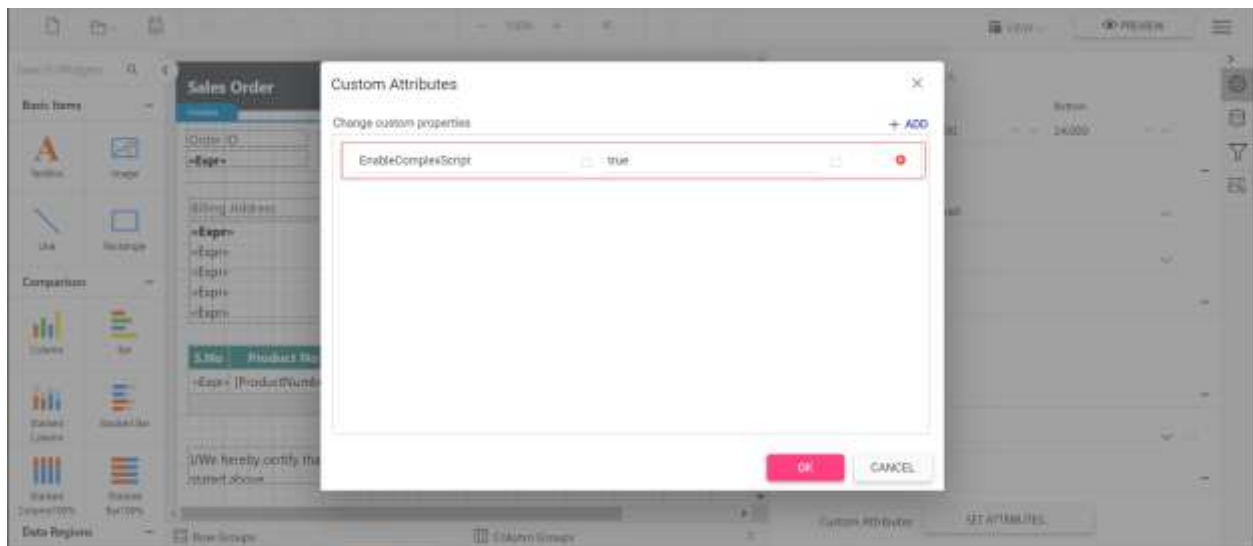
You can set the PdfConformanceLevel custom property as shown below,



Export pdf document with complex script

The `EnableComplexScript` custom property allows you to export pdf documents with complex script language texts.

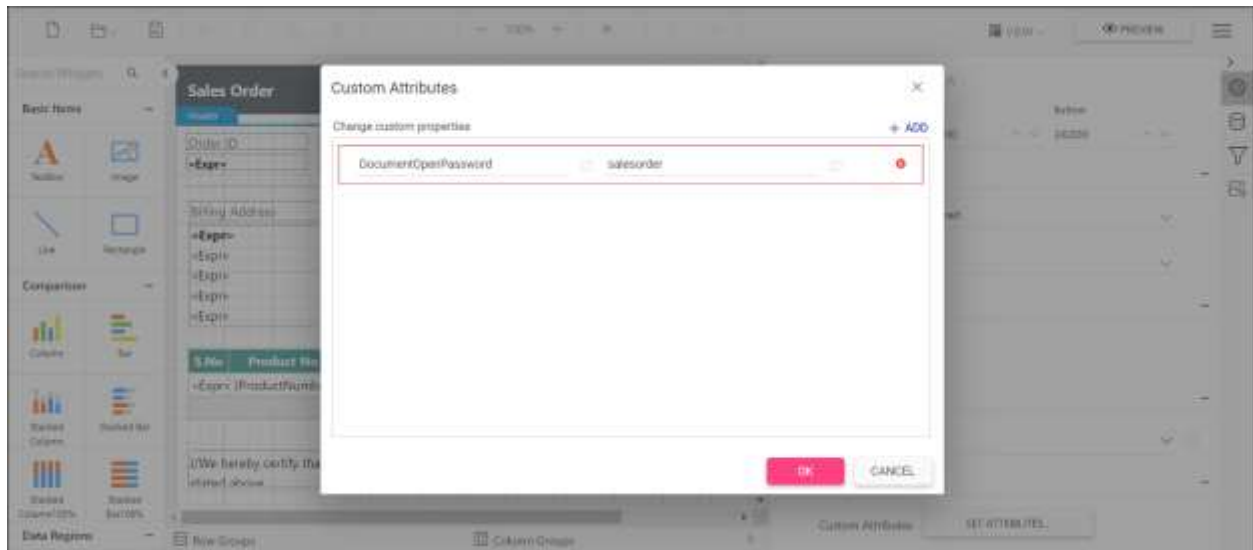
You can set the property value, as shown below,



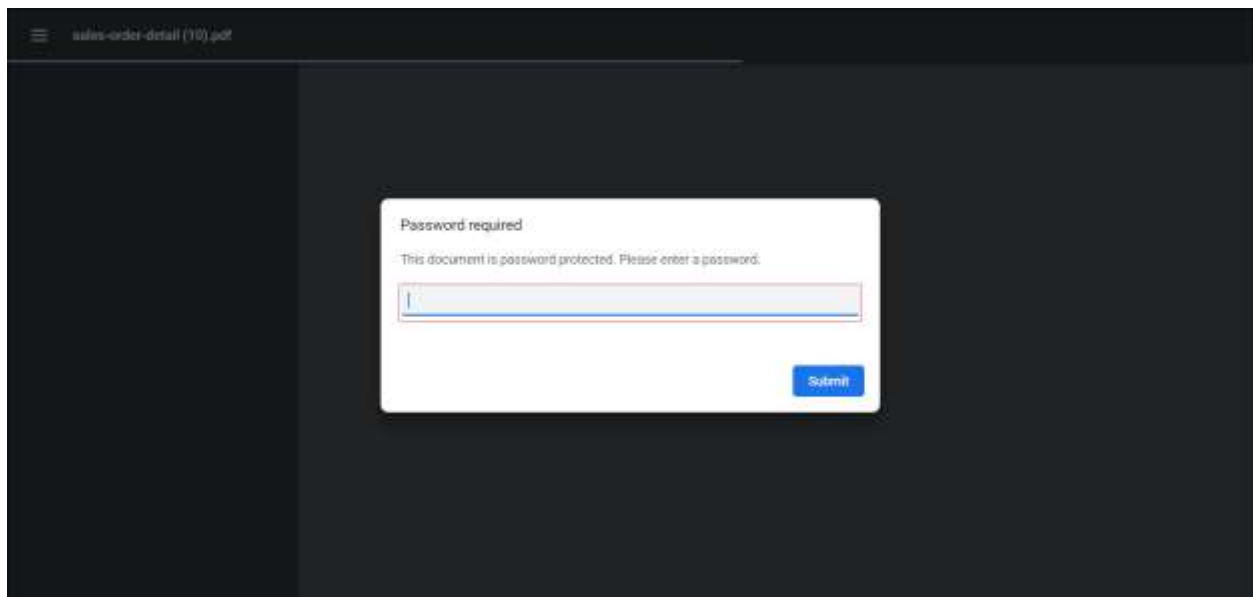
Encrypt and secure export documents

The custom property `DocumentOpenPassword` allows you to protect the exported document such as PDF, Word, and Excel from unauthorized users by encrypting the document using the encryption password.

You can set the property value, as shown below,



open the pdf document and see the below output.

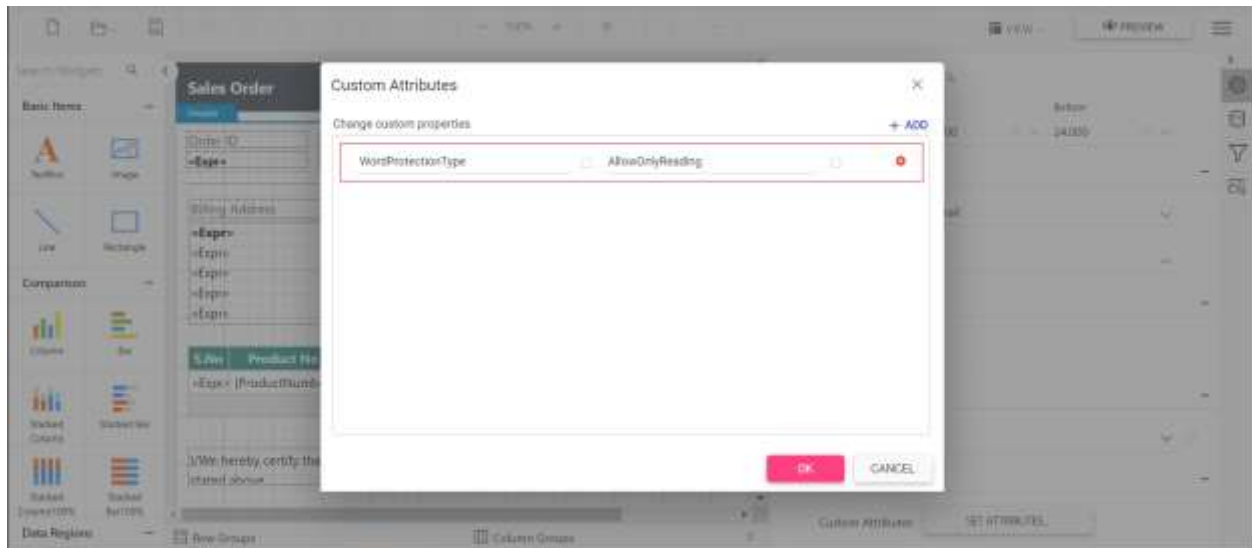


Protecting Word document from editing

The custom property `WordProtectionType` allows you to restrict a Word document from editing either by providing a password or without a password by using following types of protection.

- `AllowOnlyComments`- You can add or modify only the comments in the Word document.
- `AllowOnlyFormFields`- You can modify the form field values in the Word document.
- `AllowOnlyRevisions`- You can accept or reject the revisions in the Word document.
- `AllowOnlyReading`- You can only view the content in the Word document.
- `NoProtection`- You can access or edit the Word document contents as normally.

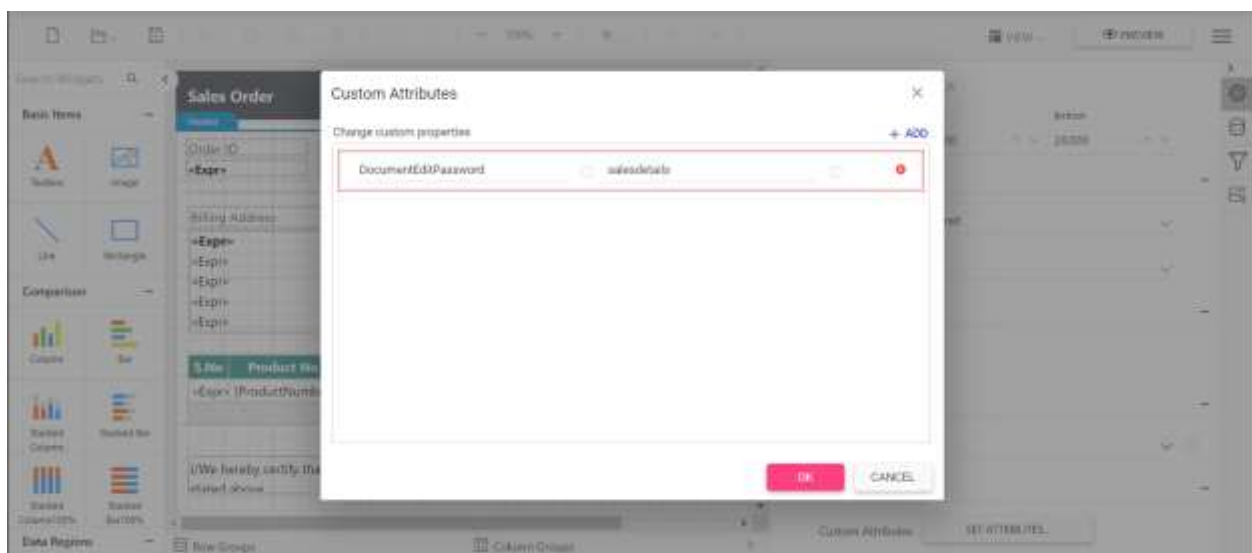
You can set the `WordProtectionType` custom property as shown below,



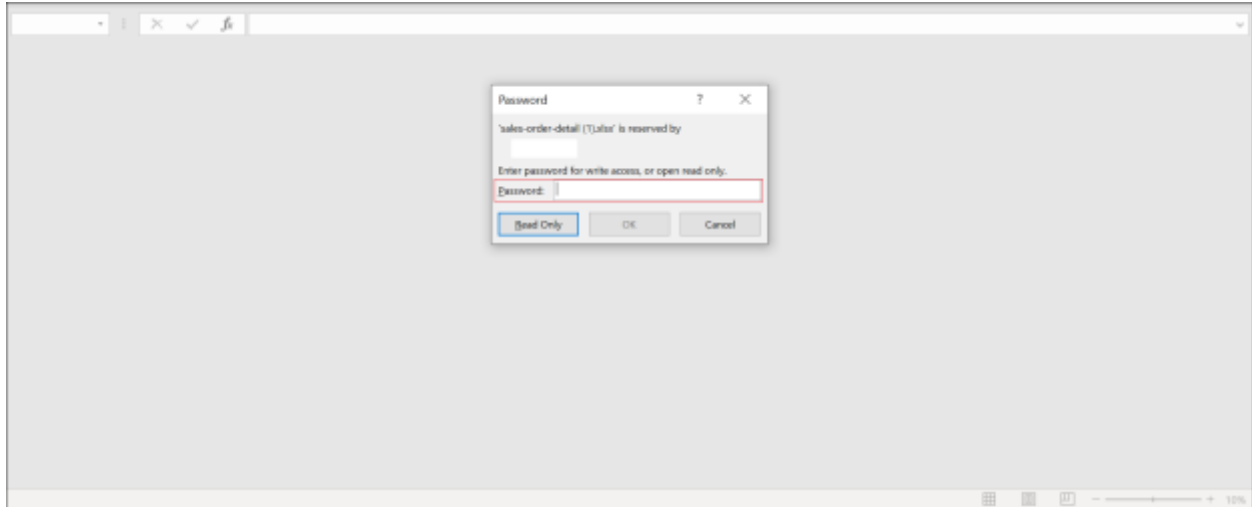
Set excel document edit password

The custom property `DocumentEditPassword` helps to allow specific users permission to modify the workbook data and save changes to the file in the excel document.

You can set the property value, as shown below,



open the excel document and see the below output.

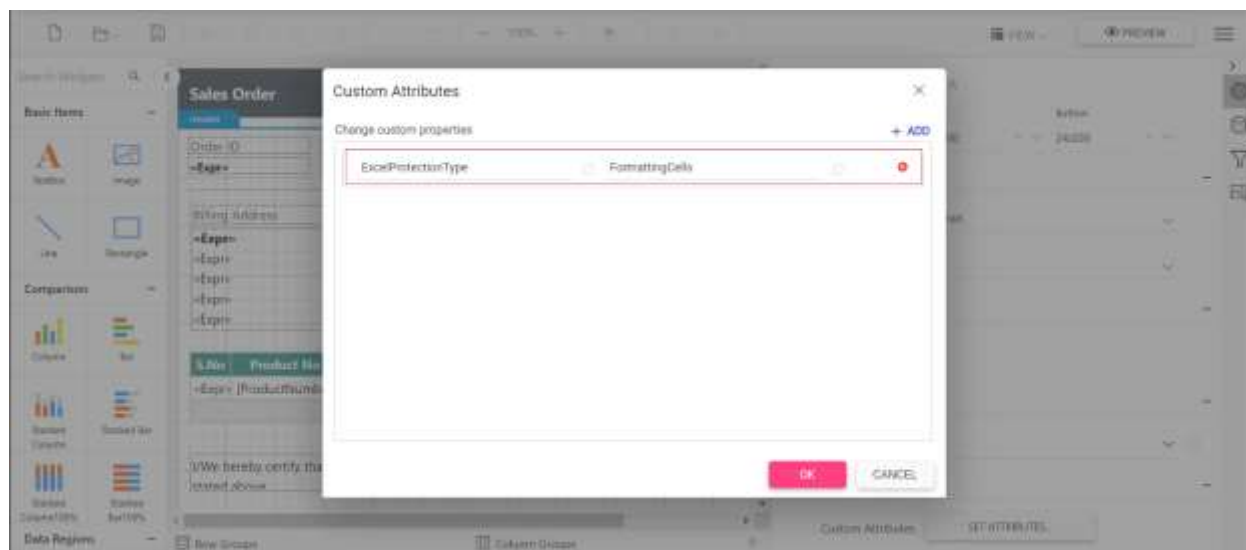


Set excel document protection

The custom property `ExcelProtectionType` allows you to protect the worksheet of excel document either by providing a password or without a password by using following types of protection.

- `None` - Represents no protection in excel sheet
- `Objects` - allows to protect shapes in excel sheet
- `Scenarios` - allows to protect scenarios.
- `FormattingCells` - allows the user to format any cell on a protected worksheet.
- `FormattingColumns` - allows the user to format any column on a protected worksheet.
- `FormattingRows` - allows the user to format any rows on a protected worksheet.
- `InsertingColumns` - allows the user to insert columns on the protected worksheet.
- `InsertingRows` - allows the user to insert rows on the protected worksheet.
- `InsertingHyperlinks` - allows the user to insert hyperlinks on the worksheet.
- `DeletingColumns` - allows the user to delete columns on the protected worksheet, where every cell in the column to be deleted is unlocked.
- `DeletingRows` - allows the user to delete rows on the protected worksheet, where every cell in the row to be deleted is unlocked.
- `LockedCells` - allows to protect locked cells.
- `Sorting` - allows the user to sort on the protected worksheet
- `Filtering` - allows the user to set filters on the protected worksheet. Users can change filter criteria but can not enable or disable an auto filter.
- `UsingPivotTables` - allows the user to use pivot table reports on the protected worksheet.
- `UnLockedCells` - allows to protect the user interface, but not macros.
- `Content` - allows to protect the contents in the excel sheet.
- `All` - allows to protect all type of protection.

You can set the `ExcelProtectionType` custom property as shown below,



UWP Report Viewer Reporting Service

The UWP Report Viewer requires a Web API service to process the report files. The following topics explains how to create reporting Web API service to preview the reports.

- [ASP.NET Web API Service](#)
- [ASP.NET Core Web API Service](#)

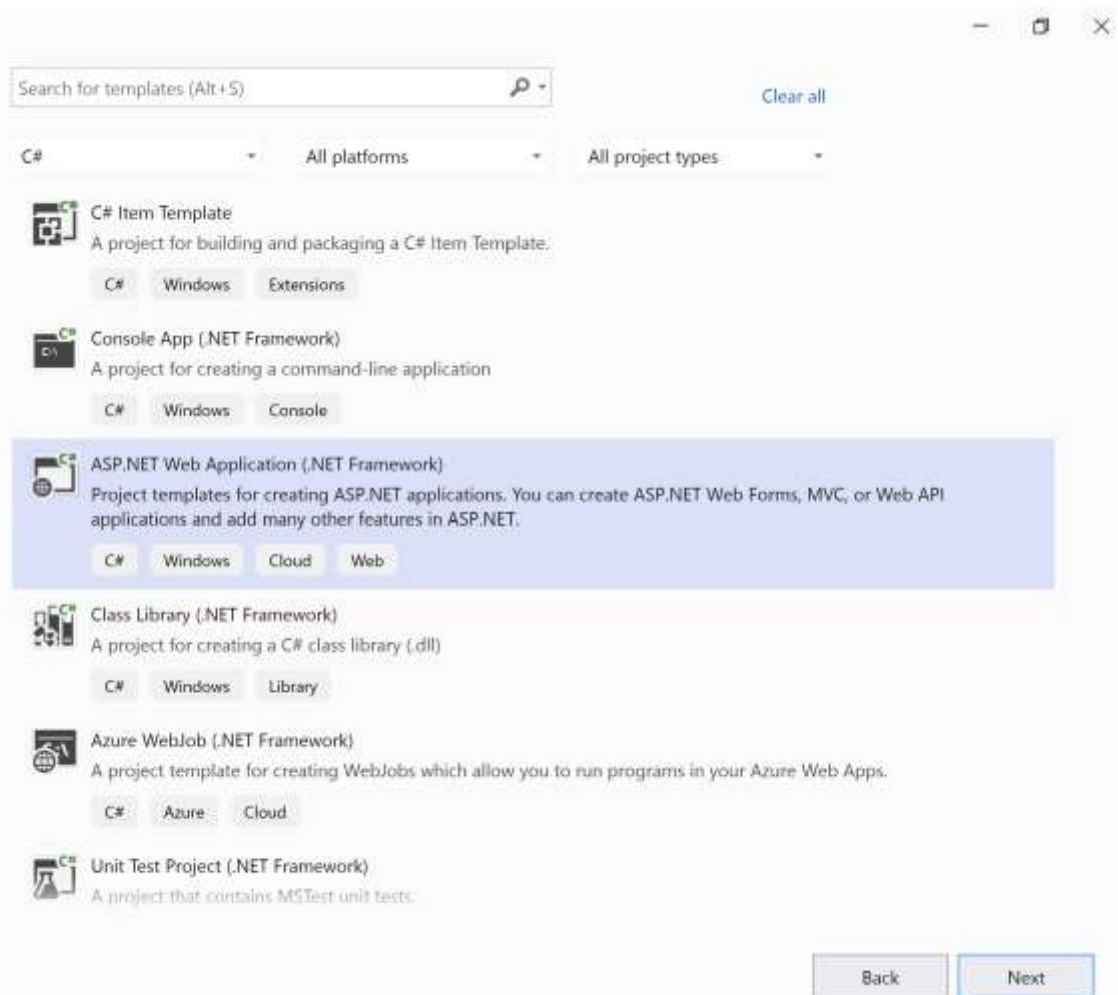
Create ASP.NET Web API service

In this section, you will learn how to create a Web API Service for Report Viewer using the new ASP.NET Empty Web Application template.

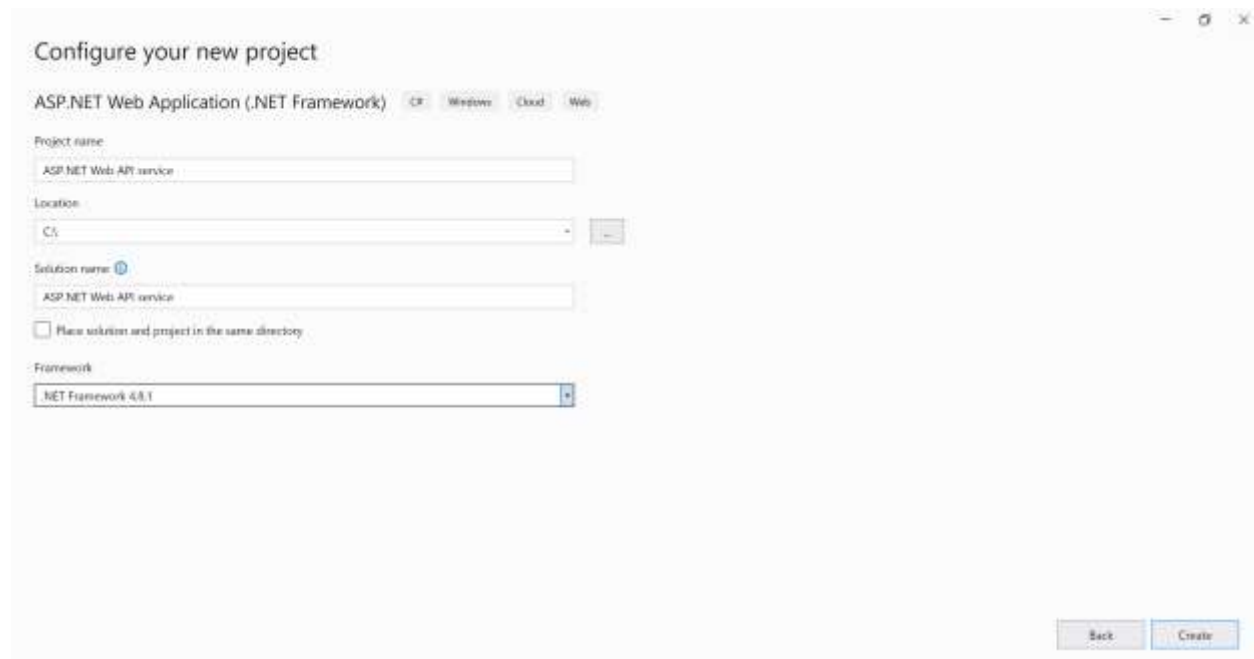
To get start quickly with Web API Service for Report Viewer, you can check on this video:

youtube: https://youtu.be/Qbho_8cnhJ8

1. Start Visual Studio 2022 and click **Create new project**.
2. Select **ASP.NET Web Application (.NET Framework)**.



3. Change the application name, and then select the required **.NET Framework** in the drop-down.



Configure your new project

ASP.NET Web Application (.NET Framework) **CR** Windows Cloud Web

Project name
ASP.NET Web API service

Location
C:\

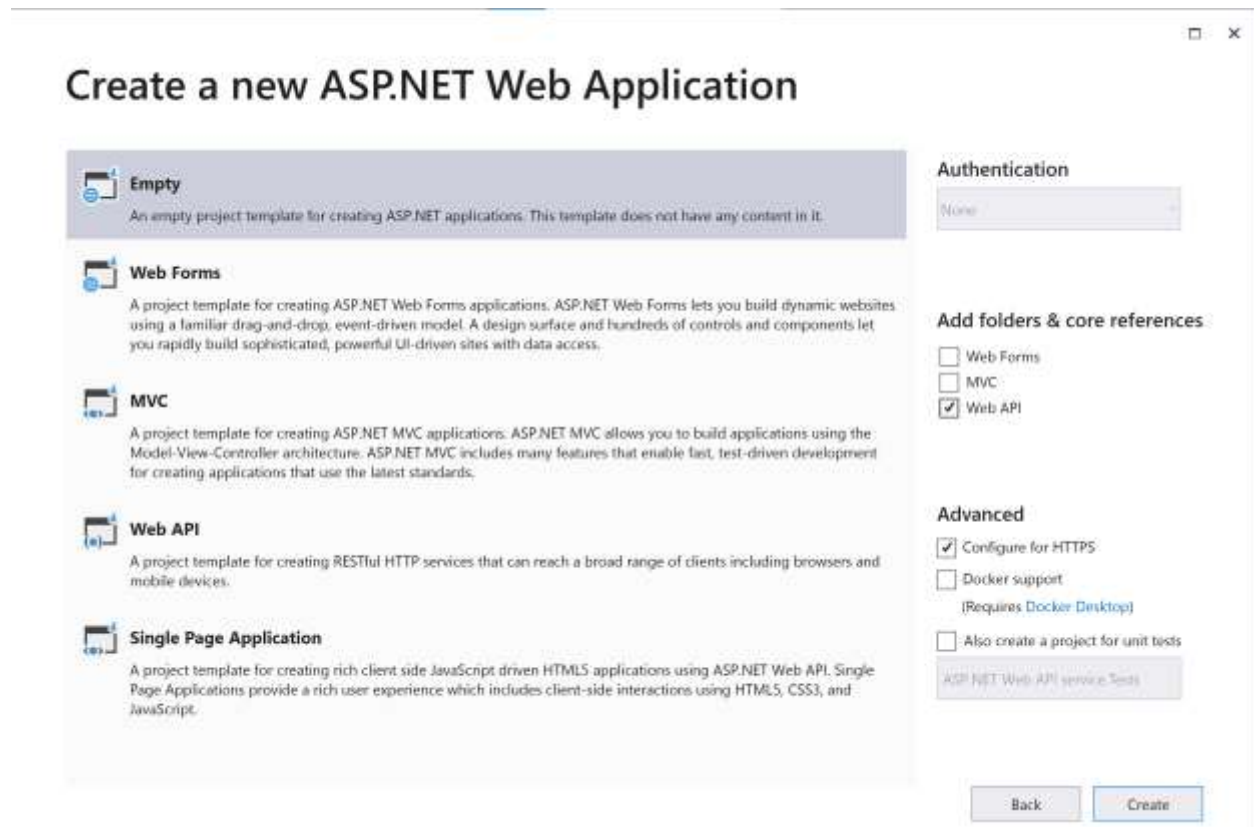
Solution name
ASP.NET Web API service

☐ Place solution and project in the same directory

Framework
.NET Framework 4.5.1

Back Create

4. Choose **Empty, Web API** and then click **OK**. Now, the Web application project is created with default ASP.NET Web template.



Create a new ASP.NET Web Application

Empty
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Web Forms
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

MVC
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Web API
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Single Page Application
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
None

Add folders & core references

☐ Web Forms
☐ MVC
☒ Web API

Advanced

☒ Configure for HTTPS
☐ Docker support
(Requires [Docker Desktop](#))
☐ Also create a project for unit tests

ASP.NET Web API service Tests

Back Create

Configure Report Viewer Web API

1. Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Refer to the [NuGet Packages](#) section to learn more details about installing and configuring Report Viewer NuGet packages.

2. Search for **BoldReports.Web** NuGet packages, and install them in your Web application.

Package | Purpose

PostReportAction | Action (HttpPost) method for posting the request in report process.

OnInitReportOptions | Report initialization method that occurs when the report is about to be processed.

OnReportLoaded | Report loaded method that occurs when the report and sub report start loading.

GetResource | Action (HttpGet) method to get resource for the report.

ReportHelper

The class **ReportHelper** contains helper methods that help to process a Post or Get request from the Report Viewer control and return the response to the Report Viewer control. It has the following methods:

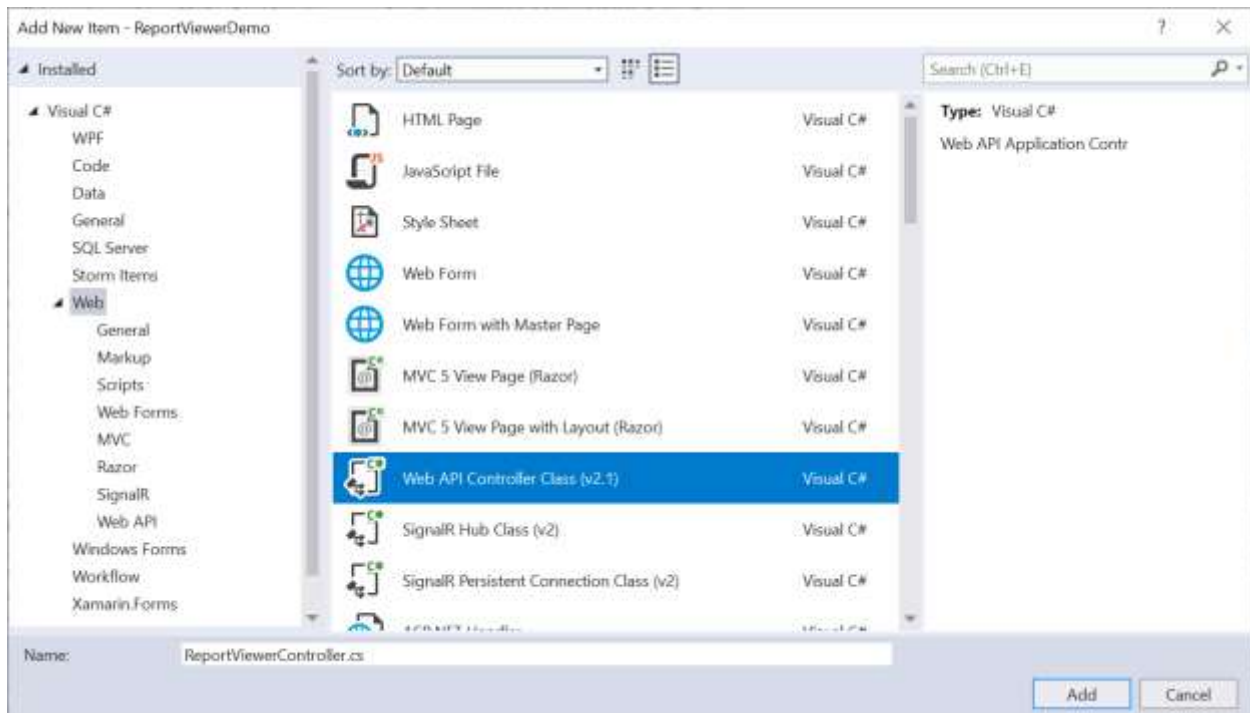
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click **Controller** folder in your project and select **Add > New Item** from the context menu.
2. Select **Web API Controller Class** from the listed templates and name it as **ReportViewerController.cs**



3. Click **Add**.

While adding Web API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```

5. Inherit the **IReportController** interface, and implement its methods (replace the following code in newly created Web API controller).

```
`csharp
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
}
```

```
[System.Web.Http.ActionName("GetResource")]
[AcceptVerbs("GET")]
public object GetResource(string key, string resourcetype, bool isPrint)
{
    return ReportHelper.GetResource(key, resourcetype, isPrint);
}
// Method that will be called when initialize the report options before start processing the report
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    // You can update report options here
}
// Method that will be called when reported is loaded
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    // You can update report options here
}
}
```

Add routing information

1. To configure routing to include an action name in the URI, open the **WebApiConfig.cs** file and change the **routeTemplate** in the **Register** method as follows,

```
`csharp
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services
        // Web API routes
        config.MapHttpAttributeRoutes();
        config.Routes.MapHttpRoute(
```

```

name: "DefaultApi",
routeTemplate: "api/{controller}/{action}/{id}",
defaults: new { id = RouteParameter.Optional }
);
}
}
,

```

2. Compile and run the Web API service application.

Enable Cross-Origin Requests

Browser security prevents Report Viewer from making requests to your Web API Service when both runs in a different domain. To allow access to your Web API service from a different domain, you must enable cross-origin requests.

1. Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages**. Alternatively, go to **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.
2. Search for `Microsoft.AspNet.WebApi.Cors` NuGet packages, and install them in your Web API application.
3. Call `EnableCors` in `WebApiConfig` to add CORS services to the **Register** method. Replace the following code to allow any origin requests.

```

`csharp
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Add Enable Cors
        config.EnableCors();

        // Web API configuration and services

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{action}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}

```

```

}
}
`

```

4. To specify the CORS policy for API controller, add the `[EnableCors]` attribute to the controller class. Specify the policy name.

```

`csharp
[EnableCors(origins: "", headers: "", methods: "*")]
public class ReportViewerController : ApiController, IReportController
{
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
    [System.Web.Http.ActionName("GetResource")]
    [AcceptVerbs("GET")]
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }
    // Method that will be called when initialize the report options before start processing the report
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
    // Method that will be called when reported is loaded
    [NonAction]
    public void OnReportLoaded(ReportViewerOptions reportOption)
    {
        // You can update report options here
    }
}

```

```
}  
,
```

Create ASP.NET Core Web API Service

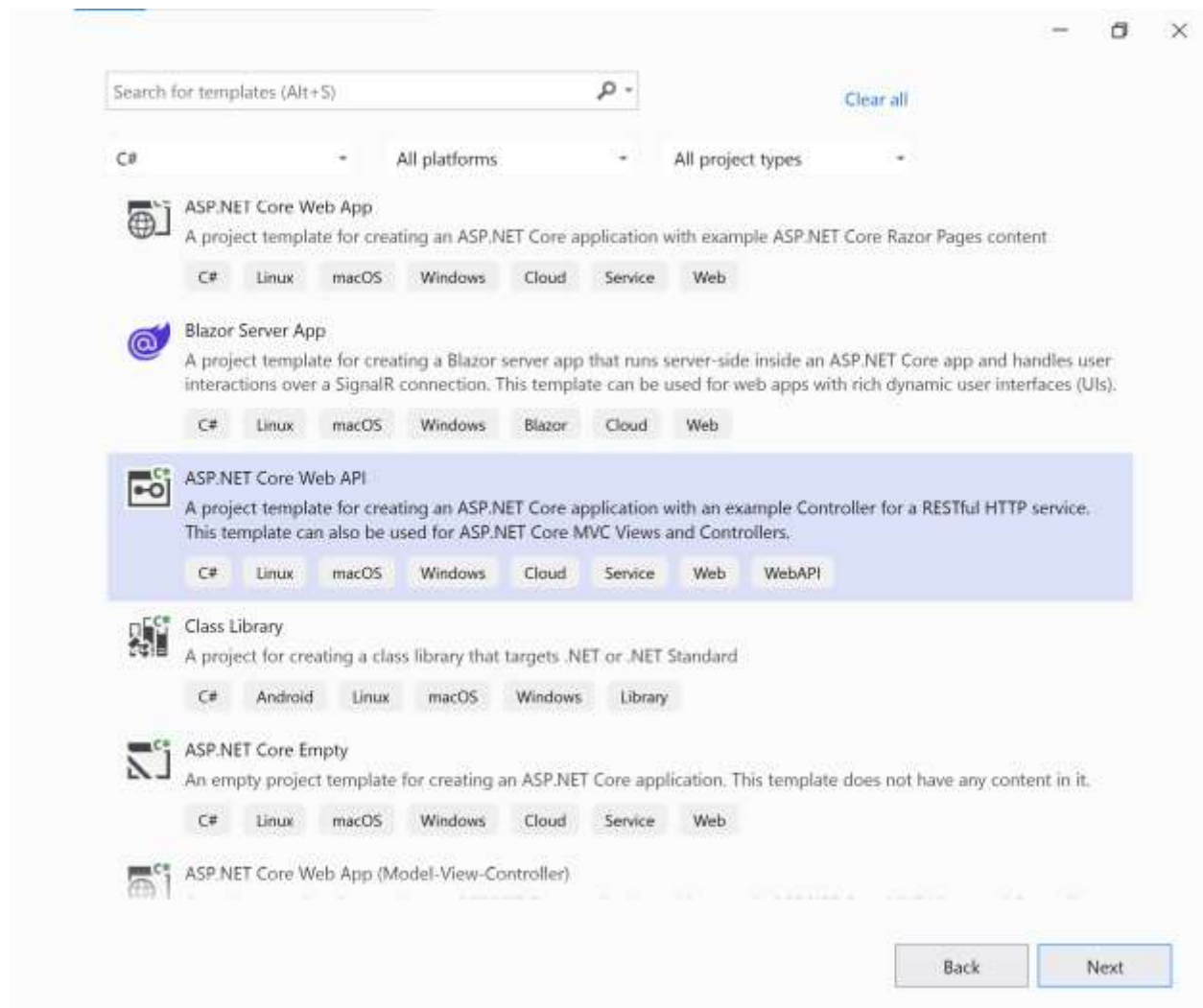
To create an ASP.NET Core Web API for Report Viewer using a new ASP.NET Core Web Application template, follow these steps:

Bold Reports ASP.NET Core supports from **.NET Core 2.1** only. So, choose the .NET Core version **ASP.NET Core 2.1** or higher versions for Viewer API creation.

To get start quickly with ASP.NET Core Web API for Report Viewer, you can check on this video:

youtube: <https://youtu.be/b1yZGAeWoHQ>

1. Start Visual Studio 2022 and click **Create new project**.
2. Choose **ASP.NET Core Web API**, and then click **Next**.



3. Change the project name, and then click **Next**.

4. In the dropdown for the **ASP.NET Core version**, choose **ASP.NET Core 6.0**, then click **Create**.

The screenshot shows the 'Additional information' configuration page for an ASP.NET Core Web API service. At the top, there are tabs for 'ASP.NET Core Web API', 'CR', 'Linux', 'macOS', 'Windows', 'Cloud', 'Service', 'Web', and 'WebAPI'. The 'Framework' dropdown is set to '.NET 6.0 (Long Term Support)'. The 'Authentication type' is set to 'None'. There are checkboxes for 'Configure for HTTPS' (checked), 'Enable Docker' (unchecked), and 'Docker OS' (set to 'Linux'). At the bottom, there are checkboxes for 'Use controllers (unchecked to use minimal APIs)' (checked), 'Enable OpenAPI support' (checked), and 'Do not use top-level statements' (unchecked). 'Back' and 'Create' buttons are at the bottom right.

If you need to use Bold Reports with ASP.NET Core on Linux or macOS, then refer to this [Can Bold Reports be used with ASP.NET Core on Linux and macOS](#) section.

List of dependency Libraries

The Web API service configuration requires the following reporting server-side packages. Right-click the project or solution in the **Solution Explorer** tab, and choose **Manage NuGet Packages** and then search for **BoldReports.Net.Core** package, and install to the application. The following provides detail of the packages and its usage.

Package | Purpose

Syncfusion.Compression.Net.Core | Supports for exporting the report to PDF, Microsoft Word, and Microsoft Excel format. It is a base library for the packages **Syncfusion.Pdf.Net.Core**, **Syncfusion.DocIO.Net.Core** and **Syncfusion.XlsIO.Net.Core**.

Syncfusion.Pdf.Net.Core | Supports for exporting the report to a PDF.

Syncfusion.DocIO.Net.Core | Supports for exporting the report to a Word.

Syncfusion.XlsIO.Net.Core | Supports for exporting the report to an Excel.

Syncfusion.OfficeChart.Net.Core | It is a base library of the **Syncfusion.XlsIO.Net.Core** package.

Newtonsoft.Json | Serialize and deserialize the data for report viewer. It is a mandatory package for the report viewer, and the package version should be higher of 10.0.1 for NET Core 2.0 and others should be higher of 9.0.1.

System.Data.SqlClient | This is an optional package for the report viewer. It should be referred in project when renders the RDL report and which contains the SQL Server and SQL Azure data source. Also, the package version should be higher of 4.1.0.

Configure Web API

The interface **IReportController** has declaration of action methods that are defined in the Web API Controller for processing the RDL, RDLC, and SSRS reports and for handling request from the Report Viewer control. The IReportController has the following action methods declaration:

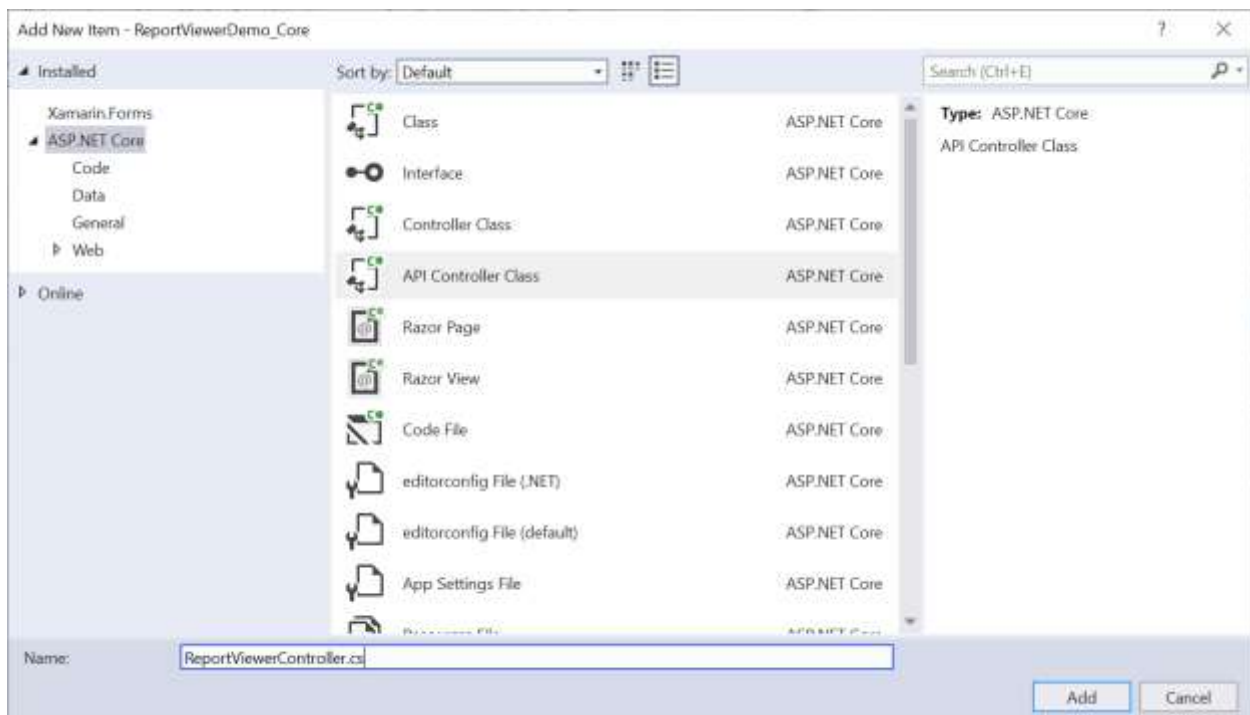
Methods | Description

GetResource | Returns the report resource to the requested key.

ProcessReport | Processes the report request and returns the result.

Add Web API Controller

1. Right-click the project and select **Add > New Item** from the context menu.
2. In the Add New Item dialog, select **API Controller** class and name it as **ReportViewerController.cs**



3. Click **Add**.

While adding API Controller class, name it with the suffix **Controller** that is mandatory.

4. Open the **ReportViewerController** and add the following using statement.

```
`csharp
using BoldReports.Web.ReportViewer;
`
```


5. Inherit the `IReportController` interface, and then implement its methods.
6. Create local references for the interfaces given in following table.

Interface | Purpose

`IMemoryCache` | Report Viewer requires a memory cache to store the information of consecutive client request and have the rendered report viewer information in server.

`IWebHostEnvironment` | `IWebHostEnvironment` used to get the report stream from application `wwwroot\Resources` folder.

7. Next, add the `[EnableCors]` attribute to the `ReportViewerController` class and specify the policy name which given in `Program.cs`.

```
`csharp
[Route("api/[controller]/[action]")]
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
}
`
```

8. You cannot load the application report with path information in ASP.NET Core service. So, you should load the report as stream in `OnInitReportOptions`.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    string basePath = _hostingEnvironment.WebRootPath;
    // Here, we have loaded the sales-order-detail.rdl report from application the folder
    wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
    System.IO.FileStream reportStream = new System.IO.FileStream(basePath + @"\Resources\sales-order-
    detail.rdl", System.IO.FileMode.Open, System.IO.FileAccess.Read);
    reportOption.ReportModel.Stream = reportStream;
}
`
```

9. You can replace the template code with the following code.

```
`csharp
```

```
[Route("api/[controller]/[action]")]
[Microsoft.AspNetCore.Cors.EnableCors("AllowAllOrigins")]
public class ReportViewerController : Controller, IReportController
{
    // Report viewer requires a memory cache to store the information of consecutive client request and
    // have the rendered report viewer information in server.
    private Microsoft.Extensions.Caching.Memory.IMemoryCache _cache;
    // IWebHostEnvironment used with sample to get the application data from wwwroot.
    private Microsoft.AspNetCore.Hosting.IWebHostEnvironment _hostingEnvironment;

    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    public ReportViewerController(Microsoft.Extensions.Caching.Memory.IMemoryCache memoryCache,
        Microsoft.AspNetCore.Hosting.IWebHostEnvironment hostingEnvironment)
    {
        _cache = memoryCache;
        _hostingEnvironment = hostingEnvironment;
    }

    // Post action to process the report from server based json parameters and send the result back to the
    // client.
    [HttpPost]
    public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)
    {
        return ReportHelper.ProcessReport(jsonArray, this, this._cache);
    }

    // Method will be called to initialize the report information to load the report with ReportHelper for
    // processing.
    [NonAction]
    public void OnInitReportOptions(ReportViewerOptions reportOption)
    {
        string basePath = _hostingEnvironment.WebRootPath;
        // Here, we have loaded the sales-order-detail.rdl report from application the folder
        // wwwroot\Resources. sales-order-detail.rdl should be there in wwwroot\Resources application folder.
        System.IO.FileStream reportStream = new System.IO.FileStream(basePath + @"\Resources\sales-order-
        detail.rdl", System.IO.FileMode.Open, System.IO.FileAccess.Read);
        reportOption.ReportModel.Stream = reportStream;
    }
}
```

```

}
// Method will be called when reported is loaded with internally to start to layout process with
ReportHelper.
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
}
//Get action for getting resources from the report
[ActionName("GetResource")]
[AcceptVerbs("GET")]
// Method will be called from Report Viewer client to get the image src for Image report item.
public object GetResource(ReportResource resource)
{
return ReportHelper.GetResource(resource, this, _cache);
}
[HttpPost]
public object PostFormReportAction()
{
return ReportHelper.ProcessReport(null, this, _cache);
}
}
`

```

The `sales-order-detail.rdl` report can be downloaded from [here](#). Also, you can add the report from Bold Reports installation location. For more information on installed sample location, see [Samples and demos](#).

10. Run the application and use the API URL in the Report Viewer `reportServiceUrl` property.

Enable Cross-Origin requests

Browser security prevents Report Viewer from making requests to your Web API Service when both runs in a different domain. To allow access to your Web API service from a different domain, you must enable cross-origin requests.

Call `AddCors` in `Program.cs` to add the CORS services to the app's service container. Replace the following code to allow any origin requests.

```

`csharp
builder.Services.AddCors(o => o.AddPolicy("AllowAllOrigins", builder =>

```

```
{  
builder.AllowAnyOrigin()  
.AllowAnyMethod()  
.AllowAnyHeader();  
});  
.....  
.....  
app.UseHttpsRedirection();  
app.UseCors();  
app.UseAuthorization();  
`
```

For more information about CORS, see [Configure CORS for API](#)

How to Create RDL/RDLC Report

The following sections explain about how to create a new RDL/RDLC report using Bold Report Designer, Microsoft Report Builder and Visual Studio Report Server project template.

- [Create a RDL report](#)
- [Create a RDLC report](#)
- [Resolve does not contains a definition issue](#)

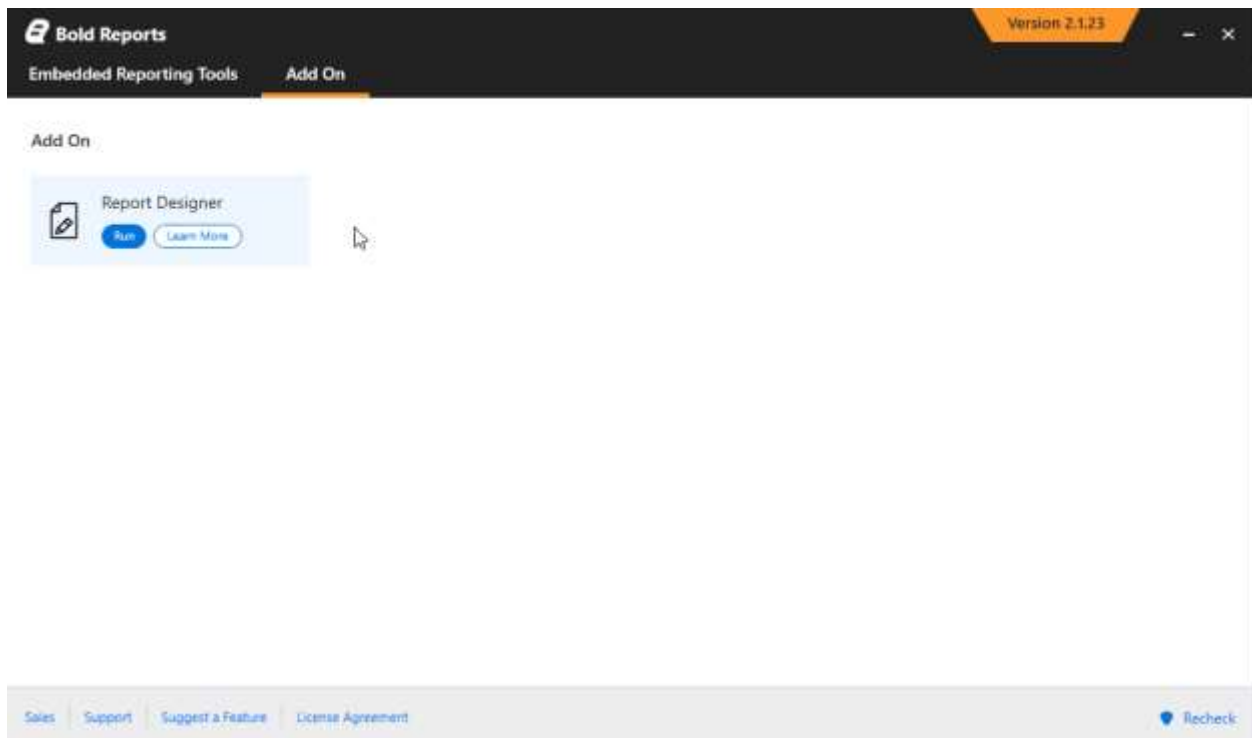
Create a SSRS RDL report

You can create an RDL report using any of the following reporting tools:

- Bold Reports Report Designer.
- Microsoft Report Builder.
- Visual Studio Report Server project template.

Bold Reports Report Designer

Bold Reports Report Designer provides the intuitive user interface to create and edit the RDL reports, which is available in Bold Embedded Reporting Tools Control Panel Add On.



Microsoft SQL Report Builder

You can create an RDL report using the Microsoft stand-alone Report Builder. For more details, refer to this [online documentation](#).

Visual Studio Report Server template

To create an RDL report in Visual Studio, a Report Server project is required where you can save your report definition (.rdl) file. For more details, refer to this [Visual Studio documentation](#).

If you do not have the Business Intelligence or Report Server Project options, you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create a RDLC report using business object data source

This section describes step by step procedure to create an RDLC report using Visual Studio Reporting project type.

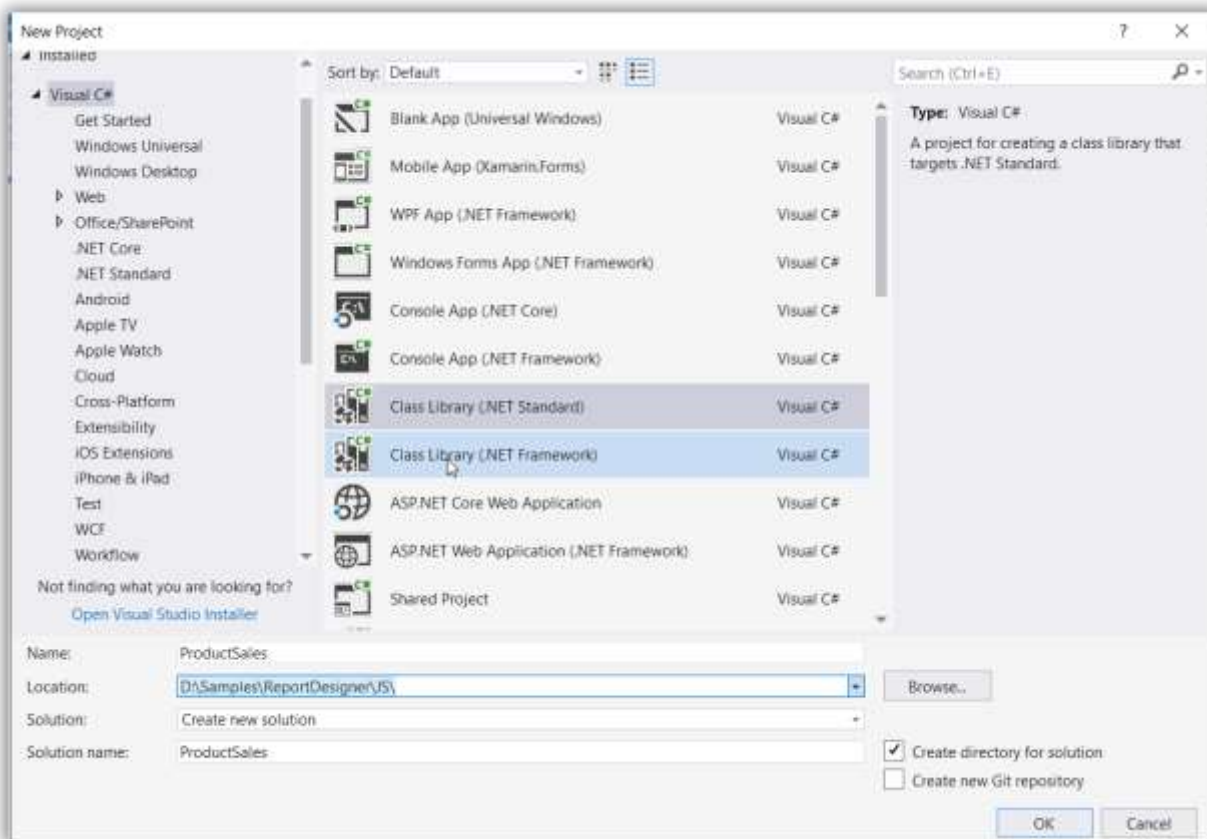
Prerequisites

- Microsoft Visual Studio 2017 or higher
- [Microsoft RDLC Report Designer](#)

If you are using Microsoft Visual Studio lower to 2017 version then you should update SSDT with the Business Intelligence templates. Refer to [Download SQL Server Data Tools](#).

Create business object class

1. Open Visual Studio from the File menu and select **New Project**.
2. Create project with class library type from the project type list



3. Create the class with necessary properties. You can find the reference below,

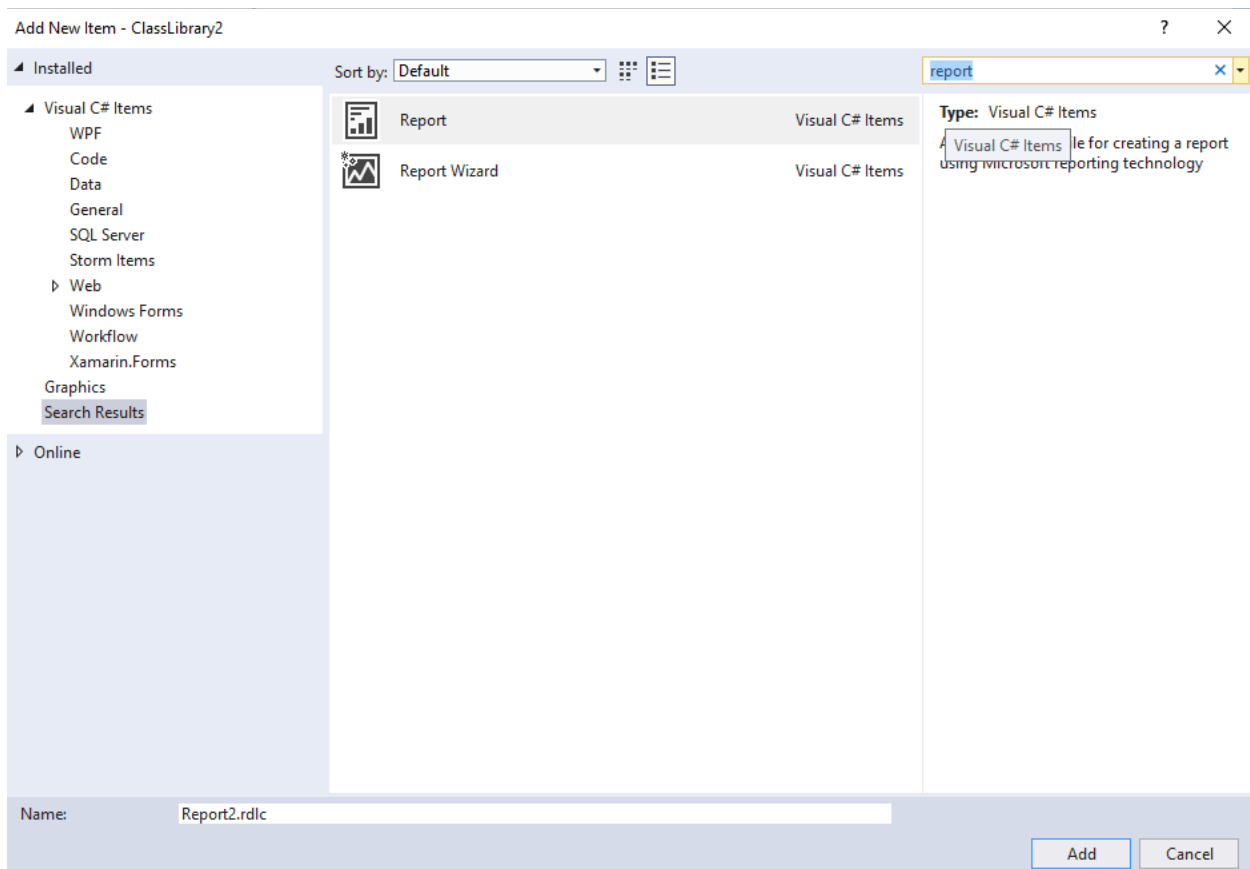
```
`csharp
public class ProductSales
{
    public string ProdCat { get; set; }
    public string SubCat { get; set; }
    public string OrderYear { get; set; }
    public string OrderQtr { get; set; }
    public double Sales { get; set; }
}
`
```

4. Clean and build the application.

Add an RDLC report

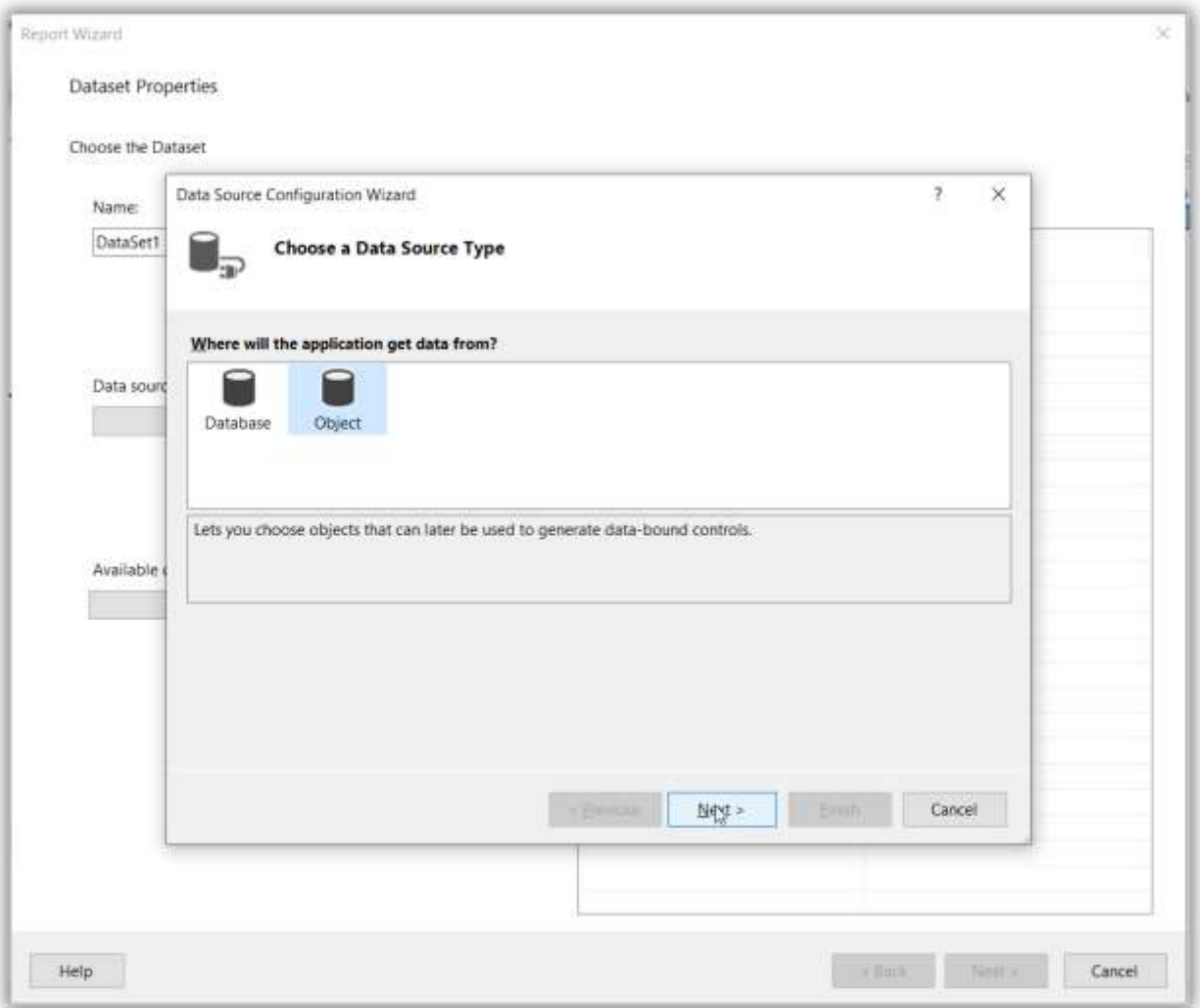
1. Right-click the project and click **Add > New Item**.

2. Search Report with new item and select **Report Wizard** to start the report creation with dataset selection.
3. Click **Add**.

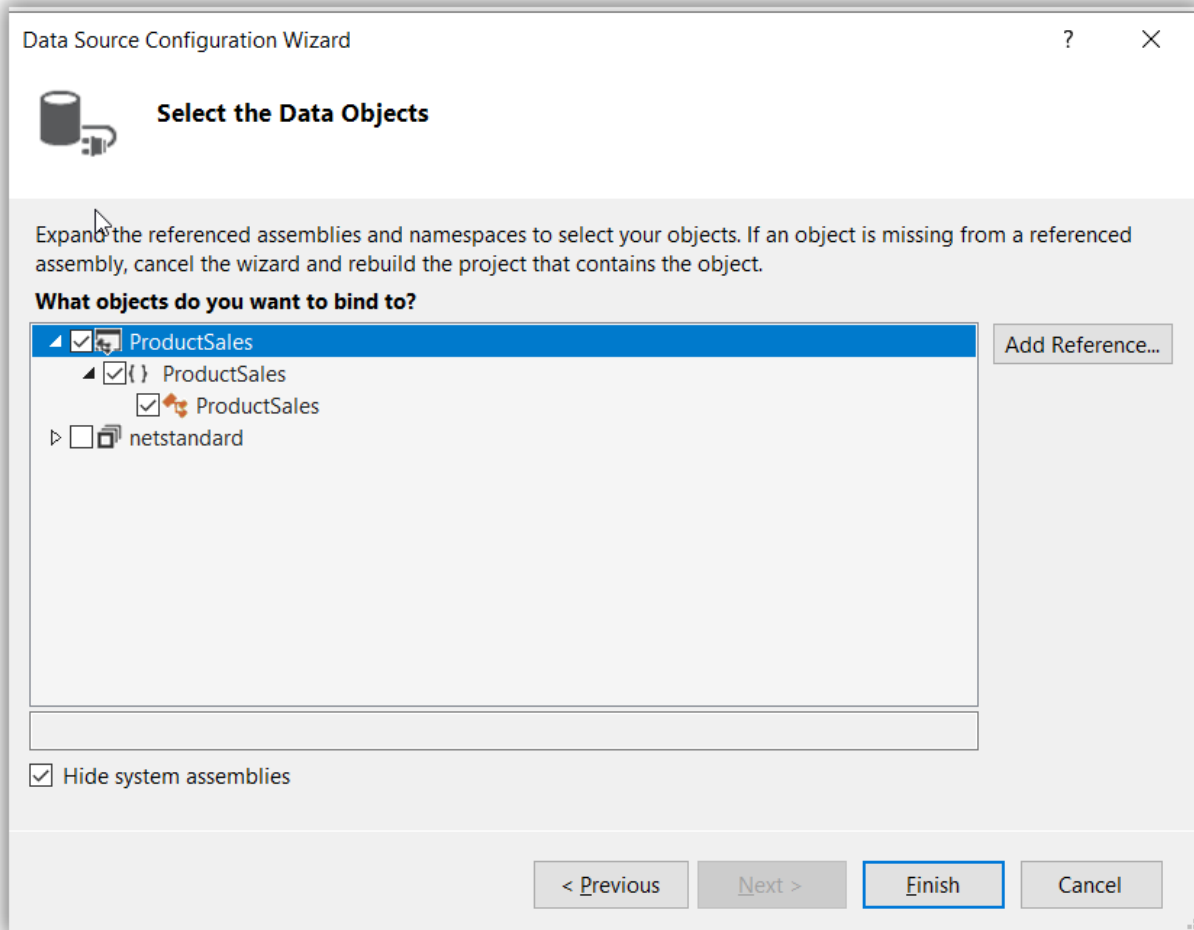


Data source and table configuration wizard

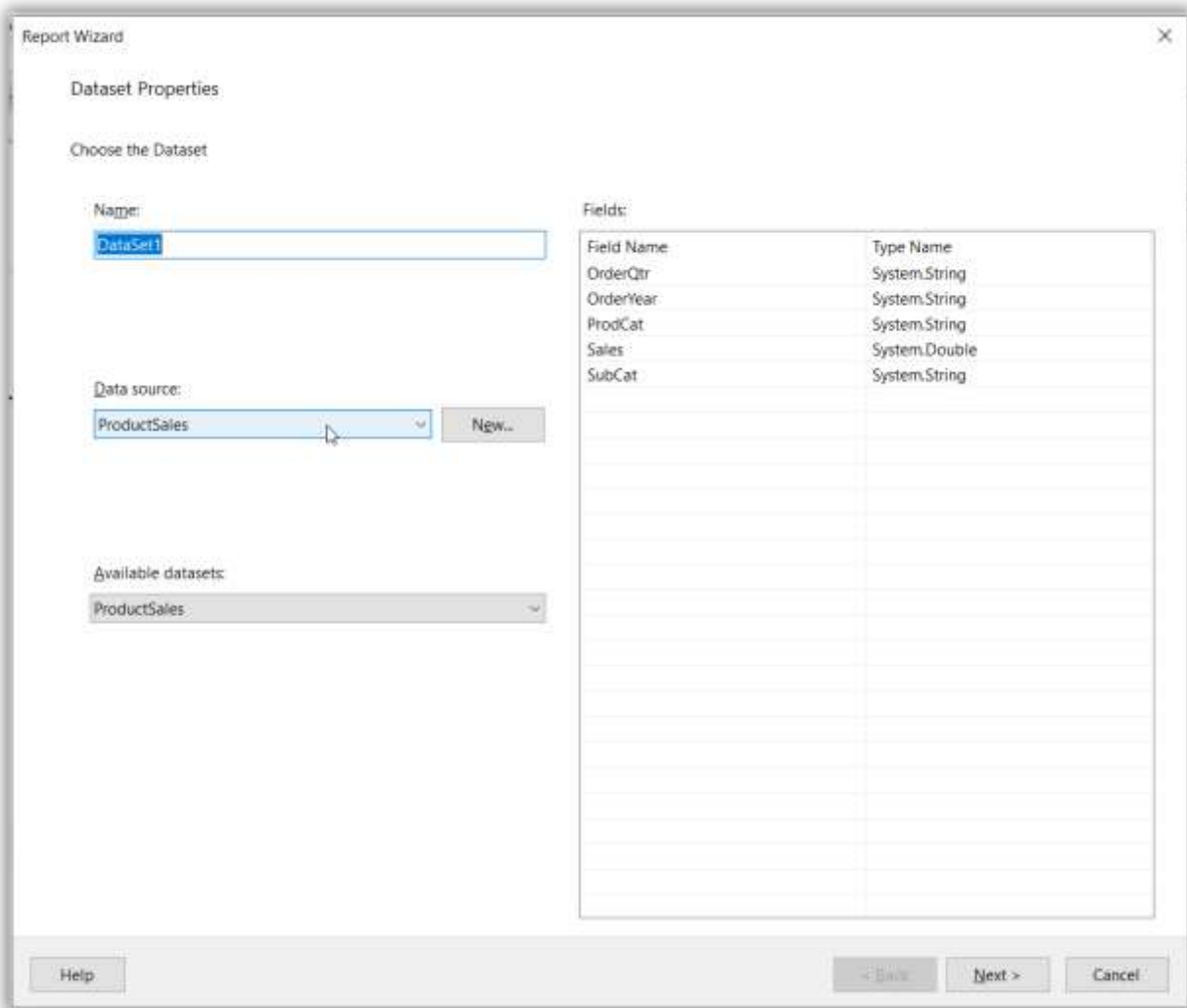
1. Choose object type from the Data Source Configuration wizard and click **Next**.



2. Expand the tree view and select **ProductSales**, and then click **Finish**.



3. In the DataSet Properties wizard, specify the dataset name as `SalesData`.

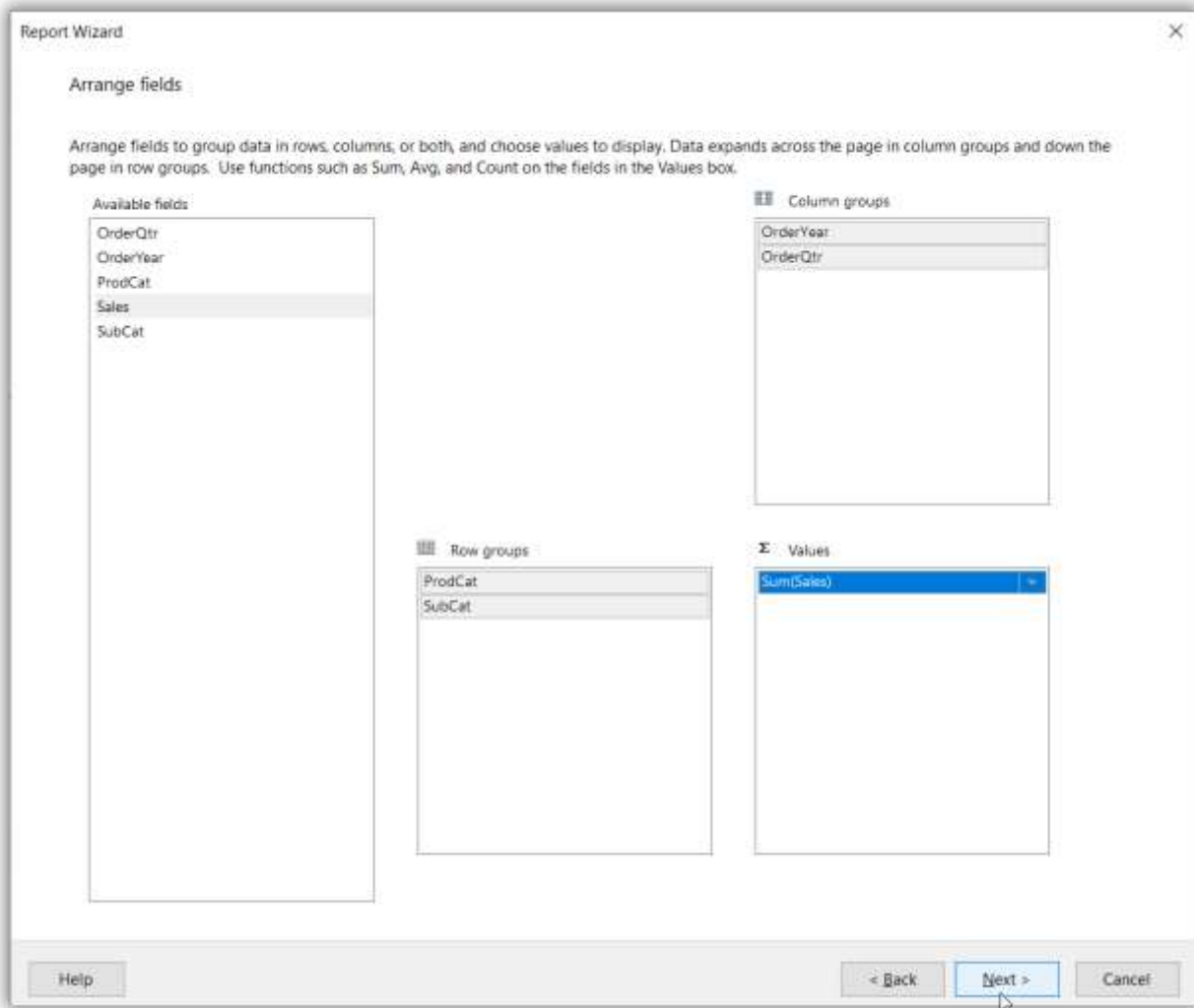


The image shows the 'Report Wizard' dialog box, specifically the 'Dataset Properties' tab. The 'Choose the Dataset' section is active. The 'Name' field contains 'DataSet1'. The 'Data source' dropdown menu is set to 'ProductSales', with a 'New...' button next to it. Below this, the 'Available datasets' list also shows 'ProductSales'. On the right, the 'Fields' table lists the following fields and their types:

Field Name	Type Name
OrderQtr	System.String
OrderYear	System.String
ProdCat	System.String
Sales	System.Double
SubCat	System.String

At the bottom of the dialog, there are 'Help', '< Back', 'Next >', and 'Cancel' buttons.

4. Drag the fields into Values, Row, and Column groups, and then click **Next**.



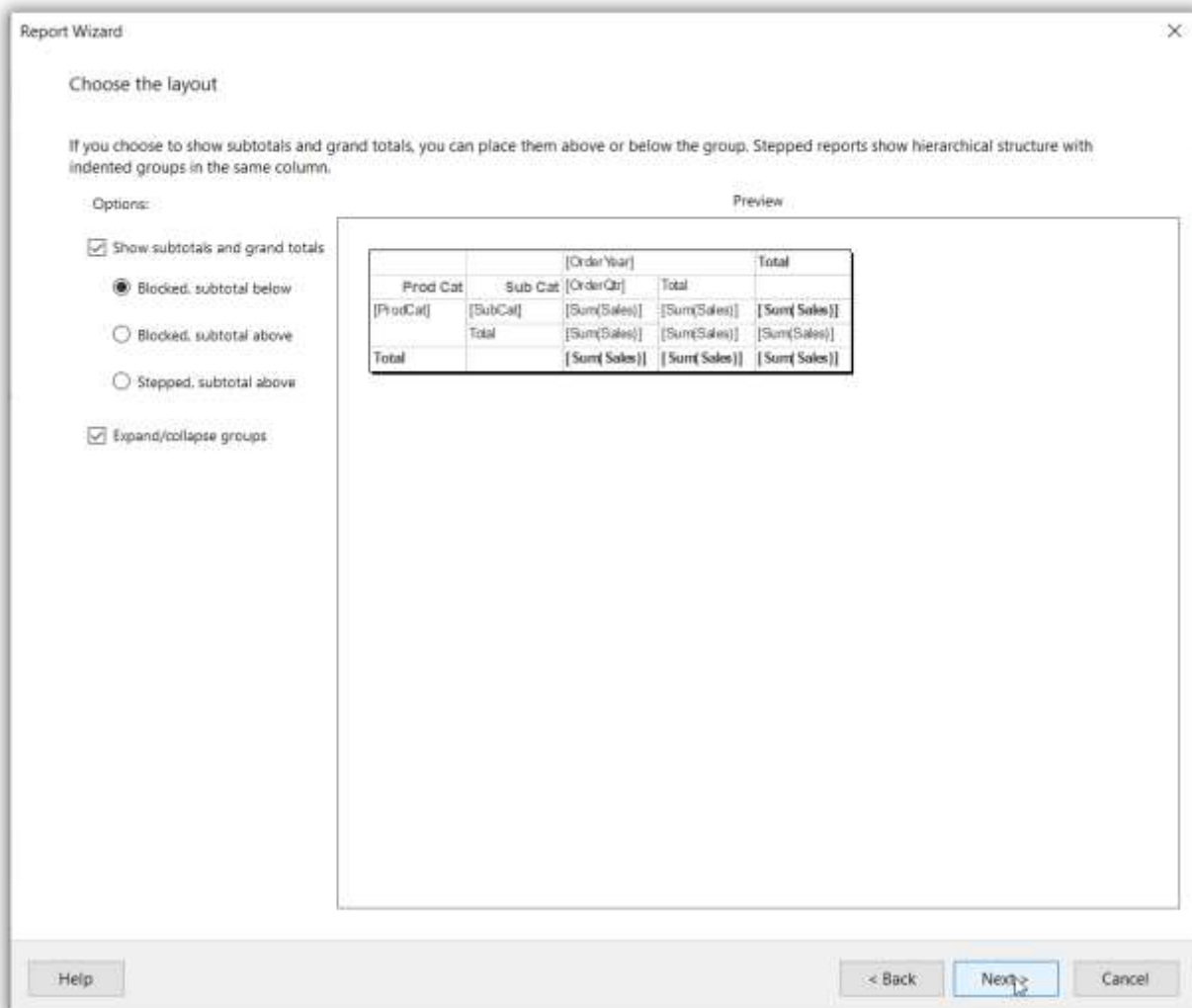
The image shows the 'Report Wizard' dialog box, specifically the 'Arrange fields' step. The dialog has a title bar with 'Report Wizard' and a close button. Below the title bar is the section 'Arrange fields' with a description: 'Arrange fields to group data in rows, columns, or both, and choose values to display. Data expands across the page in column groups and down the page in row groups. Use functions such as Sum, Avg, and Count on the fields in the Values box.'

The dialog is divided into four main areas:

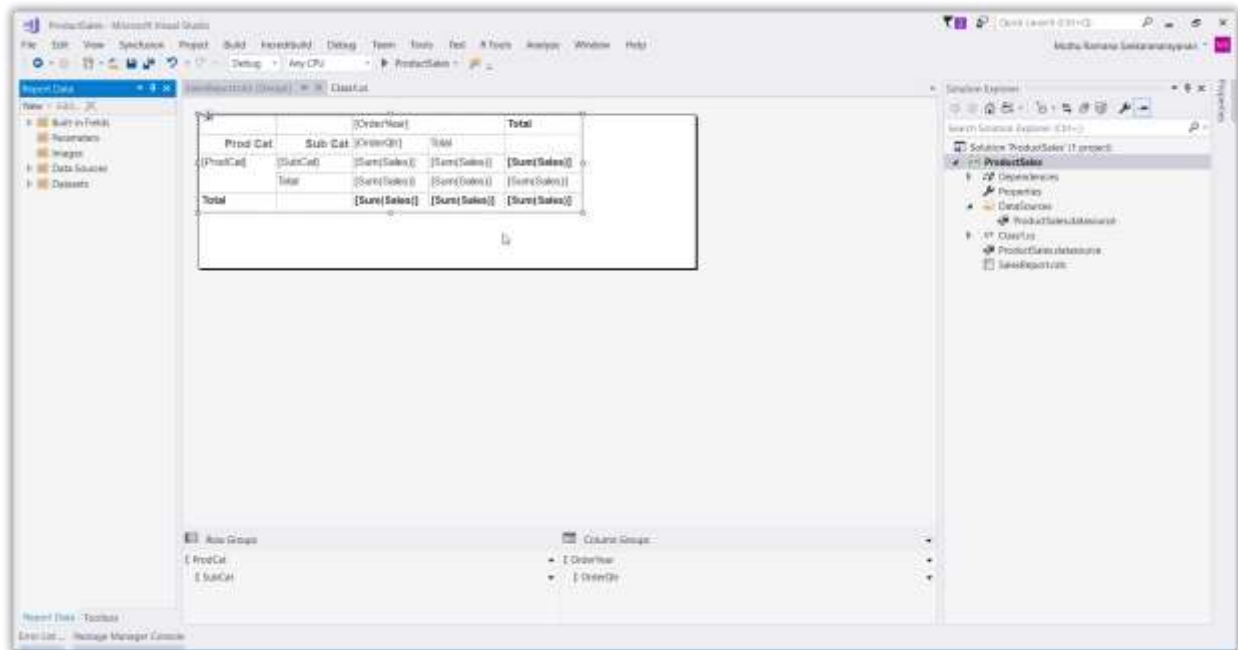
- Available fields:** A list box containing 'OrderQty', 'OrderYear', 'ProdCat', 'Sales', and 'SubCat'. 'Sales' is currently selected.
- Column groups:** A list box containing 'OrderYear' and 'OrderQty'.
- Row groups:** A list box containing 'ProdCat' and 'SubCat'.
- Σ Values:** A list box containing 'Sum(Sales)'.

At the bottom of the dialog, there are three buttons: 'Help', '< Back', and 'Next >', and a 'Cancel' button on the far right. A mouse cursor is pointing at the 'Next >' button.

5. Choose the table layout and click **Next**.
6. Select table style and click **Finish**.



Now, the RDLC report is displayed in the Visual Studio as follows.



Resolve does not contains a definition issue

Use the **x:Name** directive instead of **Name** for resolving does not contain a definition for and no extension method can be found with control initialization in XAML page. Also, You can get more details for this from the following reference.

[How to resolve does not contains a definition issue](#)

Bold Report Writer

Report Writer is a class library that enables the user to render reports defined in Microsoft's RDL format (2008 or 2008 R2) as PDF, Word, Excel or CSV documents.

The important features of UWP Report Writer are listed as follows:

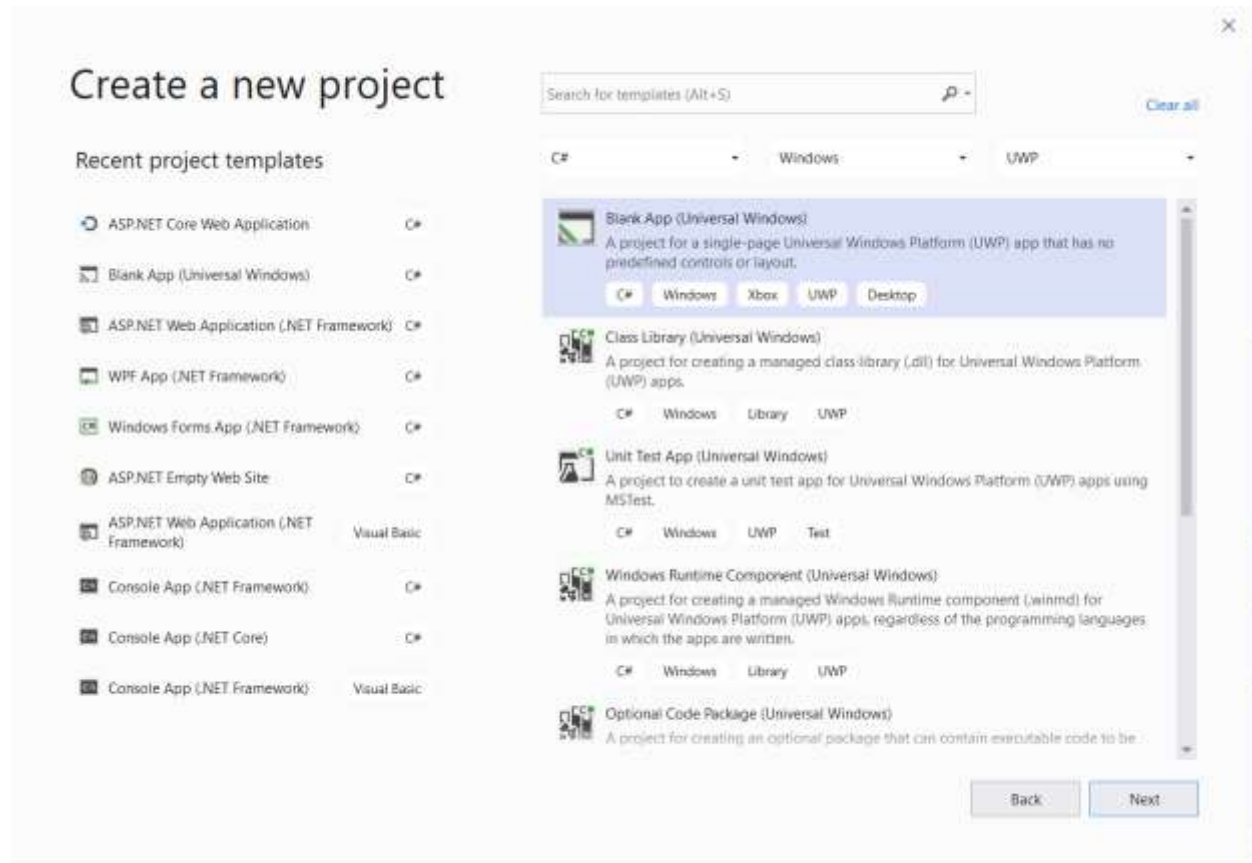
- RDL Specification - Supports RDL specification for the SQL Server 2008 and RDL specification for the SQL Server 2008 R2 only. List of available report definition formats: [msdn.microsoft.com/library/dd297486\(SQL.100\)](https://msdn.microsoft.com/library/dd297486(SQL.100)).
- Data sources - You can use advanced database servers Data Sources in Report Writer (SQL and Oracle).
- Charts - Show all basic types of charts that are available in Microsoft RDL reports.
- Tablix - Shows the summaries and simple tables.
- Gauge - Shows measurement values by using expression values.
- Textbox - Shows textbox data with expression support.
- Export - Export report as PDF, Word, Excel and CSV.
- Report Parameter - Views the report based on the report parameter value.

Export SSRS RDL Report in Bold Reports UWP Report Writer

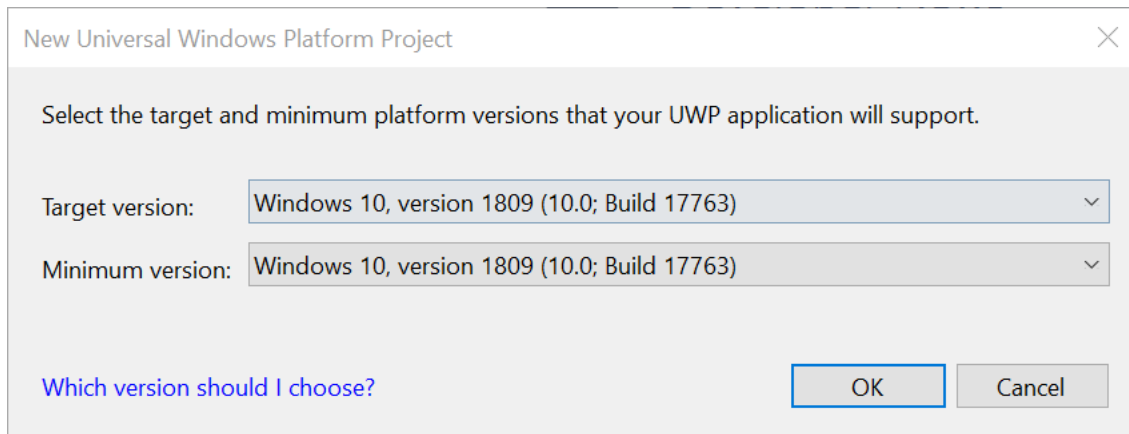
The Report Writer is a class library that is used to export the RDL report with popular file formats like PDF, Microsoft Word, Microsoft CSV, and Microsoft Excel without previewing the report on a webpage. This section describes how to export the RDL report in UWP application using the Report Writer.

Create UWP application

1. Start Visual Studio 2019 and click **Create new project**.
2. Choose **Blank App (Universal Windows)**, and then click **Next**.



3. Change the project name, and then click **Create**.
4. Select the target and minimum platform version **Windows 10, version 1809(10.0; Build 17763)** from the dropdown, and then click **OK**.



Set the same version for the target and the minimum platform version.

Configure Report Writer in an application

1. Right-click the project or solution on the Solution Explorer tab and choose **Manage NuGet Packages**. Alternatively, select **Tools > NuGet Package Manager > Manage NuGet Packages for Solution**.

Refer to the [NuGet Packages](#) to learn more details about installing and configuring the Report writer NuGet packages. You can add the reports from the Syncfusion installation location. For more information, refer to the [samples and demos](#) section.

2. Search for **BoldReports.UWP** NuGet package and install them in your UWP application.

Package | Purpose

BoldReports.UWP | Contains UWP Reporting controls (Report Writer) to preview and export the reports.

Adding already created report

In this tutorial, the **Product List.rdlc** report is used, and it can be downloaded at this [link](#).

1. Create a folder **Resources** in your application to store the RDLC reports.
2. Add already created reports to the newly created folder.

Initialize Report Writer

1. Initialize the Report Writer export type inside the `Page` tag as shown below in the **MainWindow.xaml** file,

```
`csharp
```

```
<Page
```

```
....
```

```
....
```

```

....
xmlns:BoldReports="using:BoldReports.UI.Xaml"
....>
<Grid Margin="0" Name="grd_controlPanel" Background="{StaticResource
ApplicationPageBackgroundThemeBrush}">
<StackPanel Name="pagePanel" Grid.Row="1" Margin="20,40,0,0" Background="White">
<TextBlock FontSize="15" FontFamily="Segoe UI Regular" TextWrapping="Wrap" Padding="5,5,5,5" >
<TextBlock.Text>This sample shows the capability of exporting a RDLC report into various file formats
like PDF, WORD, EXCEL and HTML using Local export mode of Report Writer. Choose a file format and
click generate button to view the selected document generated from report file.</TextBlock.Text>
</TextBlock>
<StackPanel Margin="0,10,0,0" Orientation="Vertical">
<RadioButton FontSize="15" FontFamily="Segoe UI Regular" Content="PDF" HorizontalAlignment="Left"
Margin="20,10,0,0" x:Name="pdf" IsChecked="true" VerticalAlignment="Top"/>
<RadioButton FontSize="15" FontFamily="Segoe UI Regular" Content="Word"
HorizontalAlignment="Left" Margin="20,10,0,0" x:Name="word" VerticalAlignment="Top"/>
<RadioButton FontSize="15" FontFamily="Segoe UI Regular" Content="Excel"
HorizontalAlignment="Left" Margin="20,10,0,0" x:Name="excel" VerticalAlignment="Top"/>
<RadioButton FontSize="15" FontFamily="Segoe UI Regular" Content="HTML" x:Name="html"
VerticalAlignment="Top" Margin="20,10,0,0" Width="90"/>
<Button Click="Button_Click" HorizontalAlignment="Left" Margin="20,10,0,0"
VerticalAlignment="Bottom" BorderBrush="LightBlue" Background="#8bb54a">
<StackPanel Orientation="Horizontal" Background="#8bb54a" Width="144">
<TextBlock Foreground="ffffff" FontSize="16" FontFamily="Segoe UI Bold" Text="Generate"
HorizontalAlignment="Right" Margin="30,5,0,0" Width="110" VerticalAlignment="Center"
Height="27"/>
</StackPanel>
</Button>
</StackPanel>
</StackPanel>
</Grid>
</Page>
,

```

Set Report Path

1. Open the `MainWindow.xaml.cs` file and add the following using statement.


```
`csharp
using BoldReports.Writer;
using Windows.Storage.Pickers;
using System.Reflection;
using Windows.Storage.Streams;
using Windows.Storage;
using Windows.UI.Popups;
using System.Collections;
`
```

2. Create a class and methods that returns business object data collection. Use the following code in your application.

```
`csharp
public class ReportData
{
    public string ProdCat { get; set; }
    public string SubCat { get; set; }
    public double? OrderYear { get; set; }
    public string OrderQtr { get; set; }
    public double? Sales { get; set; }
    public static IList GetData()
    {
        List<ReportData> datas = new List<ReportData>();
        ReportData data = null;
        data = new ReportData()
        {
            ProdCat = "Accessories",
            SubCat = "Helmets",
            OrderYear = 2002,
            OrderQtr = "Q1",
            Sales = 4945.6925
        };
        datas.Add(data);
        data = new ReportData()
    }
}
```

```
{
  ProdCat = "Components",
  SubCat = "Road Frames",
  OrderYear = 2002,
  OrderQtr = "Q3",
  Sales = 957715.1942
};
datas.Add(data);
data = new ReportData()
{
  ProdCat = "Components",
  SubCat = "Forks",
  OrderYear = 2002,
  OrderQtr = "Q4",
  Sales = 23543.1060
};
datas.Add(data);
data = new ReportData()
{
  ProdCat = "Bikes",
  SubCat = "Road Bikes",
  OrderYear = 2002,
  OrderQtr = "Q1",
  Sales = 3171787.6112
};
datas.Add(data);
data = new ReportData()
{
  ProdCat = "Accessories",
  SubCat = "Helmets",
  OrderYear = 2002,
  OrderQtr = "Q3",
  Sales = 33853.1033
}
```

```
};  
datas.Add(data);  
return datas;  
}  
}  
`
```

3. Initialize the ReportWriter by using the following code example in the `MainWindow.xaml.cs` file export button click event.

```
`csharp  
async void Button_Click(object sender, RoutedEventArgs e)  
{  
    FileSavePicker fileSavePicker = new FileSavePicker();  
    WriterFormat format = WriterFormat.PDF;  
    if (pdf.IsChecked == true)  
    {  
        fileSavePicker.FileTypeChoices.Add("PDF", new List<string> { ".pdf" });  
        fileSavePicker.DefaultFileExtension = ".pdf";  
        format = WriterFormat.PDF;  
    }  
    else if (excel.IsChecked == true)  
    {  
        fileSavePicker.FileTypeChoices.Add("Excel", new List<string> { ".xlsx" });  
        fileSavePicker.DefaultFileExtension = ".xlsx";  
        format = WriterFormat.Excel;  
    }  
    else if (word.IsChecked == true)  
    {  
        fileSavePicker.FileTypeChoices.Add("Word", new List<string> { ".docx" });  
        fileSavePicker.DefaultFileExtension = ".docx";  
        format = WriterFormat.Word;  
    }  
    else if (html.IsChecked == true)  
    {
```

```
fileSavePicker.FileTypeChoices.Add("Html", new List<string> { ".html" });
fileSavePicker.DefaultFileExtension = ".html";
format = WriterFormat.HTML;
}
fileSavePicker.SuggestedFileName = "ExportReport";
var savedItem = await fileSavePicker.PickSaveFileAsync();
if (savedItem != null)
{
    MemoryStream exportFileStream = new MemoryStream();
    Assembly assembly = typeof(MainPage).GetType().Assembly;
    // Ensure the report location and application name.
    Stream reportStream = assembly.GetManifestResourceStream("<application name>.Resources.Product
    List.rdlc");
    BoldReports.UI.Xaml.ReportDataSourceCollection datas = new
    BoldReports.UI.Xaml.ReportDataSourceCollection();
    datas.Add(new BoldReports.UI.Xaml.ReportDataSource { Name = "Sales", Value = ReportData.GetData()
    });
    ReportWriter writer = new ReportWriter(reportStream, datas);
    writer.ExportMode = ExportMode.Local;
    writer.ExportCompleted += Writer_ExportCompleted;
    await writer.SaveAsync(exportFileStream, format);
    try
    {
        using (IRandomAccessStream stream = await savedItem.OpenAsync(FileAccessMode.ReadWrite))
        {
            // Write compressed data from memory to file
            using (Stream outstream = stream.AsStreamForWrite())
            {
                byte[] buffer = exportFileStream.ToArray();
                outstream.Write(buffer, 0, buffer.Length);
                outstream.Flush();
            }
        }
        exportFileStream.Dispose();
    }
```

```

}
catch {}
}
}

private void Writer_ExportCompleted(object sender, byte[] e)
{
    MessageBox msgDialog = new MessageBox("Report exporting completed successfully");
    msgDialog.ShowAsync();
}

```

- Build and run the application. Choose a file format and click the generate button to view the selected document generated from the report file.

Congratulations! You have completed your first UWP Writer application! Click [here](#) to download the already created UWP Report Writer application.

Note: You can refer to our feature tour page for the [UWP Report Writer](#) to see its innovative features.

How to queries for Bold Reports ReportViewer

This section helps to get the answer for the frequently asked how to queries in Bold Reports ReportViewer.

- [How to dispose the ReportViewer object?](#)
- [How to Apply Bold Reports Custom Nuget Packages?](#)
- [How to add Report Viewer in JSP?](#)
- [How to use JSON data as report data?](#)
- [How to resolve the unsupported media type error on loading image in ASP.NET Core?](#)
- [How to resolve the image rendering and exporting issue with ASP.NET Core Authentication middleware?](#)
- [How to set the parameter in web api controller?](#)
- [How to manage reports with in application using Bold Reports Report Designer?](#)
- [How to manage reports with in SQL server using Bold Reports Report Designer?](#)
- [How to resolve the Bold Reporting components undefined issue?](#)
- [How to resolve the Syncfusion components undefined issue when using Bold Reporting components?](#)
- [How to add a WebAPI Data Processing Extension for Report Designer?](#)
- [How to resolve the image rendering and exporting issue with the ASP.NET MVC Authentication filter?](#)
- [How to change the report datasource dynamically based on customer id?](#)
- [How to change the data source dynamically using Report Serializer based on customer id?](#)
- [How to connect the Report Server Report Service with application API?](#)
- [How to customize the save and open button in the Bold Reports Report Designer?](#)
- [How to pass JSON data in the Bold Reports Report Viewer and Report Writer?](#)

- [How to serialize the report using Report Serializer](#)
- [How to customize the parameter settings by parameter name?](#)
- [How to use the Report Viewer with camel case serializer settings application?](#)
- [How to add the custom http headers for the Bold Reports Report Viewer and Report Designer?](#)
- [How to generate a PDF from report in console app?](#)
- [How to change the Parameter data type?](#)
- [How to add a PostgreSQL Data Processing Extension for Report Viewer?](#)
- [How to add a WebAPI Data Processing Extension for Report Viewer?](#)
- [How to resolve the Multiple actions were found that matches the request issue?](#)

How to dispose the Web ReportViewer object

You can destroy the client and server side report viewer processing objects using the [destroy](#) method.

Example

```
`js
<div id="report viewer"></div>
<script>
var reportviewerObj = $("#reportviewer").data("boldReportViewer");
reportviewerObj.destroy();
</script>
`
```

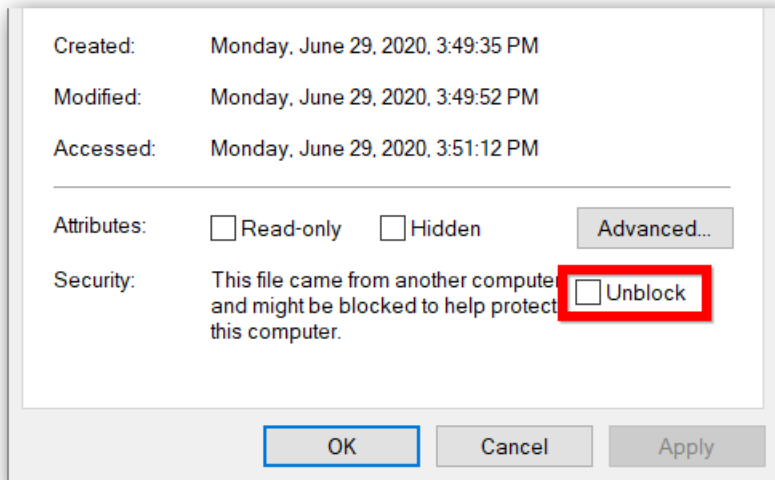
How to Apply Bold Reports Custom NuGet Packages

You are having two options to apply the patched NuGet packages in development environment. You can find the details as follows.

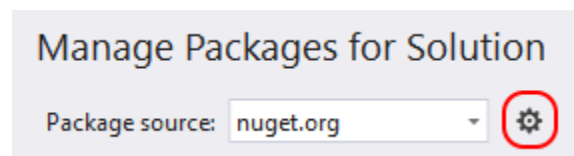
Apply Custom NuGet packages using the Visual Studio

You have to use the Visual Studio package manager to restore the custom package from the location of having the Custom NuGet packages. You can find the details as follows.

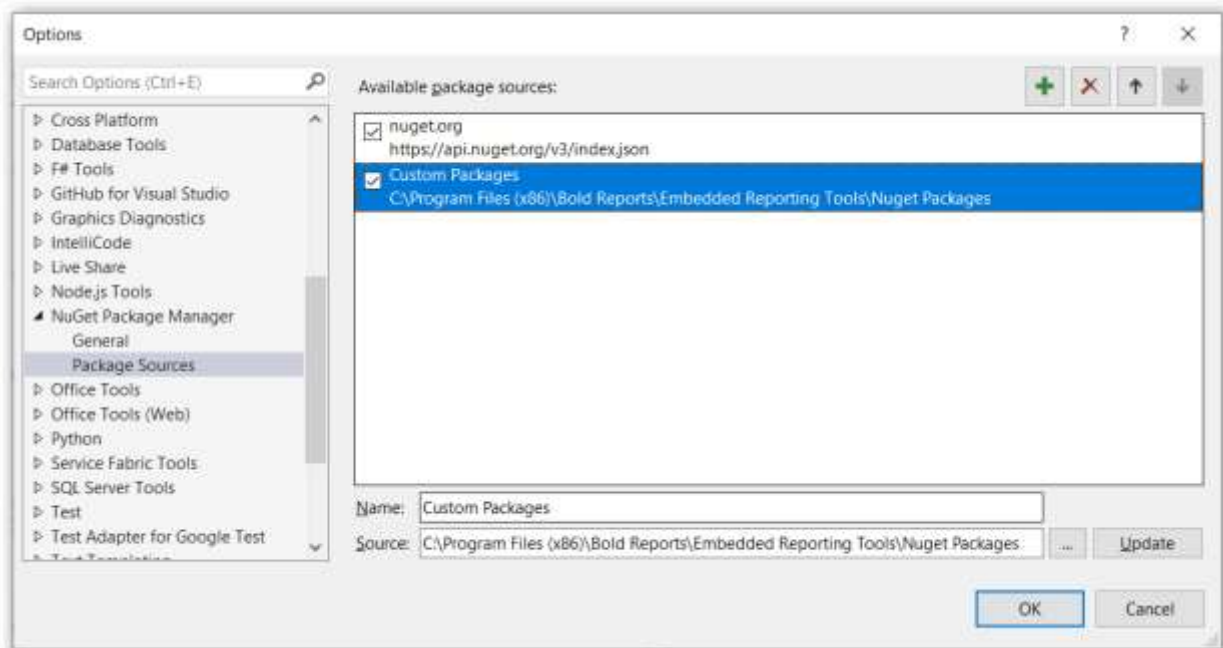
1. First, close your project in Visual Studio.
2. Delete the bin and obj folders from the project location.
3. Open the following location with help of Run. `%USERPROFILE%\nuget\packages\`. Remove the `boldreports.net.core` folder.
4. Download the provided NuGet packages.
5. Unblock the downloaded zip file and Unzip the downloaded file.



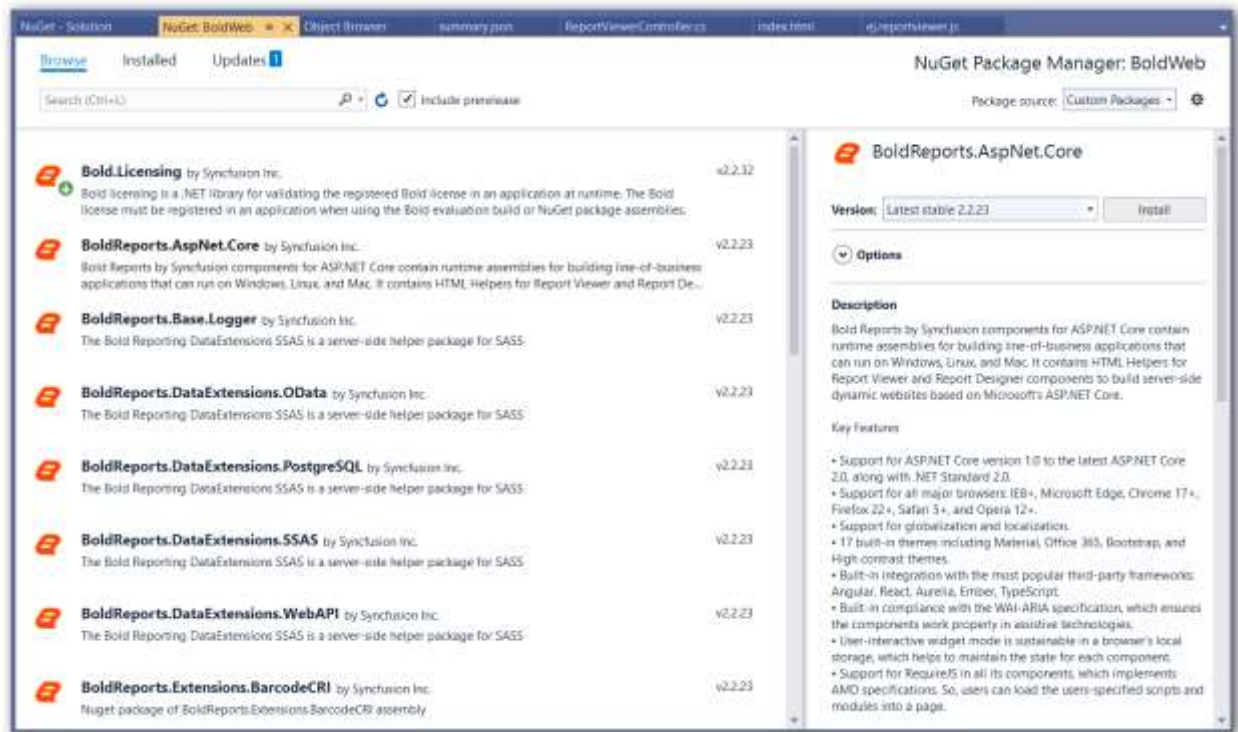
6. Open the Visual Studio NuGet package manager.
7. Select the Settings icon in the Package Manager UI outlined as follows.



8. Add a source, select +, edit the name, enter the custom package path in the Source control, and select Update. The source now appears in the selector drop-down.



- Now, choose the newly added source in package source drop-down and select it.

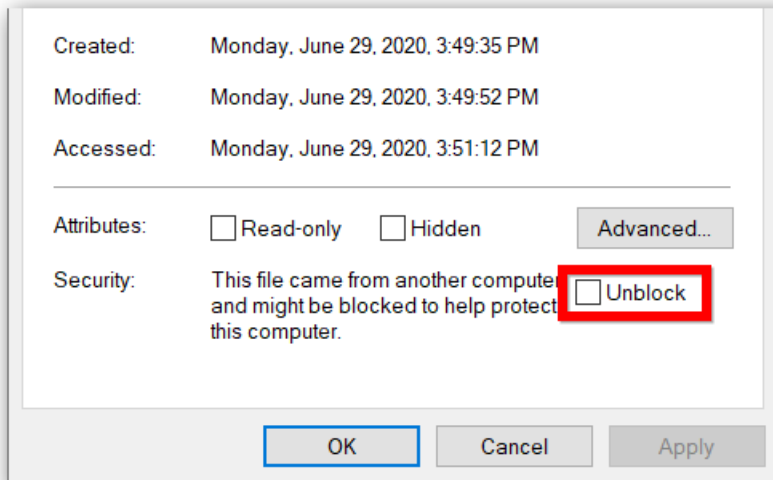


- Install the Custom NuGet Packages.

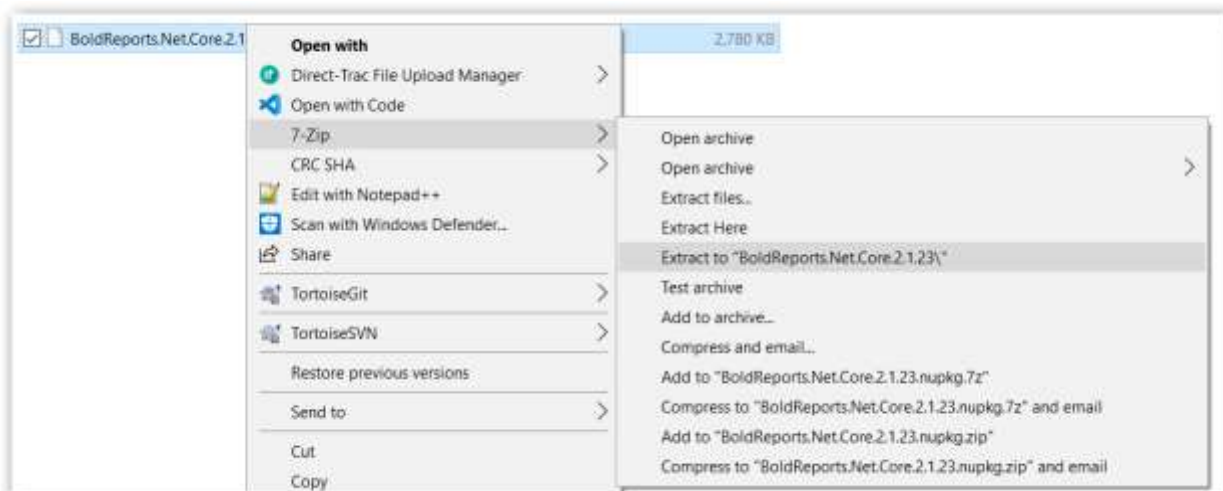
Replaced the assemblies manually

You have to use the Visual Studio package manager to restore the custom package from the location of having the Custom NuGet packages.

- First, close your project in Visual Studio.
- Delete the bin and obj folders from the project location.
- Download the provided NuGet packages.
- Unblock the downloaded zip file and Unzip the downloaded file.



5. Extract NuGet package using the 7-Zip extractor.



6. If you do not have the 7-zip, then change the `nupkg` file name as `Zip` and extract using the zip extractor.



7. Copy all folders from the lib or specific framework that you are using.
8. Open the following location with help of Run. `%USERPROFILE%\nuget\packages\bldreport s.net.core\5.4.20\lib`. Pasted the copied files.
9. Now, you can open and compile the project, this will use the provided patch assembly from the NuGet cache restored in your system.

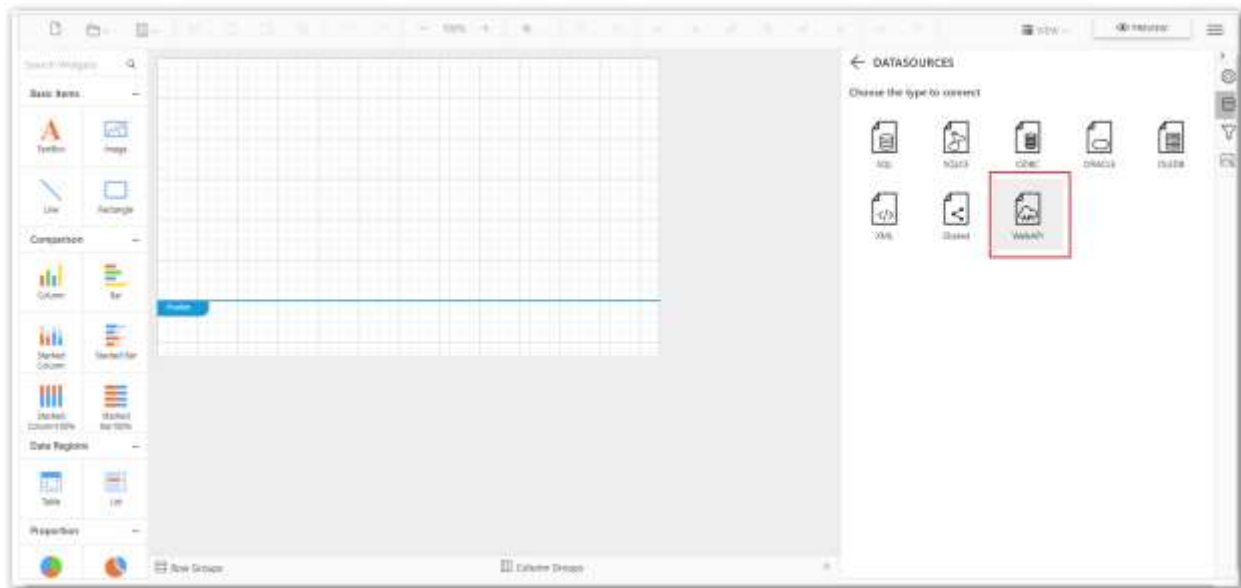
How to add Report Viewer in JSP

Use the Bold Reports [Javascript Report Viewer](#) for adding Report Viewer in the JSP application. You can find the sample from [here](#).

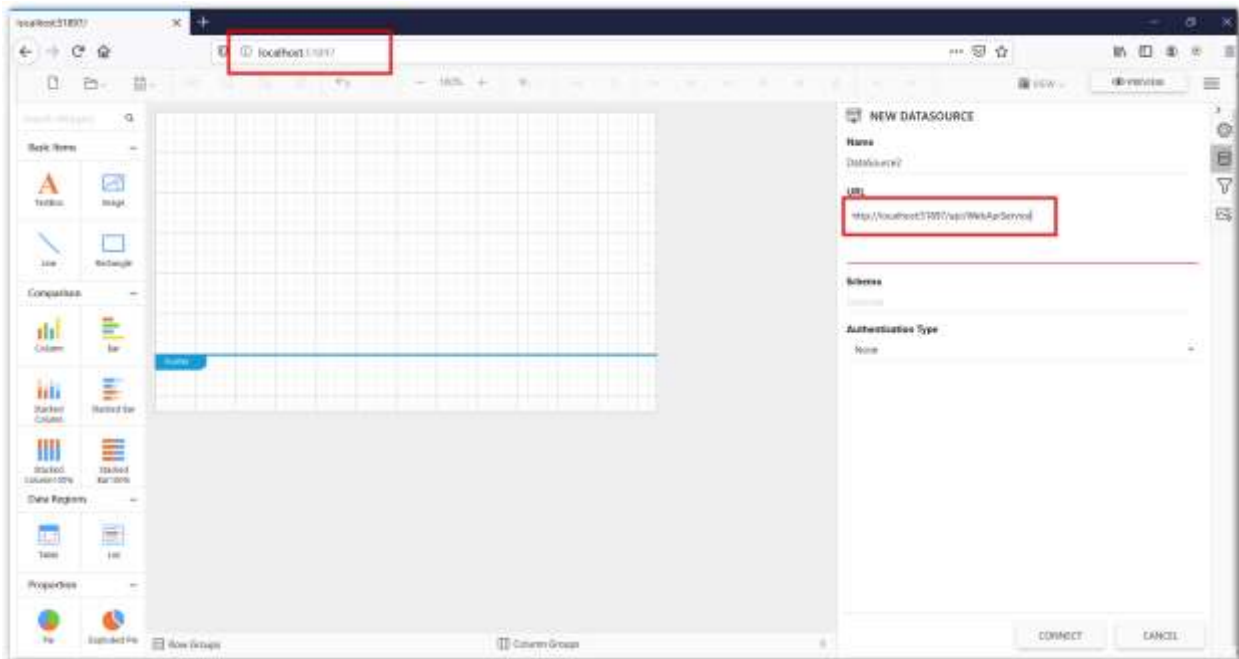
How to use JSON data as Report Data

You have to use Web Api data source for using JSON in Bold Reports.

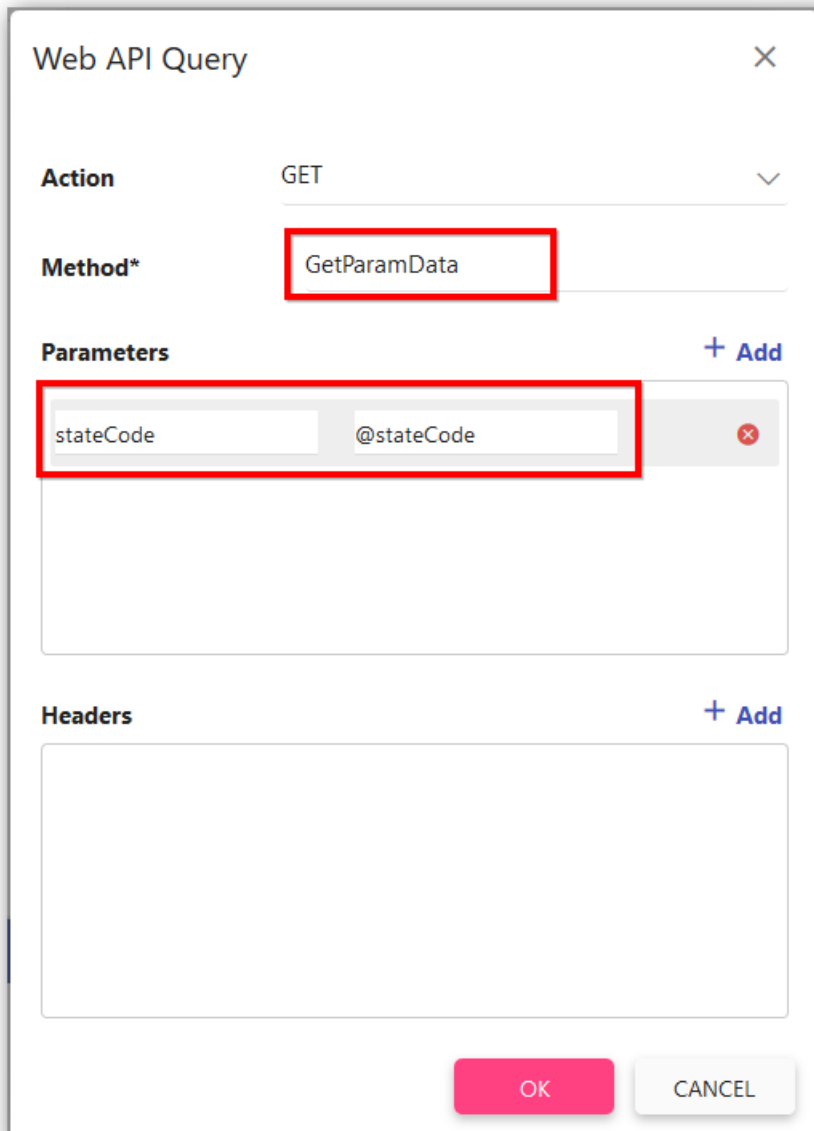
1. Run the application, add **NEW DATA** and choose **WebApi** in properties pane.



2. Provide the URL in **localhost:/api/**.



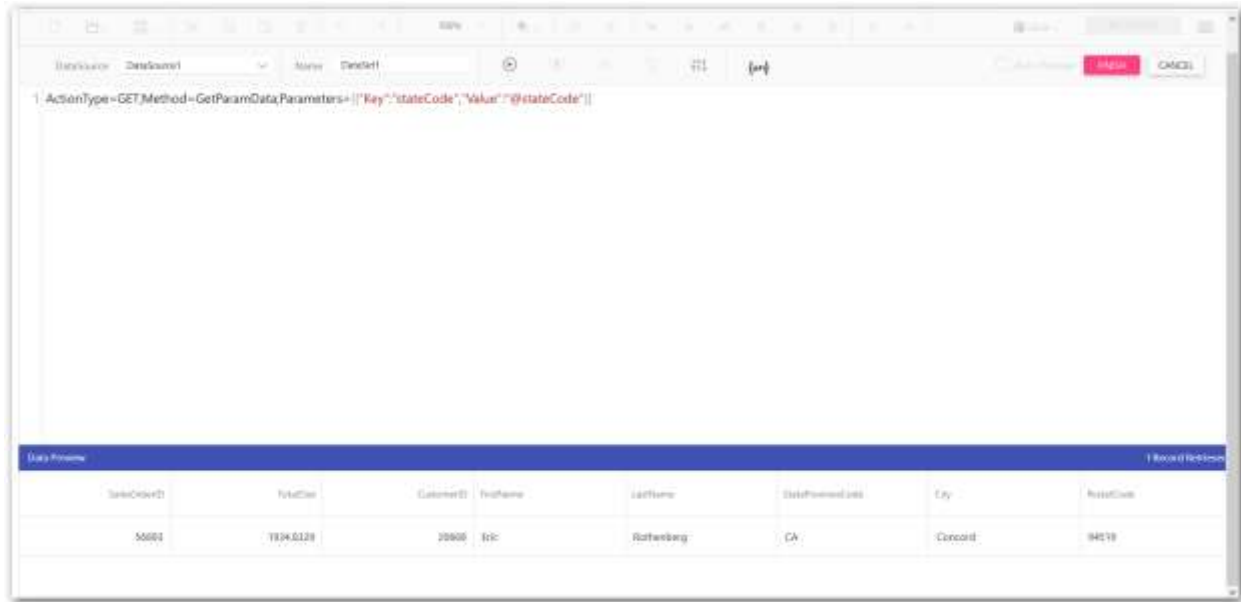
3. Mention the Web API controller method name and add parameter name and value.



The image shows a 'Web API Query' dialog box with the following fields and controls:

- Action:** A dropdown menu set to 'GET'.
- Method*:** A text input field containing 'GetParamData', which is highlighted with a red rectangle.
- Parameters:** A section with a '+ Add' button. It contains a list of parameters: 'stateCode' and '@stateCode', both highlighted with a red rectangle. A red 'X' icon is visible to the right of the list.
- Headers:** A section with a '+ Add' button and an empty list area.
- Buttons:** 'OK' and 'CANCEL' buttons at the bottom right.

4. JSON data is get from Web api controller to report data set report designer.



5. You can use this following video reference for how to use web API in Report Designer and it can be downloaded from this [link](#).
6. Use the following code sample in controller to get the JSON data.

```
`csharp
public class WebApiServiceController : ApiController
{
    [HttpPost]
    public object PostParamData([FromBody]string stateCode)
    {
        var data = StoreSales.GetData(stateCode);
        return data;
    }
    [HttpPost]
    public object PostParamInt([FromBody]int id)
    {
        return null;
    }
    [HttpGet]
    public object GetParamData(string stateCode)
    {
        var data = CustomerSales.GetData(stateCode);
    }
}
```

```
return data;
}
[HttpGet]
public object GetData()
{
    var data = StoreSales.GetData(string.Empty);
    return data;
}
}

public class CustomerSales
{
    public int SalesOrderID { get; set; }
    public double TotalDue { get; set; }
    public int CustomerID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string StateProvinceCode { get; set; }
    public string City { get; set; }
    public string PostalCode { get; set; }
    public static List<CustomerSales> GetData(string stateCode)
    {
        List<CustomerSales> customerSals = new List<CustomerSales>();
        CustomerSales cusSal = null;
        cusSal = new CustomerSales()
        {
            SalesOrderID = 56893,
            TotalDue = 1934.8329,
            CustomerID = 20668,
            FirstName = "Eric",
            LastName = "Rothenberg",
            StateProvinceCode = "CA",
            City = "Concord",
            PostalCode = "94519"
        }
    }
}
```

```
};  
customerSals.Add(cusSal);  
return customerSals;  
}  
}  
,
```

How to resolve the unsupported media type error on loading image in ASP.NET Core

Unsupported media type error occurs on the loading image in the Report Viewer and Report designer because of the `ApiController` attribute. This attribute will not allow for file byte content. You can resolve the error by following these steps.

1. Remove the `ApiController` attribute in your controller because it is not supported to get the image from API.
2. Then add the `FromBody` attribute with the `PostReportAction` action as shown below because we have removed the `ApiController` attribute from the controller.

```
[HttpPost]  
0 Verweise  
public object PostReportAction([FromBody] Dictionary<string, object> jsonArray)  
{  
    //Contains helper methods that help to process a Post or Get request from the  
    return ReportHelper.ProcessReport(jsonArray, this, this._cache);  
}
```

See also

[Display ssrs rdl report in Bold Reports ASP.NET Core Report Viewer](#)

How to resolve the image rendering and exporting issue with ASP.NET Core Authentication middleware

You could not add the authentication for export and image rendering requests from the Report Viewer and Report Designer. So, you have to ignore the authentication for the `GetResource` and `PostFormReportAction` methods using the `AllowAnonymous` attribute.

Regarding security, you will not have any issues in the aspect of security by ignoring the authentication for this `GetResource` and `PostFormReportAction` requests.

These requests are used to retrieve the file format content from the server and used with our control based on the framework suggestion to have better experience in usability in downloads and avoid the delay of rendering images with reports.

These requests will be used at the time of exporting and image rendering only, this cannot be used once again by others. This approach is similar to the [Amazon Simple Storage Service \(Amazon S3\)](#) how they are providing access to share the private files,

You can get more details of the implementation approach from these steps,

1. Before initiating a non-authentication request, we will send the authenticate request to the server to generate the export and image content.
2. The authenticated request will generate the exported with a unique server for the downloadable content and unique id will be shared with the client once the content is ready.
3. After completing the process of generation, we will get the runtime unique key generated from the client and we will do the non-authentication request post action from the client with a unique key to the content for download and image rendering.
4. Once the content revival initiated with the server, we could not make use of this URL again to get the generated content once again from the server because the files will be deleted with the server after initiating the action.

You can find the following code reference for using the `[AllowAnonymous]` attribute and sample from this [link](#).

```
`csharp
[Authorize]
[Route("api/[controller]/[action]/{id?}")]
public class ReportApiController : ControllerBase, IReportController
{
    .....
    .....
    [ActionName("GetResource")]
    [AcceptVerbs("GET")]
    [AllowAnonymous]
    public object GetResource(ReportResource resource)
    {
        return ReportHelper.GetResource(resource, this, _cache);
    }
    [HttpPost]
    [AllowAnonymous]
    public object PostFormReportAction()
    {
        return ReportHelper.ProcessReport(null, this, this._cache);
    }
    .....
    .....
```



```
}
,
```

How to set the parameter in Web API Controller

Use the `OnReportLoaded` method to set parameter default value in the Web API controller.

```
`csharp
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    List<BoldReports.Web.ReportParameter> userParameters = new
    List<BoldReports.Web.ReportParameter>();
    userParameters.Add(new BoldReports.Web.ReportParameter()
    {
        Name = "SalesOrderNumber",
        Values = new List<string>() { "SO50756" }
    });
    reportOption.ReportModel.Parameters = userParameters;
}
,
```

You can find the following help documentation for how to set the parameter at client side in various platforms.

- [Angular](#)
- [React](#)
- [Java Script](#)
- [ASP.NET Core](#)
- [ASP.NET MVC](#)
- [ASP.NET Webforms](#)

How to manage reports within application using Bold Reports Report Designer

You can use **ExternalServer** to manage the reports with in application. Find the following code for accessing the existing reports, datasources and datasets from application **App_Data** folder.

```
`csharp
public override List<CatalogItem> GetItems(string folderName, ItemTypeEnum type)
{
    List<CatalogItem> _items = new List<CatalogItem>();
```

```
string targetFolder = HttpContext.Current.Server.MapPath("~/") + @"App_Data\ReportServer\";
if (type == ItemTypeEnum.Folder || type == ItemTypeEnum.Report)
{
    targetFolder = targetFolder + @"Report\";
    if (!string.IsNullOrEmpty(folderName) || folderName.Trim() == "/")
    {
        targetFolder = targetFolder + folderName;
    }
}
if (type == ItemTypeEnum.DataSet)
{
    foreach (var file in Directory.GetFiles(targetFolder + "DataSet"))
    {
        CatalogItem catalogItem = new CatalogItem();
        catalogItem.Name = Path.GetFileNameWithoutExtension(file);
        catalogItem.Type = ItemTypeEnum.DataSet;
        catalogItem.Id = Regex.Replace(catalogItem.Name, @"[^0-9a-zA-Z]+", "_");
        _items.Add(catalogItem);
    }
}
else if (type == ItemTypeEnum.DataSource)
{
    foreach (var file in Directory.GetFiles(targetFolder + "DataSource"))
    {
        CatalogItem catalogItem = new CatalogItem();
        catalogItem.Name = Path.GetFileNameWithoutExtension(file);
        catalogItem.Type = ItemTypeEnum.DataSource;
        catalogItem.Id = Regex.Replace(catalogItem.Name, @"[^0-9a-zA-Z]+", "_");
        _items.Add(catalogItem);
    }
}
else if (type == ItemTypeEnum.Folder)
{

```

```

foreach (var file in Directory.GetDirectories(targetFolder))
{
    CatalogItem catalogItem = new CatalogItem();
    catalogItem.Name = Path.GetFileNameWithoutExtension(file);
    catalogItem.Type = ItemTypeEnum.Folder;
    catalogItem.Id = Regex.Replace(catalogItem.Name, @"[^0-9a-zA-Z]+", "_");
    _items.Add(catalogItem);
}
}

else if (type == ItemTypeEnum.Report)
{
    foreach (var file in Directory.GetFiles(targetFolder, "*.rdl"))
    {
        CatalogItem catalogItem = new CatalogItem();
        catalogItem.Name = Path.GetFileNameWithoutExtension(file);
        catalogItem.Type = ItemTypeEnum.Report;
        catalogItem.Id = Regex.Replace(catalogItem.Name, @"[^0-9a-zA-Z]+", "_");
        _items.Add(catalogItem);
    }
}

return _items;
}
`

```

You can refer the below application for using the ExternalServer

[ExternalServer Sample](#)

How to manage the reports with database using Bold Reports Report Designer

You can use **ExternalServer** to manage the reports within SQL server database. Find the following code for accessing the existing reports, datasources and datasets from SQL server ExternalServer database.

```

`csharp
public override List<CatalogItem> GetItems(string folderName, ItemTypeEnum type)
{
    List<CatalogItem> _items = new List<CatalogItem>();

```

```
string targetFolder = HttpContext.Current.Server.MapPath("~/") + @"App_Data\ReportServer\";
if (type == ItemTypeEnum.Folder || type == ItemTypeEnum.Report)
{
    targetFolder = targetFolder + @"Report\";
    if (!string.IsNullOrEmpty(folderName) || folderName.Trim() == "/")
    {
        targetFolder = targetFolder + folderName;
    }
}
if (type == ItemTypeEnum.DataSet)
{
    foreach (var file in Directory.GetFiles(targetFolder + "DataSet"))
    {
        CatalogItem catalogItem = new CatalogItem();
        catalogItem.Name = Path.GetFileNameWithoutExtension(file);
        catalogItem.Type = ItemTypeEnum.DataSet;
        catalogItem.Id = Regex.Replace(catalogItem.Name, @"[^0-9a-zA-Z]+", "_");
        _items.Add(catalogItem);
    }
}
else if (type == ItemTypeEnum.DataSource)
{
    foreach (var file in Directory.GetFiles(targetFolder + "DataSource"))
    {
        CatalogItem catalogItem = new CatalogItem();
        catalogItem.Name = Path.GetFileNameWithoutExtension(file);
        catalogItem.Type = ItemTypeEnum.DataSource;
        catalogItem.Id = Regex.Replace(catalogItem.Name, @"[^0-9a-zA-Z]+", "_");
        _items.Add(catalogItem);
    }
}
else if (type == ItemTypeEnum.Folder)
{

```

```

foreach (var file in Directory.GetDirectories(targetFolder))
{
    CatalogItem catalogItem = new CatalogItem();
    catalogItem.Name = Path.GetFileNameWithoutExtension(file);
    catalogItem.Type = ItemTypeEnum.Folder;
    catalogItem.Id = Regex.Replace(catalogItem.Name, @"[^0-9a-zA-Z]+", "_");
    _items.Add(catalogItem);
}
}

else if (type == ItemTypeEnum.Report)
{
    foreach (var file in Directory.GetFiles(targetFolder, "*.rdl"))
    {
        CatalogItem catalogItem = new CatalogItem();
        catalogItem.Name = Path.GetFileNameWithoutExtension(file);
        catalogItem.Type = ItemTypeEnum.Report;
        catalogItem.Id = Regex.Replace(catalogItem.Name, @"[^0-9a-zA-Z]+", "_");
        _items.Add(catalogItem);
    }
}

return _items;
}

```

You can refer the below application for using the ExternalServer with SQL server

[ExternalServer Sample](#)

Before the sample running you should update the below attached SQL query in your SQL server.

[SQL Query](#)

How to resolve the Bold Reporting components undefined issue

There are three scenarios in which the Bold Report Viewer or Bold Report Designer undefined issue can occur. The details of the three scenarios and how to resolve them are provided as follows.

Script reference

If the Bold Report Viewer or the Bold Report Designer scripts are not properly referred, then this issue can occur. Hence, you need to ensure that the Bold Reporting components scripts are properly referred within your application, in order to resolve this issue.

How to resolve the Syncfusion components undefined issue when using the Bold Reporting components

Using Bold Reports along with EJ1 components

Using Bold Reports along with EJ1 components

Having the Reporting components in both EJ1 and Bold Reports products, if you are referring the common `ej.web.all.min.js` then you will face a conflict with Syncfusion Reporting components and Bold Reports, due to which this issue will occur. When this issue is occurring under this scenario, you can refer to [How to use the Bold Reports along with EJ1 controls](#) to resolve the issue.

Jquery script reference order

The Bold Reporting components will get registered after the `jquery.min.js` is referred. Hence this issue can occur if the `jquery.min.js` is not referred or referred after the Bold Reporting components script references. To resolve this issue, you need to ensure that the Bold Reporting components script references are referred only after the `jquery.min.js` reference.

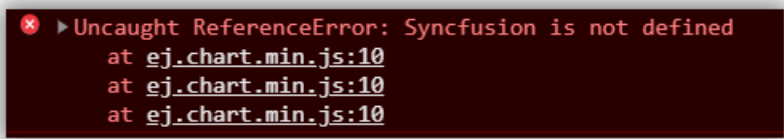
If you are using the multiple `jquery.min.js` references in your application, then you need to refer the Bold Reporting components script references at the last node of the references so that the instance of the Reporting components get properly registered.

How to resolve the Syncfusion components undefined issue when using the Bold Reporting components

There are two scenarios, in which the Syncfusion components undefined issue can occur. The details of the two scenarios and how to resolve them are provided as follows.

Script reference

If the `bold.reports.common.min.js` or the `bold.reports.widgets.min.js` scripts are not properly referred, then this issue can occur. Hence, you need to ensure that these scripts are properly referred within your application, in order to resolve this issue.



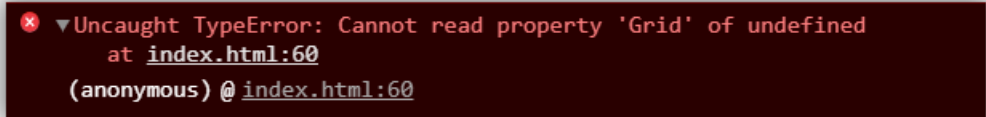
```
✖ ▶ Uncaught ReferenceError: Syncfusion is not defined
    at ej.chart.min.js:10
    at ej.chart.min.js:10
    at ej.chart.min.js:10
```

Using Bold Reports along with EJ2 components

The Bold Reports uses the Syncfusion EJ1 components for internal, which uses the `ej` as reference. Hence, while using the EJ2 components along with bold reports `ejs` should be used as the reference to initialize the EJ2 components.

For example, if you are using the EJ2 grid component along with Bold Reports, then you need to initialize the grid component using the following code sample.

```
`js
var grid = new ej.grid.Grid({dataSource: data});
`
```



```
✖ ▼ Uncaught TypeError: Cannot read property 'Grid' of undefined
    at index.html:60
    (anonymous) @ index.html:60
```

How to add a WebAPI Data Processing Extension for Report Designer **Using Bold Reports along with EJ2 components**

See also

- [How to resolve the Bold Reporting components undefined issue?](#)
- [How to use BoldReports ReportViewer with EJ2 controls?](#)
- [How to use BoldReports ReportViewer with EJ1 controls?](#)

How to add a WebAPI Data Processing Extension for Report Designer

This article explains the Data Processing Extensions providing the additional data source supports, which is not available in built-in Report Designer. You can find the following steps to add the WebAPI Data Processing Extension for Report Designer.

1. You need to refer the below attached **webapi.data.js** and **queryinputdialog.js** script files in your application.

[Extension scripts](#)

```
`html
<head>
<script src="../../Scripts/extension/webapi.data.js"></script>
<script src="../../Scripts/extension/queryinputdialog.js"></script>
</head>
`
```

2. Provide the extension details for ReportDesigner with **ReportDataExtensions** property as shown in following code example.

MVC

```
`html
<div style="width:100%; height:100%; position:absolute;">
@{Html.Bold().ReportDesigner("designer").ServiceUrl("/api/DesignerAPI"). ReportDataExtensions (ext
=> { ext.ClassName("WebAPIDataSource").Name("WebAPI". ImageClass ("e-reportdesigner-
datasource-webapi").DisplayName("WebAPI").Add(; }}). Render();}
</div>
@{Html.Bold().ScriptManager()}
`
```

3. Add the following codes in **script** tag to get a WebAPI dialog when using the WebAPI Extension.

```
`javascript
<script type="text/javascript">
var qryOptions = null;
```

```
var webApiQueryDialog = null;
if (ej.isNullOrUndefined(webApiQueryDialog)) {
webApiQueryDialog = new QueryInputDialog($('#designer'));
}
if (!ej.isNullOrUndefined(webApiQueryDialog)) {
qryOptions = {
toolbarRendering: $.proxy(webApiQueryDialog.renderToolBarItems, webApiQueryDialog),
datasetLoaded: $.proxy(webApiQueryDialog.enableButton, webApiQueryDialog),
dataModeChanged: $.proxy(webApiQueryDialog.enableButton, webApiQueryDialog)
};
}
function controllInitialized(args) {
debugger;
var designer = $('#designer').data('boldReportDesigner');
designer.setModel({queryDesignerOptions: qryOptions });
}
`
```

4. Attach the WebAPI Data Processing Extension in your application as shown in following help documentation.

[WebAPI Data Processing Extension](#)

You can download the following sample for Bold Report Designer with the WebAPI Data Processing Extension.

[MVC Report Designer with WebAPI Data Processing Extension](#)

How to resolve the image rendering and exporting issue with the ASP.NET MVC Authentication filter

You will get issue on rendering the image and exporting the report from report viewer and report designer in ASP.NET MVC application when Authentication filter has been used for your Web API. You have to ignore the Authentication validation for export and image request with condition of URL and form values.

Regarding security, you will not have any issues in the aspect of security by ignoring the authentication for this `GetResource` and `PostReportAction` requests.

These requests are used to retrieve the file format content from the server and used with our control based on the framework suggestion to have better experience in usability in downloads and avoid the delay of rendering images with reports.

These requests will be used at the time of exporting and image rendering only, this cannot be used once again by others. This approach is similar to the [Amazon Simple Storage Service \(Amazon S3\)](#) how they are providing access to share the private files,

You can get more details of the implementation approach from these steps,

1. Before initiating a non-authentication request, we will send the authenticate request to the server to generate the export and image content.
2. The authenticated request will generate the export with a unique server for the downloadable content and unique id will be shared with the client once the content is ready.
3. After completing the process of generation, we will get the runtime unique key generated from the client and we will do the non-authentication request post action from the client with a unique key to the content for download and image rendering.
4. Once the content revival initiated with the server, we could not make use of this URL again to get the generated content once again from the server because the files will be deleted with the server after initiating the action.

You can find the following code reference for ignoring the Authentication in the filter and the sample from this [link](#).

```
`csharp
if (context.Request.RequestUri.ToString().Contains("ReportApi/PostReportAction") &&
HttpContext.Current.Request.Form.Count > 0 &&
HttpContext.Current.Request.Form.GetValues("reportAction")[0] == "Export")
{
return;
}
else if (context.Request.RequestUri.ToString().Contains("ReportApi/GetResource"))
{
return;
}
`
```

How to change the data source dynamically

You have to use the `reportOption.ReportModel.DataSourceCredentials` available with the `OnInitReportOptions` method to dynamically change the data source in the web API controller. The following code sample shows how to change the connection string of the `<database>` data source in the report.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
```

```
DataSourceCredentials dataSourceCredentials = new DataSourceCredentials();

string connectionString = "Data Source = <instancename>; Initial Catalog = <database>; User ID =
'<username>'; Password = '<password>'";

//You have to provide the shared data source name used with the report or the data source name
available with the report.

dataSourceCredentials.Name = "<database>";

dataSourceCredentials.ConnectionString = connectionString;

reportOption.ReportModel.DataSourceCredentials = new List<DataSourceCredentials> {
dataSourceCredentials };

}
`
```

You can find the following help documentation for how to change data sources based on the application parameters in various platforms.

- [Angular](#)
- [React](#)
- [Java Script](#)
- [ASP.NET Core](#)
- [ASP.NET MVC](#)
- [ASP.NET Webforms](#)

How to change the data source dynamically using Report Serializer

ReportSerializer allows you to convert report to a C# object model. So, you can convert your report to Report Designer and modify the data source connection with data sources.

```
`csharp

public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    FileStream inputStream = new FileStream(@"C:\Users\Resources\sales-order-detail.rdl",
    FileMode.Open, FileAccess.Read);

    BoldReports.RDL.DOM.ReportSerializer serializer = new BoldReports.RDL.DOM.ReportSerializer();
    BoldReports.RDL.DOM.ReportDefinition reportDefinition = serializer.GetReportDefinition(inputStream);

    string connectionString = "Data Source =<instancename> ; Initial Catalog = <database>; User ID =
'<username>'; Password = '<password>'";

    foreach (var source in reportDefinition.DataSources)
    {
        source.ConnectionProperties.ConnectionString = connectionString;
    }

    MemoryStream reportStream = new MemoryStream();
```

```

serializer.SaveReportDefinition(reportStream , reportDefinition);
reportStream.Position = 0;
reportOption.ReportModel.Stream = reportStream;
}
`

```

How to connect the Report Server Report Service with application API

1. You have to follow the documentation for creating the [service](#) for your application,
2. You have to make use of the `ReportHelper.ReportServiceURL` and `ReportHelper.ServiceAuthorizationToken` to connect with the Report Server service from application service.

You have to use this download [link](#) for getting the sample service application for connecting the Report server with application API.

See also

- [How to change the report datasource dynamically based on customer id?](#)

How to customize the save and open button in the Bold Reports Report Designer

This articles explains to customize the save and open buttons in the Report Designer application. We can able to achieve this in two ways one is **Hiding the existing button and add the new buttons** another one is **Overriding the actions when clicking the inbuilt button**.

Hiding the existing button and add the new buttons

1. Hide the save and open button using the **toolbarSettings** event in our Report Designer.

ASP.NET Core

In ASP.NET Core Report Designer application, you can hide the save and open button using **ViewBag** option as shown in the following code example.

```

`html
<bold-report-designer id="reportdesigner1"
create="controlInitialized"
service-url=" ../Home"
report-data-extensions="@ViewBag.ReportDataExtensions"
toolbar-settings="@ViewBag.toolbarSettings"
report-opened="reportOpened"
ajax-before-load="ajaxBeforeLoad"

```

```
report-saved="reportSaved"
toolbar-click="toolbarClick"
report-modified="reportModified">
</bold-report-designer>
`
`csharp
public IActionResult Index()
{
    ViewBag.toolbarSettings = new BoldReports.Models.ReportDesigner.ToolbarSettings();
    ViewBag.toolbarSettings.Items = BoldReports.ReportDesignerEnums.ToolbarItems.All
    & ~BoldReports.ReportDesignerEnums.ToolbarItems.Save
    & ~BoldReports.ReportDesignerEnums.ToolbarItems.Open;
    return View();
}
```

2. Add a customized button and use your override codes to save and open the report to your custom location.

ASP.NET Core

```
`html
<button id="Open" type="button" value="Open">Open</button>
<button id="Save" type="button" value="Save" >Save</button>
<div style="height: 600px;width: 100%;">
<bold-report-designer id="reportdesigner1"
create="controllInitialized"
service-url=" ../Home"
report-data-extensions="@ViewBag.ReportDataExtensions"
toolbar-settings="@ViewBag.toolbarSettings"
report-opened="reportOpened"
ajax-before-load="ajaxBeforeLoad"
report-saved="reportSaved"
toolbar-click="toolbarClick"
report-modified="reportModified">
</bold-report-designer>
```

How to customize the save and open button in the Bold Reports Report Designer **Overriding** the actions when clicking the inbuilt button

```
</div>
<script>
$("#Open").click(function (e) {
var designer = $('#reportdesigner1').data('boldReportDesigner');
designer.openReport("/" + category + "/" + reportName);
});
$("#Save").click(function (e) {
var designer = $('#reportdesigner1').data('boldReportDesigner');
if (!designer.isNewServerReport()) {
// It is invokes the existing report save call.
designer.saveReport();
} else {
// It is invokes the SaveAs report call.
designer.saveReport("reportName");
}
});
`
```

Overriding the actions when clicking the inbuilt button

Override the Save and Open button using the **SaveReportClick** and **OpenReportClick** event as shown in the following code example.

ASP.NET Core

```
`html
<div style="height: 600px;width: 100%;">
<bold-report-designer id="reportdesigner1"
create="controllInitialized"
service-url="../Home"
report-data-extensions="@ViewBag.ReportDataExtensions"
save-report-click="saveMenuClick"
open-report-click="openMenuClick"
report-opened="reportOpened"
ajax-before-load="ajaxBeforeLoad"
report-saved="reportSaved"
toolbar-click="toolbarClick"
report-modified="reportModified">
```

How to customize the save and open button in the Bold Reports Report Designer **Overriding** the actions when clicking the inbuilt button

```
</bold-report-designer>
</div>
<script>
switch (args.select) {
case 'Device':
// It is invokes when opening the report from device.
this.browseFromClient();
args.cancel = true;
break;
case 'Server':
// It is invokes when opening the report from server.
this.browseReport(ej.ReportDesigner.BrowseType.Open);
args.cancel = true;
break;
}
function saveMenuClick(args) {
switch (args.select) {
case 'Save':
// It is invokes the existing report save call.
saveReport();
args.cancel = true;
break;
case 'SaveAsDisk':
// It is invokes the file download.
downloadReport();
args.cancel = true;
break;
case 'SaveAsServer':
// It is invokes the save to server call.
saveAsServer(catagory, reportName);
args.cancel = true;
break;
}
}
```

```
}  
,
```

How to pass JSON data in the Bold Reports Report Viewer and Report Writer

You have to Convert the objects from JSON data as shown in the following code example.

```
`csharp  
string json = System.IO.File.ReadAllText(@"<App Location>\App_Data\categories.json");  
Categories Categories = JsonConvert.DeserializeObject<Categories>(json);  
writer.DataSources.Clear();  
writer.DataSources.Add(new ReportDataSource( Name = "Categories", Value = Categories.ToList()));  
public class Categories : List<Category>  
{  
}  
public class Category  
{  
    public int CategoryID { get; set; }  
    public string Description { get; set; }  
    public string Name { get; set; }  
}  
,
```

You can refer the following documentation for passing the business object data for RDLC report in the Report Viewer and Report Writer.

[Report Viewer](#)

[JavaScript Report Viewer](#)

[ASP.NET WebForms Report Viewer](#)

[ASP.NET MVC Report Viewer](#)

[ASP.NET Core Report Viewer](#)

[Report Writer](#)

[ASP.NET Core Report Writer](#)

How to serialize the report using Report Serializer

ReportSerializer allows you to convert a report to a C# object model. Please follow these steps to serialize the report using Report Serializer.

```
`csharp
```

```

FileStream readStream = new FileStream(@"C:\Users\Resources\sales.rsd", FileMode.Open,
FileStream.Read);

BoldReports.RDL.DOM.ReportSerializer reportSerializer = new BoldReports.RDL.DOM.ReportSerializer();
var dataset = reportSerializer.GetSharedDataSet(readStream);
var reportDefintion = reportSerializer.GetReportDefinition(readStream);

```

How to customize parameter setting by parameter name

Use the `beforeParameterAdd` event for this customization. You can refer the following code sample for customizing the parameter setting by `StartDate` and `EndDate` parameter name.

```

`html
<div id="viewer"></div>

<script>
$("#viewer").boldReportViewer({
beforeParameterAdd: "onBeforeParameterAdd"
});
</script>
</html>
`

`html
<bold-report-viewer id="viewer" report-service-url="/api/ReportViewer" report-path="product-line-
sales.rdl" before-parameter-add="beforeParameterAdd"> </bold-report-viewer>

<script type="text/javascript">
function beforeParameterAdd(args) {
if (args.parameterModel.Name === "StartDate") {
args.parameterSettings.minDateTime = new Date("4/5/2003 5:00:00 AM");
args.parameterSettings.maxDateTime = new Date("4/15/2003 5:00:00 AM");
}
if (args.parameterModel.Name === "EndDate") {
args.parameterSettings.minDateTime = new Date("5/10/2003 5:00:00 AM");
args.parameterSettings.maxDateTime = new Date("5/20/2003 5:00:00 AM");
}
}
</script>
`

```


See also

- [How to add the Null parameter with DatePicker for DateTime parameter?](#)
- [How to group the values in parameter drop down?](#)
- [How to customize the boolean parameter UI with parameter pane?](#)

How to use the Report Viewer with camel case serializer settings application

If you are using the camel serializer for our application, then will have a problem in processing API result with Report Viewer. So, you have to use the default resolver for using the Report Viewer and Designer.

`csharp

```
config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new  
CamelCasePropertyNamesContractResolver();  
`
```

You can refer the following code snippet to resolve to set the default serializer for Report Viewer Web API with ASP.NET MVC.

`csharp

```
public class DefaultCaseControllerConfigAttribute : Attribute,  
System.Web.Http.Controllers.IControllerConfiguration  
{  
  
    public void Initialize(System.Web.Http.Controllers.HttpControllerSettings controllerSettings,  
        System.Web.Http.Controllers.HttpControllerDescriptor controllerDescriptor)  
    {  
  
        var formats =  
            controllerSettings.Formatters.OfType<System.Net.Http.Formatting.JsonMediaTypeFormatter>().ToList();  
  
        foreach (var format in formats)  
        {  
            controllerSettings.Formatters.Remove(format);  
        }  
  
        var formatter = new System.Net.Http.Formatting.JsonMediaTypeFormatter  
        {  
            SerializerSettings = { ContractResolver = new Newtonsoft.Json.Serialization.DefaultContractResolver() }  
        };  
  
        controllerSettings.Formatters.Add(formatter);  
    }  
}
```

[DefaultCaseControllerConfigAttribute]

```
public class RreportApiController : ApiController, IReportController
{
}
,
```

How to have the save alone with report designer

1. Hide the save button from the designer using the `toolbarSettings`.
2. Use the `toolbarRendering` and `toolbarClick` event to use our custom save button along with save option.

You can refer the following code snippet.

```
`html
<div id="container"></div>
<script>
$("#container").boldReportDesigner({
toolbarSettings: { items: ej.ReportDesigner.ToolbarItems.All & ~ej.ReportDesigner.ToolbarItems.Save },
toolbarClick: function(args) {
if ($ (args.target).hasClass('e-rptdesigner-toolbarcontainer')) {
var saveButton = ej.buildTag('li.e-rptdesigner-toolbarli e-designer-toolbar-align e-tooltxt', '', {}, {});
var savelcon = ej.buildTag('span.e-rptdesigner-toolbar-icon e-toolbarfonticonbasic e-rptdesigner-
toolbar-save e-li-item', '', {}, { title: 'Save' });
args.target.find('ul:first').append(saveButton.append(savelcon));
}
},
toolbarRendering: function(args) {
if (args.click === 'Save') {
var designer = $('#designer').data('boldReportDesigner');
args.cancel = true;
designer.saveReport();
}
}
});
</script>
,
```

How to show alert for users to save the changes before closing the designer

Use `hasReportChanges` to know the unsaved changes from the designer and `window beforeunload` event needs to be used for showing the alert along with unsaved changes in validation. You can find the following code sample.

```
`html
<script type="text/javascript">
var isFormSubmit = true;
$(document).ready(function () {
$(document.body).bind('submit', $.proxy(winformSubmit, this));
$(window).bind('beforeunload', $.proxy(beforeWindowUnload, this));
});
function winformSubmit(args) {
isFormSubmit = false;
}
function beforeWindowUnload(args) {
if (isFormSubmit) {
var designer = $('#designer').data('boldReportDesigner');
if (designer.hasReportChanges()) {
return 'Changes you made may not be saved';
}
}
isFormSubmit = true;
}
</script>
`
```

How to add the custom http headers for the Bold Reports Report Viewer and Report Designer

You can add the custom http headers from client side to server side using the **ajaxBeforeLoad** event in our Report Viewer and Report Designer components. You can find the following help documentation for adding the custom http headers for the Bold Reports Report Viewer and Report Designer.

[Custom http headers for JavaScript Report Viewer](#)

[Custom http headers for ASP.NET Web Forms Report Viewer](#)

[Custom http headers for ASP.NET MVC Report Viewer](#)

[Custom http headers for ASP.NET Core Report Viewer](#)

[Custom http headers for Angular Report Viewer](#)

[Custom http headers for React Report Viewer](#)

[Custom http headers for JavaScript Report Designer](#)

[Custom http headers for ASP.NET Web Forms Report Designer](#)

[Custom http headers for ASP.NET MVC Report Designer](#)

[Custom http headers for ASP.NET Core Report Designer](#)

[Custom http headers for Angular Report Designer](#)

[Custom http headers for React Report Designer](#)

How to generate a PDF from the report in console app

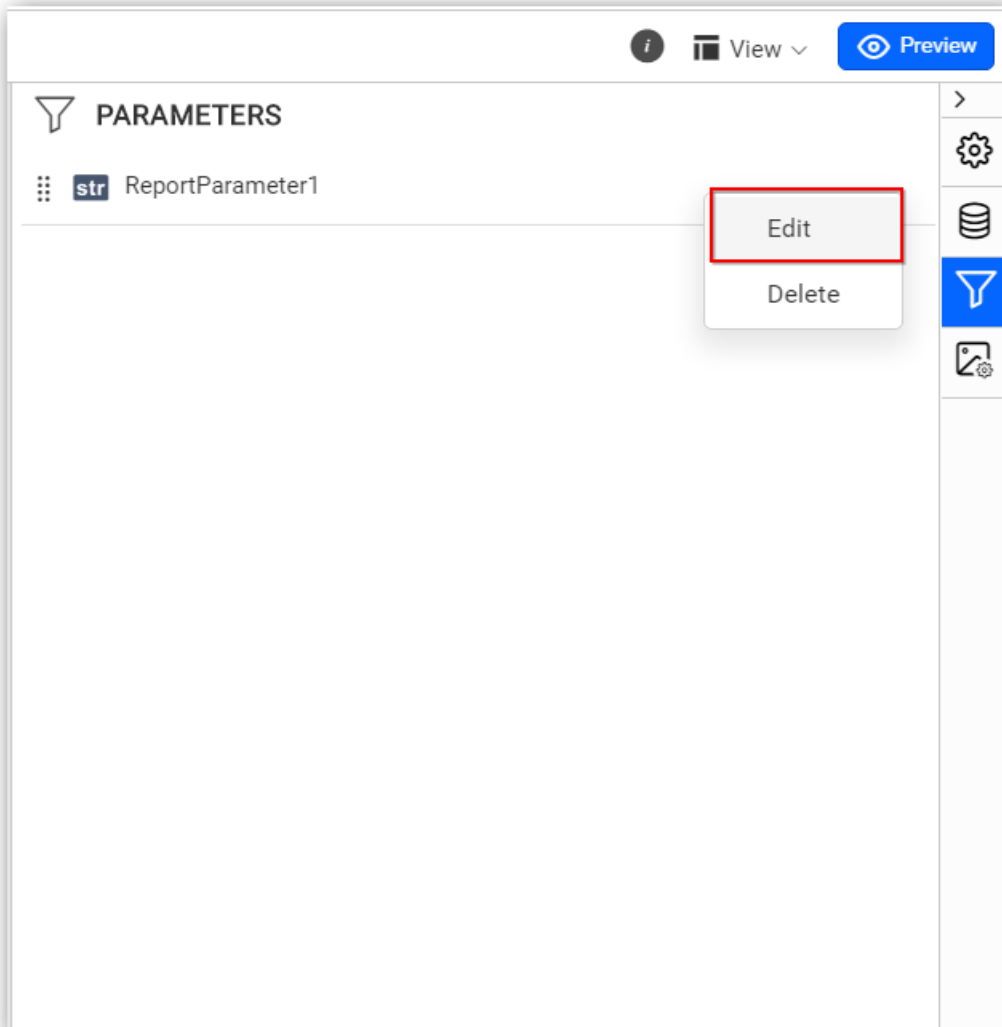
Use **Report Writer** component to generate a PDF documentation from the report in console application. You can find the following documentation for your reference.

[Report Writer](#)

How to change the parameter data type

Use the parameter edit option in Bold Report Designer to change the Report parameter data type existing one to required one. Follow these steps, to change parameter data type. In Bold Reports, supported data types are Boolean, DateTime, Integer, Float, and String.

1. Open the report in our Bold Report Designer.
2. Open the Parameter pane and click the Parameter edit option.



3. Change the Report parameter data type and save the parameter.

EDIT PARAMETER

Name
ReportParameter1

Prompt
ReportParameter1

Data Type
DateTime
String
Boolean
DateTime
Integer
Float
Visible

[Assign Values >>](#)

How to configure PostgreSQL Data Processing Extension for Report Viewer

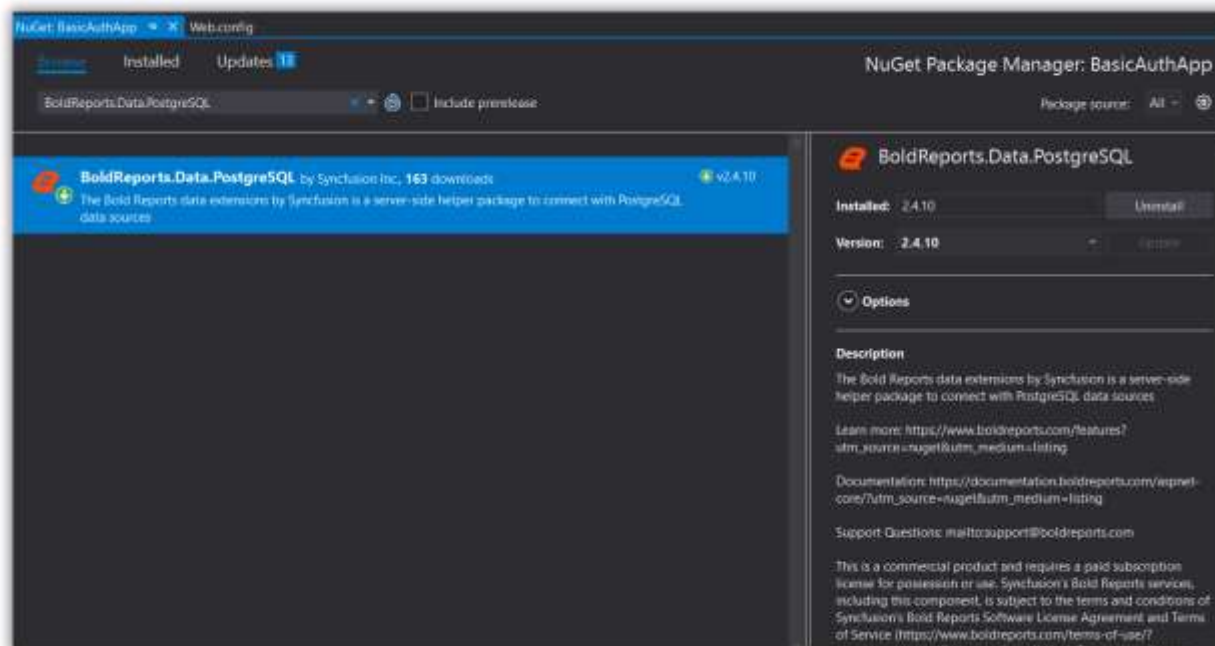
This section explains the steps required to register and load PostgreSQL data extensions in Web Report Viewer application.

Install PostgreSQL data source extension NuGet

To register and load PostgreSQL data sources in the application install **BoldReports.Data.PostgreSQL** package in the application.

Right-click the project or solution in the *Solution Explorer* tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Search for **BoldReports.Data.PostgreSQL** NuGet package, and install it in your application.



Register PostgreSQL data source extension in application startup

ASP.NET

To register the extension in the ASP.NET Web application, follow these steps.

1. Open the code-behind file `Global.asax.cs` and add the following using statement.

```
`csharp
using BoldReports.Web;
`
```

2. Then register the assembly name `BoldReports.Data.PostgreSQL` in `Application_Start` using `ReportConfig.DefaultSettings` as follows to use the PostgreSQL data extension.

```
`csharp
protected void Application_Start(object sender, EventArgs e)
{
    System.Web.Http.GlobalConfiguration.Configuration.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{action}/{id}",
        defaults: new { id = RouteParameter.Optional });
    AppDomain.CurrentDomain.SetData("SQLServerCompactEditionUnderWebHosting", true);
    //Use the below code to register extensions assembly into report designer
}
```

```
ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {  
"BoldReports.Data.PostgreSQL" });  
}  
`
```

ASP.NET Core

To register the extension in the ASP.NET Core application, follow these steps.

1. Open the file `Startup.cs` and add the following using statement.

```
`csharp  
using BoldReports.Web;  
`
```

2. Then register the package name `BoldReports.Data.PostgreSQL` in `Startup` as follows using `ReportConfig.DefaultSettings` as follows to use the PostgreSQL data extension.

```
`csharp  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, ILoggerFactory loggerFactory)  
{  
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));  
    loggerFactory.AddDebug();  
    AppDomain.CurrentDomain.SetData("SQLServerCompactEditionUnderWebHosting", true);  
    //Use the below code to register extensions assembly into report designer  
    ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {  
        "BoldReports.Data.PostgreSQL" });  
}  
`
```

To register multiple data extensions in the application, provide the assembly names as a list of strings.

How to configure WebAPI Data Processing Extension for Report Viewer

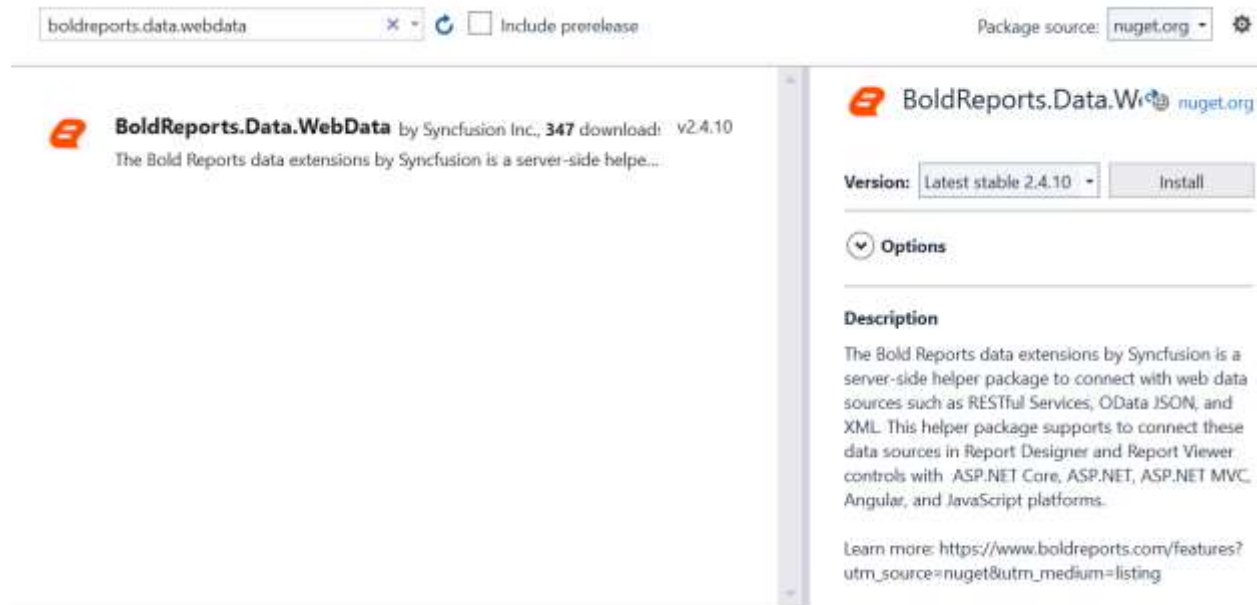
This section explains the steps required to register and load WebAPI data extensions in Web Report Viewer application.

Install WebAPI data source extension NuGet

To register and load WebAPI data sources in the application install **BoldReports.Data.WebData** package in the application.

Right-click the project or solution in the *Solution Explorer* tab, and choose **Manage NuGet Packages**. Alternatively, select the **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** menu command.

Search for **BoldReports.Data.WebData** NuGet package, and install it in your application.



Register WebAPI data source extension in application startup

ASP.NET

To register the extension in the ASP.NET Web application, follow these steps.

1. Open the code-behind file `Global.asax.cs` and add the following using statement.

```
`csharp
using BoldReports.Web;
`
```

2. Then register the assembly name `BoldReports.Data.WebData` in **Application_Start** using `ReportConfig.DefaultSettings` as follows to use the WebAPI data extension.

```
`csharp
protected void Application_Start(object sender, EventArgs e)
{
    System.Web.Http.GlobalConfiguration.Configuration.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{action}/{id}",
        defaults: new { id = RouteParameter.Optional });
    AppDomain.CurrentDomain.SetData("SQLServerCompactEditionUnderWebHosting", true);
    //Use the below code to register extensions assembly into report designer
    ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {
        "BoldReports.Data.WebData" });
}
```

How to resolve the Multiple actions were found that matches the request issue when using ASP.NET MVC application

Register WebAPI data source extension in application startup

```
}  
,
```

ASP.NET Core

To register the extension in the ASP.NET Core application, follow these steps.

1. Open the file `Startup.cs` and add the following using statement.

```
`csharp  
using BoldReports.Web;  
,
```

2. Then register the package name `BoldReports.Data.WebData` in **Startup** as follows using `ReportConfig.DefaultSettings` as follows to use the WebAPI data extension.

```
`csharp  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, ILoggerFactory loggerFactory)  
{  
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));  
    loggerFactory.AddDebug();  
    AppDomain.CurrentDomain.SetData("SQLServerCompactEditionUnderWebHosting", true);  
    //Use the below code to register extensions assembly into report designer  
    ReportConfig.DefaultSettings = new ReportSettings().RegisterExtensions(new List<string> {  
        "BoldReports.Data.WebData" });  
}  
,
```

To register multiple data extensions in the application, provide the assembly name's as list of strings.

How to resolve the Multiple actions were found that matches the request issue when using ASP.NET MVC application

You will get the issue `Multiple actions were found that matches the request` for report viewer and report designer Web API in the ASP.NET MVC application controller, when Web API is not routed properly with action. You have to create all the Web API methods with `ActionName` and `NonAction` attributes as like below,

```
`csharp  
public class ReportApiController : ApiController, IReportController  
{  
    [System.Web.Http.ActionName("PostReportAction")]  
    // Post action for processing the RDL/RDLC report
```

```

public object PostReportAction(Dictionary<string, object> jsonResult)
{
    return ReportHelper.ProcessReport(jsonResult, this);
}

// Get action for getting resources from the report
[System.Web.Http.ActionName("GetResource")]
[AcceptVerbs("GET")]
public object GetResource(string key, string resourcetype, bool isPrint)
{
    return ReportHelper.GetResource(key, resourcetype, isPrint);
}

[NonAction]

// Method that will be called when initialize the report options before start processing the report
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    // You can update report options here
}

[NonAction]

// Method that will be called when reported is loaded
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    // You can update report options here
}
}
,

```

How to resolve the image not rendered issue in .NET 6.0 on Linux Environment

Image not rendered issue occurred in .Net 6.0 and above versions on Linux environment due to lack of [System.Drawing.Common](#) package support in these versions of .NET 6.0 and above on Linux.

To resolve this issue, you can follow the steps outlined in the provided instructions:

Step 1 : Open the [Application Name].runtimeconfig.json file in a text editor.

Step 2 : In the runtimeconfig.json file, set the System.Drawing.EnableUnixSupport runtime configuration switch to true by adding it to the configProperties section, as shown in the following code sample.

```

`json
{
  "runtimeOptions": {
    "tfm": "net6.0",
    "frameworks": [
    ],
    "configProperties": {
      .....
      .....
      "System.Drawing.EnableUnixSupport": true
    }
  }
}
`

```

```

{
  "runtimeOptions": {
    "tfm": "net6.0",
    "frameworks": [
      {
        "name": "Microsoft.NETCore.App",
        "version": "6.0.0"
      },
      {
        "name": "Microsoft.AspNetCore.App",
        "version": "6.0.0"
      }
    ],
    "configProperties": {
      "System.GC.Server": true,
      "System.Reflection.Metadata.MetadataUpdater.IsSupported": false,
      "System.Runtime.Serialization.EnableUnsafeBinaryFormatterSerialization": false,
      "System.Drawing.EnableUnixSupport": true
    }
  }
}

```

Frequently asked questions

This section helps to get the answer for the frequently asked questions in **Embedded Reporting Tools**. We discussed the below topics in this section,

- [How to install Embedded Reporting Tools in silent mode?](#)
- [Is Bold Reports Embedded Reporting Tools or Viewer SDK requires additional licensing when running application with Azure App service?](#)
- [How to provide the permission for user to access the SSRS Report Server reports?](#)
- [How to use BoldReports ReportViewer with EJ2 controls?](#)

- [What are all the SSRS versions supported in Bold Reports?](#)
- [Is Bold Reports have a centralized report authoring system?](#)
- [How to get the user from database and pass the user as parameter for filtering the data from database?](#)
- [Is Bold Reports Embedded Reporting Tools supports with .NET Core and Docker on Linux?](#)
- [Licensing procedure for deployment](#)
- [Is it possible to create RDLC reports from business object datasource?](#)
- [Can we use the Report Designer component from Syncfusion Community license?](#)
- [How to add header authorization for report service?](#)
- [Can the Bold Reports be used with .NET Core Linux and macOS?](#)
- [Is it possible to use the .NET class objects with Report Viewer?](#)
- [Is theme studio available for Bold Reports to generate the custom theme?](#)
- [Is it possible to create WPF .Net Core application with Report Viewer?](#)
- [Why should you migrate to Bold Reports from Syncfusion Report platform](#)
- [Is it possible to load the million records report with Report Viewer and Report Writer?](#)
- [Can specify ReportServer URL in Web API controller?](#)
- [What is the difference between Report Service URL and Report Server URL?](#)
- [How to use BoldReports ReportViewer with EJ1 controls?](#)
- [Can Bold Reports be used with Java Spring Framework?](#)
- [What are all the supported HTML tags in Report Viewer?](#)
- [How to hide the license watermarks?](#)
- [How to use report viewer and designer with unobtrusive javascript enabled](#)
- [How to get the dataset name in Web API Service?](#)
- [where can i find the installation error and debug log files?](#)
- [Is it possible to show the selected all text instead of showing all values from a parameter?](#)
- [How to avoid the extra blank pages in print and print preview?](#)
- [How to use a frame border-image for PDF exported documentation?](#)
- [What are the differences in Embedded Reporting and Report Viewer SDK?](#)
- [Does Bold Report Viewer use SSRS Report processing?](#)
- [Why should not use produces attribute in web api controller?](#)
- [How to convert Crystal Reports to RDL for Bold Reports?](#)
- [How to use the Report Viewer and Designer with camel case serializer settings application?](#)
- [Is it possible to use a data table in Bold Reports?](#)
- [Why http authorization headers are not being set for all requests?](#)
- [CDN links for Localization and Culture](#)
- [How to resolve the Routing issue in ASP.NET MVC application?](#)
- [How to print the report using external button](#)
- [How to install PhantomJS manually?](#)

Troubleshooting Embedded Reporting Tools

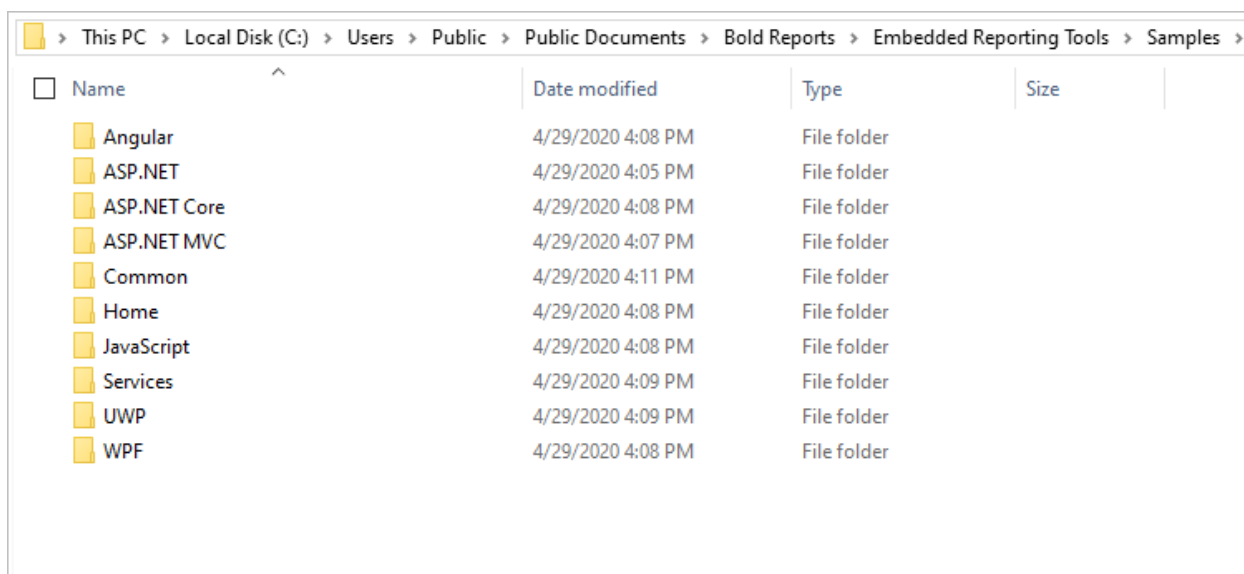
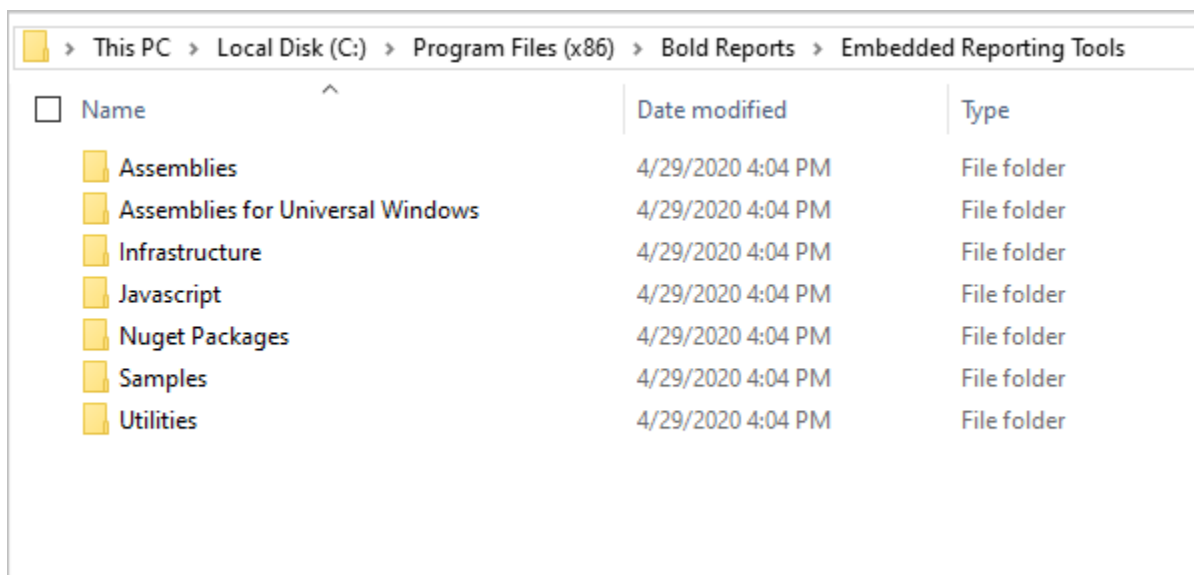
If you are facing issues with launching the **Bold Reports Embedded Reporting Tools** local samples follow the below troubleshooting mechanism.

- **If you face problem regarding controlled folder access(Ransomware protection) permission for Bold Reports Embedded Reporting Tools?**

We recommend you to refer how to enable permission to [allow-access-protected-folders](#).

- **Even though you have allowed permission for blocking apps, still not able to launch the Embedded Reporting Tools Sample Browser?**

We recommend you to ensure the below folders present in the installation path `C:\Program Files (x86)\Bold Reports\Embedded Reporting Tools` and `C:\Users\Public\Documents\Bold Reports\Embedded Reporting Tools\Samples`.

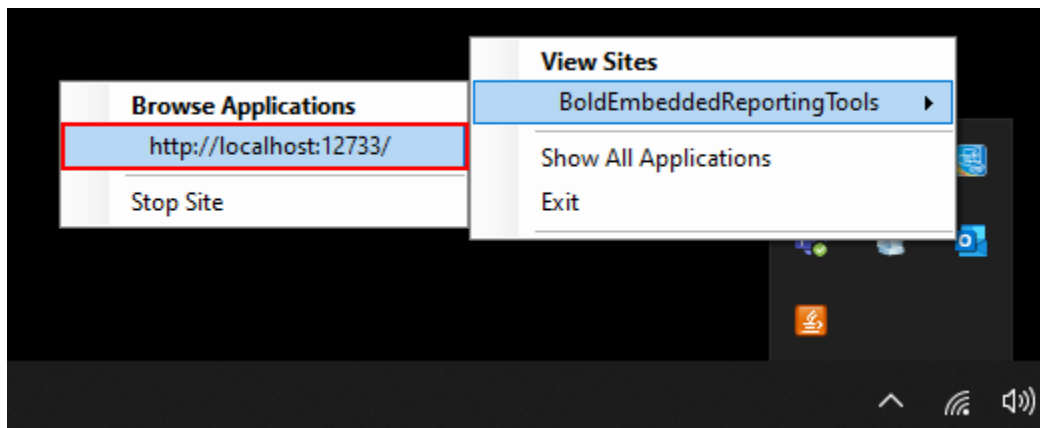
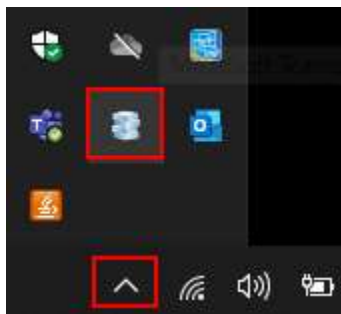


- **Even you have properly shipped folders, but still not able to launch Bold Reports Embedded Reporting Tools Sample Browser?**

Open command prompt in administrator mode and navigate to the path `C:\Program Files (x86)\Bold Reports\Embedded Reporting Tools\Utilities\StartSampleBrowserReportingTools` and run anyone of the below command through the `StartSampleBrowserReportingTools.exe`.

Platform	Arguments
JavaScript	StartSampleBrowserReportingTools.exe "JAVASCRIPT"
Angular	StartSampleBrowserReportingTools.exe "ANGULAR"
ASP.NET	StartSampleBrowserReportingTools.exe "ASPNET"
ASP.NET MVC	StartSampleBrowserReportingTools.exe "ASPNETMVC"
ASP.NET Core	StartSampleBrowserReportingTools.exe "ASPNETCORE"

After running the above command, ensure the System tray contains **IISEXPRESS.exe** application which hosts the samples.



- If none of the above steps doesn't help to launch the Samples?

You can able to find the error log file in the path **C:\Program Files (x86)\Bold Reports\Embedded Reporting Tools\Utilities\StartSampleBrowserReportingTools**, kindly [contact us](#) by creating a support ticket and share the generated log file with us, we will reach you soon.

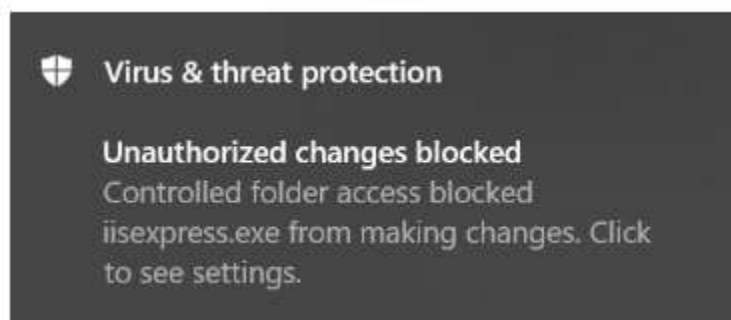
Controlled folder access and the protected file usage

[Controlled folder access](#) helps you protect valuable data from malicious apps and threats, such as [ransomware](#). Controlled folder access is included with Windows 10 and Windows Server 2019. When you enable the ransomware protection in your machine, the applications will be restricted to access the protected folders in your machine.

Controlled folder access and the protected file usage **Steps** to allow Bold Reports Embedded Reporting Tools to access protected folders

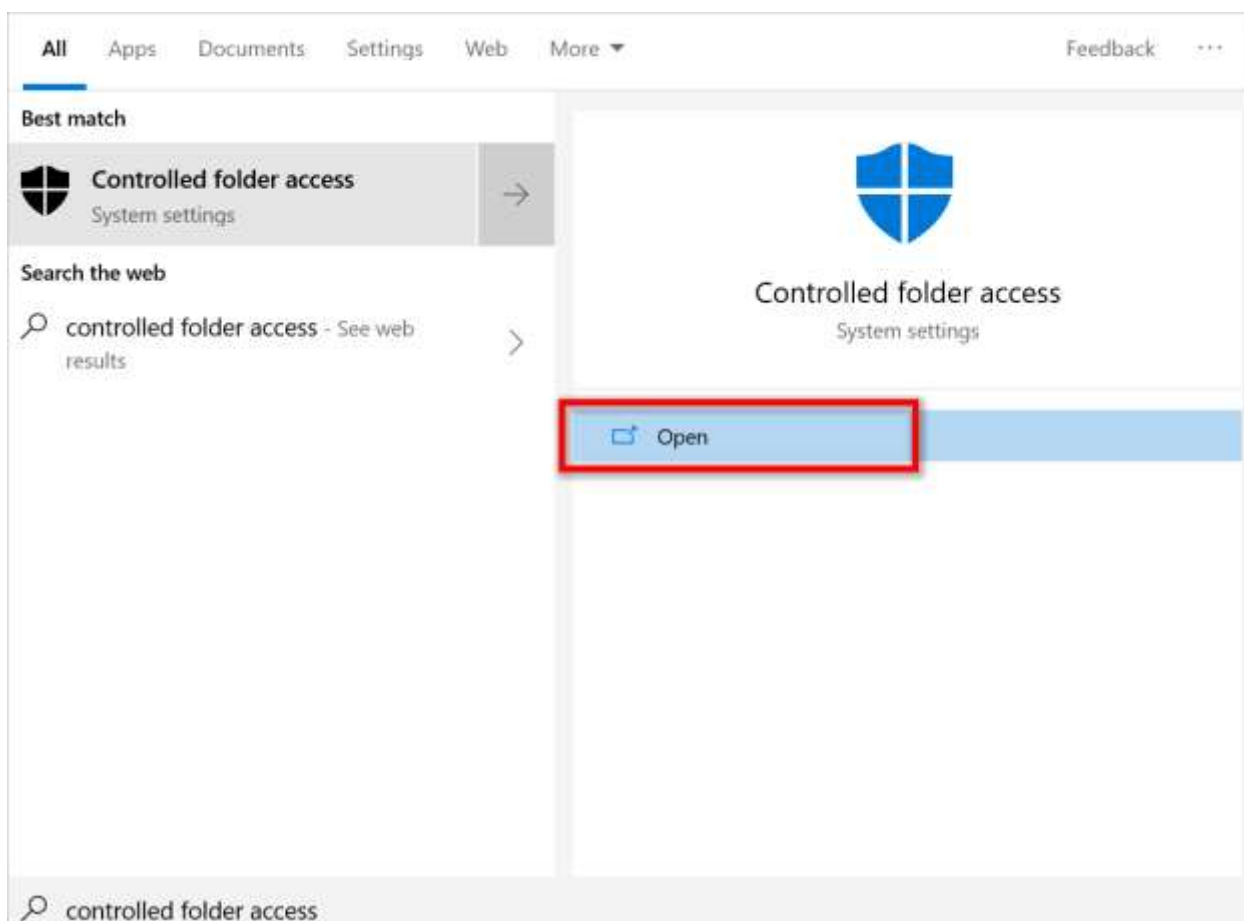
If the Bold Reports Embedded Reporting Tools or its related executable are not allowed to access the protected folders in your machine when the ransomware protection is enabled, follow the below-mentioned steps to provide access to Bold Reports application for accessing the protected folders.

You will receive notifications from Windows when an application is blocked from accessing the protected folders. For example, the following message will be displayed when you are trying to preview the dashboard with ransomware protection enabled.



Steps to allow Bold Reports Embedded Reporting Tools to access protected folders

- Open the Ransomware protection in Windows security using the start menu.



- Click the option **Allow an app through Controlled folder access**.

Ransomware protection

Protect your files against threats like ransomware, and see how to restore files in case of an attack.

Controlled folder access

Protect files, folders, and memory areas on your device from unauthorized changes by unfriendly applications.



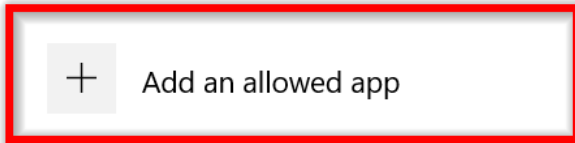
Protected folders

Allow an app through Controlled folder access

- The list of applications will be shown that can access the protected folders in your machine and click the **Add an allowed app** option.

Allow an app through Controlled folder access

If Controlled folder access has blocked an app you trust, you can add it as an allowed app. This allows the app to make changes to protected folders.





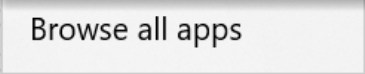
Most of your apps will be allowed by Controlled folder access without adding them here. Apps determined by Microsoft as friendly are always allowed.

- It will help you choose which application should be allowed to access the protected folder by showing recently blocked apps and all apps.

Allow an app through Controlled folder access

If Controlled folder access has blocked an app you trust, you can add it as an allowed app. This allows the app to make changes to protected folders.

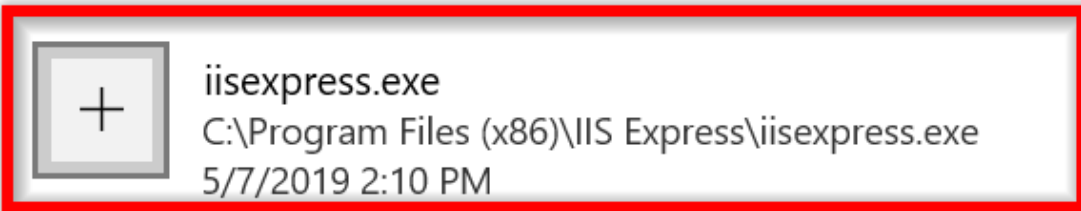
 Add an allowed app

Mod  lowed by Controlled folder access without
add  etermined by Microsoft as friendly are always
all

You can click any one of the options and choose the Bold Reports Embedded Reporting Tools related applications.

Recently blocked apps

The following apps were recently blocked by Controlled folder access. Make sure that you trust an app before you allow it.



- Allow the iisexpress and the Bold Reports Embedded Reporting Tools related apps to access your protected folders if the application is blocked in your machine.

Application	Path
-------------	------

iisexpress.exe	C:\Program Files (x86)\IIS Express
Bold Reports Embedded Reporting Tools Sample Browser Launcher	C:\Program Files (x86)\Bold Reports\Embedded Reporting Tools\Utilities\StartSampleBrowserReportingTools\StartSampleBrowserReportingTools.exe

- Now, you can continue to use the Bold Reports Embedded Reporting Tools without blocking any operations to your protected folders.

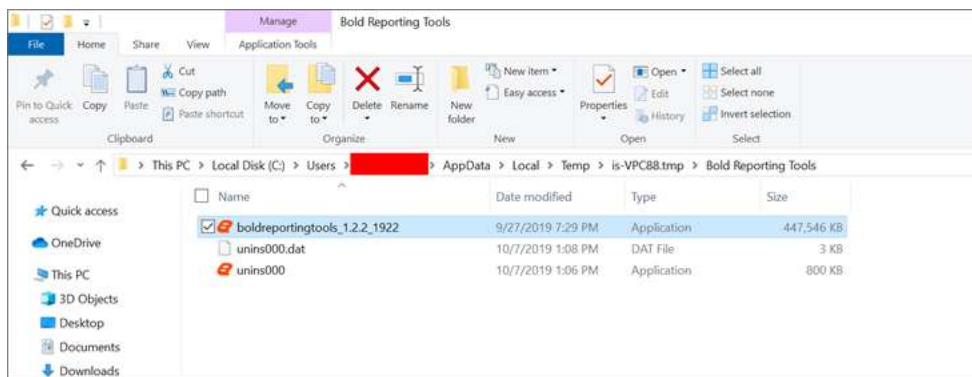
References

- <https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/enable-controlled-folders>
- <https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/customize-controlled-folders#allow-specific-apps-to-make-changes-to-controlled-folders>

Silent installation

The silent installation steps are applicable only for version 1.2.7 and below.

1. Double click the Embedded Reporting Tools setup.
2. Embedded Reporting Tools setup will be extracted in temp location (%temp%).



3. Copy that extracted Embedded Reporting Tools setup to some other location and cancel the installation.
4. Open the command prompt with administrative privileges and run the extracted Embedded Reporting Tools setup with the following arguments.

Arguments:

Is Bold Reports Embedded Reporting Tools or Viewer SDK requires additional licensing when running application with Azure App service **Site Setting of Report Server**

```
/Install silent /InstallPath:{InstallationPath} /pidkey:{unlock_key} /isdesktopdhortcut:{TRUE or FALSE}/Log "{LogFilepath\filename.log}"
```

Example:

```
/Install silent /InstallPath:C:\Program Files (x86)\New\Report /pidkey:@1243453sdffdfvv /isdesktopshortcut:TRUE /Log "C:\Program Files (x86)\New\Install.log"
```

```
E:\Bold Reports Tools>BoldReportingTools 1.2.2.1912.exe/Install silent /InstallPath:C:\Program Files (x86)\New\Report /pidkey:@1243453sdffdfvv /isdesktopshortcut:TRUE /Log "C:\Program Files (x86)\New\Install.log"
```

5. Now, Embedded Reporting Tools setup has been installed in silent mode.

Is Bold Reports Embedded Reporting Tools or Viewer SDK requires additional licensing when running application with Azure App service

No, if you are already having the Bold Reports for your team, then you do not have a need to buy additional license for using your application with Azure App service. As per our instruction, we should register the license token in startup of application as explained in the following documentation.

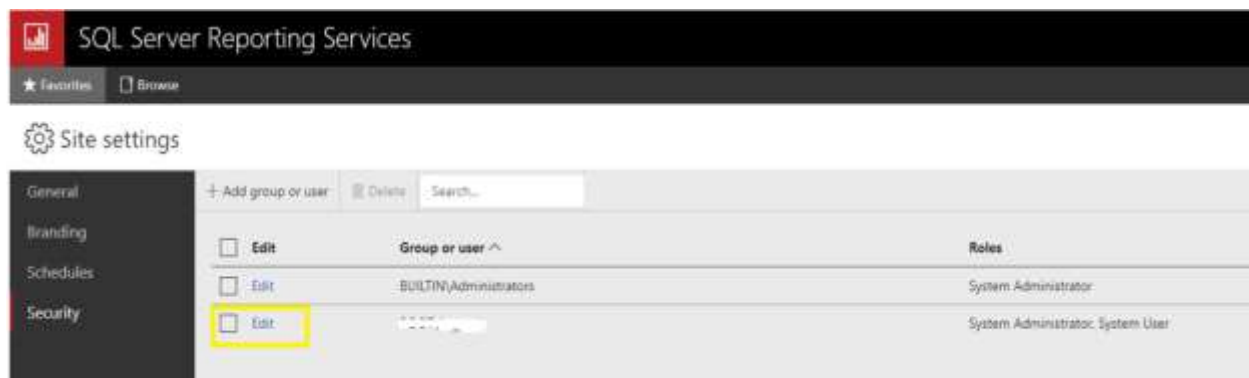
[License Token](#)

How to provide the permission for user to access the SSRS Report Server reports

The user what we are using for the **ReportServerCredential** that should have permission to get the report, datasource , resources from server. So, we make sure we are having the content manager permission available for the folder from where are going to get the reports and the user has the permission to access the Report Server.

Site Setting of Report Server

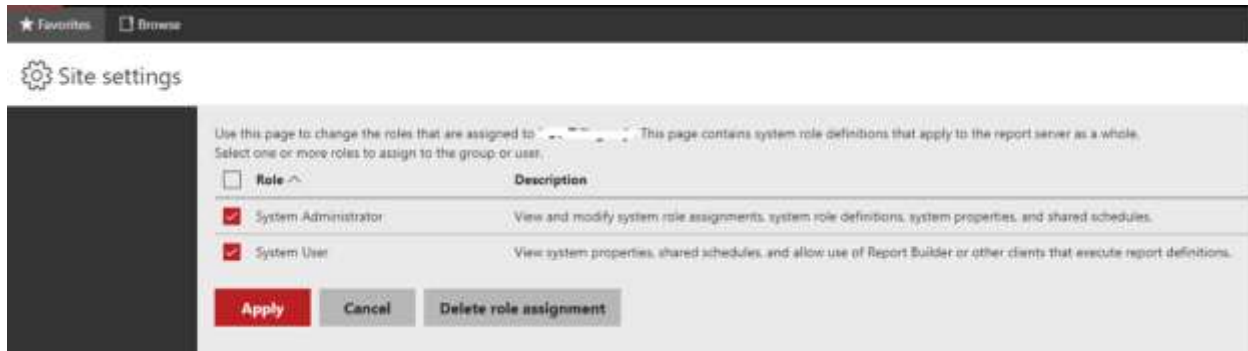
Within the SSRS website, the first item to setup is to create system level permissions; these permissions are assigned to the main administrators of SSRS and the "power" users who publish reports. Two main roles, System Administrator and System User are predefined. Assignment to these roles is made by clicking on Site Setting in the upper right corner of the report server site; next click on the Security link from the left menu.



Permission of user

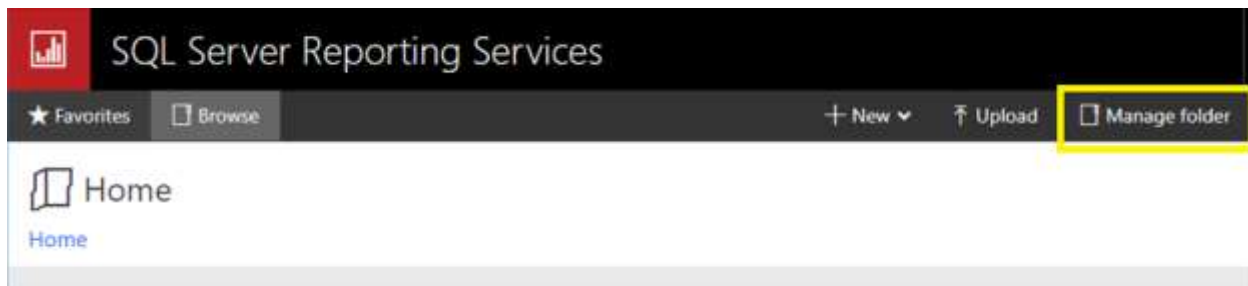
Clicking on the Edit option allows you to add, edit, or remove the roles assigned to the user or group as displayed in the below figure. The System Administrator role is reserved for those who need to have full

control over the Report Server whereas the System User role is applied to users / groups who are power users of the Report Server.



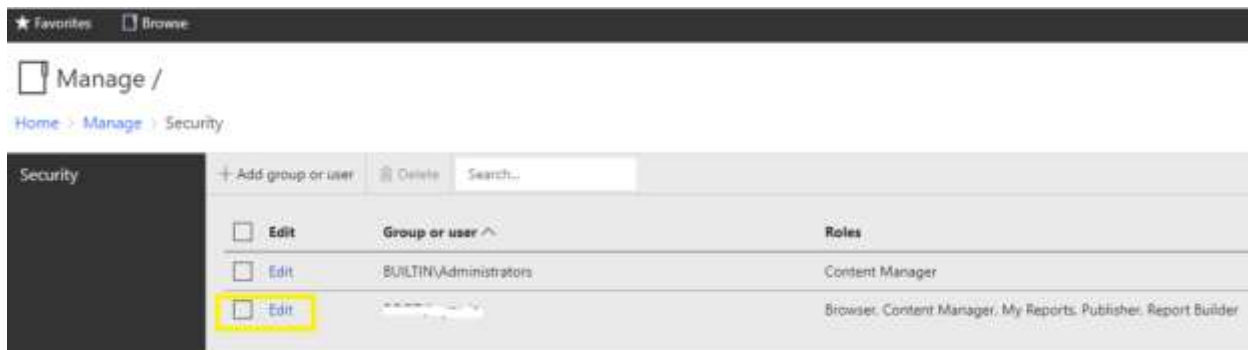
Folder

Click on the Manage Folder button upper right corner of the Report Server.



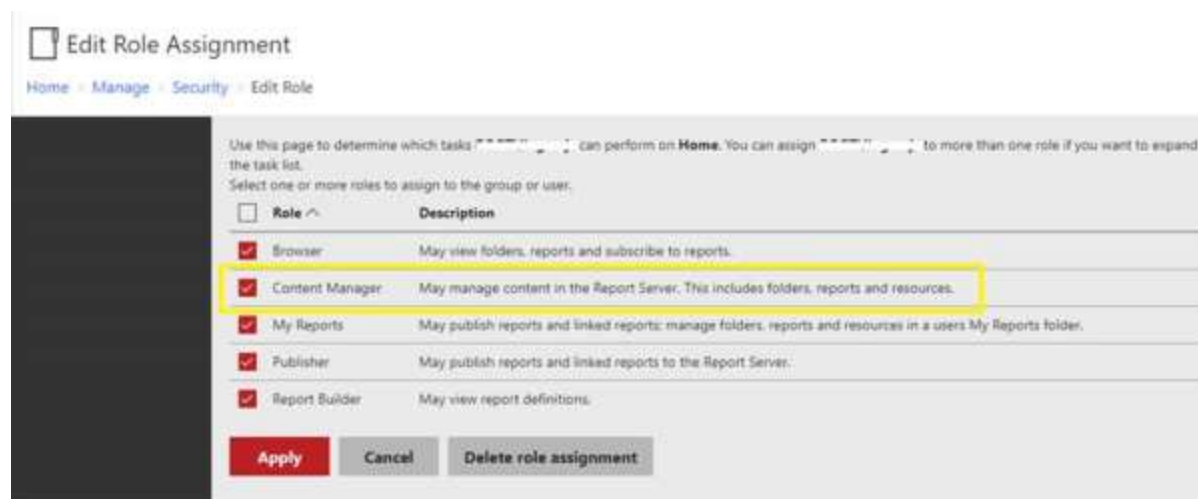
Permission for Folder

Provide the permission for the Group or users



Permission of user with Folder

Provide the Role for the user



How to use the ReportViewer along with EJ2 controls

For using the Bold Reports with EJ1, you need to ensure the compatibility styles, the script order is maintained with following suggestions:

Styles

For style, you need to refer the compatibility styles of EJ2 from Bold Reports and Bold Reports styles should be referred before EJ2 styles.

```
`html
```

```
<link href="https://cdn.syncfusion.com/ej2/styles/compatibility/material.css" rel="stylesheet" />
```

```
<link href="https://cdn.boldreports.com/5.4.20/content/bold.widgets.core.compatibility.min.css"
rel="stylesheet" />
```

```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.theme.compatibility.min.css"
rel="stylesheet" />
```

```
,
```

Scripts

You should refer EJ2 scripts before Bold Reports scripts as follows.

```
`html
```

```
<head>
```

```
<meta charset="utf-8" />
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
<title>BoldReports ReportViewer and EJ2 controls</title>
```

```
@ Syncfusion Essential JS 2 Styles @
```

```
<link rel="stylesheet" href="https://cdn.syncfusion.com/ej2/styles/compatibility/material.css" />
```

```
@ BoldReports Styles @
```

```
<link href="https://cdn.boldreports.com/5.4.20/content/bold.widgets.core.compatibility.min.css"
rel="stylesheet" />
```



```
<link href="https://cdn.boldreports.com/5.4.20/content/material/bold.theme.compatibility.min.css"
rel="stylesheet" />
```

```
@Default scripts@
```

```
<script src="http://code.jquery.com/jquery-1.10.2.min.js" type="text/javascript"></script>
```

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jquery-easing/1.3/jquery.easing.min.js"
type="text/javascript"></script>
```

```
<script src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
```

```
@ Syncfusion Essential JS 2 Scripts @
```

```
<script src="https://cdn.syncfusion.com/ej2/dist/ej2.min.js"></script>
```

```
@ BoldReports Scripts @
```

```
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.common.min.js"></script>
```

```
<script
src="https://cdn.boldreports.com/5.4.20/scripts/common/bold.reports.widgets.min.js"></script>
```

```
<!--Used to render the chart item. Add this script only if your report contains the chart report item.-->
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/data-visualization/ej.chart.min.js"></script>
```

```
<!-- Report Viewer component script-->
```

```
<script src="https://cdn.boldreports.com/5.4.20/scripts/bold.report-viewer.min.js"></script>
```

```
</head>
```

```
,
```

If your application is already using any of the essential JS2 without `ej2.min.js`, then refer the Bold Reports EJ2 dependent scripts from Syncfusion.

```
`html
```

```
<head>
```

```
@ Syncfusion Essential JS 2 Scripts @
```

```
<script src="https://cdn.syncfusion.com/ej2/ej2-buttons/dist/global/ej2-buttons.min.js"></script>
```

```
@ BoldReports Scripts @
```

```
<!--Used to render the gauge item. Add this script only if your report contains the gauge report item. -->
```

```
<script src="https://cdn.syncfusion.com/ej2/ej2-base/dist/global/ej2-base.min.js"></script>
```

```
<script src="https://cdn.syncfusion.com/ej2/ej2-data/dist/global/ej2-data.min.js"></script>
```

```
<script src="https://cdn.syncfusion.com/ej2/ej2-pdf-export/dist/global/ej2-pdf-export.min.js"></script>
```

```
<script src="https://cdn.syncfusion.com/ej2/ej2-svg-base/dist/global/ej2-svg-base.min.js"></script>
```

```
<script src="https://cdn.syncfusion.com/ej2/ej2-lineargauge/dist/global/ej2-
lineargauge.min.js"></script>
```

```
<script src="https://cdn.syncfusion.com/ej2/ej2-circulargauge/dist/global/ej2-circulargauge.min.js"></script>
</head>
,
```

Reference of EJ2 scripts from Bold Reports and Syncfusion components

```
`html
//Ej2
<script src="https://cdn.syncfusion.com/ej2/ej2-maps.min.js"></script>
//BoldReports
<script src="https://cdn.boldreports.com/ej2/ej2-maps.min.js"></script>
,
```

Script compatibility for ASP.NET Core

Add compatibility, use the following code in the **Layout.cshtml** page. Since BoldReports components and EJ2 controls have same library names to perform different actions, conflicts may occur when you refer these both controls in same application. To overcome this, extend these libraries in ej namespace in ASP.NET Core platform.

```
`html
<script>
var dataCopy = Object.assign({}, ej.data);
$.extend(ej, Syncfusion);
$.extend(ej.data, dataCopy);
</script>
,
```

Using EJ2 components with JavaScript

Hence, while using the EJ2 components along with Bold Reports, **ejs** should be used as the reference to initialize the EJ2 components.

For example, if you are using the EJ2 grid component along with Bold Reports, then you need to initialize the grid component using the following code sample.

```
`js
var grid = new ej.grids.Grid({dataSource: data});
,
```

See also

- [How to use Bold Reports with Syncfusion Blazor?](#)

How to use the Bold Reports with ASP.NET Core 3.x

This is applicable only for for the Bold Reports version before **2.3.27**. Since, the latest version is not required this setting.

We are using `Json.NET serializer` for Report Service which has been removed from ASP.NET Core 3.0 shared framework. So, you have to use `AddNewtonsoftJson()` with services to works with `Json.NET serializer` as per the migration information from .NET Core 3.0

[Migrate from ASP.NET Core 2.2 to 3.0](#)

- Add a package reference to `Microsoft.AspNetCore.Mvc.NewtonsoftJson`
- Update `Startup.ConfigureServices` to call `AddNewtonsoftJson`.

```
`csharp
public void ConfigureServices(IServiceCollection services)
{
    ...
    ...
    services.AddNewtonsoftJson();
}
```

What are all the SSRS versions are supported in Bold Reports

Find the list of SSRS server versions are supported in Bold Reports.

Platform	Supported SSRS version
ASP.NET Core	SQL Server Reporting Services 2008SQL Server Reporting Services 2008 R2SQL Server Reporting Services 2012SQL Server Reporting Services 2014SQL Server Reporting Services 2016SQL Server Reporting Services 2017SQL Server Reporting Services 2019
ASP.NET MVC	SQL Server Reporting Services 2008SQL Server Reporting Services 2008 R2SQL Server Reporting Services 2012SQL Server Reporting Services 2014SQL Server Reporting Services 2016SQL Server Reporting Services 2017SQL Server Reporting Services 2019
WPF	SQL Server Reporting Services 2008SQL Server Reporting Services 2008 R2SQL Server Reporting Services 2012SQL Server Reporting Services 2014SQL Server Reporting Services 2016SQL Server Reporting Services 2017SQL Server Reporting Services 2019

You will use the ASP.NET Core or ASP.NET MVC services for other platforms such as JavaScript, Angular, React, and UWP platforms. So, do not have separate support list for the other platforms.

Centralized report authoring

Centralized authoring explains how to use the report with authors and multiple developers with several access.

Is Bold Reports have a centralized report authoring system

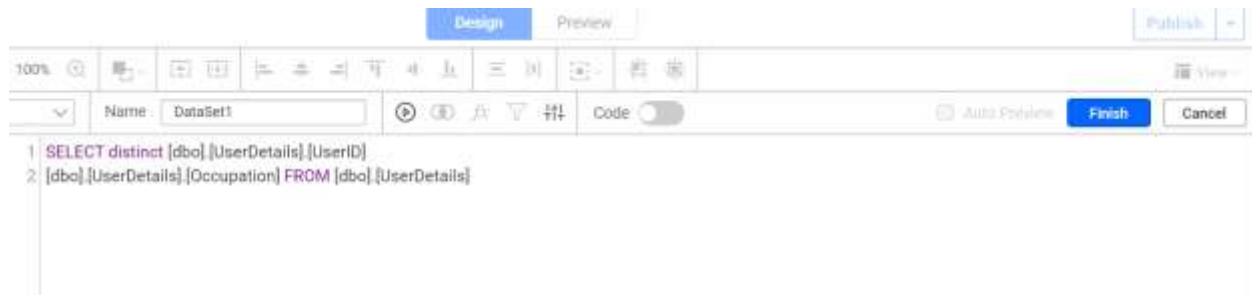
Yes, Bold Reports having On-Premise and Cloud version Report Server to have the reports in centralized report authoring.

You can refer more details of this from [Bold Report Cloud](#) and [Bold Reports On-Premise](#).

Get the user from database and pass the user as parameter for filtering the data from database

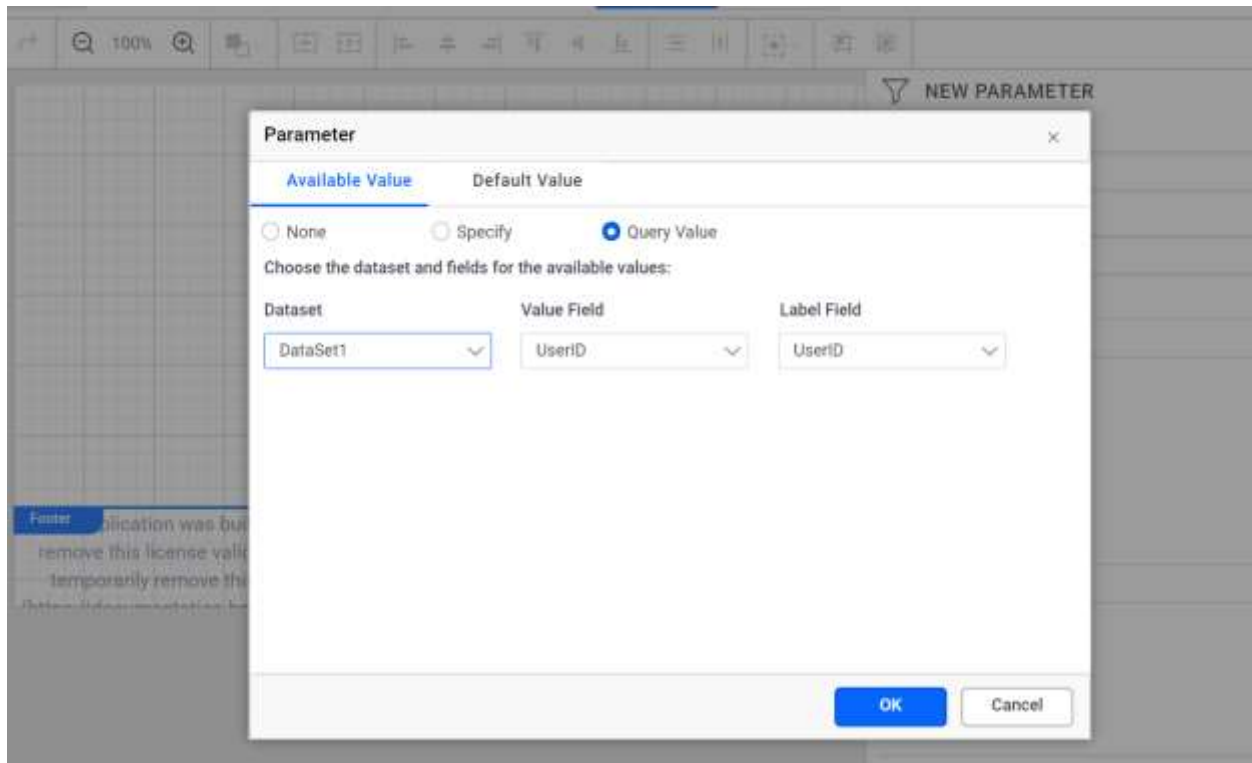
Find the following steps to get the user from database and pass the user as parameter for filtering the data from database.

1. Create a dataset with getting the user id from database as shown in the following image.



2. Pass the user id field value from dataset to parameter available value as shown in the following image.

Does Bold Reports Embedded Reporting Tools support .NET Core and Docker on Linux Is Bold Reports have a centralized report authoring system



3. Create a new dataset with filtering the data based on the parameter as shown in the following image.



Does Bold Reports Embedded Reporting Tools support .NET Core and Docker on Linux

Yes, Bold Reports Embedded Reporting Tools support .NET Core on Linux Docker. You have to ensure the following details while using our component with Docker environment.

We are using the `System.Drawing` to measure text size for `CanGrow` feature supports `TextBox` `ReportItem` and it requires native `libgdiplus` library, which does not contain in default `microsoft/dotnet` image. So, you must add native `libgdiplus` library with install command as follows in `DockerFile`.

`command

install System.Drawing native dependencies

```
RUN apt-get update \
&& apt-get install -y --allow-unauthenticated \
libc6-dev \
libgdiplus \
libx11-dev \
&& rm -rf /var/lib/apt/lists/*
\
```

The sample docker file can be downloaded from [here](#).

For .NET 6.0 or above version on Linux

You need to set the `System.Drawing.EnableUnixSupport` as `true` in the `ConfigProperties` section of the `runtimeconfig.json` file, as shown in the following code sample.

```
`json
{
  "runtimeOptions": {
    "tfm": "net6.0",
    "frameworks": [
    ],
    "configProperties": {
      .....
      .....
      "System.Drawing.EnableUnixSupport": true
    }
  }
}
```

How to use the Bold Reports with ASP.NET Core 3.1

This is applicable only for the Bold Reports version before **2.3.27**. Since, the latest version is not required this setting.

We are using `Json.NET` serializer for Report Service which has been removed from ASP.NET Core 3.1 shared framework. So, you have to use `AddNewtonsoftJson()` with services to works with `Json.NET` serializer as per the migration information from .NET Core 3.1

[Migrate from ASP.NET Core 2.2 to 3.1](#)

- Add a package reference to `Microsoft.AspNetCore.Mvc.NewtonsoftJson`

Does Bold Reports have any licensing procedure for deployment **For** .NET 6.0 or above version on Linux

- Update `Startup.ConfigureServices` to call `AddNewtonsoftJson`.

```
`csharp
public void ConfigureServices(IServiceCollection services)
{
    ...
    ...
    services.AddNewtonsoftJson();
}
```

Does Bold Reports have any licensing procedure for deployment

Yes, you need licensing token to be registered for Bold Reports development environment. If you are using the Bold reports component in the application level, even if you use the dependencies of the installed build, the licensing will not be imported. The following licensing error will be displayed, if the license token is missing in your projects.

This application was built using a trial version of Bold Reports. Include a valid license to remove this license validation message permanently. You can also obtain a free 30 days evaluation license to temporarily remove this message during the evaluation periods.

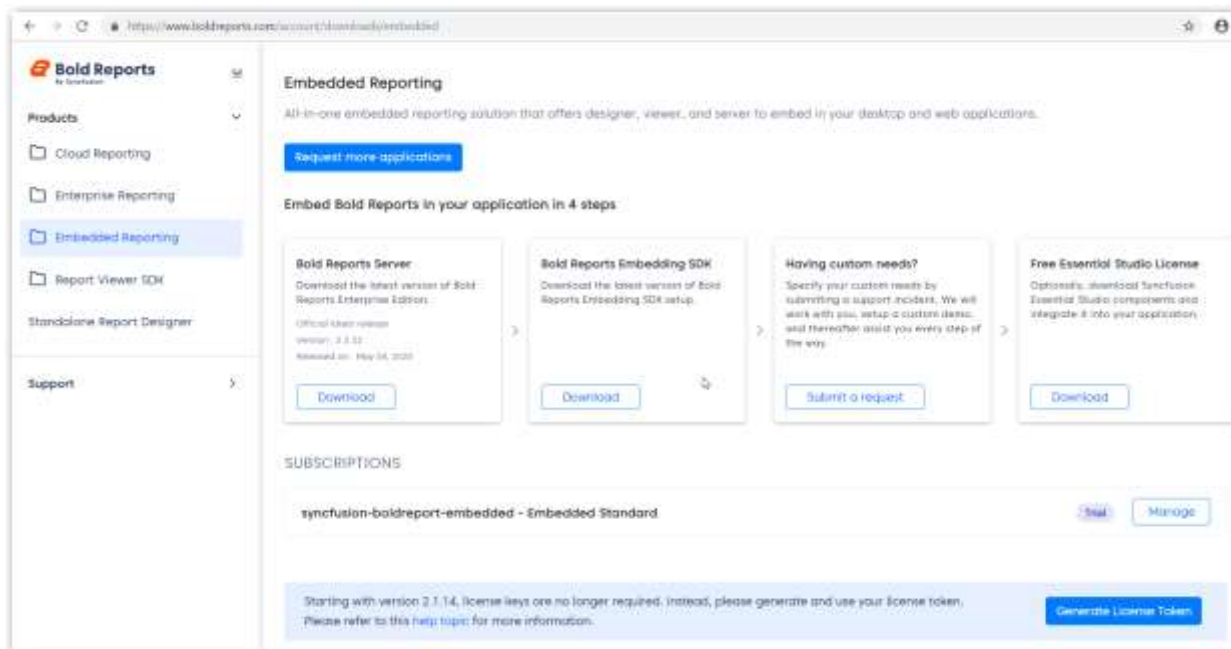
You need to register the license tokens in your projects. To learn about registering the license token in your projects, refer to this [how to register the Bold Reports license token](#) section.

Is it possible to create the RDLC reports using the business object data source in Report Designer

The Bold Report Designer does not have support to create the RDLC reports using the business object data source. In Bold Report Designer, RDLC reports can be created and the data can be assigned using the JSON array collection only. If you need to create RDLC report using the business object data source, then refer to this [RDLC report creation](#) section.

Can you use the Report Designer component from Syncfusion community license

No, Syncfusion community license cannot be used for Report Designer and you can use your community license for Report Viewer SDK access only. You should have Bold Reports Embedded plan to use Report Designer in application.



How to add a header authorization for report service

You can add the headers from client side to server side using the **ajaxBeforeLoad** event in our Report Viewer and Report Designer components. You can find the following help documentation for adding the authorization headers for the Bold Reports Report Viewer and Report Designer.

[Header authorization for JavaScript Report Viewer](#)

[Header authorization for ASP.NET Web Forms Report Viewer](#)

[Header authorization for ASP.NET MVC Report Viewer](#)

[Header authorization for ASP.NET Core Report Viewer](#)

[Header authorization for Angular Report Viewer](#)

[Header authorization for React Report Viewer](#)

Can the Bold Reports be used with ASP.NET Core on Linux and macOS

Yes, you can make use of our Bold Reports Embedded Reporting tools with Linux and macOS. You have to ensure the following while using our component.

Bold Reports uses the **System.Drawing** to measure text size for **CanGrow** feature supports available TextBox ReportItem and it requires native libgdiplus library. So, you should install the libgdiplus dependent library, which is not available with non-Windows operating systems.

Linux

Install the libgdiplus by executing the following commands at a terminal (command) prompt:

```
`bash
```

```
sudo apt install libc6-dev
```

```
sudo apt install libgdiplus
```


Is it possible to use the .NET class objects in ReportViewer

For .NET 6.0 or above version on Linux

`

For .NET 6.0 or above version on Linux

You need to set the `System.Drawing.EnableUnixSupport` as `true` in the `ConfigProperties` section of the `runtimeconfig.json` file, as shown in the following code sample.

```
`json
{
  "runtimeOptions": {
    "tfm": "net6.0",
    "frameworks": [
    ],
    "configProperties": {
      .....
      .....
      "System.Drawing.EnableUnixSupport": true
    }
  }
}
```

`

macOS

Use the Homebrew ("brew") package manager for installing libgdipplus. After installing brew, install the libgdipplus by executing the following commands at a terminal (command) prompt:

```
`bash
brew update
brew install mono-libgdipplus
`
```

See Also

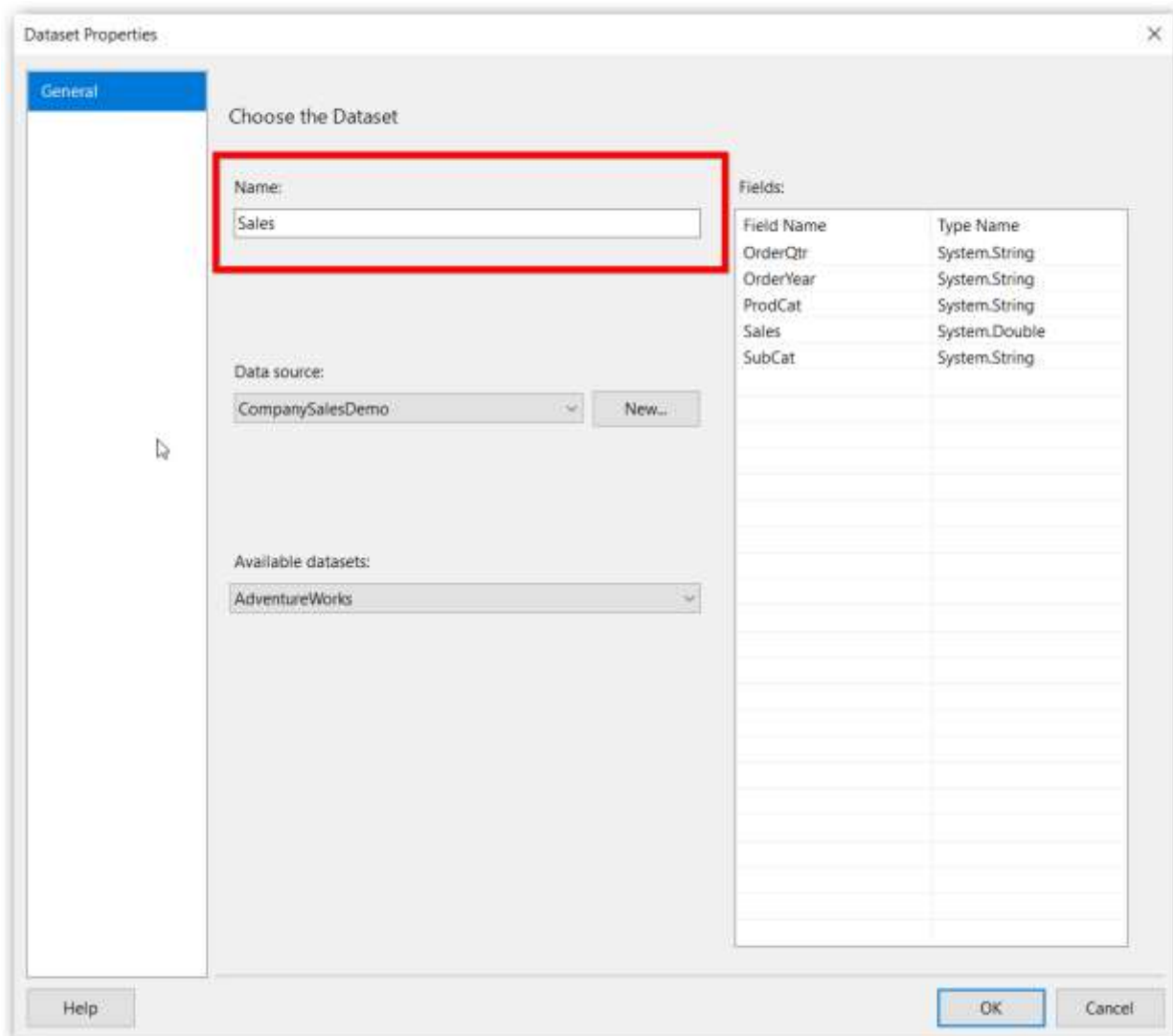
[.NET Core dependencies requirements on .NET ubuntu for System.Drawing.Common assembly](#)

[.NET Core dependencies requirements on macOS for System.Drawing.Common assembly](#)

[System.Drawing for .NET Core from NuGet packages](#)

Is it possible to use the .NET class objects in ReportViewer

Yes, it is possible to use the .Net class objects as the `DataSource` in BoldReports ReportViewer. For this, you have to use the `ProcessingMode` property with `ProcessingMode.Local` and your objects need to pass using the `DataSources` property along with the data set name used with RDLC or RDL report.



See Also

[.NET Class objects in Angular ReportViewer](#)

[.NET Class objects in JavaScript ReportViewer](#)

[.NET Class objects in React ReportViewer](#)

[.NET Class objects in ASP.NET Core ReportViewer](#)

[.NET Class objects in ASP.NET MVC ReportViewer](#)

[.NET Class objects in ASP.NET Web Forms ReportViewer](#)

[.NET Class objects in WPF ReportViewer](#)

[.NET Class objects in UWP ReportViewer](#)

Is theme studio available for Bold Reports to generate the custom theme

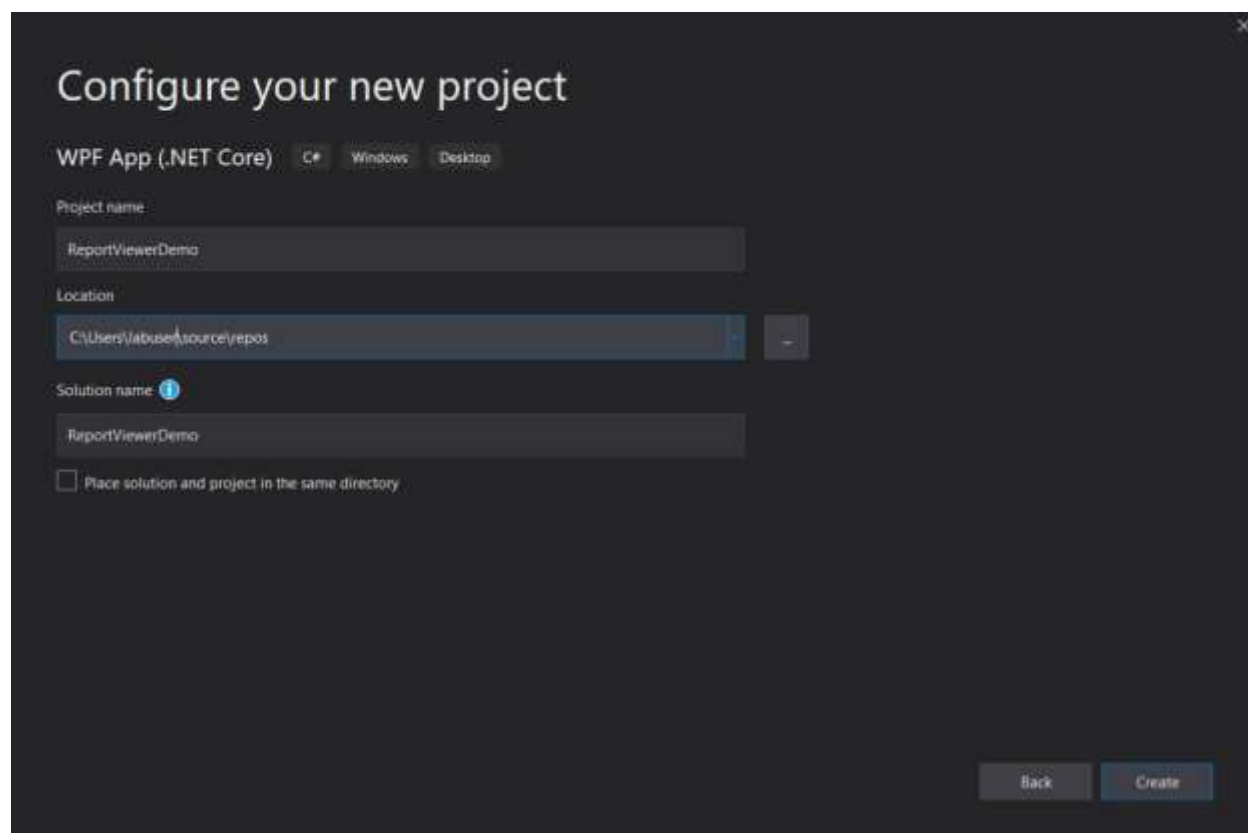
No, theme studio is not available for Bold Reports to generate the custom theme. As of now you have to contact our Support team with your requirement to get customized theme.

Is it possible to create a WPF .Net Core application with Bold Report Viewer

Yes, it is possible to create a WPF .Net Core application with Bold Report Viewer. For creating a WPF .Net Core application with Bold Report Viewer, you need to ensure that Visual Studio 2019 is installed in your machine.

Please follow these mentioned steps to create a WPF .Net Core application with Bold Report Viewer:

1. Open the Visual Studio 2019 and select **Create a new project**.
2. Go to **Installed > Visual C# > Windows Desktop**.
3. Select **WPF App (.NET Core)**, then click **NEXT**.
4. Change the application name, and then click **Create**.



5. Visual Studio creates the project and opens the designer for the default application window named as MainWindow.xaml.

You can follow the remaining steps from the [Getting started](#) section to complete the process of creating WPF .Net Core application with Bold Report Viewer.

Why should you migrate to Bold Reports from Syncfusion Report Platform

As per our plan to have a separate suite for the reporting components, we have introduced Bold Reports from Syncfusion. Due to the migration to Bold reports, there will be no further feature updates for Syncfusion Report Platform. So, you need to consider using our Bold Reports for your developments.

See also

- [What are the differences in Bold Reports licensing models](#)
- [Can Syncfusion licenses be used with Bold Reports?](#)
- [Can the Syncfusion community license be used with Bold Reports](#)
- [Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license](#)
- [Does Bold Reports Embedded Reporting Tools or Viewer SDK require additional licensing when deploying an application to Azure App service](#)
- [Migrating Reporting Application](#)

Is it possible to load the million records report with Report Viewer and Report Writer

Yes, you can load the million records reports with Bold Reports Report Viewer and Report Writer components.

You should use **EnableVirtualEvaluation** and **DisablePageSplitting** API with Report Viewer and Report Writer for handling million records. Find the following reference document.

[Handle larger amount of data with Report Viewer](#)

Can specify ReportServer URL in Web API controller

Yes, you can specify the [reportServerUrl](#) property in Web API controller as shown in the following code snippet.

```
`csharp
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
    reportOption.ReportModel.ReportServerUrl = "https://testing.SSRS.Server.com";
}
`
```

Difference between Report Service URL and Report Server URL

Report Service URL

Bold Reports Reporting components are built with the architecture of client-server technology. You should have server side Web API service with ASP.NET Core or ASP.NET MVC for using our components and need to assign created API URL with `reportServiceURL` API for server integration.

You can make use of the following references, to create or use existing Web API service for our Reporting components:

- [Bold Reports Report Server built-in service for Report Viewer](#)
- [Bold Reports Report Server built-in service for Report Designer](#)
- [Create ASP.NET Core Web API for Report Viewer](#)
- [Create ASP.NET MVC Web API for Report Viewer](#)

Report Server URL

Bold Reports Reporting components provides support to use the reports directly from SQL Server Reporting Services (SSRS) and Bold Reports Report Server. If you are going to use the feature of using the reports from SQL Server Reporting Services (SSRS) and Bold Reports Report Server, then you have to provide the server URL information to our Reporting components using the `reportServerURL` API for report processing.

You can refer to the following sections to make use of the report server URL, in order to use the reports from Report Server in our Reporting components:

- [Bold Reports Report Server Reports with Report Viewer](#)
- [SSRS Reports with Report Viewer](#)

How to use the Bold Reports along with EJ1 controls

We are having the Reports component in both EJ1 and Bold Reports products. So, if you referring the common **web.all.min.js** then Report Viewer conflict with EJ and Bold Reports product. So you can specify the component script instead of referring the **web.all.min.js** script in your application when using the EJ1 component along with the Bold Reports component.

For example, if you want to use the EJ1 Grid item along with BoldReports Report Viewer component then you can specify the **ej.grid.min.js**. You can find the following example for your reference.

```
`html
```

```
<script src="https://cdn.boldreports.com/external/jquery-1.10.2.min.js"
type="text/javascript"></script>
```

```
<script
src="https://cdn.boldreports.com/2.2.32/scripts/common/bold.reports.common.min.js"></script>
```

```
<script
src="https://cdn.boldreports.com/2.2.32/scripts/common/bold.reports.widgets.min.js"></script>
```

```
<!--EJ1 Grid component script reference.-->
```

```
<script src="https://cdn.syncfusion.com/18.1.0.42/js/web/ej.grid.min.js"></script>
```

```
<!--Bold Reports Report Viewer component script-->
```

```
<script src="https://cdn.boldreports.com/2.2.32/scripts/bold.report-viewer.min.js"></script>
```

Can use Java Spring Framework for Bold Reports

Yes, you can use Bold Reports with Java Spring Framework applications. Bold Reports do not support directly by using our component with Java Spring Web application, you have to use our JavaScript jQuery component with Java Spring Web application. You can refer the following article how can use the JQuery with Java Spring Web application.

[How to use AJAX and jQuery in Spring Web MVC \(.jsp\) Application](#)

[Report Viewer Getting Started with JavaScript](#)

What are all the supported HTML tags in Report Viewer

This section explaining the supported HTML tags and Limitations of CSS Attributes.

Supported HTML Tags

1. Hyperlinks:
2. Fonts:
3. Header, style and block elements: ` ,

‘ ,

’

’

4. , `

5. Text format: , , ,

Limitations of Cascading Style Sheet Attributes

The following is a list of attributes that are supported:

1. text-align, text-indent
2. font-family
3. font-size
4. Only valid RDL size values are supported in absolute CSS length units. Supported units are: in, cm, mm, pt, pc, px, ex, and em.
5. Relative CSS length units are ignored and are not supported. Unsupported units include percentage (%), and rem.
6. color
7. padding, padding-bottom, padding-top, padding-right, and padding-left
8. font-weight

How to hide the license watermarks

The license watermarks will be shown based on license validation in reporting components as per [licensing](#). Bold Reports licensing requires internet connection for the license validation. So, you should ensure the firewall is not blocking the site <http://websiteapi.boldbi.com/> from your network for license validation. If you don't have the network problem, then you should ensure the license token registered, as properly with your application as explained in below documentation.

[Register license token for an application](#)

Since, the process of our license validation will be at the startup of every application, you have to register the license in the startup of application. The internet license validation will be processed in the place of registering the license token only.

See Also

[How to upgrade from Trial version after purchasing a license](#)

[Where can I get a license token](#)

[Embedded reporting tools licensing version 1.2.x](#)

Unobtrusive

Many uncertainties and difficulties are involved in a usual JavaScript programming environment like - some of the browsers may ignore the javascript codes under the scripts section completely or partially due to its complexity and so on. To overcome all such inconveniences, [Unobtrusive JavaScript](#) support has been introduced in order to make it easier for the users to create all our Bold Reports components with basic level HTML tag-like structure.

One of the main goal of the unobtrusive support is to achieve the clear separation of both the HTML content and behavior, so as to enhance the page loading time and to make the code updating easier. Bold Reports JavaScript have separate integration library to achieve the Unobtrusive JS support. To make use of Unobtrusive support with our Bold Reports JavaScript components, it is necessary to refer the `ej.unobtrusive.min.js` file in your application.

The `ej.unobtrusive.min.js` file can be accessed from the following location, which can then be copied and referred in your application.

```
`html
```

```
<script src=http://cdn.boldreports.com/5.4.20/scripts/common/ej.unobtrusive.min.js></script>
```

```
,
```

[How to use report viewer and designer with unobtrusive JavaScript Enabled](#)

Here, the HTML5 syntax is used for defining any of the control and its properties, instead of manually converting the HTML elements into Bold Reports widgets through JavaScript. All the components can be initialized with usual HTML mark-up tags, by setting the name of the component to the `data-role` attribute in lowercase.

The `data-role` type is enabled by default. Therefore, while making use of the data-role in control creation, `data-bold` keyword should be prepended to all the properties that we need to define for a control.

The demonstration of such data-role declaration with a simple **ReportViewer** control creation is shown below,

Refer the **ej.unobtrusive.min.js** file in your application along with the other script and CSS reference section and add the code for defining the ReportViewer control with the basic HTML mark-up tags along with its attributes defined with **data-bold** keyword prepended as shown below,

```
`html
<html>
<head>
...
<script src=http://cdn.boldreports.com/5.4.20/scripts/common/ej.unobtrusive.min.js></script>
</head>
<body>
<!-- Report Viewer component script-->
<div style="width:100%; height:60%; position:absolute;">
<div id="viewer" data-role="boldreportviewer" data-bold-processingmode='local' data-bold-
enablenotificationbar='true' ></div>
</div>
</body>
</html>
`
```

In the above code, **processingmode** is one of the **ReportViewer** property to set the processing mode for the control, which is defined here with **data-bold** keyword prepended to it.

How to get a dataset name in Web API Service

You can get a dataset name in Web API Service when using the ***ReportHelper** class in our Bold Reports Report Viewer component. You can refer the below code snippet for how to get Dataset name at controller side in ASP.NET and ASP.NET Core.

ASP.NET

Store the **jsonResult** in local property and use that property in **ReportHelper.GetDataSetNames**. The following code sample demonstrates to get the dataset name in the **OnReportLoaded** method.

```
`csharp
private Dictionary<string, object> _jsonResult;
//Post action for processing the rdl/rdlc report
public object PostReportAction(Dictionary<string, object> jsonResult)
{
if (jsonResult != null && jsonResult.Keys.Count > 0)
```



```

{
    _jsonResult = jsonResult;
}
return ReportHelper.ProcessReport(jsonResult, this);
}
public object GetResource(string key, string resourcetype, bool isPrint)
{
    return ReportHelper.GetResource(key, resourcetype, isPrint);
}
//Method will be called when initialize the report options before start processing the report
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
}
//Method will be called when reported is loaded
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    var datasetName = ReportHelper.GetDataSetNames(_jsonResult, this);
}
`

```

ASP.NET Core

Store the **jsonResult** in local property and use that property in **ReportHelper.GetDataSetNames**. The following code sample demonstrates to get the dataset name in the **OnReportLoaded** method.

```

`csharp
private Dictionary<string, object> _jsonResult;
//Post action for processing the rdل/rdlc report
public object PostReportAction([FromBody]Dictionary<string, object> jsonResult)
{
    if (jsonResult != null && jsonResult.Keys.Count > 0)
    {
        _jsonResult = jsonResult;
    }
    return ReportHelper.ProcessReport(jsonResult, this);
}

```

```

}
public object GetResource(string key, string resourcetype, bool isPrint)
{
return ReportHelper.GetResource(key, resourcetype, isPrint);
}
//Method will be called when initialize the report options before start processing the report
[NonAction]
public void OnInitReportOptions(ReportViewerOptions reportOption)
{
}
//Method will be called when reported is loaded
[NonAction]
public void OnReportLoaded(ReportViewerOptions reportOption)
{
var datasetName = ReportHelper.GetDataSetNames(jsonResult, this, cache);
}
,

```

Where can i find the installation error and debug log files

You can find the error log files for Embedded Reporting Tools installation failures in the following location.

- C:\Program Files (x86)\Bold Reports\Embedded Reporting Tools\Infrastructure\Install Log
- C:\Program Files (x86)\Bold Reports\Embedded Reporting Tools\Utilities\InstallInfoGenerator

Is it possible to showing the selected all text instead of showing all values from a parameter

Yes, you can able to show the **Selected All** text instead of showing all values from parameter when we choose the select all option in Parameter using expression. You can find the below expression for achieving this.

```

`csharp
=iif(Parameters!<ParameterName>.Count == Count(Fields!<FieldName>.Value,"<DatasetName>"),
"Selected All",Join(Parameters!<ParameterName>.value,""))
,

```

How to avoid the extra blank pages in print and print preview

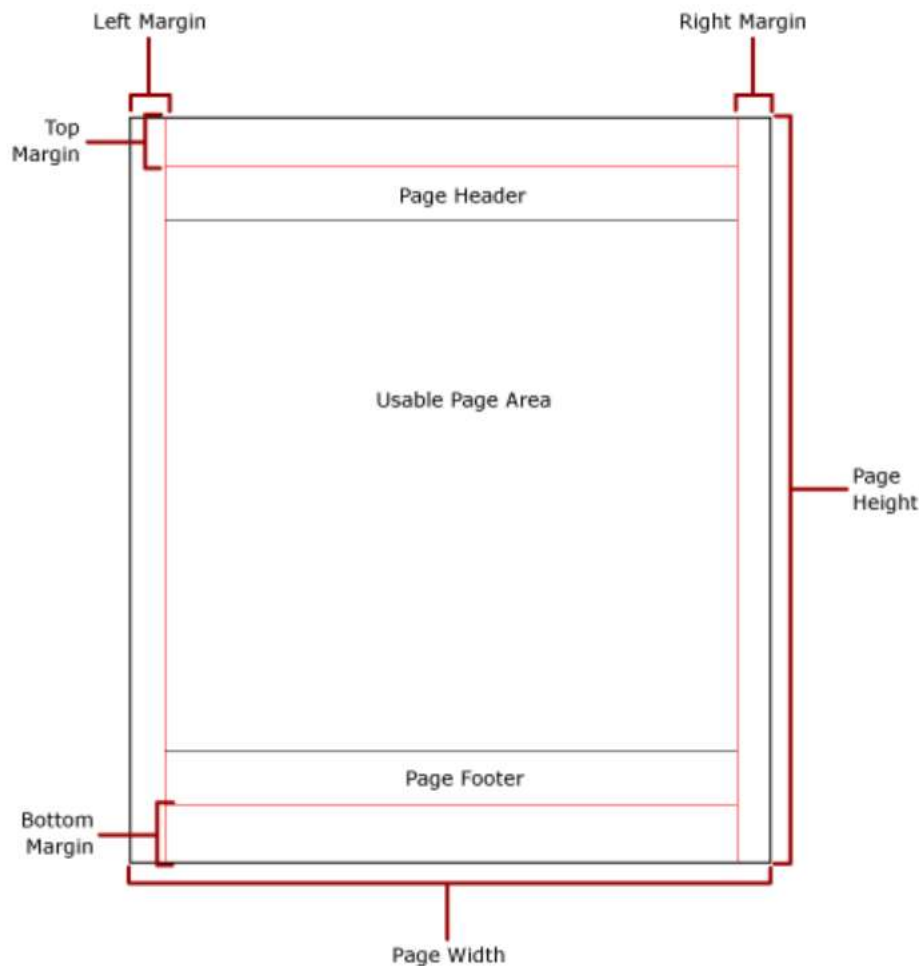
The paper size that you specify for the report in the **Report Properties** will define the pagination for the report while printing and rendering report in the print layout. The extra blank page is created when the body of your report is too wide for your page. If you want the report to appear on a single page, all the content within the report body must fit on the physical page and the body width should be lesser or equal to the following formula:

```
`csharp
```

```
Body Width <= Page Width - (Left Margin + Right Margin)
```

```
`
```

For physical page renders, the concept of Usable Area should be important to keep in mind. The area of the physical page that remains after the space is allocated for margins, column spacing, and page header and footer is called the usable page area. Margins are applied only when you render the report in the print layout and print reports. The following image indicates the margin and usable page area of a physical page.



The following formula is used to calculate the usable area of the report rendering:

Horizontal usable area

```
`csharp
```

```
X = Page.Width - (Left Margin + Right Margin + Column Spacing)
```

```
,
```

Vertical usable area

```
`csharp
```

```
Y = Page.Height - (Top Margin + Bottom Margin + Header Height + Footer Height)
```

```
,
```

Consider the report width is 21 cm, the left margin of the report is 0.5 cm, and the right margin of the report is 0.5 cm. To avoid an extra printed page in the exported PDF file, the following formula is used:

```
`csharp
```

```
width of body (20) + left margin (0.5) + right margin (0.5) <= report width (21)
```

```
,
```

If the width of body is 20 or lesser, it will be rendered without extra pages. When it uses greater than 20, it will add extra pages.

How to use a frame border-image for PDF exported documentation

This article explains to use the frame border-image for PDF exported documentation using Bold Report Report Writer component with Syncfusion PDF libraries.

You can refer the [Report Writer documentation](#) for getting the PDF document from Report.

You have to use **Syncfusion.Pdf** for use the frame border-image for PDF exported documentation. You can refer the following code snippet for how to use the frame border-image for PDF exported documentation.

```
`csharp
```

```
[HttpPost]
```

```
public IActionResult RDLPDF()
```

```
{
```

```
// Export the RDL To PDF.
```

```
FileStream mainReportStream = new FileStream(_hostingEnvironment.WebRootPath +  
@"\Reports\SampleReport.rdl", FileMode.Open, FileAccess.Read);
```

```
BoldReports.Writer.ReportWriter writer = new BoldReports.Writer.ReportWriter();
```

```
writer.LoadReport(mainReportStream);
```

```
MemoryStream memoryStream = new MemoryStream();
```

```
writer.Save(memoryStream, BoldReports.Writer.WriterFormat.PDF);
```

```
// Download the generated export document to the client side.
```

```
memoryStream.Position = 0;
```

```
FileStreamResult fileStreamResult = new FileStreamResult(memoryStream, "application/pdf");
var fileStream1 = fileStreamResult.FileStream;
// Combine the Exported PDF and templatecombine
PdfDocument finalDoc = new PdfDocument();
FileStream fileStream2 = new FileStream(_hostingEnvironment.WebRootPath + @"\PDF\Template.pdf",
    FileMode.Open, FileAccess.Read);
// Creates a PDF stream for merging
Stream[] streams = { fileStream1, fileStream2 };
// Merges the PDF Document.
PdfDocumentBase.Merge(finalDoc, streams);
//Save the document into the stream
MemoryStream combinestream = new MemoryStream();
finalDoc.Save(combinestream);
combinestream.Position = 0;
FileStreamResult fileStreamResult1 = new FileStreamResult(combinestream, "application/pdf");
var pagenumber = fileStreamResult1.FileStream;
// Add a Page number for combined PDF
PdfLoadedDocument loadedDoc = new PdfLoadedDocument(pagenumber);
// Create a new PDF document
PdfDocument doc = new PdfDocument();
doc.PageSettings.Margins.All = 0;
//Add a page to the document
PdfPage page = doc.Pages.Add();
//Create PDF graphics for the page
PdfGraphics graphics = page.Graphics;
FileStream fileStream3 = new FileStream(_hostingEnvironment.WebRootPath +
    @"\f25e9269edfa67ef53e840eea0a98c30.jpg", FileMode.Open, FileAccess.Read);
//Load the image from the disk
PdfBitmap image = new PdfBitmap(fileStream3);
//Draw the image
graphics.DrawImage(image, 0, 0, page.GetClientSize().Width, page.GetClientSize().Height);
//Save the document
MemoryStream ms = new MemoryStream();
doc.Save(ms);
```

```
//Close the document
doc.Close(true);

//Load the border design PDF document
PdfLoadedDocument loadedDocument1 = new PdfLoadedDocument(ms);

//Create a new document
PdfDocument document = new PdfDocument();
document.PageSettings.Margins.All = 0;

//Add the border design in each page of an existing PDF document
for (int i = 0; i < loadedDoc.Pages.Count; i++)
{
    //Add a PDF page
    PdfPage pdfPage = document.Pages.Add();

    //Create a template from the first document
    PdfPageBase loadedPage = loadedDocument1.Pages[0];
    PdfTemplate template = loadedPage.CreateTemplate();

    //Draw the loaded template into a new document
    pdfPage.Graphics.DrawPdfTemplate(template, PointF.Empty, page.GetClientSize());

    //Create a template from the second document
    loadedPage = loadedDoc.Pages[i];
    template = loadedPage.CreateTemplate();

    //Draw the loaded template into a new document
    pdfPage.Graphics.DrawPdfTemplate(template, PointF.Empty, page.GetClientSize());
}

MemoryStream pagenumberstream = new MemoryStream();
document.Save(pagenumberstream);

//Close the PDF documents
document.Close(true);
loadedDocument1.Close(true);
loadedDoc.Close(true);

string contentType = "application/pdf";

//Define the file name
string fileName = "Combinewithpagenumber.pdf";

//Creates a FileContentResult object by using the file contents, content type, and file name
```

```

pagenumberstream.Position = 0;
return File(pagenumberstream, contentType, fileName);
}
`

```

[PDF document with the frame border sample](#)

What are the differences in Embedded Reporting and Report Viewer SDK

Difference between the Embedded Reporting tools and Report Viewer SDK in Bold Reports Component to refer the following table for more details.

	Embedded Reporting	Report Viewer SDK
Bold Reports Cloud Report Server	No	No
Bold Reports Enterprise Report Server	Yes	No
Report Designer component	Yes	No
Report Viewer for application	Yes	Yes
Report Writer library (Export Reports to different formats without view)		Yes
Standalone Report Designer for creating Reports	Yes	Yes

See also

- [Can Syncfusion licenses be used with Bold Reports](#)
- [Can the Syncfusion community license be used with Bold Reports](#)
- [Can the Report Viewer component be accessed from Bold Reports using Syncfusion community license](#)
- [Does Bold Reports Embedded Reporting Tools or Viewer SDK require additional licensing when deploying an application to Azure App service](#)

Does Bold Report Viewer use SSRS Report processing

Bold Report Viewer can load reports from SSRS, but it does not use SSRS report processing for rendering of reports. The Bold Report Viewer will get only the definitions from SSRS such as reports, data sources and datasets. The data processing will happen in the application server that serves as the backend for the Bold Report Viewer. A connection is made to the data source using resource details retrieved from the SSRS Report Server and the connection string, query details used in the report. You can see the processing flow of a report loading in Bold Report Viewer from SSRS in the following image.



See also

[How to provide the permission for user to access the SSRS Report Server reports?](#)

Why should not use Produces attribute in web API controller

Produces attribute forces all actions within the controller to return JSON-formatted responses.

ReportHelper itself will provide the processing data with JSON string. So, there is no need to use produces attribute in Web API controller and actions for converting the JSON also should not be changed for other types.

```
`csharp
```

```
public object PostReportAction(Dictionary<string, object> jsonResult)
{
    return ReportHelper.ProcessReport(jsonResult, this);
}
`
```


How to convert Crystal Reports to RDL for Bold Reports

Bold Reports does not have an automated option or tool to convert Crystal Reports to RDL. You need to manually convert the Crystal Reports. You can refer [here](#) to convert your Crystal Reports manually.

Is it possible to use a data table in Bold Reports

Yes, it is possible to use data table in Bold Reports. The data source can be added as a data table using `BoldReports.Web.ReportDataSource`. The steps involved in adding a sample data source using data table are provided as follows.

- Create a SQL connection and add the data source to the report in the `OnReportLoaded` function.

```
`csharp
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    System.Data.SqlClient.SqlConnection connection = new System.Data.SqlClient.SqlConnection(@"Data
    Source=<instancename>;Initial Catalog=<database>;User id=<username>;Password=<password>;");
    using (connection)
    {
        reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource()
        {
            ...
        });
    }
}
```

- While adding the data source, the dataset name used in the report must be provided along with the data table as shown in the following code sample.

```
`csharp
public void OnReportLoaded(ReportViewerOptions reportOption)
{
    System.Data.SqlClient.SqlConnection connection = new System.Data.SqlClient.SqlConnection(@"Data
    Source=<instancename>;Initial Catalog=<database>;User id=<username>;Password=<password>;");
    using (connection)
    {
        reportOption.ReportModel.DataSources.Add(new BoldReports.Web.ReportDataSource()
```

```

{
    Name = "DataSet1",
    Value = this.GetDataTable(connection)
});
}
}

public System.Data.DataTable GetDataTable(System.Data.SqlClient.SqlConnection connection)
{
    System.Data.DataSet dataset = new System.Data.DataSet();
    System.Data.SqlClient.SqlDataAdapter adapter = new System.Data.SqlClient.SqlDataAdapter();
    adapter.SelectCommand = new System.Data.SqlClient.SqlCommand(
        @"SELECT top 10
        [HumanResources].[Department].[DepartmentID],[HumanResources].[Department].[Name],
        [HumanResources].[Department].[GroupName],[HumanResources].[Department].[ModifiedDate]
        FROM[HumanResources].[Department]",
        connection);
    adapter.Fill(dataset);
    return dataset.Tables[0];
}

```

Here the **Name** is case-sensitive and it should be same as in the dataset name in the report definition.

The **Value** also accepts IList and DataSet inputs.

Why http authorization headers are not being set for all requests

Http authorization headers not being set for all requests on Report Viewer and Report Designer in ASP.NET Core and ASP.NET MVC applications because of the following reasons mentioned in the table.

Report Action	Action Name	Request type	Component	Headers	Comments
Report Rendering	PostReportAction	POST	Report Viewer, Report Designer	Yes	No
Report Exporting (ASP.NET MVC)	PostReportAction	POST	Report Viewer, Report Designer	No	This request is to download the exporting document from Report Viewer and Report Designer applications with

					Form post action. So, the request headers cannot be passed with this action. Regarding the security, you can refer more details from this link .
Report Exporting (ASP.NET Core)	PostFormReportAction	POST	Report Viewer, Report Designer	No	This request is to download the exporting document from Report Viewer and Report Designer applications with Form post action. So, the request headers cannot be passed with this action. Regarding the security, you can refer more details from this link .
Image Rendering	Get Resource	GET	Report Viewer, Report Designer	No	This request is to render the image items with Report Viewer and Report Designer applications. Since, this request URL is used with HTML image src attribute, you cannot add additional headers for this request and you can refer more details from this link .
Report Designing	PostDesignerAction	POST	Report Designer	Yes	No

Report Save (ASP.NET MVC)	PostDesignerAction	POST	Report Designer	No	This request method is used for the download process (save the report in the device), upload process (open the report from the device), and add an image (open the image from the device). You cannot use further to get the data from the server.
Report Save (ASP.NET Core)	PostFormDesignerAction	POST	Report Designer	No	This request method is used for the download process (save the report in the device), upload process (open the report from the device), and add an image (open the image from the device). You cannot use further to get the data from the server.
Open Report and Add image	UploadReportAction	POST	Report Designer	Yes	No
Image Manager Images Reporting	GetImage	GET	Report Designer	No	This request is to render the image items with Report Viewer and Report Designer applications. Since, this request URL is used with HTML image src attribute, you cannot add additional

					headers for this request and you can refer more details from this link .
--	--	--	--	--	--

CDN links for Localization and Culture

You can get the CDN links for Localization and Culture scripts in the following list based on the Culture Code.

ar-AE

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.ar-AE.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.ar-AE.min.js"><script>
```

,

cs-CZ

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.cs-CZ.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.cs-CZ.min.js"><script>
```

,

da-DK

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.da-DK.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.da-DK.min.js"><script>
```

,

de-DE

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.de-DE.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.de-DE.min.js"><script>
```

,

en-GB

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.en-GB.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.en-GB.min.js"><script>
```

,

[en-US](#)``html``<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.en-US.min.js"></script>``<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.en-US.min.js"></script>``,`[es-ES](#)``html``<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.es-ES.min.js"></script>``<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.es-ES.min.js"></script>``,`[fa-IR](#)``html``<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.fa-IR.min.js"></script>``<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.fa-IR.min.js"></script>``,`[fi-FI](#)``html``<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.fi-FI.min.js"></script>``<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.fi-FI.min.js"></script>``,`[fr-FR](#)``html``<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.fr-FR.min.js"></script>``<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.fr-FR.min.js"></script>``,`[he-IL](#)``html``<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.he-IL.min.js"></script>``<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.he-IL.min.js"></script>``,`[hr-HR](#)``html``<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.hr-HR.min.js"></script>``<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.hr-HR.min.js"></script>`

,

hu-HU

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.hu-HU.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.hu-HU.min.js"><script>
```

,

it-IT

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.it-IT.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.it-IT.min.js"><script>
```

,

ja-JP

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.ja-JP.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.ja-JP.min.js"><script>
```

,

ko-KR

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.ko-KR.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.ko-KR.min.js"><script>
```

,

ms-MY

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.ms-MY.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.ms-MY.min.js"><script>
```

,

nb-NO

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.nb-NO.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.nb-NO.min.js"><script>
```

,

nl-NL

`html

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.nl-NL.min.js"><script>
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.nl-NL.min.js"></script>  
,
```

pl-PL

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.pl-PL.min.js"></script>  
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.pl-PL.min.js"></script>  
,
```

pt-PT

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.pt-PT.min.js"></script>  
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.pt-PT.min.js"></script>  
,
```

ro-RO

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.ro-RO.min.js"></script>  
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.ro-RO.min.js"></script>  
,
```

ru-RU

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.ru-RU.min.js"></script>  
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.ru-RU.min.js"></script>  
,
```

sk-SK

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.sk-SK.min.js"></script>  
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.sk-SK.min.js"></script>  
,
```

sv-SE

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.sv-SE.min.js"></script>  
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.sv-SE.min.js"></script>  
,
```

tr-TR

```
`html
```



```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.tr-TR.min.js"></script>
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.tr-TR.min.js"></script>
,
```

vi-VN

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.vi-VN.min.js"></script>
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.vi-VN.min.js"></script>
,
```

zh-CN

```
`html
```

```
<script src="http://cdn.boldreports.com/5.4.20/scripts/l10n/ej.localetexts.zh-CN.min.js"></script>
<script src="http://cdn.boldreports.com/5.4.20/scripts/i18n/ej.culture.zh-CN.min.js"></script>
,
```

How to resolve the Routing issue in ASP.NET MVC application

You will get the issue **Multiple actions were found that matches the request** for report viewer and report designer Web API in the ASP.NET MVC application controller, when Web API is not routed properly with action. You have to create all the Web API methods with **ActionName** and **NonAction** attributes as like below,

```
`csharp
```

```
public class ReportApiController : ApiController, IReportController
{
    [System.Web.Http.ActionName("PostReportAction")]
    // Post action for processing the RDL/RDLC report
    public object PostReportAction(Dictionary<string, object> jsonResult)
    {
        return ReportHelper.ProcessReport(jsonResult, this);
    }
    // Get action for getting resources from the report
    [System.Web.Http.ActionName("GetResource")]
    [AcceptVerbs("GET")]
    public object GetResource(string key, string resourcetype, bool isPrint)
    {
        return ReportHelper.GetResource(key, resourcetype, isPrint);
    }
}
```

[NonAction]

// Method that will be called when initialize the report options before start processing the report

```
public void OnInitReportOptions(ReportViewerOptions reportOption)
```

```
{
```

```
// You can update report options here
```

```
}
```

[NonAction]

// Method that will be called when reported is loaded

```
public void OnReportLoaded(ReportViewerOptions reportOption)
```

```
{
```

```
// You can update report options here
```

```
}
```

```
}
```

```
,
```

How to print the report using the external button

You can print the report with external button using [Print](#) method.

See also

[How to hide the print button from Report viewer toolbar](#)

How to deploy PhantomJS WebKit manually

PhantomJS is a headless WebKit script able with JavaScript. It is a free software/open source that may contain MIT, BSD, LGPL or GPL, or other similar licenses It contains third-party code. This executable file is necessary to export the data visualization report items during report export. Without this, the data visualization report items no longer available in the report exported documents. It is your decision if you choose to download Phantom JS, but you must accept all of their terms and conditions if you want to use it with Syncfusion's products.

To download the PhantomJS application and deploy it on your machine, you should accept its license terms on [LICENSE](#) and [Third-Party](#) document. Then, you can download PhantomJS by clicking this [link](#).

Once download completed, extract the zip file and then copy the PhantomJS application from {Extracted Location}\PhantomJS-2.1.1-windows\bin and paste it in the below mentioned deploy location.

Install Location:

Please place the PhantomJS executable file in the location.

```
{Deployed Location}\Bold Reports\Embedded Reporting\Samples\ASP.NET
Core\wwwroot\PhantomJS
```

Example:

C:\Users\Public\Documents\Bold Reports\Embedded Reporting\Samples\ASP.NET Core\wwwroot\PhantomJS\phantomjs.exe

Overview

The Report Viewer SDK provides rich set of embedded reporting tools Report Viewer, Report Writer and Report Server in a single point of access to design, preview, manage, schedule, export and print the reports.

Key features

[Display Reports](#) --- You can preview the new or already created reports to take a look on generated report within your applications (Javascript, ASP.NET CORE, Angular, ASP.NET, ASP.NET MVC, WPF, UWP) using Report Viewer.

[Export Reports](#) --- You can export the RDL / RDLC report to popular formats like PDF, Excel, CSV, PowerPoint, Word and HTML using Report Writer library. Also you can export your report from Report Designer, Report Viewer and Report Server.

Troubleshoot installation or upgrade problems

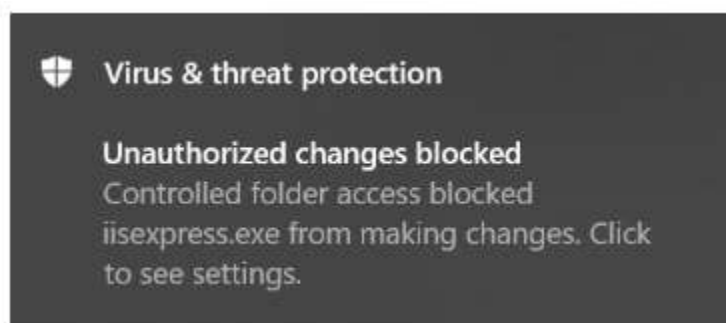
This troubleshooting guide provides step-by-step procedure to resolve the problems encountered while installing and upgrading the Report Viewer SDK.

Allow controlled folder access and the protected file usage

[Controlled folder access](#) helps you protect valuable data from malicious apps and threats, such as [ransomware](#). Controlled folder access is included with Windows 10 and Windows Server 2019. When you enable the ransomware protection in your machine, the applications will be restricted to access the protected folders in your machine.

If the Bold Reports Report Viewer SDK or its related executable are not allowed to access the protected folders in your machine when the ransomware protection is enabled, follow the below-mentioned steps to provide access to Bold Reports application for accessing the protected folders.

You will receive notifications from Windows when an application is blocked from accessing the protected folders. For example, the following message will be displayed when you are trying to preview the dashboard with ransomware protection enabled.

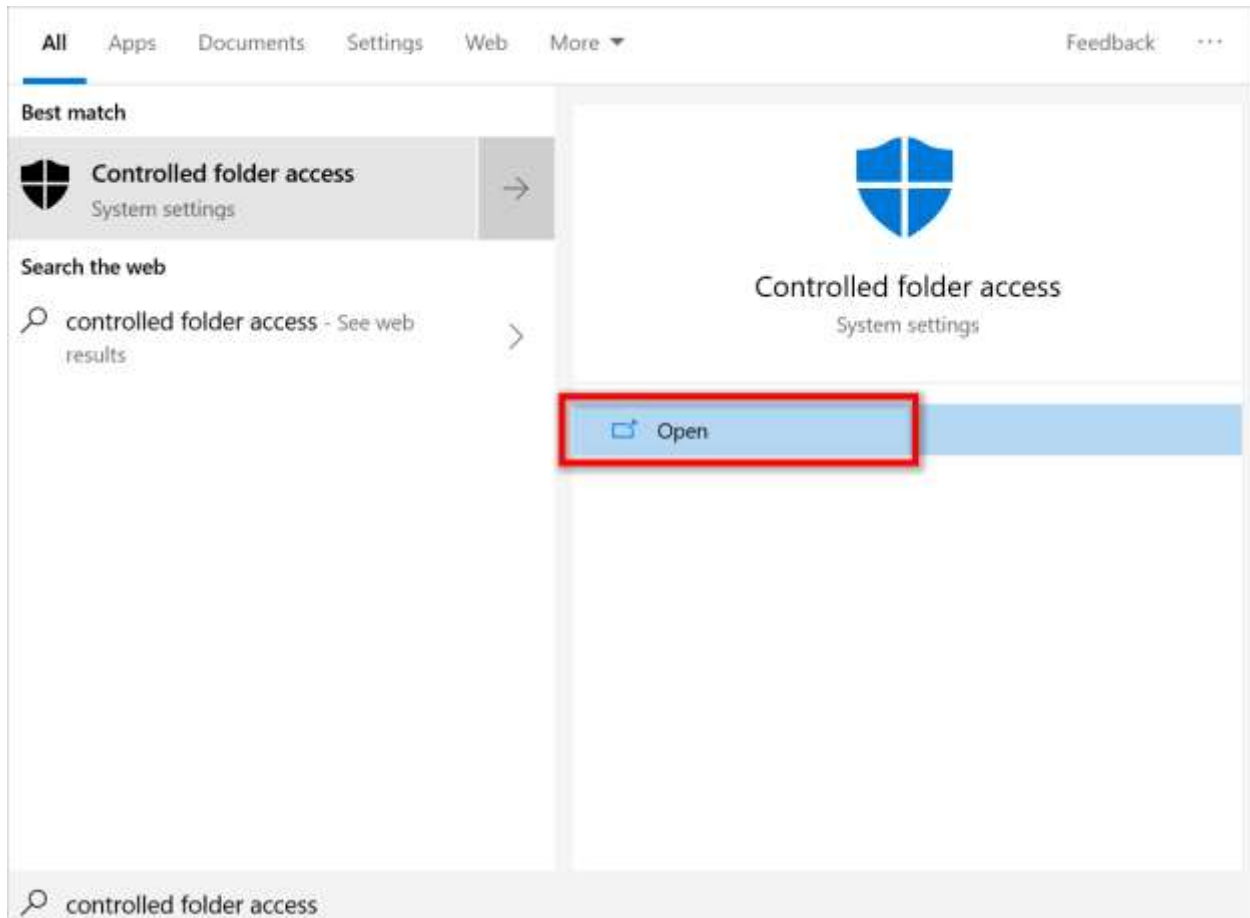


Steps to allow Bold Reports Report Viewer SDK to access protected folders

- Open the Ransomware protection in Windows security using the start menu.

Troubleshoot installation or upgrade problems
usage

Allow controlled folder access and the protected file



- Click the option **Allow an app through Controlled folder access**.

Ransomware protection

Protect your files against threats like ransomware, and see how to restore files in case of an attack.

Controlled folder access

Protect files, folders, and memory areas on your device from unauthorized changes by unfriendly applications.



On

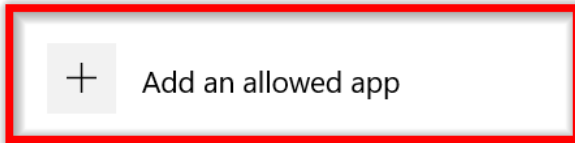
Protected folders

[Allow an app through Controlled folder access](#)

- The list of applications will be shown that can access the protected folders in your machine and click the **Add an allowed app** option.

Allow an app through Controlled folder access

If Controlled folder access has blocked an app you trust, you can add it as an allowed app. This allows the app to make changes to protected folders.




Most of your apps will be allowed by Controlled folder access without adding them here. Apps determined by Microsoft as friendly are always allowed.

- It will help you choose which application should be allowed to access the protected folder by showing recently blocked apps and all apps.

Allow an app through Controlled folder access

If Controlled folder access has blocked an app you trust, you can add it as an allowed app. This allows the app to make changes to protected folders.

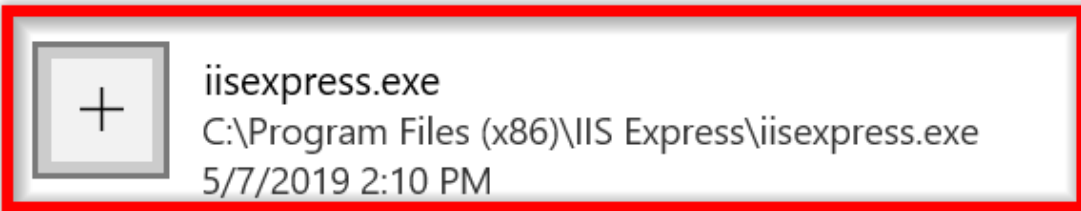
 Add an allowed app

Mod... Recently blocked apps ... allowed by Controlled folder access without
add... Browse all apps ... determined by Microsoft as friendly are always
allo...

You can click any one of the options and choose the Bold Reports Report Viewer SDK related applications.

Recently blocked apps

The following apps were recently blocked by Controlled folder access. Make sure that you trust an app before you allow it.



- Allow the iisexpress and the Bold Reports Report Viewer SDK related apps to access your protected folders if the application is blocked in your machine.

Application	Path
-------------	------

iisexpress.exe	C:\Program Files (x86)\IIS Express
Bold Reports Report Viewer SDK Sample Browser Launcher	C:\Program Files (x86)\Bold Reports\Report Viewer SDK\Utilities\StartSampleBrowserReportingTools\StartSampleBrowserReportingTools.exe

- Now, you can continue to use the Bold Reports Report Viewer SDK without blocking any operations to your protected folders.

References

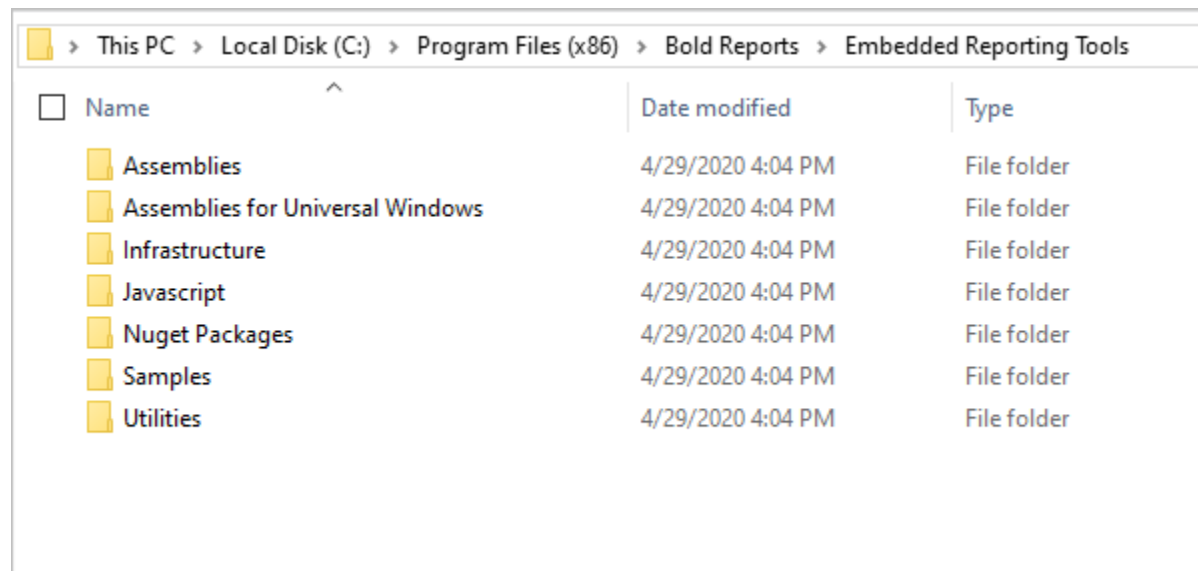
- <https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/enable-controlled-folders>
- <https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/customize-controlled-folders#allow-specific-apps-to-make-changes-to-controlled-folders>

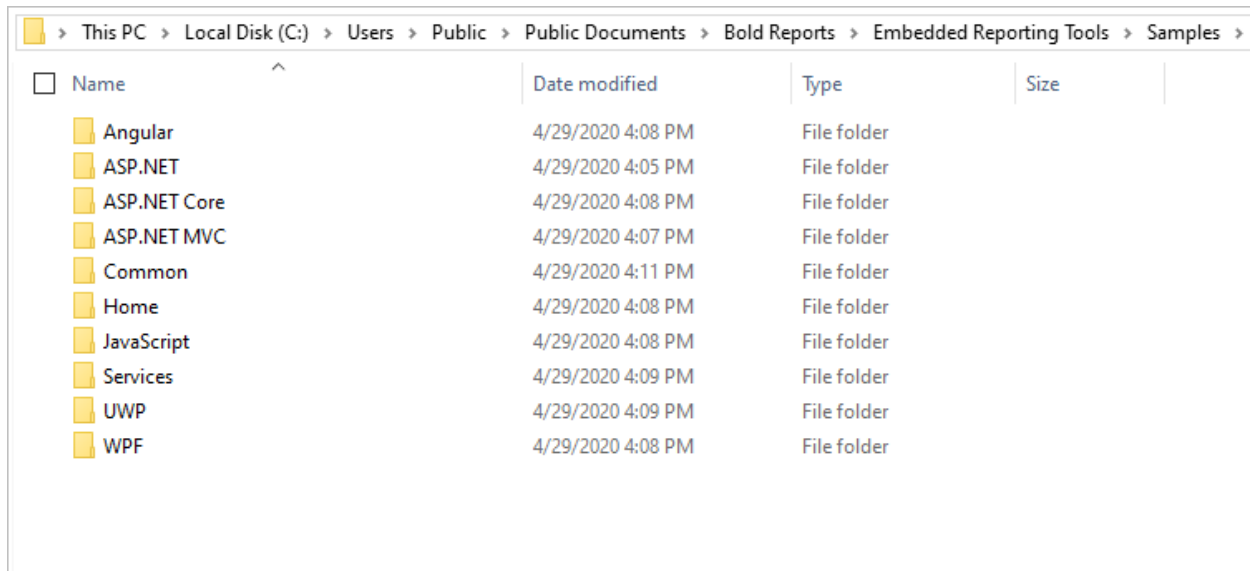
Troubleshooting Report Viewer SDK sample browsers

If you are facing issues with launching the **Bold Reports Report Viewer SDK** local samples follow the below troubleshooting mechanism.

- **Even though you have allowed permission for blocking apps, still not able to launch the Report Viewer SDK Sample Browser?**

We recommend you to ensure the below folders present in the installation path **C:\Program Files (x86)\Bold Reports\Report Viewer SDK** and **C:\Users\Public\Documents\Bold Reports\Report Viewer SDK\Samples**.



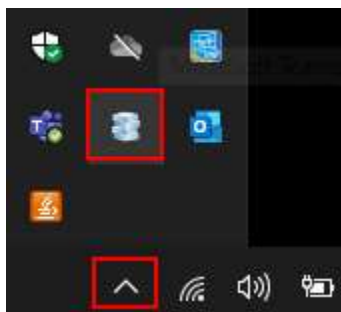


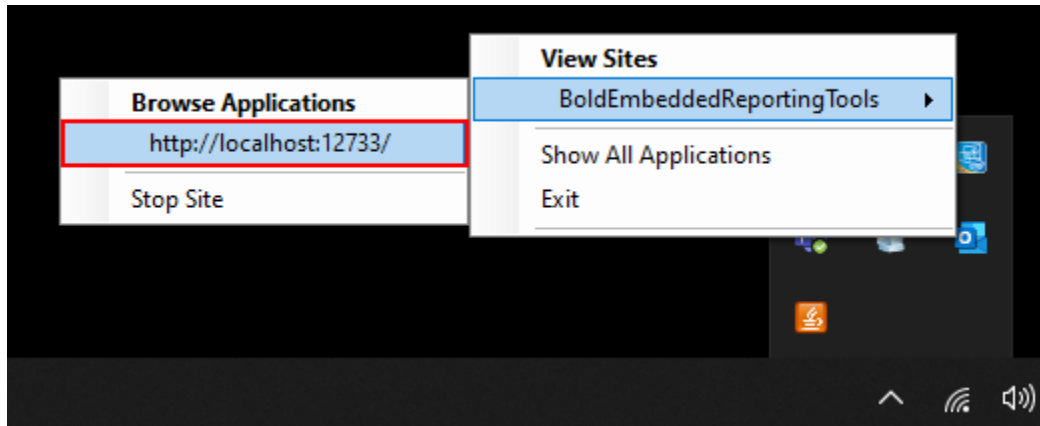
- **Even you have properly shipped folders, but still not able to launch Bold Reports Report Viewer SDK Sample Browser?**

Open command prompt in administrator mode and navigate to the path `C:\Program Files (x86)\Bold Reports\Report Viewer SDK\Utilities\StartSampleBrowserReportingTools` and run anyone of the below command through the `StartSampleBrowserReportingTools.exe`.

Platform	Arguments
JavaScript	<code>StartSampleBrowserReportingTools.exe "JAVASCRIPT"</code>
Angular	<code>StartSampleBrowserReportingTools.exe "ANGULAR"</code>
ASP.NET	<code>StartSampleBrowserReportingTools.exe "ASPNET"</code>
ASP.NET MVC	<code>StartSampleBrowserReportingTools.exe "ASPNETMVC"</code>
ASP.NET Core	<code>StartSampleBrowserReportingTools.exe "ASPNETCORE"</code>

After running the above command, ensure the System tray contains `IISEXPRESS.exe` application which hosts the samples.





- **If none of the above steps doesn't help to launch the Samples?**

You can able to find the error log file in the path `C:\Program Files (x86)\Bold Reports\Report Viewer SDK\Utilities\StartSampleBrowserReportingTools`, kindly [contact us](#) by creating a support ticket and share the generated log file with us, we will reach you soon.

Error codes in report parsing and processing

This section provides a detailed explanation and solution to the respective problems in the Report Viewer for parsing and processing the report.

RV0001

The source of the report definition has not been specified.

Description

The Report Viewer will not render the report when either the report path or stream is not set, or the given path is incorrect.

Solution

1. Make sure that the report path is specified and ensure whether the name or report path is correct.
2. Check whether the report exists in the specified directory or not and also it should be accessible.

RV0009

Provide Report Server information to get shared data source information.

Description

Report with a shared data source must contain the Report Server information and credentials that are provided to render in the Report Viewer.

Solution

Make sure that the credentials in the Report Server URL are not empty and ensure that the correct information is provided to the Report Viewer. For example, refer to the below code.

```
`csharp
```

```
Viewer.ReportServerUrl = @"https://demos.boldreports.com/services/api/ReportViewer";
```

```
Viewer.ReportServerCredential = new System.Net.NetworkCredential("username", "password");  
`
```

RV0010

An error occurred while sending a request to the reporting service.

Description

Report will not be rendered when there is a problem while connecting or retrieving the data from the provided reporting service's URL.

Solution

1. Make sure that you are connected to the internet.
2. Ensure that the specified reporting service URL is valid.
3. whether the reporting service is running or not.

If not, start the reporting service from the desktop shortcut (for example: Start BoldReports in IIS) or from an installed location.

Installed location path: (C:\Program Files (x86)\Bold Reports\Enterprise Reporting\Utilities\DeployIIS)

RV0013

The report definition has an invalid schema version.

Description

Report definition should contain the supported schema version (for example, 2008) to render the report in the Report Viewer.

Solution

Make sure that the provided schema in your report definition is supported in the Report Viewer.

Supported versions are 2008, 2010, and 2016.

RV0014

The requested file or assembly was not found. It may have either moved or renamed.

Description

Any of the assembly related to the report platform is missing at the installation location.

Solution

1. If you are previewing the report from the Report Designer, check whether any assembly is missing at the installed location (C:\Program Files (x86)\Bold Reports\Report Designer) or reinstall the Report Designer application.
2. If you are loading the report from SDK sample, check whether any assembly is missing at the installed location (C:\Program Files (x86)\Bold Reports\Embedded Reporting\Assemblies) or reinstall the Report Platform SDK application.
3. If you are loading the report from the Report Server, check whether any assembly is missing at the installed location (C:\Program Files (x86)\Bold Reports\Enterprise Reporting\Utilities\DeployIIS) or reinstall the Report Server application.

Note: Please contact Syncfusion support if you cannot resolve the issue.

RV0023

Expressions are not used within the scope.

Description

Report Viewer encounters an error when an expression must be used outside the scope.

Solution

Provide a valid expression within the scope.

Note: Detailed description with the line number and position is mentioned in the Report Viewer instance error dialog.

RV0024

An error occurred during the local report processing. There is an error that occurred while rendering the custom code.

Description

Report Viewer encounters an error when compiling the custom code used in the report.

Solution

Provide valid custom code in your report based on detailed information mentioned in the Report Viewer instance error dialog.

Note: Custom code exists in report properties.

RV0034

The Value property of an image report item contains an invalid value.

Description

Report will not be rendered to the Report Viewer when you try to use invalid image data or a non-existing image in the report.

Solution

Ensure that the value provided to the image report item is specified properly, and make sure that the image is available.

1. Embedded: Image data should be available in a report definition.
2. External: External images can be obtained by specifying a URL and make sure that credentials are sufficient to access the image.
3. Database: Images can be referred from the dataset, and make sure that credentials are sufficient to access the database.

RV0051

The data source used in a report does not exist or not accessible.

Description

Report will not be rendered in the Report Viewer when the data source used in a report does not exist or it is not accessible due to permission rights.

Solution

Ensure whether the data source in a report exists in the referred location.

1. For embedded data source, the data source is referred in the local machine.
2. For shared data source, the data source exists on the server and used on the local machine.
Make sure that the authentication is sufficient to access the data source.

RV0062

Fields cannot be used in report parameter expressions.

Description

Report Viewer does not render the report when the parameter contains field expressions and fields cannot be used in report parameter expressions.

Solution

Remove the field expressions used in parameter expressions.

RV0072

Forward dependencies are not valid.

Description

Report will not be rendered when there is an existence of forward dependencies between parameters.

Solution

Make sure to remove the forward dependencies in your report definition before previewing the report.

Example: First Parameter depends on Dataset 1 and Dataset 2 depends on Second Parameter, and then it results in forward dependency.

Note: Parameters are executed based on the order.

RV0073

Please select a value for the report parameter.

Description

Report parameters must be supplied with values if it is **non-nullable** while trying to preview the report in the Report Viewer.

Solution

Enter the value of the parameter in a parameter block when it is **non-nullable** or allow blank.

RV0075

The expression that references the parameter is not valid and does not exist in the parameter collection or forward dependencies.

Description

Report Viewer throws errors when you try to use a non-existing report parameter or with forward dependencies.

Solution

Make sure to use the parameters that exist report definition and avoid forward dependencies in the report.

Note: Letters in the names of parameters should be in the correct case.

RV0083

The value provided for the report parameter is not valid for its type.

Description

The report will not be rendered by the Report Viewer when the specified value for the report parameter is irrespective of its type.

Solution

Provide an appropriate value for the report parameter based on its data type while rendering the report.

Example: A parameter with an integer data type allows integer values, not float values.

RV0084

Report item refers to an invalid `DataSetName`.

Description

Report will not be rendered in the Report Viewer when a non-existing dataset is used in the report.

Solution

1. Embedded: Make sure that the report item referred dataset exists in the report definition.
2. Shared: Make sure that the referred dataset exists on the server and accessible.

RV0089

Collection class or data table input is needed.

Description

To render the RDLC report in the Report Viewer, the report must have a collection class or data table input to rendering the report.

Solution

Make sure to provide dataset collection to render the report. Supported collections are `IEnumerable`, `System.Data.DataTable`, and `System.Data.Dataset`.

RV0152

Report variable or group variable reference cannot be used in report parameter expressions.

Description

Report Viewer does not render the report when the parameter contains group or report variables. Variable values cannot be used in report parameter expressions.

Solution

Remove the variable reference used in the parameter expressions.

RV0156

An invalid operator is used in expression.

Description

Report Viewer encounters an error if it contains an invalid operator in an expression.

Solution

Make sure to use appropriate and supported operators in the expression.

Note: Detailed information is provided with the expression, and the report item name is mentioned in the Report Viewer instance error dialog.

RV0164

Data region items are not allowed in reports without datasets to render in the Report Viewer.

Description

Report Viewer will not render the report when the data region items does not refer to the dataset.

Solution

Make sure to provide the dataset name property value in your report definition.

Note: Report will render when the report definition has a single dataset; When the dataset count is greater than one, you must specify the dataset name property value to data region items (Tablix, Chart, Gauge, etc..).

RV0167

Toggle items must be text boxes that share the same scope, and they are not allowed in page headers or footers.

Description

In our Report Viewer, toggle items must be shared within the same scope, and they are not allowed in page headers and page footers.

Solution

Provide toggle items in the report in the same scope, not from their page header or page footer.

Example: If a report item in the page header contains a toggle item, then the referred toggle item should be in the header region.

RV0175

A **ReportName** cannot be an empty string or just white space.

Description

In SubReport item, the **ReportName** property value cannot be an empty or just white space.

Solution

Make sure that the **ReportName** property value for the subreport item is not empty.

RV0293

Aggregate and lookup functions cannot be used in report parameter expressions.

Description

The value expression is used for report parameter, it includes an aggregate or lookup function.

Aggregate and lookup functions cannot be used in report parameter expressions.

Solution

Remove **Aggregate** and **Lookup** functions in your report parameter expressions.

RV0331

Lookup function has an incorrect number of parameters.

Description

Provided lookup function has an incorrect number of parameters in report definition.

Solution

Provide the correct number of arguments for lookup expressions in respective report item expressions and refer to the syntax of the lookup expressions.

RV0334

ReportItem expressions can be referred only to fields within the current dataset.

Description

ReportItem expressions can be referred only to fields within the current dataset.

Make sure that the field expression is referred to the current dataset.

Solution

Provide current dataset fields for report item expression.

Note: Tablix report item with scope **Dataset1** must contain the fields in the **dataset1**. If field expression specified in another dataset is required, then you should specify the scope.

Example: `=First(Fields!Name.Value, "Dataset2")`

RV0347

Each dataset, data region, or grouping in the report has different name for their report item.

Dataset, data region, and grouping names must be unique within a report.

Description

Report item referred dataset, data region, and grouping names must be unique within a report.

Solution

Provide a unique dataset, data region, and grouping name for their report item in your report definition.

1. Incorrect: Dataset name: Item 1, Tablix name: Item 1
2. Correct: Dataset name: Item 1, Tablix name: Item 1

RV0362

An error has occurred on the Report Server.

Description

Specified reporting service encounters an error while rendering the report in the Report Viewer.

Solution

1. Check the error log file(s) generated in the below mentioned location. `(C:\Program Files\Bold Reports\Enterprise Reporting\idp\web\Logs)`
2. For more information about the types of status code, refer to the available status code in online.

RV0394

The value expression for the **TextRun** contains a colon or a line terminator.

Description

The Textbox report item value expression contains a colon or a line terminator.

Colons and line terminators are not valid in expressions.

Solution

Remove colons and line terminator used in the specified report item expression.

RV0444

Cannot load sub report.

Description

Sub report cannot be loaded with an invalid name or report path.

Solution

1. Make sure that the sub report path is specified and ensure whether the name or path is correct.
2. Check whether the report exists in the specified directory or not, and also it should be accessible.

RV0445

The subreport cannot be found at the specified location.

Description

Report will not be rendered in the Report Viewer when the report does not exist in the specified location.

Solution

Make sure that the report name or path is correct, and the report exists in the specified directory.

RV0446

Subreport cannot be shown.

Description

Report Viewer will not render the sub report without the `ReportPath` or `ReportServerUrl`.

Solution

Provide a subreport `ReportPath` or `ReportServerUrl` in your report.

RV0451

The tablix has a detail member with inner members.

Detail members contain only static inner members.

Description

In the tablix report item, detail members contain only static inner members.

Solution

In the tablix report item, provide static inner members for their detail members.

RV0488

The tablix report item has an invalid Tablix member in the column hierarchy, and the `KeepTogether` property is not set to `None`.

Description

All tablix member elements in a `TablixColumnHierarchy` must have the `KeepWithGroup` property as `None`.

Solution

Set `KeepWithGroup` property as `None` in a `TablixColumnHierarchy` for all Tablix members in the tablix report item.

RV0534

Report Deserialization failed.

Description

An error occurred during the deserialization of the report definition.

Solution

1. XML deserialization is failed due to incorrect item tags or missed elements. Report item elements should have its required attributes.
2. Note: Detailed description with line number and position is mentioned in the Report Viewer instance error dialog.

RV0535

Provide Report Server information to get shared dataset information.

Description

Report with shared datasets must contain the Report Server information and credentials to render in the Report Viewer.

Solution

Make sure that the credentials in the Report Server URL are not empty and ensure that the correct information is provided to the Report Viewer.

```
Example: Viewer.ReportServerUrl =  
@"https://demos.boldreports.com/services/api/ReportViewer";  
Viewer.ReportServerCredential = new System.Net.NetworkCredential("username",  
"password");
```

RV0537

The `ReportServerUrl` or `ReportServerCredential` is missing.

Description

Server report or server referred items used in the report will not be rendered in the Report Viewer without `ReportServerUrl` and `ReportServerCredential`.

Solution

Make sure that the `ReportServerUrl` and `ReportServerCredential` are not empty and provided with proper data and are accessible.

RV0538

The value for the `DataSetName` property is missing.

Description

Data region items are not allowed in reports without datasets to render in the Viewer.

Solution

Make sure to provide a dataset name property value in your report definition and ensure that the property value is not empty.

RV0539

Provide dataset inputs for the report.

Description

To render the RDLC report, you must add the data source collection in the Report Viewer if any dataset is used in the report.

Solution

Make sure to add the dataset to render the report in Report Viewer if dataset fields are used.

Example: `Viewer.DataSources.Add("Dataset1", object collection of data)`

Note: Here, name ("Dataset 1") is the dataset name used in the report definition.

RV0540

Invalid Expression

Description

Report Viewer does not render the report when an invalid expression is used.

Solution

Provide valid expressions in the respective report item.

Note: Detailed description with line number and position is mentioned in the Report Viewer instance error dialog.

RV0542

Unsupported Expression

Description

Provided expression does not support in the Report Viewer.

Object expressions do not have complete support.

Solution

Please use supported expressions in the Report Viewer to render the report.

RV0543

The `DateAdd` function should have arguments.

Description

Report Viewer encounters an error if `DateAdd` function does not contain any arguments.

Solution

Provide arguments for `DateAdd` function in the respective report item expression and refer to the syntax of the `DateAdd` function.

Example: `=DateAdd("d",3,Fields!BirthDate.Value)`

RV0544

The `DateDiff` function should contain arguments.

Description

Report Viewer encounters an error if `DateDiff` function does not contain arguments in an expression.

Solution

Provide arguments for `DateDiff` function in the respective report item expression and refer to the syntax of the `DateDiff` function.

Example: `=DateDiff("yyyy",Fields!BirthDate.Value,"1/1/2010")`

RV0546

`PaymentPeriod` must be between 1 and `numberOfPayments`.

Description

Report Viewer encounters an error if the payment period does not fall within the range.

Solution

Provide payment period value between 1 and `numberOfPayments`.

RV0547

This implementation does not handle zero interest rate.

Description

Report is not rendered in the Report Viewer when it holds the zero interest rate.

Solution

Provide greater than 0 or less than 0 interest rate. Does not provide 0 value for their interest rate.

RV0548

Invalid Expression

Description

The expression used in the report is incorrect scope, syntax, or unsupported.

Solution

Ensure that the expression used in the report is correct with valid scope and syntax.

Note: Detailed information of report item along with line number and position is mentioned in the Report Viewer instance error dialog.

RV0549

Method name is not a member in custom code.

Description

Evaluated method name is not a member in your custom code.

Solution

Provide a valid method name. Make sure that the provided method name is present in your custom code.

RV0552

Parameter is missing a value.

Description

Report Viewer will not render the report when the hidden parameter does not contain a value that is non-nullable.

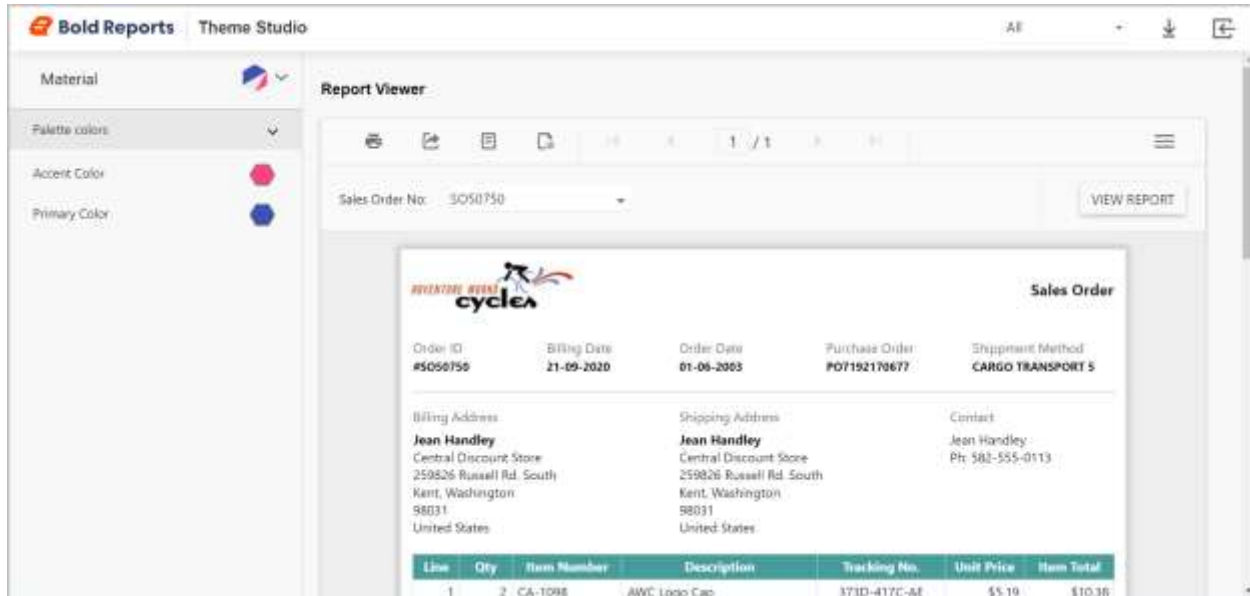
Solution

Provide value to hidden parameters in your report definition before rendering the report.

Note: Hidden parameters must contain a default value, or it should be **nullable**.

Bold Reports Theme Studio

[Bold Reports Theme Studio](#) allows us to build a new theme based on an existing Bold Reports control theme. Also, we can import our customized theme into the Theme Studio and customize the colors further and download the newly customized theme. This comes with Bootstrap, Material, High Contrast 1, High Contrast 2, and Office 365 theme customizations.



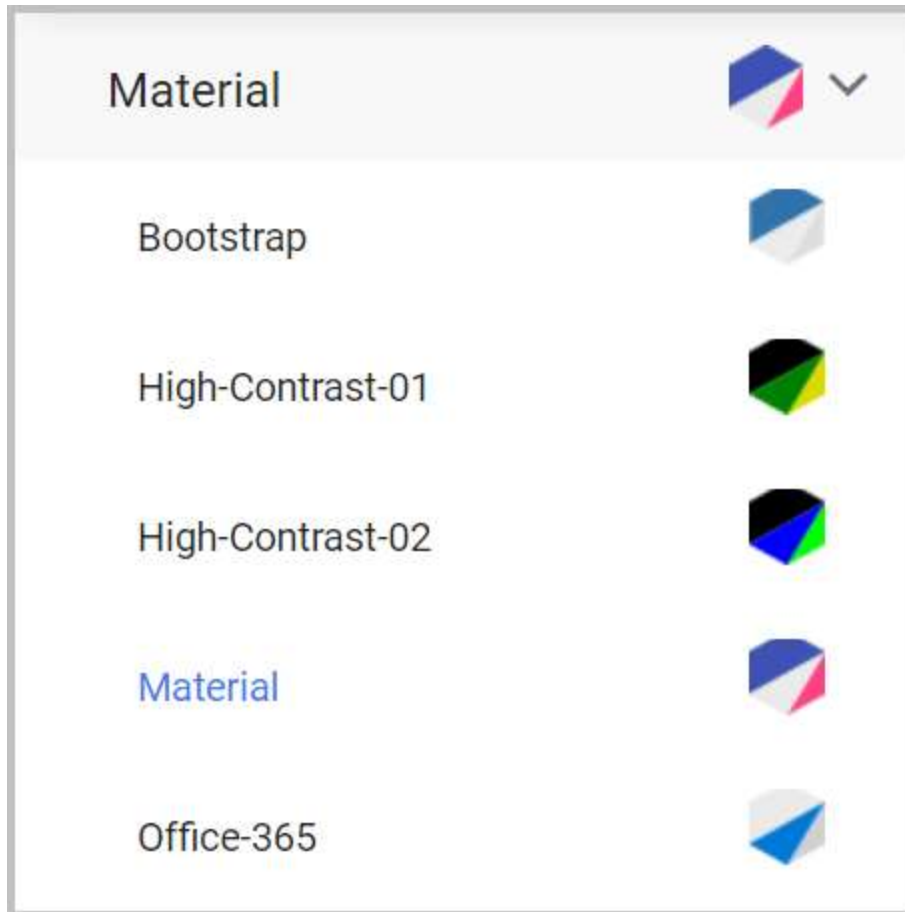
Create a new theme

We can create a [new theme](#) based on the existing Bold Reports theme. In this section, We will explain about creating a new theme, customizing colors, filtering Bold Reports control, downloading the customized theme, importing the customized theme for further customization and using the customized theme in web application. You can customize the report using our online [theme studio](#)

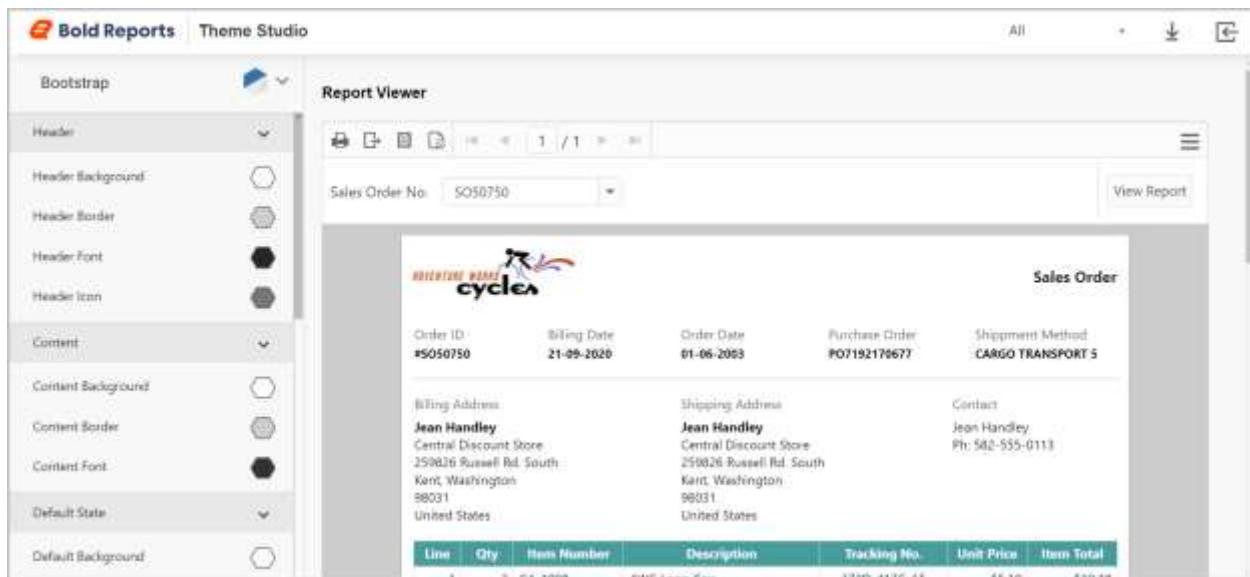
The customized theme will be used for web applications only not supported for UWP and WPF platforms.

Choose an existing Bold Reports theme

To create a new theme, we need to choose any one of the desired built-in themes from the theme switcher. Bold Reports Theme Studio comes with Bootstrap, Material, High Contrast 1, High Contrast 2, and Office 365.



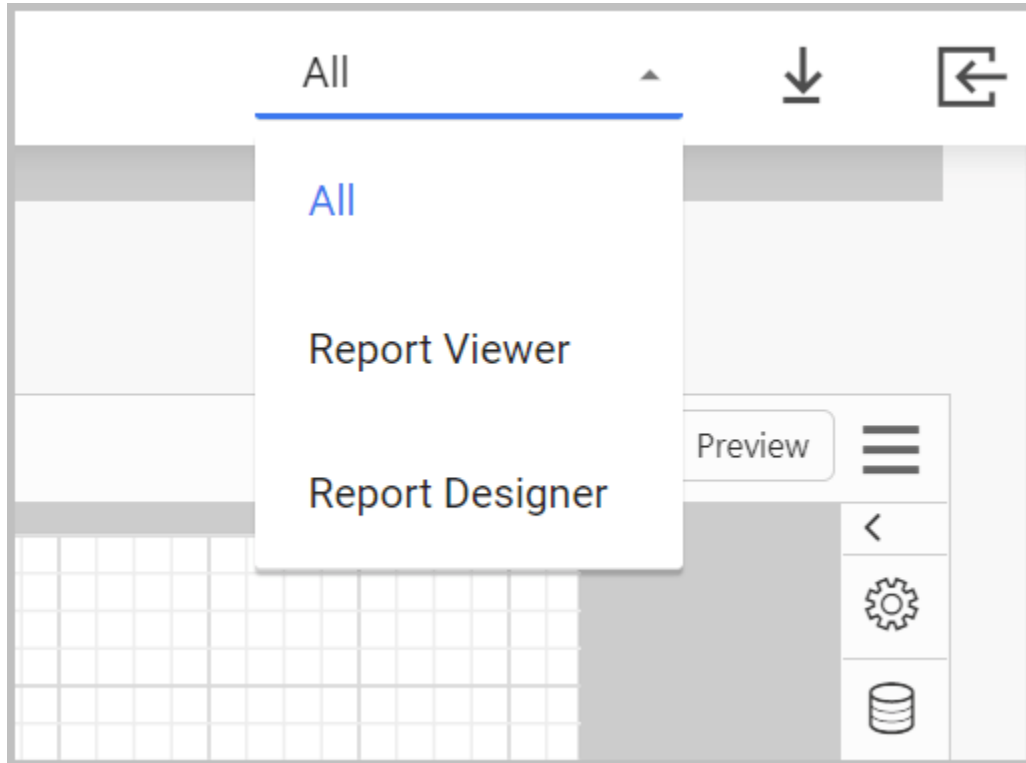
For demonstration, we are choosing **Bootstrap** theme.



Filter Bold Reports control

Bold Reports Theme Studio provides a filter option to customize theme for the specific Bold Reports controls. We have following options in filter.

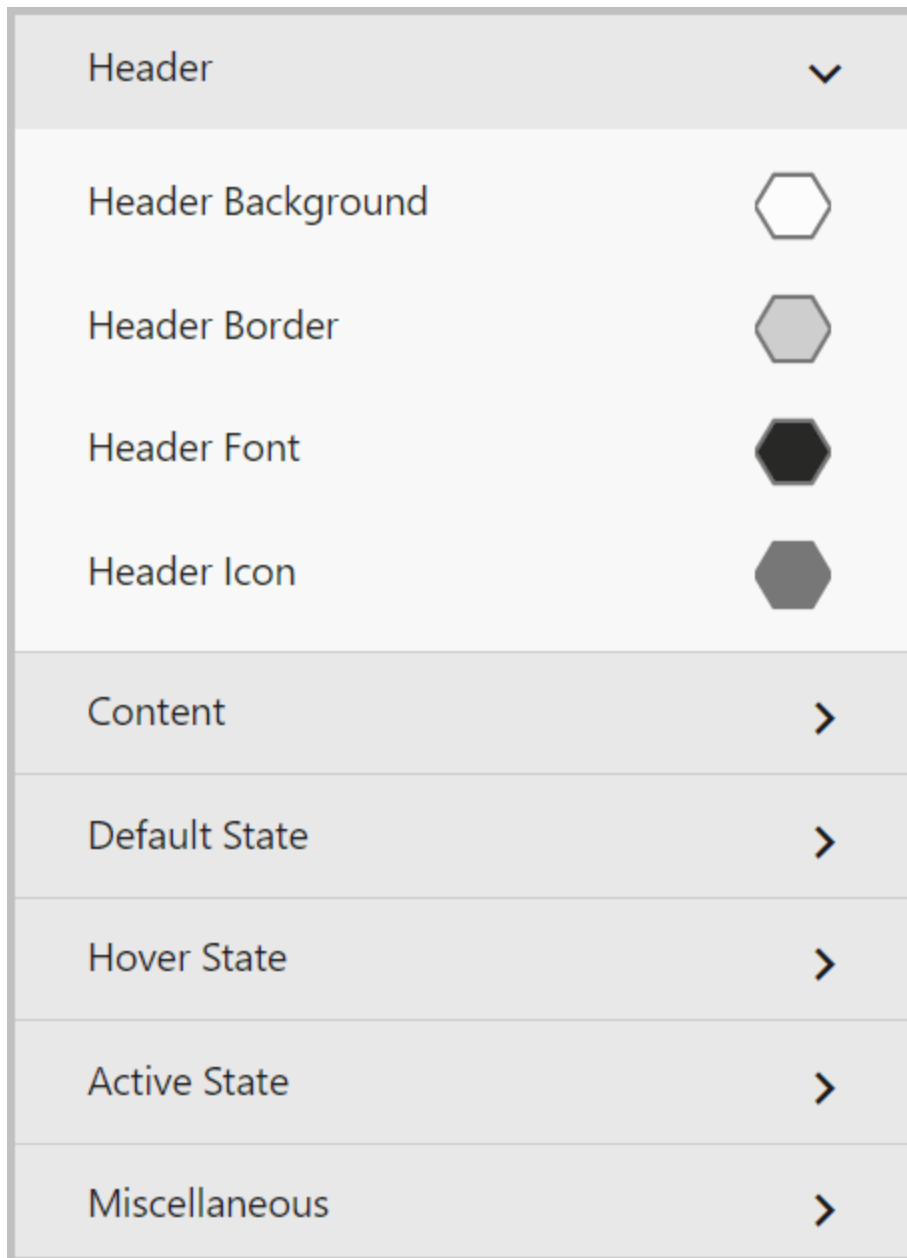
- All - Both Report Viewer and Report Designer will be customized. This is the default selected option.
- Report Viewer - Bold Report Viewer will be customized.
- Report Designer - Bold Report Designer will be customized.



Customize property

We can customize the selected theme properties from the customization property panel. Customization option will differ for each built-in themes. In our selected Bootstrap theme, we have following options for customization.

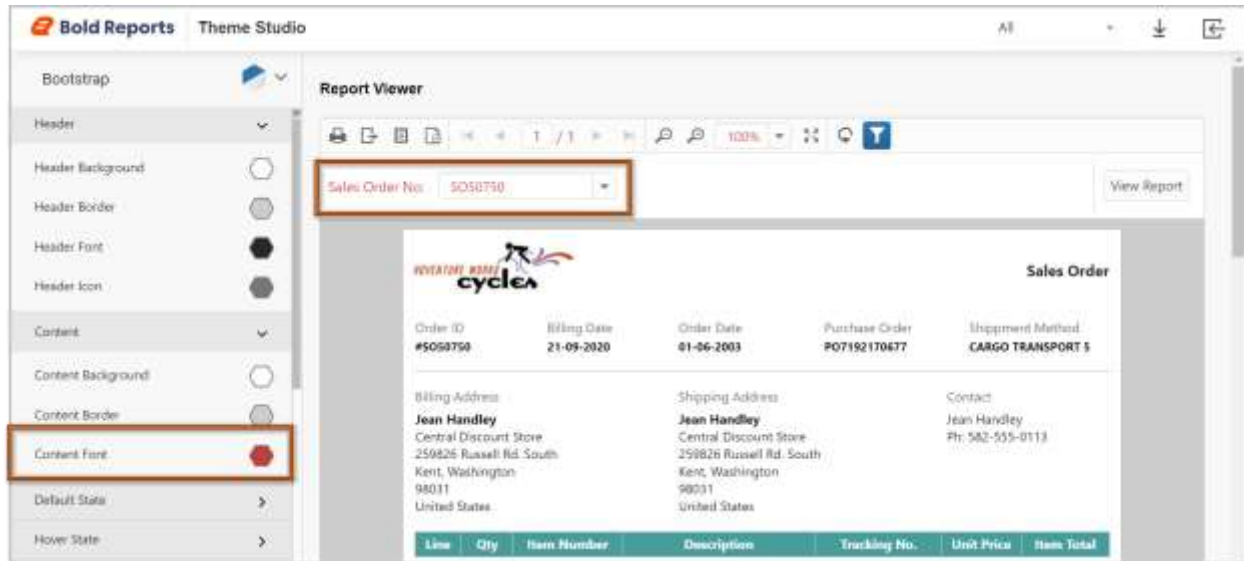
- Header
- Content
- Default State
- Hover State
- Active State
- Miscellaneous



Here, we are customizing content font color to red from black. This will automatically refresh the reporting controls to preview our customized theme.

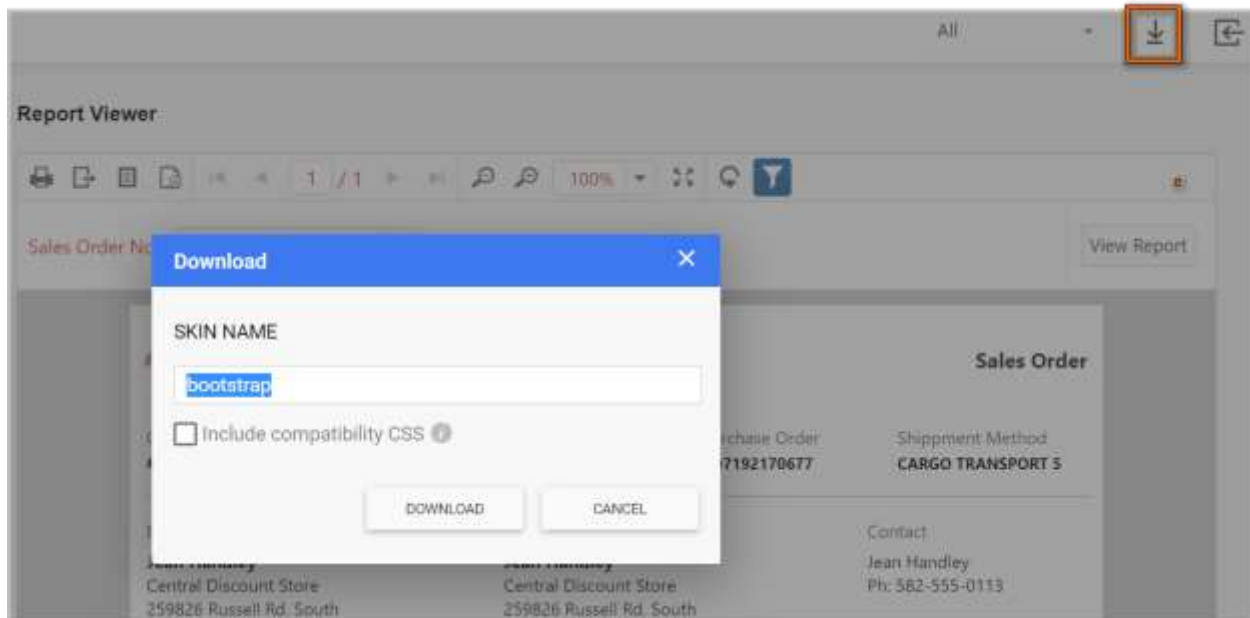
Create a new theme

Download the customized theme



[Download the customized theme](#)

Once customized the theme for our reporting controls, we can download the customized theme files directly through the **Download** option.



We have also provided compatibility CSS to use both Bold Reports and [Essential JS 2](#) controls in a same application. If you are using Bold Reports and Essential JS 2 controls in a same application, we strongly suggest you to use the compatibility CSS in your application by checking **Include compatibility CSS**.

The downloaded theme consists of following files.

|Files|Details|

|-----|-----|

|properties.json| It contains the configuration of the customized theme. we can reuse the customized theme by importing this file in theme studio and customize the theme further. |

|bootstrap-theme/bold.reports.all.min.css, bootstrap-theme/bold.reportdesigner.min.css| Its a minified Report viewer and Designer theme files. |














|bootstrap-theme/bold.reports.all.compatibility.min.css, bootstrap-theme/bold.reportdesigner.compatibility.min.css| Its a minified Report viewer and Designer compatibility theme files. |

|unmin/bold.reports.all.css, unmin/bold.reportdesigner.css| Its an unminified Report viewer and Designer theme files. |

|unmin/bold.reports.all.compatibility.css, unmin/bold.reportdesigner.compatibility.css| Its an unminified Report viewer and Designer compatibility theme files. |

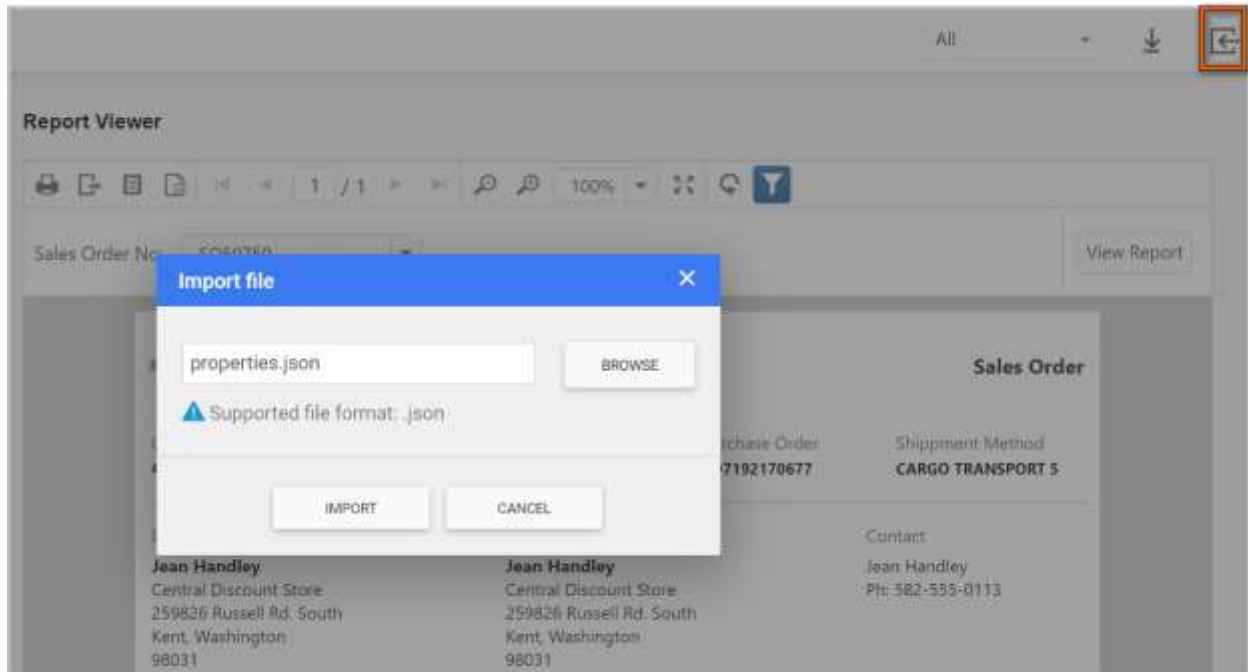
|less/bold.reports.all.less, less/bold.reportdesigner.less| Its a Report viewer and Designer less files. We can manually change the color code in this LESS file and generate the CSS by using LESS to CSS generator. |

|less/bold.reports.all.compatibility.less, less/bold.reportdesigner.compatibility.less| Its a Report viewer and Designer compatibility LESS files. We can manually change the color code in this LESS file and generate the CSS by using LESS to CSS. |

 bootstrap-theme	File folder
 common-images	File folder
 less	File folder
 unmin	File folder
 bold.reportdesigner.core.bootstrap	Cascading Style Sheet Docum...
 bold.reportdesigner.core.bootstrap.c...	Cascading Style Sheet Docum...
 bold.reportdesigner.core.bootstrap.c...	Cascading Style Sheet Docum...
 bold.reportdesigner.core.bootstrap.min	Cascading Style Sheet Docum...
 bold.widget.core.bootstrap	Cascading Style Sheet Docum...
 bold.widget.core.bootstrap.compatibi...	Cascading Style Sheet Docum...
 bold.widget.core.bootstrap.compatibi...	Cascading Style Sheet Docum...
 bold.widget.core.bootstrap.min	Cascading Style Sheet Docum...
 properties	JSON Source File

Import the customized theme

We can also customize the downloaded theme further by importing **properties.json** file in theme studio. After customizing the theme, we can again download the newly customized theme.



Use the downloaded CSS in the application

The downloaded folder will contains minified CSS file which suits for production sites and unminified CSS file which suits for development sites.

Referring minified CSS

- Create the following folders in the same structure under your application folder.

```
`bash
```

```
content/bold-reports/{theme specific folder}
```

```
`
```

- Copy {downloaded folder}/*.css to content/bold-reports/.
- Copy {downloaded folder}/{theme specific folder}/* to content/bold-reports/{theme specific folder}/.
- Copy {downloaded folder}/common-images/* to content/bold-reports/common-images/.

let say, we have downloaded bootstrap theme and the theme specific folder for bootstrap is **bootstrap-theme**. then, we need to refer the following style references in the application.

```
`html
```

```
<!-- Report Viewer -->
```

```
<link href="content/bold-reports/bootstrap-theme/bold.reports.all.min.css" rel="stylesheet" />
```

```
<!-- Report Designer -->
```

```
<link href="content/bold-reports/bootstrap-theme/bold.reportdesigner.min.css" rel="stylesheet" />
```

,

For compatibility, we need to refer the following style references instead of the above style references.

```
`html
```

```
<!-- Report Viewer -->
```

```
<link href="content/bold-reports/bootstrap-theme/bold.reports.all.compatibility.min.css"
rel="stylesheet" />
```

```
<!-- Report Designer -->
```

```
<link href="content/bold-reports/bootstrap-theme/bold.reportdesigner.compatibility.min.css"
rel="stylesheet" />
```

,

Referring unminified CSS

- Create the following folders in the same structure under your application folder.

```
`bash
```

```
content/bold-reports/{theme specific folder}
```

,

- Copy {downloaded folder}/*.css to content/bold-reports/.
- Copy {downloaded folder}/unmin/* to content/bold-reports/{theme specific folder}/.
- Copy {downloaded folder}/common-images/* to content/bold-reports/common-images/.

let say, we have downloaded bootstrap theme and the theme specific folder for bootstrap is **bootstrap-theme**. then, we need to refer the following style references in the application.

```
`html
```

```
<!-- Report Viewer -->
```

```
<link href="content/bold-reports/bootstrap-theme/bold.reports.all.css" rel="stylesheet" />
```

```
<!-- Report Designer -->
```

```
<link href="content/bold-reports/bootstrap-theme/bold.reportdesigner.css" rel="stylesheet" />
```

,

For compatibility, we need to refer the following style references instead of the above style references.

```
`html
```

```
<!-- Report Viewer -->
```

```
<link href="content/bold-reports/bootstrap-theme/bold.reports.all.compatibility.css" rel="stylesheet"
/>
```

```
<!-- Report Designer -->
```

```
<link href="content/bold-reports/bootstrap-theme/bold.reportdesigner.compatibility.css"
rel="stylesheet" />
```

Overview

Bold Reports provides a Source Code Add-On installer that allows you to modify the Syncfusion source code and use customized assemblies. This installer contains all of the Essential Studio sources, and you can easily build the customized source. The Source License is required in order to download and install the Source Code Add-on installer.

Here, you can explore the deployment of the Bold Reports Source Code Add-On in detail.

[Prerequisites](#) - Describes the system requirements of Bold Reports Source Code Add-On.

[Download and Installation](#) - Provides information on login and download and installation of Bold Reports Source Code Add-On.

[Generate Nugets](#) - Provides the list of Nugets and how to build and generate Nuget packages in Bold Reports Source Code Add-On.

Overview

Bold Reports provides a Source Code Add-On installer that allows you to modify the Syncfusion source code and use customized assemblies. This installer contains all of the Essential Studio sources, and you can easily build the customized source. The Source License is required in order to download and install the Source Code Add-on installer.

Here, you can explore the deployment of the Bold Reports Source Code Add-On in detail.

[Prerequisites](#) - Describes the system requirements of Bold Reports Source Code Add-On.

[Download and Installation](#) - Provides information on login, download and installation of Bold Reports Source Code Add-On.

[Generate Nugets](#) - Provides the list of Nugets and how to build and generate Nuget packages in Bold Reports Source Code Add-On.

System Requirements

This topic describes the software and hardware requirements for the source code add-on setup in the Bold Reports Source Code Add-On.

Hardware Environments

The following hardware environments are necessary to run the Bold Reports Source Code Add-On.

- 2.4 GHZ or faster, 32 bit or 64 bit processor.
- 8 GB RAM for 32 bit or 2 GB RAM for 64 bit.
- 1.68 GB Hard Disk space (Installation files).

Development Environments

The following development environments are necessary to run the Bold Reports Source Code Add-On.

- [SQLCE 3.5, 4.0](#)

- [NODE JS 10.x or above](#)
- [Visual Studio Build Tools - MSBuild 15.0](#)
- SDK for Bold Reports Projects
- [ASP .NET MVC SDK](#)
- [.NET CORE SDK 2.1, 3.1, 5.0, 6.0](#)
- [.NET FrameWork SDK 4.0, 4.5, 4.5.1, 4.6](#)
- [.NET](#)
- [UWP SDK](#)
- IIS (Internet Information Services)
- [IIS Manager](#)
- [IIS Express](#)
- GULP - `npm i gulp@3.9.1 -g`
- GATSBY JS - `npm i gatsby-cli -g`

See Also

[Download and Installation](#)

[Generate Nugets](#)

Download and installation of Bold Reports Source Code Add-On

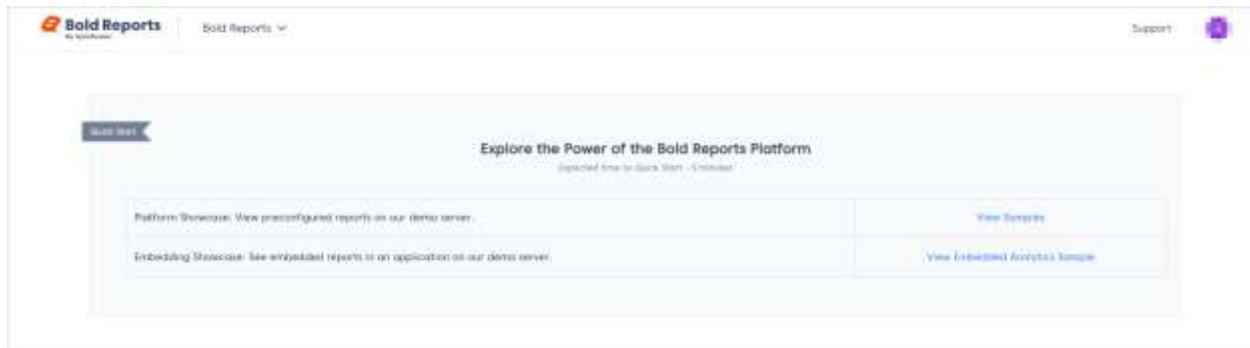
This section briefly describes the steps involved in the download and installation of the Bold Reports Source Code Add-On setup. Before installing the Bold Reports Source Code Add-On, make sure that your machine meets the [requirements](#).

Register and Download

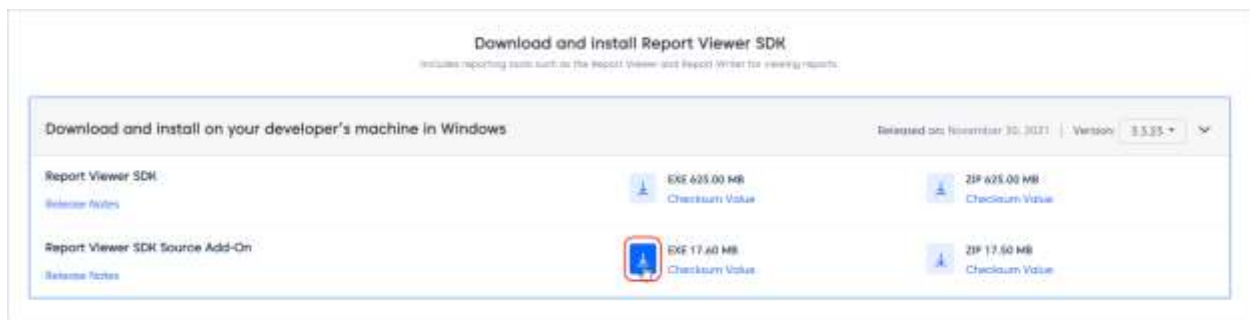
1. Go to the Bold Reports [Home](#) page to login.



2. Sign up to create a new account creation or sign in with an existing account.
3. You will be redirected to your downloads page.

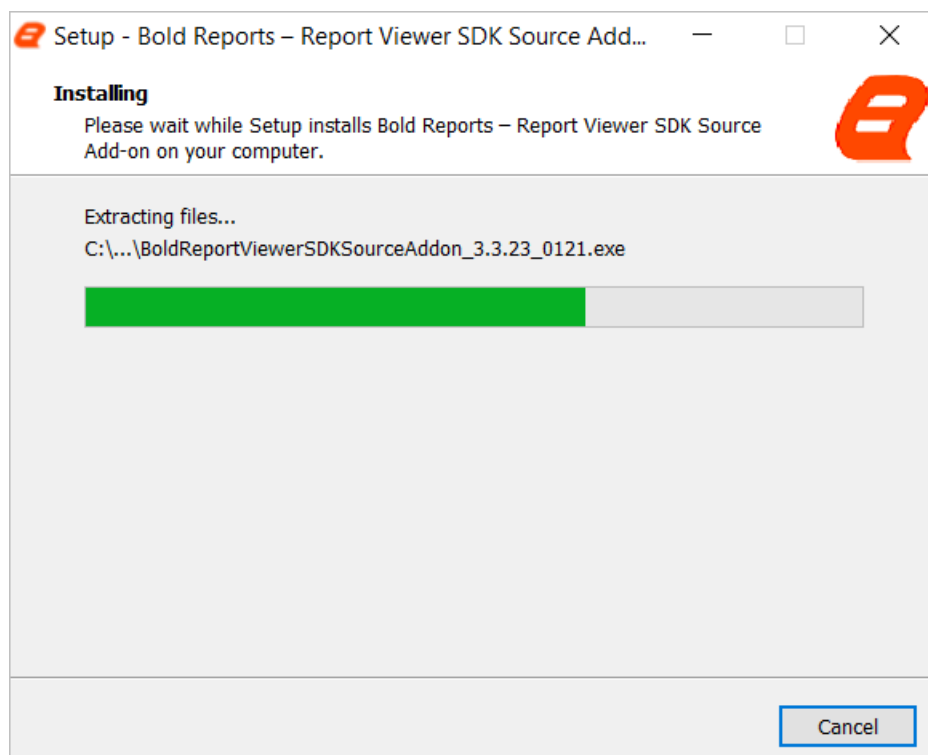


4. By clicking the download button, you can download the Bold Reports Source Code Add-On exe file.



Step-by-Step Installation

1. Once the setup is downloaded, run the installer from the saved location by clicking the **Run** button or by double-clicking the EXE file.



2. Sign-in with your registered e-mail address to unlock the setup.

Bold Reports

Bold Reports - Report Viewer SDK Source Add-on

Version: 3.3.23 Created on: 11/30/2021

Thank you for choosing Bold Reports - Report Viewer SDK Source Add-on.
Please enter the required information to proceed further.

Login to install Offline install

Email

Password

[Forgot password?](#)

or Sign in with

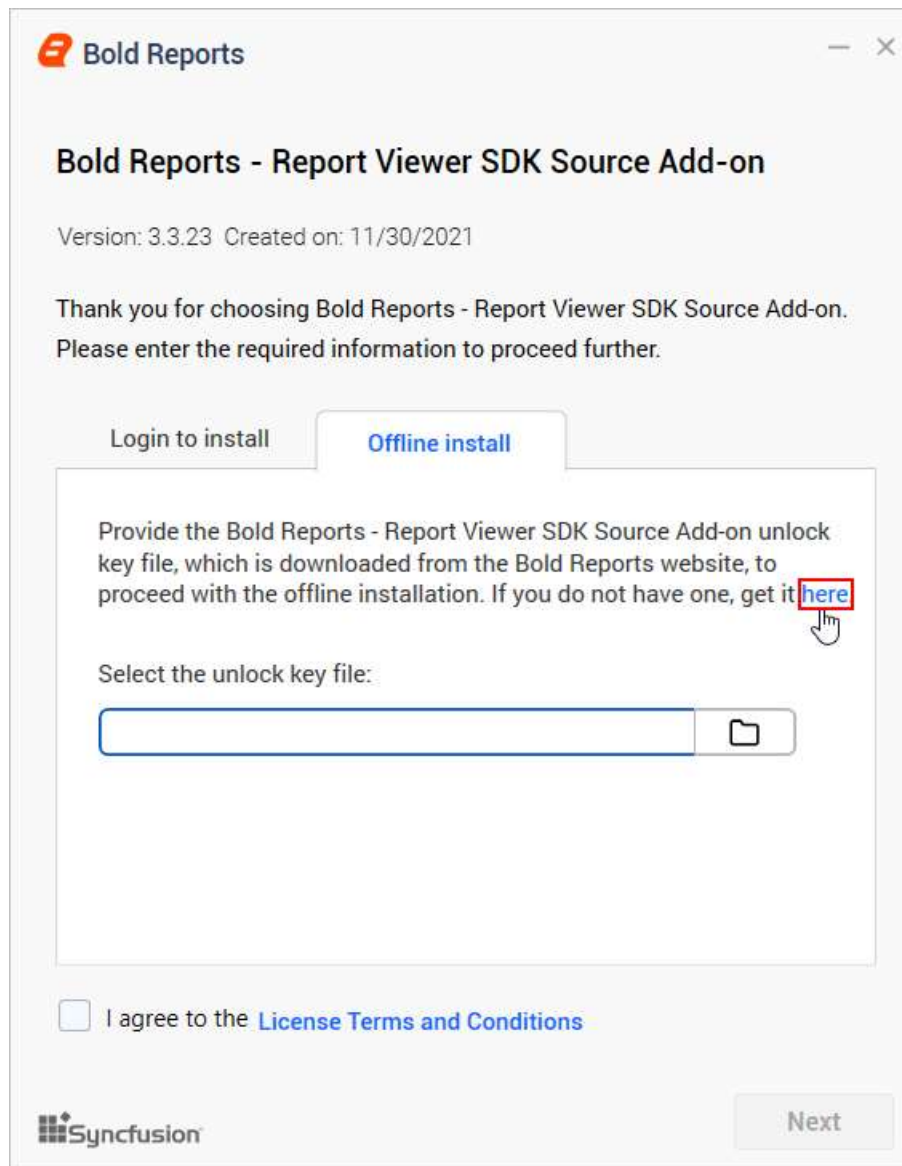
Microsoft ADFS

☐ I agree to the [License Terms and Conditions](#)

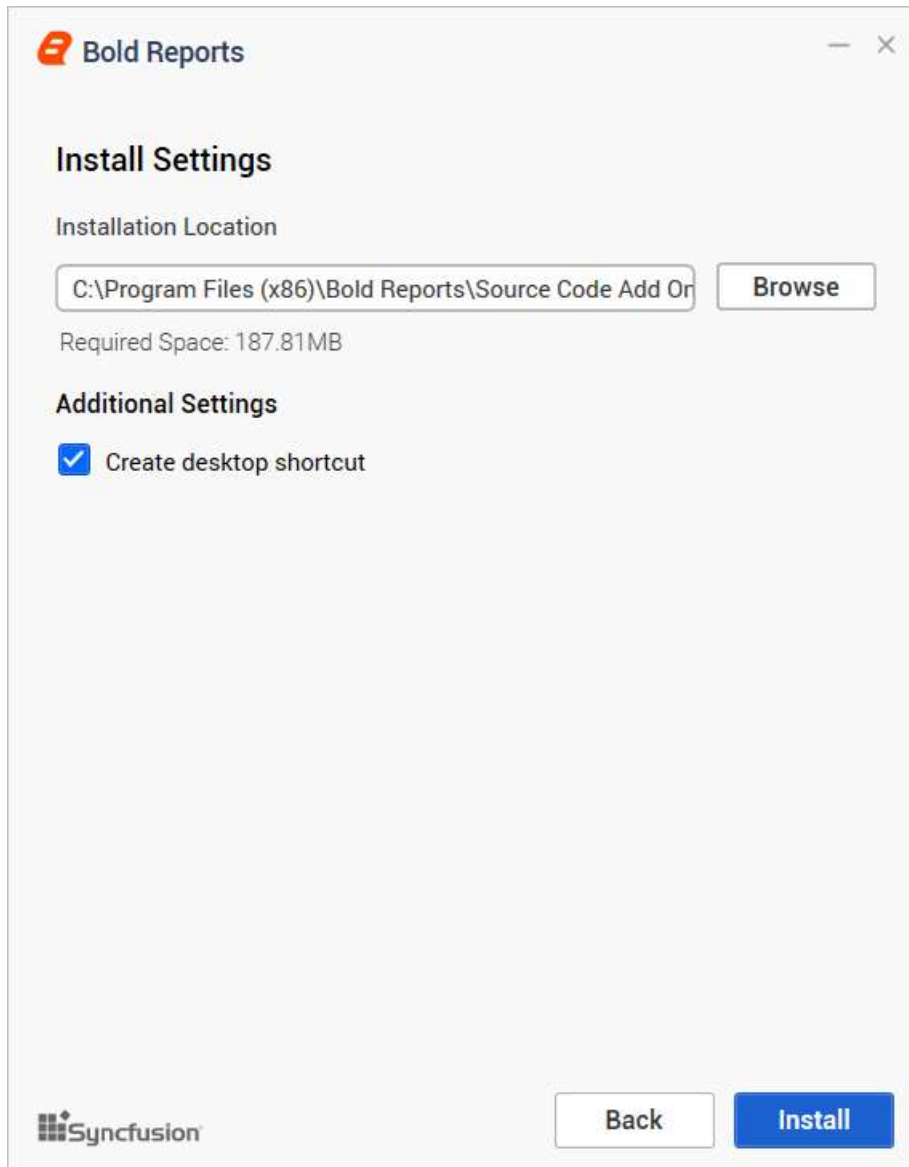
Next

Syncfusion

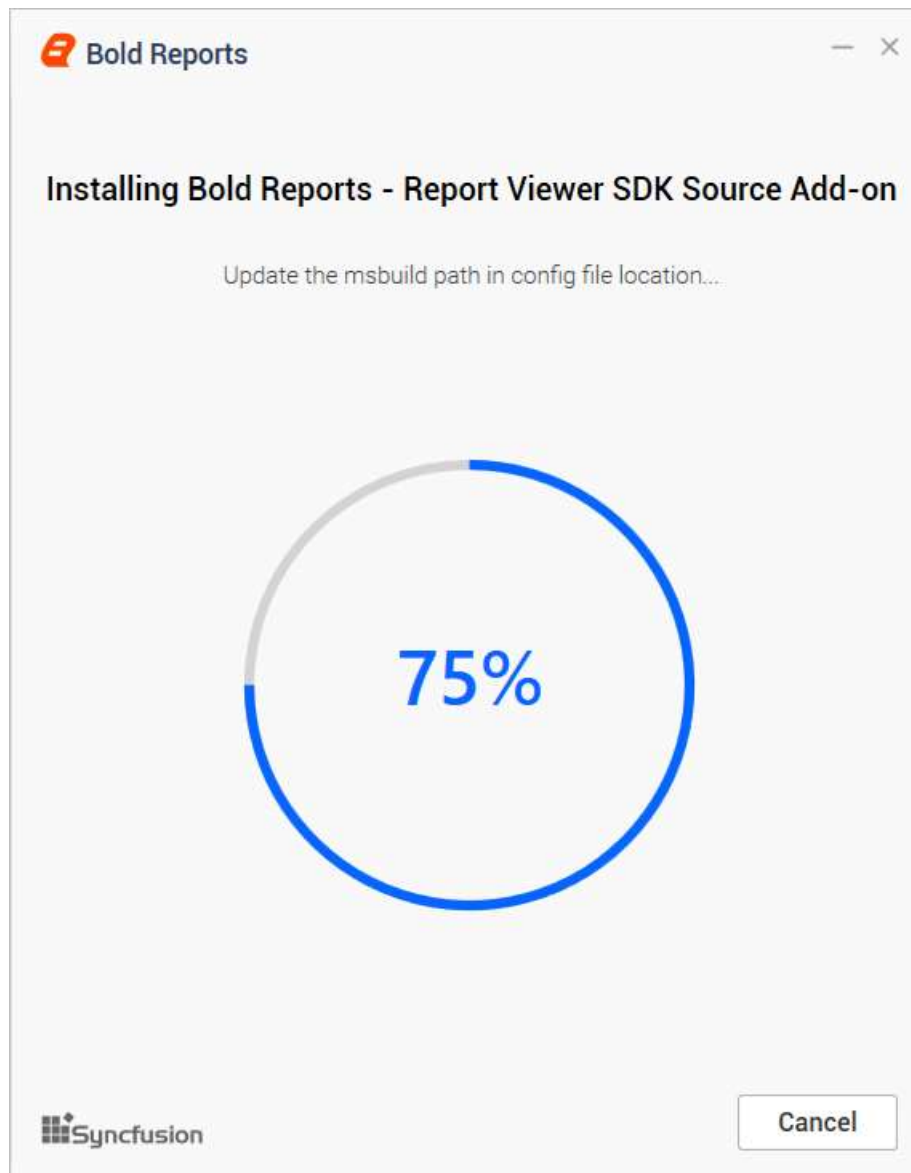
- Using an offline unlock key, you can also unlock the setup. If you don't have the unlock key, you can download it by clicking the [here](#) option as shown in the below image. It will redirect you to the Bold Reports website page and then expand the **Subscriptions** section. Then, click [Get Unlock Key File](#). A file will be downloaded with the file extension **.lic**. To unlock the setup, upload the downloaded **.lic** file. Then, enable the license terms and conditions checkbox and click **Next**.



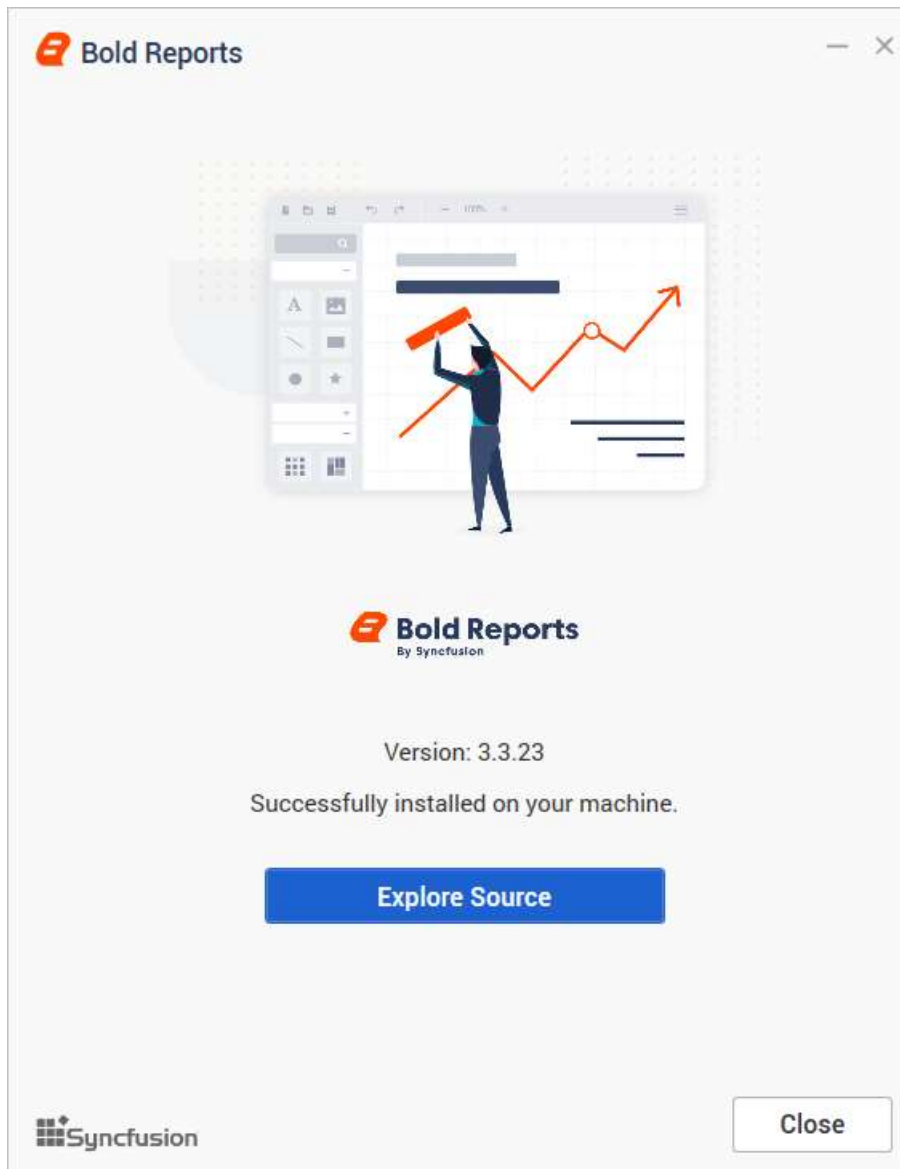
4. Choose the location where you would like to install the Bold Reports Source Code Add-On setup and samples. If you want to perform additional tasks like desktop shortcut creation, you can check the option. Otherwise, you can uncheck it and click Install.



5. Now, the installation begins. You can cancel the installation at anytime by pressing **Cancel**, if you prefer.



6. On successful installation, the below screen appears. Click the **Explore Source** button to run the Sample Browser or Explore Samples.



See Also

[Generate Nugets](#)

Generate Nuget

This topic explains how to build the source code add-on and how to generate Nuget packages in the source code add-on.

Build

The below command will output a Source Code Add-On folder inside the base path.

```
`console
```

```
npm install & gulp source-code-addon
```

```
`
```

After successfully executing the above command, you will get the compiled folder.

Folder Structure

```
`console
-->Base(1)
-->build(9)
-->compressor(2)
-->config(1)
-->js(5)
-->cireports(11)
-->Common(2)
-->nuget-spec(8)
-->UWP(2)
-->Web
-->JavaScript(8)
-->Reporting.web(2)
-->Reports.ASPNET(1)
-->WPF
`
```

Generate Nuget-packages

1. Go to the **Source Code Add On** path.

{system drive}:\Program Files (x86)\Bold Reports\Source Code Add On

2. Open the **config.json** file in the **Src** folder.
3. Mention the Nuget packages name in the **config.json** under the **nugetPackages** array, since support is provided for the below mentioned Nuget packages.

```
`console
```

- "BoldReports.Web",
- "BoldReports.UWP",
- "BoldReports.Net.Core",
- "BoldReports.WPF",
- "BoldReports.AspNet.Core",
- "BoldReports.Mvc4",
- "BoldReports.Mvc5",
- "BoldReports.WebForms",
- "BoldReports.Global",
- "BoldReports.Javascript",
- "BoldReports.Javascript.Extensions"

`

4. Execute the below command to generate Nuget packages under `./Resources/nuget-packages`.

`console

`npm install & gulp build`

`